Real-Time Bidding with Multi-Agent Reinforcement Learning in Display Advertising

Junqi Jin[§], Chengru Song [§], Han Li[§], Kun Gai[§], Jun Wang[†], Weinan Zhang[‡]
[§]Alibaba Group, [†]University College London, [‡]Shanghai JiaoTong University
{junqi.jjq,chengru.scr,lihan.lh,jingshi.gk}@alibaba-inc.com,jun.wang@cs.ucl.ac.uk,wnzhang@sjtu.edu.cn

ABSTRACT

Real-time advertising allows advertisers to bid for each impression for a visiting user. To optimize a specific goal such as maximizing the revenue led by ad placements, advertisers not only need to estimate the relevance between the ads and user's interests, but most importantly require a strategic response with respect to other advertisers bidding in the market. In this paper, we formulate bidding optimization with multi-agent reinforcement learning. To deal with a large number of advertisers, we propose a clustering method and assign each cluster with a strategic bidding agent. A practical Distributed Coordinated Multi-Agent Bidding (DCMAB) has been proposed and implemented to balance the tradeoff between the competition and cooperation among advertisers. The empirical study on our industry-scaled real-world data has demonstrated the effectiveness of our modeling methods. Our results show that a cluster based bidding would largely outperform single-agent and bandit approaches, and the coordinated bidding achieves better overall objectives than the purely self-interested bidding agents.

KEYWORDS

Bid Optimization, Real-Time Bidding, Multi-Agent Reinforcement Learning, Display Advertising

1 INTRODUCTION

Online advertising [5, 9] is a marketing paradigm utilizing the Internet to target audience and drive conversions. Real-time bidding (RTB) [26] allows advertisers to bid for every individual impression in realtime when being generated. A typical RTB ad exchange employs the second price sealed-bid auction [30], and in theory (under strong assumptions) the second price auction would encourage truthful bidding. In practice, however, the optimal or equilibrium bids are largely unknown, depending on various factors, including the availability of market bid prices, the existence of budget constraints, performance objectives, (ir)rationality of opponent bidders. As such, how to *strategically* optimize bidding becomes a central question in RTB advertising [31].

The research on optimal bidding strategies so far has been focused largely on statistical solutions, making a strong assumption that the market data is *stationary* (i.e. their probability distribution does not change over time in response to the current bidder's behaviors) [1, 21, 28, 32, 33]. Specially, Zhang et al. [32] shows that budget-constrained optimal bidding can be achieved under the condition that the environment (along with other ad bidders) is stationary. Zhu et al. [33] proposes a two-stage bandit modeling where each bidding decision is independent over time. Cai et al. [1] and Wang et al. [28] leverage reinforcement learning to model the bid optimization as a sequential decision procedure. Nonetheless, in ad auctions, ad campaign bidders not only interact with the auction environment but, most critically, with each other. The changes in the strategy of one bidder would affect the strategies of other bidders and vice versa [25]. In addition, existing computational bidding methods [21, 32] are mainly concerned with micro-level optimization of one party (a specific advertiser or merchant)'s benefit. But given the competition in the RTB auction, optimizing one party's benefit may ignore and hurt other parties' benefits. From the ad system's viewpoint, the micro-level optimization may not fully utilize the dynamics of the ad ecosystem in order to achieve better social optimality [28, 33].

In this paper, we address the above issue by taking a game-theoretical approach [20]. RTB bidding is solved by multi-agent reinforcement learning (MARL) [12], where bidding agents interactions are modeled. A significant advantage over the previous methods [1, 21, 28, 32, 33] is that our proposed MARL bidding strategy is *rational* as each bidding agent is motivated by maximizing their own payoff; it is also *strategic* as each bidding agent will also provide a best response to the strategic change of other bidders to eventually reach to an equilibrium stage.

Our study is large-scale, and is developed in the context of a realistic industry setting in Taobao (taobao.com), one of the largest e-commerce platform in China. Taobao serves more than four hundred million active users. To our best knowledge, this is the first study of employing MARL for such large scale online advertising case, evaluated over real data. Previous studies on MARL are mostly in the theoretical nature, and the majority experiments are done by simulated games [12, 17]. Our RTB bidding can be considered one of the earliest realistic applications of MARL.

Modeling large scale bidding by MARL is, however, difficult. In Taobao e-commerce platform, there are a large number of consumers and merchants. Modeling each merchant as a strategic agent is computationally infeasible. To tackle this issue, we propose that bidding agents operate in the clustering level. We cluster consumers into several groups, each of which is considered as a "super-consumer", and also cluster merchants into groups, each of which is represented by a common bidding agent. The multi-agent formulation is thus based on the interactions between super-consumers and cluster-level bidding agents, as well as the interactions among bidding agents. A technical challenge is the convergence of MARL as all the cluster bidding agents explore the auction system simultaneously, which makes the auction environment non-stationary and noisy for each agent to learn a stable policy. Inspired by multi-agent deep deterministic policy gradient (MADDPG) techniques [17], we propose Distributed Coordinated Multi-Agent Bidding (referred as DCMAB) method to stabilize the convergence by feeding all agents' bidding actions to the Q function. During learning, each bidding agent's Q function evaluates future value according to all agents' actions rather than only itself's action.

Our solution is fully distributed, and has been integrated with Taobao's distributed-worker system, which has high-concurrency and asynchronous requests from our consumers. Experiments are conducted on real world industrial data. The results demonstrate our DCMAB's advantage over several strong baselines including a deployed baselines in our system.

We also find that when bidding agents act from only self-interested motivations, the equilibrium that converged to may not necessarily represent a socially optimal solution [14, 27]. We thus develop a fully coordinated bidding model that learns the strategy by specifying a common objective function as a whole.

The empirical study shows our DCMAB's ability of making merchants coordinated to reach a higher cooperative goal.

2 RELATED WORK

Bid Optimization in RTB. Bidding optimization is one of the most concerned problems in RTB, which aims to help advertiser set right bidding price for each auctioned impression to maximize the campaign's key performance indicator (KPI) such as click or profit [26]. Perlich et al. [21] first introduced a linear bidding strategy based on impression evaluation, which has been widely used in real-word applications since then. Zhang et al. [32] went beyond linear formulation. They found out the non-linear relationship between the optimal bid and the impression evaluation, and derived effective non-linear bidding functions. These methods regard bidding optimization as a static problem, thus fail to deal with dynamic situations and the rationality of bidding agents.

A more intelligent bidding strategy manages to optimize the KPI under certain constraints and make real-time adaption during campaign's lifetime, most of which are met with reinforcement learning. Cai et al. [1] formulated a Markov Decision Process (MDP) bidding framework to learn how to sequentially allocate campaign budget along impressions, with impression evaluation and campaign's real-time conditions as state. Du et al. [3] tackled budget constraint by means of Constrained MDP. Wang et al. [28] utilized deep reinforcement learning, specifically DQN, to optimize the bidding strategy in DSP. They set high-level semantic information as state, and consider no budget constraint. These tasks share a common task setting, i.e., bid optimization serves for one single advertiser, with its competitors as part of the environment, which significantly differs from our settings.

Another popular method for budget allocation is the pacing algorithm [13, 29] which smooths budget spending across time according to traffic intensity fluctuation. Compared with our method, pacing can be considered as a single agent optimization method which does not explicitly model the influence from other agents' actions in the auction environment. In addition, pacing cannot coordinate agents to cooperate for a better equilibrium.

Like many other ad exchanges, in Taobao display advertising system, we treat advertisers equally. Meanwhile, we need to balance the interests among consumers, advertisers and the platform. Therefore, we are motivated to construct a framework that simultaneously takes different interests into consideration. Advertisers compete with each other for high quality impressions, while they should cooperate in the sense of providing better user experience and improving the overall revenue of the platform. In our work, we adopt multi-agent reinforcement learning to achieve such a goal.

Multi-agent Reinforcement Learning. In multi-agent literature, how to design mechanisms and learning algorithms to make agents well cooperate is the focus. Tan [25] compared cooperation with independent Q-learning, drawing the conclusion that additional information from other agents, if used properly, is beneficial for a collective reward. Many studies afterwards focused on how to effectively coordinate agents to achieve the common goal, either by means of sharing parameters [10] or learning communication protocol [7, 19]. Some of these studies [7, 10] adopted the framework of centralized training with decentralized execution, allowing for involving extra information to ease training. Lowe et al. [17] studied further in this direction and proposed MADDPG (Multi-agent DDPG), in which the centralized critic is augmented with policies of other agents. However, MADDPG was applied in a toy simulation environment where the states update and transition tuple saving can be performed frequently.

In our task, the most serious challenge we face is that there are a huge number of advertisers in Taobao display advertising, which exceeds the processing capacity of almost all current multi-agent reinforcement learning methods. Furthermore, if we model each advertiser as an individual agent, the reward would be sparse for most agents. Besides, our bidding system is implemented on distributed workers which process requests in parallel and asynchronously. Considering all these factors, we extend the deterministic policy gradient (DPG) algorithm [16, 17, 22] to our solution with improvements including 1) a clustering method to model a large number of merchants as multiple agents and 2) distributed architecture design to enable our framework to process requests in distributed workers in parallel and asynchronously.

3 TAOBAO DISPLAY AD SYSTEM

In Taobao ad ecosystem, the advertisers are mostly the merchants who not only advertise but also sell their products on Taobao's e-commerce platform. In the remaining of this paper, we call them merchants. Taobao ad system can be divided into three parts as shown in Figure 1: First in the matching stage, user preferences are obtained by mining behavior data, and when receiving a user request, matching part recalls candidate ads (typically in the order of hundreds) from the entire ad corpus in real time based on their relevancy. Different from recommender systems, the recall of the ads also have to reflect the advertisers' willingness to join the bid, i.e., their behavior targeting settings. Second, the follow-up real-time prediction (RTP) engine predicts the click-through rate (pCTR) and the conversion rate (pCVR) for each eligible ad. Third, after real-time bidding for each candidate ad is received, these candidate ads are ranked by the descending order of $bid \times pCTR$, which is called effective cost-per-mille (eCPM) sorting mechanism. Finally, the final ranked ads are displayed. For a general RTB auction setting, we refer to [26].

As stated above, the change of bids will influence the ranking of candidate ads, and further have the impact on the connections built between the consumers and merchants. An ideal mapping is that the consumers find their ideal products and the merchants target the right consumers who have the intent to buy the advertised products. When demands are precisely met by the supplies, the platform creates higher connection value for the society and in the

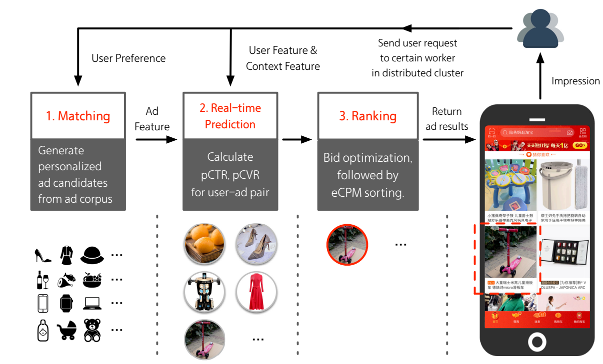

Figure 1: An Overview of Taobao Display Advertising System. Matching, RTP and Ranking modules sequentially process user requests, and finally return specified quantity of ads. These ads are shown in *Guess What You Like* of Taobao App, tagged by *Hot* (as shown in red dashed box) and surrounded with recommendation results.

meantime gets higher revenue from facilitating the matching. For better revenue optimization, merchants also authorize the platform to adjust their manually set bids within an acceptable range. In summary, bids as key control variables in the online advertising system and, if adjusted well, can achieve a win-win-win situation for all consumers, merchants and the platform's interest.

In our e-commerce RTB system, there are a large number of registered merchants and registered consumers. Each auction is launched by one consumer. According to information in this auction, each merchant under its budget constraint gives a bid price. If a merchant win an auction, the corresponding ad would be delivered to a consumer. This consumer has a probability to click the ad (click-through rate, CTR) to enter a detailed landing page for the product, and further has a probability (conversion rate, CVR) to buy the merchant's product with price *ppb* (pay-per-buy) forming the merchants' revenue. Given predefined budget to achieve higher revenue is a general goal of merchants. To get higher total revenue is also consumers' and platform's motivations: for consumers, they are connected to the products they want while for the platform, larger gross merchandise volume (GMV) means larger long-term advertising revenue. Whenever a merchant's ad is clicked, the corresponding merchant unspent budget will be subtracted the advertising *cost* according to the generalized second price (GSP) auction with CPC mechanism [4]. If a merchant loses an auction, he gets no reward and pays nothing. If the merchant's budget runs out, the merchant will not participate in any rest auctions.

Bidding in display advertising is often regarded as an episodic process [1]. Each episode includes many auctions and each auction is about one consumer's page view in a very specific context. They are sequentially sent to the bidding agents. Each merchant's goal is to allocate its budget for the right consumers at the right time to maximize its total revenue. All the merchants competing together forms a multi-agent game. However, when budgets are limited, the game of merchants' bidding may result in a suboptimal equilibrium. For example, the merchants compete severely in early auctions and many merchants have to quit early. Thus the low competition depth in the late bidding results in low matching efficiency of consumers and merchants. Therefore, all merchants setting bids for different consumers in different time according to different competition environments is essential for Taobao ad system to achieve a socially optimal situation.

4 MULTI-AGENT ADVERTISING BIDDING

We first formulate RTB as a Stochastic Game and then present our MARL approach and finally discuss our implementation details.

4.1 RTB as a Stochastic Game

We formulate RTB as a *Stochastic Game*, a.k.a. Markov Game [6], where there are N bidding agents on behalf of merchants to bid ad impressions. A Markov game is defined by a set of states $\mathcal{S}$ describing the possible status of all bidding agents, a set of actions $\mathcal{A}_1, ..., \mathcal{A}_N$ where $\mathcal{A}_i$ represents action spaces of agent i. An action $a \in \mathcal{A}_i$ is the bid adjustment ratio. According to t-th timestep state s_t, each bidding agent i uses a policy $\pi_i : \mathcal{S} \mapsto \mathcal{A}_i$ to determine an action a_i. After the execution of a_i, the bidding agent i transfers to a next state according to the state transition function $\mathcal{T} : \mathcal{S} \times \mathcal{A}_1 \times ... \times \mathcal{A}_N \mapsto \Omega(\mathcal{S})$ where $\Omega(\mathcal{S})$ indicates the collection of probability distributions over the state space. Each agent i obtains a reward (i.e., revenue) based on a function of the state and all agents' actions as $r_i : \mathcal{S} \times \mathcal{A}_1 \times ... \times \mathcal{A}_N \mapsto \mathcal{R}$. The initial states are determined by a predefined distribution. Each agent i aims to maximize its own total expected return $R_i = \Sigma_{t=0}^{T} \gamma^t r_i^t$ where γ is a discount factor and T is the time horizon. We describe the details of agents, states, actions, rewards and objective functions in our setting as follows.

Agent Clusters. In our system, n registered merchants are denoted as $m_1, m_2, ..., m_n$ and l registered consumers are denoted as $c_1, c_2, ..., c_l$. Each auction is launched by one consumer with a feature x describing the consumer's information in this auction. The merchant's product's price is denoted as *ppb* (pay-per-buy). The ideal way to formulate all merchants is to model each of them as an agent. However, such arrangement is computationally expensive, and in fact there are sparse interactions between a specific pair of a consumer and a merchant. Also as the growth of the number of agents, the exploration noise becomes difficult to control. We thus propose a *clustering method* to model the involved entities. According to the cumulative revenue during a period, n merchants are categorized as N clusters $M_1, \ldots, M_N$. Similarly, l consumers are categorized as L clusters $C_1, \ldots, C_L$. We cluster consumers for building agents' states and computation for static features so as to enable the agents to evaluate features of auctions from different consumers and adjust bids accordingly. Hereinafter, we use i as subscript of merchant cluster, and j for consumer cluster. Normally $N \ll n$, $L \ll l$, and when we shrink the cluster size and enlarge the cluster number, it approximates the ideal case. The diagram of this modeling is shown in Figure 2.

State. Our state design aims to let bidding agents optimize their budgets allocation based on both each impression's value and spending trends along time. We denote cumulative cost and revenue between merchants M_i and consumers C_j from the beginning of an episode up to the decision moment as $g_{ij} = (cost_{ij}, revenue_{ij})$ as the general information state. This is because all these g_{ij} vectors characterize important information as: (1) the budget spent status for an agent to plan for the rest auctions; (2) the (cost, revenue) distribution of consumers for an agent to distinguish quality from different consumer clusters; (3) the (cost, revenue) distribution of

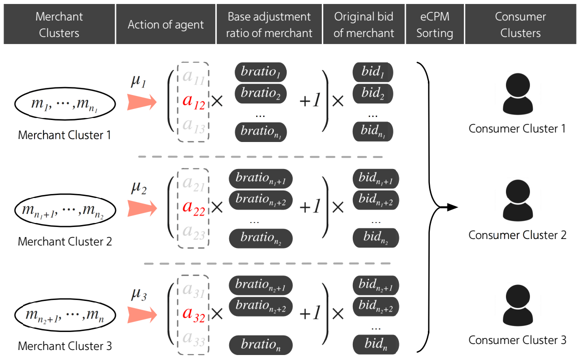

Figure 2: Merchants and consumers are grouped into clusters separately. Each merchant cluster is an agent, which adjusts ad bids of included merchants for different consumer clusters. For action a_{ij}, i iterates the number of merchant clusters, as j does for consumer clusters. $bratio_k$ stands for base adjustment ratio of merchant k.

other agents for an agent to evaluate the competitive or cooperative environment. Besides, the consumer feature x is also added which includes slowly-changed consumer features such as their total (cost, revenue) status updated every a period of time. This feature x helps agents evaluate the auction. We concatenate all g_{ij} as $g = [g_{11}, g_{12}, \dots, g_{NL}]$ with x to form the state $s = [g, x]$. We suppose each merchant's budget is predefined, therefore their spent and unspent budgets information is also maintained in the state. The diagram of this modeling is shown in Figure 3.

Action. Every merchant can manually set different fixed bids for different consumer crowds and different creatives. Without loss of generality, we denote the fixed bid as bid_k across all the auctions, where k iterates over n merchants hereinafter. For better budget allocation, the platform is authorized to adjust it with a scalar α to generate $bid_k \times (1 + \alpha)$ where $\alpha \in [-range, range], 0 < range < 1$ and we use $range = 0.9$ in our experiment. As stated above, we cluster n merchants into N clusters, then α should have N different values for different merchant clusters denoted as a_i, which is exactly the action of agent i. However, applying agent's action to all its included merchants is lack of flexibility. Hence the actually conducted bid adjust ratio is $a_i \times bratio_k$, as shown in Figure 2. The calculation of $bratio_k$ is predefined and we would discuss it in detail in Section 4.3.

Reward and Transition. Reward is defined on the agent level not on the merchant level. When a merchant k belonging to agent i executes bid and wins an auction with delivering an ad to consumer of C_j, the reward of agent i increases by the revenue directly caused by this ad from this consumer. And after the ad was clicked, the budget of merchant k decreases by $cost = pCTR_{next(k)} \times bid_{next(k)}/pCTR_k$ according to GSP mechanism where merchant $next(k)$ is the next ranked merchant of merchant k according to maximum eCPM ranking score of $pCTR \times bid$. The g_{ij} in state is updated by accumulating this $(revenue, cost)$. Changes of g_{ij} for all i, j including consumer feature x changing form the transition of the states. If a merchant loses the auction, then no contribution to its agent's reward.

4.2 Bidding by Multi-Agent RL

Since the output action (bid adjustment) is in a continuous space, we adopt deterministic policy gradient for learning the bidding strategy. In the MARL setting, the Q function for agent i is given as

$$Q_i^{\boldsymbol{\pi}}(s, \boldsymbol{a}) = \mathbb{E}_{\boldsymbol{\pi}, \mathcal{T}}[\Sigma_{t=0}^{T} \gamma^t r_i^t | s_0 = s, \boldsymbol{a}], \tag{1}$$

where $\boldsymbol{\pi} = \{\pi_1, ..., \pi_N\}$ denotes the joint policy across all agents and the joint action is given as $\boldsymbol{a} = [a_1, ..., a_N]$ and initial state is as s_0. RL makes use of the temporal difference recursive relationship known as the Bellman equation as:

$$Q_i^{\boldsymbol{\pi}}(s, \boldsymbol{a}) = \mathbb{E}_{r_i, s'}[r_i(s, \boldsymbol{a}) + \gamma \mathbb{E}_{\boldsymbol{a'} \sim \boldsymbol{\pi}}[Q_i^{\boldsymbol{\pi}}(s', \boldsymbol{a'})]]. \tag{2}$$

When policy is deterministic, with a deterministic mapping from state s to bidding action a as Eq.(3) for agent i with parameter θ_i^{μ},

$$a_i = \mu_i(s) = \mu_i([g, x]) \tag{3}$$

Above Eq.(2) becomes:

$$\begin{aligned} &Q_i^{\boldsymbol{\mu}}(s, a_1, ..., a_N) \\ &\quad = \mathbb{E}_{r_i, s'}[r_i(s, a_1, ..., a_N) + \gamma Q_i^{\boldsymbol{\mu}}(s', \mu_1(s'), ..., \mu_N(s'))], \end{aligned} \tag{4}$$

Eq.(3) is commonly called *actor*. In MARL, the goal is to learn an optimal strategy for each agent, which may have a different or even conflicted goal. The notion of Nash equilibrium [11] is important, which is represented as a set of policies $\mu^* = \{\mu_1^*, ..., \mu_2^*\}$ such that $\forall \mu_i$, it satisfies:

$$\begin{aligned} &Q_i^{\boldsymbol{\mu}^*}(s, \mu_1^*(s), ..., \mu_N^*(s)) \\ &\quad = Q_i^{\boldsymbol{\mu}^*}(s, \mu_i^*(s), \mu_{-i}^*(s)) \geq Q_i^{\boldsymbol{\mu}^*}(s, \mu_i(s), \mu_{-i}^*(s)), \end{aligned} \tag{5}$$

where we use compact notations for the joint policy of all agents except i as $\mu_{-i}^*(s) = \{\mu_1^*(s), ..., \mu_{i-1}^*(s), \mu_{i+1}^*(s), ..., \mu_N^*(s)\}$. In a Nash equilibrium, each agent acts as the best response μ_i^* to others, provided that all other agents follow the policy $\mu_{-i}^*(s)$. This gives the optimal action at each state s for agent i and leads to the equilibrium bidding strategy.

We solve $Q_i^{\boldsymbol{\mu}^*}$ and $\mu_i^*(s)$ in Eq. (5) by using an alternative gradient descent approach, similar to the ones introduced in [17, 23], where we gradient update agent's Q_i^{μ} and $\mu_i(s)$ while fixing all other agent's parameters (thus their outputs). Specifically, the *critic* Q_i^{μ} with parameter θ_i^Q is learned

by minimizing loss $L(\theta_i^Q)$ defined as

$$L(\theta_i^Q) = \mathbb{E}_{s,a,r,s'}[(Q_i^{\boldsymbol{\mu}}(s, a_1, ..., a_N) - y)^2], \tag{6}$$

$$y = r_i + \gamma Q_i^{\boldsymbol{\mu'}}(s', \mu_1'(s'), ..., \mu_N'(s')), \tag{7}$$

where $\boldsymbol{\mu'} = [\mu_1', \dots, \mu_N']$ is target policies with delayed parameters $\theta_i^{\mu'}$, $Q_i^{\boldsymbol{\mu'}}$ is target critic function with delayed parameters $\theta_i^{Q'}$, and $(s, a_1, \dots, a_N, r_i, s')$ is a transition tuple saved in experience replay memory D. And each agent's policy μ_i with parameters θ_i^{μ} is learned as

$$\nabla_{\theta_i^{\mu}} J(\mu_i) = \mathbb{E}_s \big[\nabla_{\theta_i^{\mu}} \mu_i(s) \nabla_{a_i} Q_i^{\boldsymbol{\mu}}(s, a_1, ..., a_N)|_{a_i = \mu_i(s)}\big]. \tag{8}$$

In the next section, we present a distributed implementation of Eqs. (6), (7), and (8) within our distributed architecture.

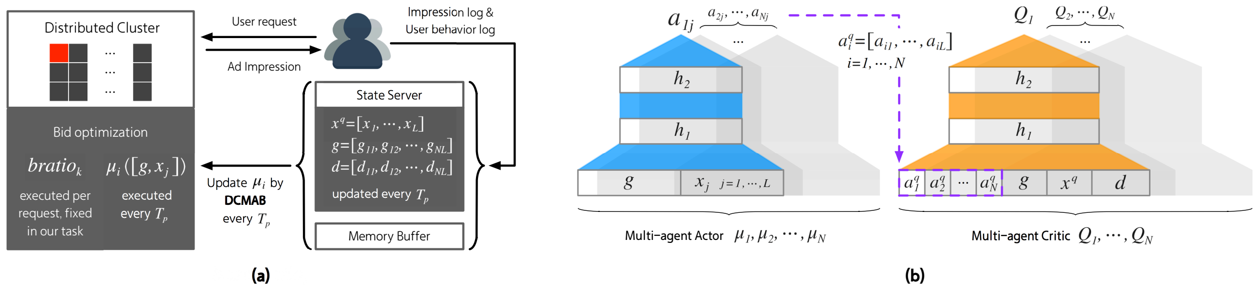

Figure 3: An Illustration of DCMAB. ***(a)*** **DCMAB workflow in our advertising system. The State Server maintains agents' states including general information g, consumer distribution d and consumer static feature x^q. Every T_p, states are merged and agents' actors are updated by DCMAB. After that, $\mu_i\left([g, x_j]\right)$ is calculated for merchant cluster i and consumer cluster j, further multiplied by $bratio_k$ to form the final bid adjustment ratio.** ***(b)*** **DCMAB network design. We have separate Actor and Q network for each agent. a_{ij} is calculated through μ_i, using g and x_j as input. In addition to states and actions, consumer distribution d is collected as input of all agents' Q function.**

4.3 Implementation & Distributed Architecture

A typical RL method such as original DPG saves a transition tuple after every state transition, which is difficult to implement in a real-world RTB platform for the following reasons. (i) An operational RTB system usually consists of many distributed workers which process consumers' requests in parallel and asynchronously, which demands to merge all workers' state transitions. (ii) The states change frequently and saving every request as a transition tuple would cost unnecessary computation. In this section, we extend the original gradient updates to be adapted to our real-world distributed-worker platform.

The system transition update and the action execution are maintained asynchronously. In other words, transition tuples and executing actions are operated with different frequencies, where the states that merge among workers and tuples are saved periodically for every a time gap T_p. During each T_p, there are many requests to be processed. For every request, according to different request features, the actor μ generates different actions for execution. With our method, the merge of the states and transition updates at every T_p interval can be handled by current industrial computation ability of distributed workers. Note that although the states are updated every T_p, the actions are generated for every auction in real time. This framework brings different frequencies of critic updates and actor executions. We propose to split consumer-based features and aggregated actions to make critic and actor well organized, which will be explained next.

Split consumer-based features. As stated in State definition, consumer feature x consists of the static feature containing slowly changed information that can be obtained before one T_p starts. And real-time feature such as $pCVR$ which can only be acquired when the request comes to the worker is also utilized.

As shown in Actor definition, we factorize the final bid adjust ratio as $a_i \times bratio_k$. The a_i is computed every T_p by $\mu_i\left([g, x]\right)$, where x is the static consumer feature. While the real-time part is used for $bratio_k$ calculation in every auction. The concrete formulation is $bratio_k = pCVR_k / pCVR_k^{avg}$, where $pCVR_k$ is on merchant level (not merchant cluster level) for merchant k and $pCVR_k^{avg}$ is the historical (7 days) average $pCVR_k$ of this merchant k. $pCVR_k / pCVR_k^{avg}$ provides auction level information and enables a merchant to apply auction level bid adjustment for high quality consumer request as demonstrated by Zhu et al. [33]. In such settings, factor a_i applies coarse adjustment to merchants within a cluster, and factor $bratio_k$ discriminates among merchants within the same cluster and reflects the value of conversion in real time.

Next, we only focus on the learnable component a_i, i.e. μ_i. Computing $\mu_i([g, x])$ for every consumer is computationally costly before every time interval T_p because of large numbers of consumers. Our solution is to utilize the consumer clusters. For L consumer clusters, we design L cluster-specific versions of features for x as $x^q = [x_1, ..., x_L]$. Each x_j contains a one-hot embedding of consumer cluster j with dimension L, and its history $(revenue, cost)$. We design consumer cluster with above one-hot embedding to enhance the discriminative ability on the basis of $(revenue, cost)$. Before the beginning of each T_p, we compute $a_{ij} = \mu_i([g, x_j])$ for every merchant cluster i and consumer cluster j pair for $i = 1, ..., N, j = 1, ..., L$. Within one interval T_p, for candidate ad of merchant k, we select a_{ij} according to the merchant cluster and consumer cluster pair, then multiplied by $bratio_k$ and clipped by $[-range, range]$ to form final adjusting ratio $\alpha = \min\{\max\{a_{ij} \times bratio_k, -range\}, range\}$ for $bid_k(1+\alpha)$. It is worth mentioning that, a_i and x in Eq.(3) are replaced by a_{ij} and x_j due to extra dimensionality of consumer cluster.

Aggregated actions. We save transition tuples to replay memory every time interval T_p, which requires to aggregate the information during T_p. Thus, we propose an aggregation method to summarize the executions within T_p where we maintain a discrete distribution of a_{ij} as $d_{ij} = \#a_{ij} / tot\_num$ where $\#a_{ij}$ stands for executed number of a_{ij} and tot_num for all executed number. We concatenate all d_{ij} as $d = [d_{11}, d_{12}, ..., d_{NL}]$ and save d as a part of tuple every T_p. And the critic function Q's input is augmented as $Q(s^q, a_1^q, ..., a_N^q, d)$ where $s^q = [g, x^q]$ and $a_i^q = [a_{i1}, ..., a_{iL}]$.

Our distributed gradient update aims to let the agents optimize their budgets allocation according to consumer distributions and consumer features every T_p while utilizing real-time feature such as $pCVR$ in every auction. We call our algorithm Distributed Coordinated Multi-Agent Bidding (DCMAB) with critic and actor update

Algorithm 1: DCMAB Algorithm

Initialize $Q_i(s^q, a_1^q, ..., a_N^q, d|\theta_i^Q)$, actor $\mu_i([g,x]|\theta_i^\mu)$ and target network Q_i', μ_i' with weights $\theta_i^{Q'} \leftarrow \theta_i^Q$, $\theta_i^{\mu'} \leftarrow \theta_i^\mu$ for each agent i.
Initialize replay memory D
for *episode = 1 to E* **do**
  Initialize a random process $\mathcal{N}$ for action exploration
  Receive initial state s for all agents
  for t = *1 to* T **do**
    For each agent i, compute a_i^q and add $\mathcal{N}_t$.
    for *auctions in parallel workers in* T_p **do**
      For each agent i, compute $bratio$ and combined with a_i^q compute adjusting ratio α and execute.
      For each agent i, save reward, cost and maintain distribution d.
    end
    For each agent i, merge rewards, cost in last T_p to get reward r_i and update state to $s^{q\prime}$. Store $(s^q, d, a_i^q, r_i, s^{q\prime})$ to replay memory.
    $s^{q\prime} \leftarrow s^q$
    for *agent i=1 to N* **do**
      Sample a random minibatch of S samples $(s^q, d, a_1^q, ..., a_N^q, r_i, s^{q\prime}, d')$ from D
      Update critic by minimizing loss with Eqs.(9),(10).
      Update actor with Eq. (11).
      Update target network: $\theta' \leftarrow \tau\theta + (1-\tau)\theta$
    end
  end
end

rules as:

$$y = r_i + \gamma Q_i'(s^{q\prime}, a_1^{q\prime}, ..., a_N^{q\prime}, d')\Big|_{a_o^{q\prime}=[\mu_o'([g',x_1']),\ldots,\mu_o'([g',x_L'])]} \tag{9}$$

$$L(\theta_i^Q) = (y - Q_i(s^q, a_1^q, ..., a_N^q, d))^2 \tag{10}$$

$$\nabla_{\theta_i^\mu} J \approx \Sigma_j \nabla_{\theta_i^\mu} \mu_i([g, x_j]) \nabla_{a_{ij}} Q_i(s^q, a_1^q, ..., a_N^q, d) \tag{11}$$

The solution is illustrated in Figure 3, and the corresponding pseudo code is presented in Algorithm 1.

5 EXPERIMENTS

Our experiments are conducted over the data sets collected from Taobao display ad system. The data is collected in *Guess What You Like* column of Taobao App Homepage where three display ads slots and hundreds of recommendation slots are allocated. As we have collected the bid prices traded in the market as well as the feedback and conversions from the consumers for the placed ads, we would be able to replay the data to train and test our proposed the MARL bidding in an offline fashion. The similar settings can be found in other research work [1, 21, 28, 32, 33].

5.1 Data Sets and Evaluation Setup

Data sets. The data sets we used for experiments come from real-world production. The display ads are located in *Guess What You Like* column of Taobao App Homepage where three display ads slots and hundreds of recommendation slots are well organized. Based on the log data, the saved procedures of consumers' requests including pCTR, pCVR, ppb along with the requests are used as procedure replay to form an offline simulation platform. And pCTR, pCVR, ppb are used to simulate the consumers' behaviors for computing states and rewards. We use a uniformly sampled fraction of first three hours' logged data from date of 20180110 as training data, and a uniformly sampled fraction of first three hours' logged data from 20180111 as test data. Training and test of our algorithm are both based on the offline simulation system due to the lack of real consumer feedback data. All results reported are based on test data.

For merchants, when the budget is unlimited, each merchant will adjust their bid price to the highest number and the solution is trivial. To test the budget optimized allocation along time, the budget for each merchant should not be too large. Similar to the work setting in [32], we determine the budget as follows: let all merchants use human manually set bid with unlimited budgets and accumulate the total cost C_T. And each merchant's budget is set as a fraction of C_T.

With the notion of C_T, here are some statistics of the data: for training set there are 203,195 impressions, 18,532 revenue (C_T) and 5,300 revenue ($C_T/3$) where (C_T) means the setting where merchants are endowed with unlimited budgets (in real situation this is impossible, when merchants have limited budgets, they quit bidding when budgets run out and the market depth decreases); for testing set there are 212,910 impressions, 18,984 revenue (C_T) and 5,347 revenue ($C_T/3$).

Evaluation metrics. The evaluation is based on the agents' revenue and total revenue under predefined budgets and a number of auctions. The agent's objective is to maximize its revenue given the budget. In our experiments, revenue is set as the primary evaluation measure. We also analyze the influences of the agents' rewards changes on the converged equilibrium.

Evaluation flow. We built an offline simulation system close to the real online system with distributed workers processing consumers' requests. In each auction, according to the maximum eCPM ranking, the top-ranked three merchants win. During our training, as the model learns, the model's different bids lead to different ranking results. Due to lack of consumers' real feedback of all different ranking results for all merchants, we use expected CPC ($cost_k \times pCTR_k$ where $cost_k = pCTR_{next(k)} \times bid_{next(k)} / pCTR_k$ is based on GSP mechanism) and expected revenue ($pCTR_k \times pCVR_k \times ppb_k$) for offline simulation[1]. The system is based on 40-node computation cluster each node of which has Intel(R) Xeon(R) CPU E5-2682 v4, 2.50GHz and 16 CPU cores with 250 GB memory on CentOS. The model is implemented with distributed TensorFlow in Python. Our offline platform is consistent with the online platform, in future deployment we only need to change the reward from expectation to real feedback.

Episode length. To simulate the real online system, our simulation platform updates states every hour. We use three hours' auctions for evaluation. The length of an episode includes three steps which is the number of state transitions. The three-hour training data includes totally 203,195 impressions which is the number of actor executions. Each experiment training takes about 4 hours with 40 distributed workers.

[1]The unit of revenue and cost in our experiments is CNY.

5.2 Compared Methods

With the same evaluation flow, the following algorithms are compared with our DCMAB algorithm. Except manually setting bids, all other 4 algorithms use neural networks as approximators. We also build a reward estimator for contextual bandit method as a critic. All 4 algorithms' critics include two hidden layers with 100 neurons for the first hidden layer and 100 neurons for the second hidden layer with states as inputs to the first layer and actions as inputs to the first hidden layer. All 4 algorithms' actors include 300 neurons for the first hidden layer and 300 neurons for the second hidden layer with states as inputs and actions as outputs. The activation function for hidden layers is *relu*, for output layer of actors is *tanh* and for output layer of critics is linear.

- **Manually Set Bids.** They are the real bids set manually by human according to their experiences.
- **Contextual Bandit.** This algorithm [15] optimizes each time step independently. Each impression's bid is adjusted according to only the feature in the impression which is called contextual feature. To compare with our DCMAB, we also add other agents' actions as a part of the contextual feature. The key difference between this algorithm and ours is that it does not optimize budgets allocation along time.
- **Advantageous Actor-critic (A2C).** This method [2, 18, 24] is an on-policy actor-critic algorithm which does not utilize a memory replay. The critic function Q of this algorithm does not take other agents' actions as input.
- **DDPG.** DDPG [16] is an off-policy learning algorithm with a memory replay. The critic function Q of this algorithm doesn't take other agents' actions as input.
- **DCMAB.** This is our algorithm. We upgrade MADDPG [17] with clustered agents modeling and redesign actor and critic structures to adapt to distributed workers' platform. The critic function Q of this algorithm takes all agents' actions as input.

5.3 Hyperparameter Tuning

5.3.1 Clustering Method. When a consumer request comes, according to its requested merchant criteria, our system firstly selects n_c candidates of merchants from all n registered merchants where $n_c \ll n$. And these n_c candidates attend the bidding stage while other $n - n_c$ merchants are filtered out. We consider one merchant who is present in bidding stage as one presence. We rank all n merchants according to their revenues in training data and group them into clusters where these clusters have approximately equal presences respect to all consumer requests in training data. This clustering method makes the competitions among agent clusters relatively balanced. The example of three clusters is as Figure 4. Usually, clusters with higher revenues consist of small numbers of merchants and contribute larger amount of revenue. The reason is that most high-revenue merchants attend the bidding stage more frequently. Consumers are also ranked according to their revenues and grouped into clusters with each cluster having equal proportion of requests to the ad platform.

5.3.2 The Number of Clusters. In our formulation, theoretically, more clusters and smaller cluster sizes provide more possible adjusting ratios, which means better possible solutions. We tried different cluster numbers as $\{1, 2, 3, 4, 5, 10, 30\}$, shown in Figure 5(a). Two kinds of rewards are used. "Coord" means all clusters' rewards are the same as total traffic revenue. "Self-Interest" means each

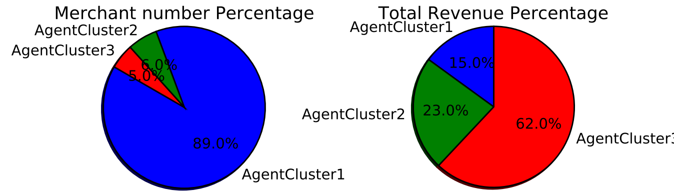

Figure 4: Clusters of Merchants

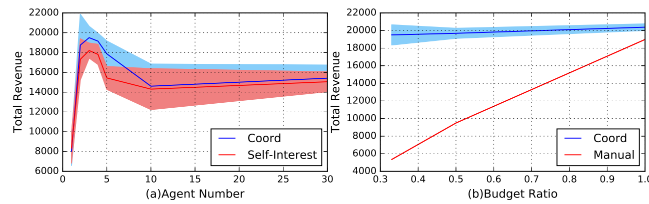

Figure 5: Revenue VS. (a) Agent Number & (b) Budget Ratio

cluster's reward is its own revenue. For both rewards, we consider total traffic revenue as the metric.

In Figure 5(a), the horizontal axis is the number of agent clusters, and the vertical axis represents total traffic revenues. We draw the mean episode reward as the blue and red curves with corresponding colored area as standard deviations. From the results, we find the best performance is achieved when the number of the clusters is 3 and 4. When the cluster number increases from 1 to 3, the performance increases, which shows the benefits of shrinking cluster size and adding more clusters. When we further increase the cluster number from 4 to 30, we find the performance drops. We observe that as we increased the number of agents, the agents' policies learning easily converged to worse equilibria as many agents competed severely in early stage with high bid prices and quited the auctions earlier. There exists better strategies for these agents such as lowering bids in early stage and competing the cheaper auctions in the late stage. After the cluster number tuning, cluster number 3 appears to perform the best, and our following-up experiments shall fix the number of clusters as 3.

5.3.3 Budget Search. With the three agent clusters fixed, we now measure the total revenue performance of our DCMAB with manually set bids shown in Figure 5(b) where 'Coord' means all agents' rewards are total revenue. The budget for each merchant is searched from one-third to full amount of unlimited budget. Compared with manually setting, our DCMAB with coordinated rewards consistently maintain a higher revenue even when budget is low due to the better budget allocation. Manually setting bids acquires more revenue as the budget increases because higher budget makes more merchants stay in the market and deliver their ads to the consumers.

5.4 Experimental Results

In this section, we compare our DCMAB algorithm with the baselines to understand their learning abilities and performance.

5.4.1 Performance Comparisons. For performance test, we set the best hyperparameters as tuned in the previous section. For instance, we group merchants and consumers into 3 clusters, respectively. Each merchant's budget is set as $C_T/3$. Each agent

Table 1: Revenue from Self-Interest Bidding Agents

	AgentC1	AgentC2	AgentC3	Total
Manual	231	817	4299	5347
Bandit	281±21	1300±50	4422±171	6003±123
A2C	238±7	872±104	7365±2387	8477±2427
DDPG	1333±1471	7938±2538	7087±4311	16359±1818
DCMAB	1590±891	4763±721	11845±1291	18199±757

cluster's reward is set as its own episode revenue, which is a self-interest reward. The results are reported in Figure 6 and Table 1.

Table 1 lists the converged performances of different algorithms (we consider the situation where the training performance does not improve in the last 50 episodes as converged). Each row shows an algorithm's results. The columns represent the results of different agent clusters' and their summed total revenue in one algorithm's experiment. We conducted 4 times of experiments for each algorithm and gave the average revenues and standard deviations in Table 1.

We use Pareto improvement [8] as one cluster can improve its revenue without hurting other clusters' revenues. Among all algorithms, our DCMAB has Pareto improvement over all other algorithms except DDPG, which means all clusters' revenue and total revenue are improved. This verifies the effectiveness of our algorithm. DDPG has Pareto improvement than Manual and Bandit. Compared with on-policy algorithm A2C, DDPG and our DCMAB perform better, illustrating the usefulness of sample memory. Compared with Bandit, other algorithms as A2C, DDPG and our DCMAB verify the importance of budget allocation among different hours, which points out the necessity of reinforcement learning modeling rather than bandit modeling. Manually setting bids perform the worst as it is a non-learning baseline.

DCMAB and DDPG result in different equilibria. AgentC1 and AgentC3 get more revenue in DCMAB than in DDPG while AgentC2 gets more revenue in DDPG than in DCMAB. Comparing these two equilibria, we find DCMAB achieves a higher total revenue of 18199 than DDPG of 16359. From the perspective of total matching efficiency for connecting consumers to products, our DCMAB gives a better result. Moreover, our DCMAB gives a more stable equilibrium with all agents' revenues and total revenue's standard deviation lower than DDPG, which verifies the merits of modeling all agents' actions in our DCMAB rather than only modeling the own action in DDPG.

The learning is illustrated in Figure 6. We find our DCMAB converges more stable than DDPG, verifying the effectiveness of modeling all agents' actions as inputs to action-value functions. DCMAB and DDPG learn faster than A2C and bandit, showing the merits of the deterministic policy gradient with a memory replay.

5.4.2 Coordination vs. Self-interest. In this part, we study how different reward settings can influence the equilibrium reached. First, we compare two kinds of reward settings as shown in Table 2 and Figure 7(a). Self-Interest stands for each agent reward set with its own revenue; Coord stands for all agents' rewards set as total traffic revenue where all agents are fully coordinated to maximize the same goal. We find Coord achieves better total revenue than Self-Interest. Compared to the Self-Interest equilibrium, in Coord's equilibrium, while Agent1 and Agent2 obtain less revenues, Agent3's revenue is improved largely, resulting in a total revenue improvement. The total revenue improvement of Coord demonstrates the ability of DCMAB to coordinate all agents to achieve a better result for the overall social benefits.

Table 2: Revenue from Fully Coordinated Bidding Agents

	Agent1	Agent2	Agent3	Total
Self-Interest	1590±891	4763±721	11845±1291	18199±757
Coord	1185±1359	698±100	17617±2583	19501±1144

Table 3: Revenue for Different Coordination Levels

	Agent1	Agent2	Agent3	Total
All Manual	231	817	4299	5347
1 PartiallyCoord	4040±2732	806±28	4157±145	9004±2728
2 PartiallyCoord	3370±218	7088±395	4110±16	14569±195
Fully Coord	1185±1359	698±100	17617±2583	19501±1144

In Table 3 and Figure 7(b), we analyze the performance when we gradually add learned agents' bids with coordination reward while keeping other agents' bids manually set. In Figure 7(b), Manual means all agents are self-interested with manually set bids; Coord1 stands for that only bids of agent cluster 1 are learned with total revenue reward while other two agents' bids are manually set; Coord2 stands for Agent1 and Agent2's bids are learned with rewards of total revenue while Agent3's bids are manually set; Coord means all agents' bids are learned with rewards of the total revenue.

Compared to Manual, the total revenue of Coord1 setting is improved from 5347 to 9004. The improvement mainly comes from Agent1 (from 231 revenue to 4040 revenue), while Agent2 (817 to 806) and Agent3 (4299 to 4157) do not contribute to the total improvement. This illustrates that the flexibility of the MARL framework from our approach in adjusting the coordination level depending on the specific needs in practice.

As for Coord2 setting, the total revenue is improved more than Coord1 and it mainly comes from Agent1 (from 231 to 3370) and Agent2 (from 817 to 7088) while Agent3 drops a little. As more merchants join in the cooperation, the total revenue is further improved from Coord1 of 9004 to Coord2 of 14569. By comparing Coord2 and Coord1, we find that Agent2's revenue increases largely from 806 to 7088, while Agent1's revenue unfortunately drops from 4040 to 3370. This shows the Coord2 rearranges the traffic allocation and would inevitably harm the performance of some agents to achieve better overall revenue.

Finally, when all agents cooperate for total revenue, it achieves the highest total revenue. As all agents' rewards aim at total revenue, we find Agent1 and Agent2 reach a compromise with dropped revenue compared to Coord1 and Coord2. And Coord rearranges the traffic to unleash Agent3's potential to improve the total revenue resulting in a larger improvement of total revenue from Coord2 14569 to 19501. In terms of total revenue, from Coord1, Coord2 to Coord, the gradually added coordination verifies our DCMAB's ability to reinforce all agents to cooperate for a predefined global goal. From a system perspective, higher total revenue means the consumers' better experiences for better connections to the commodities they like. From a long-term perspective, maximizing total revenue also encourages merchants to improve their business operational efficiency and provide better products to consumers.

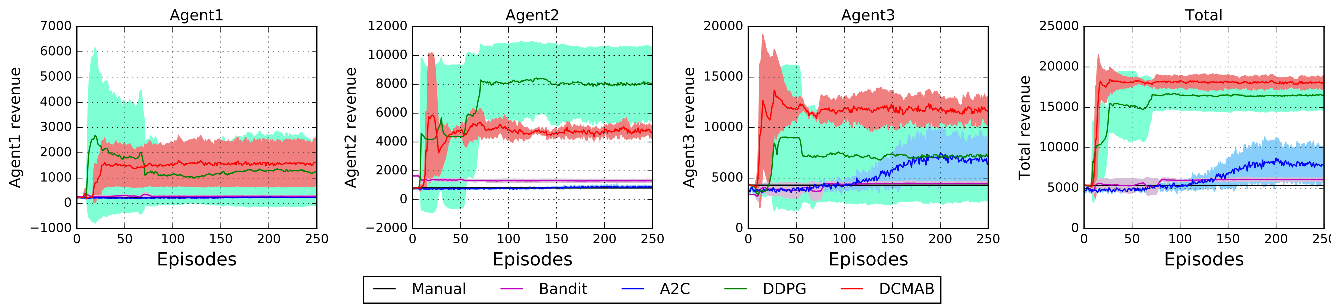

Figure 6: Learning Curves Compared with Baselines

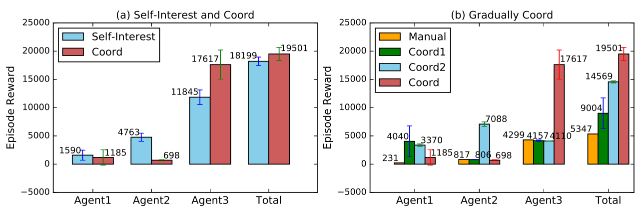

Figure 7: (a): Self-Interest VS. Coord; (b): Gradually Coord

6 CONCLUSIONS

In this paper, we proposed a distributed cluster-based multi-agent bidding solution (DCMAB) for real-time bidding based display advertising. The MARL approach is novel and for the first time takes into the interactions of all merchants bidding together to optimize their bidding strategies. It utilizes rich information from other agents' actions, the features of each historic auction and user feedback, and the budget constraints etc. Our DCMAB is flexible as it can adjust the bidding that is fully self-interested or fully coordinated. The fully coordinated version is of great interest for the ad platform as a whole because it can coordinate the merchants to reach a better socially-optimal equilibrium for balancing the benefits of consumers, merchants and the platform all together. We realized our model in a product scale distributed-worker system, and integrated it with the process auctions in parallel and asynchronously. Experimental results show that our DCMAB outperforms the state-of-the-art single agent reinforcement learning approaches. With fully cooperative rewards, DCMAB demonstrates its ability of coordinating all agents to achieve a global socially better objective.

As the results from the offline evaluation are promising, we are in the process of deploying an online version. We plan to conduct a live A/B test in Taobao ad platform with a particular focus on mobile display ads.

ACKNOWLEDGMENTS

The authors would like to thank Dr. Qing Cui, Lvyin Niu and Jin Li for insightful discussions, code contributions, and other efforts.

REFERENCES

[1] Han Cai, Kan Ren, Weinan Zhang, Kleanthis Malialis, Jun Wang, Yong Yu, and Defeng Guo. 2017. Real-Time Bidding by Reinforcement Learning in Display Advertising. In *10th WSDM*. ACM, 661–670.

[2] Thomas Degris, Patrick M Pilarski, and Richard S Sutton. 2012. Model-free reinforcement learning with continuous action in practice. In *ACC, 2012*. IEEE.

[3] Manxing Du, Redouane Sassioui, Georgios Varisteas, Mats Brorsson, Omar Cherkaoui, et al. 2017. Improving Real-Time Bidding Using a Constrained Markov Decision Process. In *ICADMA*. Springer, 711–726.

[4] Benjamin Edelman, Michael Ostrovsky, and Michael Schwarz. 2007. Internet advertising and the generalized second-price auction: Selling billions of dollars worth of keywords. *American economic review* 97, 1 (2007), 242–259.

[5] David S Evans. 2009. The online advertising industry: Economics, evolution, and privacy. *Journal of Economic Perspectives* 23, 3 (2009), 37–60.

[6] 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.

[7] Jakob Foerster, Yannis Assael, Nando de Freitas, and Shimon Whiteson. 2016. Learning to communicate with deep multi-agent reinforcement learning. In *NIPS*. 2137–2145.

[8] Drew Fudenberg and Jean Tirole. 1991. Game Theory Cambridge MA. (1991).

[9] Avi Goldfarb and Catherine Tucker. 2011. Online display advertising: Targeting and obtrusiveness. *Marketing Science* 30, 3 (2011), 389–404.

[10] Jayesh K Gupta, Maxim Egorov, and Mykel Kochenderfer. 2017. Cooperative multi-agent control using deep reinforcement learning. In *AAMAS*. Springer.

[11] Junling Hu and Michael P. Wellman. 2003. Nash Q-learning for general-sum stochastic games. In *JMLR*. ., 1039–1069.

[12] Junling Hu, Michael P Wellman, et al. 1998. Multiagent reinforcement learning: theoretical framework and an algorithm.. In *ICML*, Vol. 98. Citeseer, 242–250.

[13] Kuang-Chih Lee, Ali Jalali, and Ali Dasdan. 2013. Real time bid optimization with smooth budget delivery in online advertising. In *7th ADKDD*. ACM, 1.

[14] Joel Z Leibo, Vinicius Zambaldi, Marc Lanctot, Janusz Marecki, and Thore Graepel. 2017. Multi-agent Reinforcement Learning in Sequential Social Dilemmas. In *16th AAMAS*. 464–473.

[15] Lihong Li, Wei Chu, John Langford, and Robert E Schapire. 2010. A contextual-bandit approach to personalized news article recommendation. In *19th WWW*.

[16] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. 2015. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971* (2015).

[17] 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 *NIPS*. 6382–6393.

[18] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous methods for deep reinforcement learning. In *ICML*. 1928–1937.

[19] Igor Mordatch and Pieter Abbeel. 2017. Emergence of grounded compositional language in multi-agent populations. *arXiv preprint arXiv:1703.04908* (2017).

[20] Noam Nisan, Tim Roughgarden, Eva Tardos, and Vijay V Vazirani. 2007. *Algorithmic game theory*. Vol. 1. Cambridge University Press Cambridge.

[21] Claudia Perlich, Brian Dalessandro, Rod Hook, Ori Stitelman, Troy Raeder, and Foster Provost. 2012. Bid optimizing and inventory scoring in targeted online advertising. In *18th SIGKDD*. ACM, 804–812.

[22] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. 2014. Deterministic policy gradient algorithms. In *ICML*.

[23] Satinder Singh, Michael Kearns, and Yishay Mansour. 2000. Nash convergence of gradient dynamics in general-sum games. In *Proceedings of the Sixteenth conference on Uncertainty in artificial intelligence*. Morgan Kaufmann Publishers Inc., 541–548.

[24] Richard S Sutton and Andrew G Barto. 1998. *Reinforcement learning: An introduction*. Vol. 1. MIT press Cambridge.

[25] Ming Tan. 1993. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *10th ICML*. 330–337.

[26] Jun Wang, Weinan Zhang, and Shuai Yuan. 2016. Display advertising with real-time bidding (RTB) and behavioural targeting. *arXiv:1610.03013* (2016).

[27] Xiaofeng Wang and Tuomas Sandholm. 2003. Reinforcement learning to play an optimal Nash equilibrium in team Markov games. In *NIPS*. 1603–1610.

[28] Yu Wang, Jiayi Liu, Yuxiang Liu, Jun Hao, Yang He, Jinghe Hu, Weipeng Yan, and Mantian Li. 2017. LADDER: A Human-Level Bidding Agent for Large-Scale Real-Time Online Auctions. *arXiv preprint arXiv:1708.05565* (2017).

[29] Jian Xu, Kuang-chih Lee, Wentong Li, Hang Qi, and Quan Lu. 2015. Smart pacing for effective online ad campaign optimization. In *21st SIGKDD*. ACM, 2217–2226.
[30] Shuai Yuan, Jun Wang, Bowei Chen, Peter Mason, and Sam Seljan. 2014. An empirical study of reserve price optimisation in real-time bidding. In *20th SIGKDD*.
[31] Shuai Yuan, Jun Wang, and Xiaoxue Zhao. 2013. Real-time bidding for online advertising: measurement and analysis. In *7th ADKDD*. ACM, 3.
[32] Weinan Zhang, Shuai Yuan, and Jun Wang. 2014. Optimal real-time bidding for display advertising. In *20th SIGKDD*. 1077–1086.
[33] Han Zhu, Junqi Jin, Chang Tan, Fei Pan, Yifan Zeng, Han Li, and Kun Gai. 2017. Optimized cost per click in taobao display advertising. In *23rd SIGKDD*. ACM.