Reinforcement Learning to Improve Ad Experience in Guaranteed Delivery Advertising

ABSTRACT

Guaranteed delivery and inventory allocation is an important topic in the advertising domain. Most of the systems model it as a bipartite matching and only optimize Click Through Rate (CTR) with contract delivery constraints. Ad experience impacts user's engagement, dwell time thus is also important and needs to be considered during ad delivery. However, It is hard for the current allocation framework to deal with such long-term rewards. Reinforcement learning methods are more suitable to model long-term rewards. To the authors' knowledge, there are no related studies explicitly modeling optimizing long-term rewards in guaranteed delivery systems. In this paper, we propose to utilize reinforcement learning to model these long-term rewards in guaranteed ad delivery. 1) We show that the long-term reward and short-term reward can be represented by a generalized reward; 2) We use a constraint reinforcement learning method to consider both the allocation optimization and the reward optimization 3) Output actions have the same form as bipartite matching solution. The experimental results on real-world ad impression data prove the effectiveness of our proposed method.

CCS CONCEPTS

• **soft constraints**; • **reinforcement learning**; • **guaranteed delivery**; • **multiple timescale**;

KEYWORDS

constrained reinforcement learning, guaranteed delivery, bipartite matching

ACM Reference Format:

. 2018. Reinforcement Learning to Improve Ad Experience in Guaranteed Delivery Advertising. In . ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Guaranteed delivery still owns lots of market in online advertising industry. In the traditional display advertising setting, a large publisher like Youtube, TikTok and Hulu, enters into contracts with individual advertisers guaranteeing to show their advertisements to a pre-specified number of users matching the advertiser's targeting constraints. As the publisher faces penalties for under delivering, it is in its best interest to deliver on every contract it promises. The main metric measuring the performance of guaranteed ad delivery systems is under delivery ratio. In the guaranteed delivery setting, there will be an under delivery penalty if some ad contracts' goals are not satisfied.

Most existing methods model the guaranteed delivery problem as a bipartite graph allocation problem[12][3][1][14]. An provably nearly optimal bipartite matching solution is provided for online allocation[12]. The bipartite graph consists of supply and demand nodes. The supply node i represents the individual user visits, demand node j represents the guaranteed ad contracts, and there is an edge $(i, j) \in E$ if user i matches the targeting constraints of contract j. Moreover, each advertiser has an overall demand d_j; while a supply node has a supply parameter s_i, representing how many times the user appears during the time period. The goal of the publisher is to find an allocation of users to advertisers so that all of the supply and demand constraints are satisfied: at most one ad is shown on each user visit (supply constraint), and each advertiser fulfills its demand (demand constraint). A toy example of such a graph is shown in Figure 1.

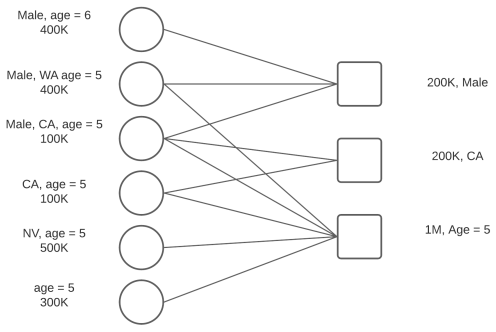

Figure 1: Example of a bipartite graph

Therefore the publishers try to reach each contract's delivery goal with best effort. Some publishers like Alibaba[3] and Tencent[14] also take user interest into account and try to optimize the click through rate (CTR) and conversion rate (CVR) for each contract, so that the overall click through count is maximized while at the same time guarantees delivery and pacing. These methods add CTR/CVR into the objective of the bipartite matching optimization problem, and derive a solution similar to the original bipartite matching solution, with an additional CTR/CVR term.

Besides short-term rewards such as CTR and CVR, ad experience will also have impact on long-term rewards. For example, a bad ad experience will have a bad influence on user experience at Hulu, therefore reducing the user watch time and total ad inventory. Most existing methods are designed to maximize the immediate (short-term) reward and recommend items following fixed greedy strategies. They overlook the long-term user experience and revenue. Short-term rewards such as click through rate (CTR) metric

KDD 2021, August 14-18,

https://doi.org/10.1145/1122445.1122456

is not enough to measure the performance of the guaranteed delivery systems[17]. Long-term rewards like user dwell time and user retention will also be influenced by the overall ad experience.

Recently there are some studies on modeling long-term user experience metrics in advertising systems and recommendation systems[16][18][17][19]. There is increasing interest in modeling long-term rewards instead of fixed greedy strategies. In [16][18] the authors focus on jointly optimizing selection and ranking of items and ads. We can also use reinforcement learning to improve recommendation. In order to exploit other labels in the data and avoid keeping recommending similar recommendations for users, [19] uses reinforcement learning to model the recommendation problem. [8] use combination-bandit to model the pay-per-click advertising problem, they provide an algorithm to decide the values of the bid and the budget to maximize the value of a campaign. [2] models the user as the environment, and proposes a model-based Q-network to handle a large number of candidate items efficiently.

Modeling user experience with guaranteed delivery constraint has many challenges. Firstly, in modern advertising systems there are often more than 10,000 contracts and 1 Million users. This significantly increased the problem size for the bipartite graph matching solution. For example, a user may be targeted by many advertisers, during the user's visit to the publisher's website, a list of qualified ads will compete for the impression opportunity. Secondly, the ad inventory on the publisher website will vary through time, a static allocation plan may not capture this trend. For example, Hulu users may tend to watch shows at night, causing increasing ad inventory in these time periods.

We introduce reinforcement learning to solve the problem of optimizing long-term reward in the guaranteed delivery advertising system. Reinforcement learning is more suitable to model the long-term rewards compared to supervised learning methods used for recommendations. It is proven that reinforcement learning can learn policies whose performance is competitive with those found by a planning algorithm with full access to the dynamics of the domain and its derivatives[5]. Reinforcement learning agents can also solve complicated problems with super human performance[4][13][10]. To the authors' knowledge, there are no other studies that explicitly models the long-term rewards in guaranteed ad delivery systems.

In this paper, we model and solve this problem using constrained reinforcement learning. Our algorithm can continuously update ad allocation and delivery strategies during the interactions with users, and the optimal strategy is designed to maximize the expected long-term cumulative reward from users [15] [20] and satisfy delivery constraints at the same time. Meanwhile, we design our neural network to reduce the state and action space of the problem, which makes the neural network implementation easier to converge. We conduct experiments with real-world data to demonstrate the effectiveness of the proposed framework.

We first formalize our problem as a bipartite graph allocation problem with long-term rewards, then propose a novel reinforcement learning algorithm inspired by the solution to the bipartite graph matching problem.

Our contributions are:

- We are the first to propose using reinforcement learning to model long-term rewards in a guaranteed ad delivery system.
- We use a novel method to represent guaranteed delivery constraints in our reinforcement learning algorithm.
- We propose a novel reinforcement learning agent to output feasible allocation plans as actions.

The content of the paper is organized as below. After formally defining the problem in section 2, in section 3 we introduce the reinforcement learning method and in section 4 we show the performance of our algorithm compared to the baseline methods.

2 REINFORCEMENT LEARNING FRAMEWORK

In this section, we begin by modeling the guaranteed delivery advertising problem as an Markov Decision Process (MDP) in 2.1, in this definition, we propose two novel design: 1) the action of the MDP is an allocation plan 2) the reward of the MDP is a generalized reward that takes delivery constraints into consideration. Next, in section 2.2 we introduce the state-of-the-art bipartite graph matching modeling of the problem, we also introduce the relation of our action design in MDP to the solution of bipartite graph matching problem solution. In section 2.3 we introduce how to serve a bipartite graph matching model. Similarly, in section 2.4 we introduce MDP model serving. Our goal will be serving as close as possible to the optimal allocation and maximizing long-term rewards.

2.1 Markov Decision Process Modeling

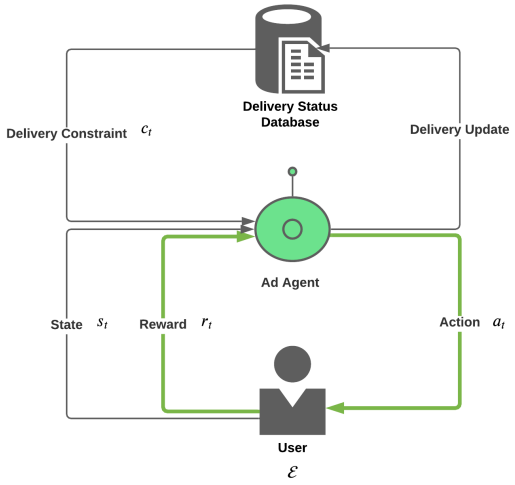

Figure 2: Reinforcement Learning Agent Structure

To solve the above mentioned challenges, inspired by bipartite graph modeling, we model the guaranteed delivery with long-term reward problem by Markov Decision Process (MDP). As is shown in figure 2, when a user visits the publisher's website, a centralized reinforcement learning agent takes action based on the state information collected from the user and ad server. The action produces a feasible allocation plan to serve ad to the user.

We formally model the problem as a Markov Decision Process (MDP) where the ad agent sequentially interacts with users (environment $\mathcal{E}$) by generating a feasible allocation plan, so as to

maximize the cumulative reward from $\mathcal{E}$. Next, we define the five elements $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$ of the MDP.

State space $\mathcal{S}$: A state $s_t \in \mathcal{S}$ includes a user's recommendation and advertisement browsing history before time t and the contextual information of current ad impression at time t . The delivery status, such as demand, supply, delivery percentage is also part of the state. We define the initial state distribution as $\mu : \mathcal{S} \mapsto [0, +\infty)$:

$$s_0 \sim \mu \tag{1}$$

Action space $\mathcal{A}$: $a_t = \{(\hat{\alpha}_j, \hat{q}_{ij}) \in \mathcal{A}, \forall j \in \mathbb{J}\}$ is the action of the agent. Where $\hat{\alpha}_j$ and $\hat{q}_{ij}$ are the parameters in the solution of bipartite graph matching problem for each contract, which can be used to derive a feasible allocation plan. The detail of bipartite matching modeling is in section 2.2.

Reward $\mathcal{R}$: We use reward to represent long-term reward and constraints at the same time. Long-term rewards R_l is the cumulative reward the system receives in the long run. Formally, R_l can be represented by expected cumulative long-term returns of the system. On the contrary, short-term rewards R_s is exactly defined by the immediate returns of the system. For example, in a recommendation system, click through feedback is a short-term reward, since users initiate the click behavior based on the current context and content. However, the user dwell time is a long-term rewards, since the user may decide to leave the publisher website after browsing a series of recommended contents or irrelevant ads. Therefore the dwell time is affected by a series of recommendation decisions made by the system.

We model the long-term rewards R^l and short-term reward R^s by a reward R^g, where $R^g = R^l + R^s$. Since the short-term reward can be represented by an additional term added to the long-term reward, without loss of generality, we only consider the long-term rewards R^l in the rest of the paper.

In this paper we propose a novel generalized reward R_t to take delivery constraints into consideration. The reward at time step t consists of two parts: long-term reward R_t^l and constrained reward R_t^c.

- The long-term reward R_t^l represents user feedback after an action a_t served at the state s_t in a time window T_q. For example, we can define the reward as the total watch time of the user watch session on Hulu. If the user leaves early, we will receive a small reward; otherwise we will receive a large reward.
- The constrained reward R_t^c is a novel reward design that reflects the cost of unsatisfied constraints. This penalty term acts similarly to the dual term of a Lagrangian, and it is dynamic during training, R_t^c will decrease R_t when the allocation plan does not satisfy the constraint.

The final reward R_t is the sum of rewards R_t^l and R_t^c:

$$R_t = R_t^l + R_t^c \tag{2}$$

If the sequence of rewards received after time step t is denoted $R_{t+1}, R_{t+2}, R_{t+3}, \ldots$, then we seek to maximize the expected return, where the return, denoted G_t, is defined as some specific function of the reward sequence. In this paper we define the return is the sum of the rewards:

$$G_t = R_{t+1} + \gamma \cdot R_{t+2} + \gamma^2 \cdot R_{t+3} + \ldots + R_{T_q} \tag{3}$$

We will discuss more details about the reward in section 3.2.

probability $\mathcal{P}$: $\mathcal{P}(s_t + 1|s_t, a_t)$ is the state transition probability from s_t to s_{t+1} after executing action a_t . The MDP is assumed to satisfy Markov property $P(s_{t+1}|s_t, a_t, \ldots, s_1, a_1) = P(s_{t+1}|s_t, a_t)$.

Discount factor γ: Discount factor $\gamma \in [0, 1]$ defines the expected discounted returns. The agent tries to select actions so that the sum of the discounted rewards in equation 3 it receives over the future is maximized.

Figure 2 illustrates the user-agent interactions in MDP. With the above definitions, the problem can be formally defined as follows: Given the historical MDP,i.e.,$(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$, the goal is to find a policy to generate action a_t at each time step t for $\mathbb{J}$ contracts running in the ad server and simultaneously optimizing the user experience and the advertising revenue.

A key novelty of this design is that the agent considers the delivery constraints and produces actions that maximize long-term rewards. This is implemented with the constrained reward R^c in equation 2. We model the problem as a constraint control problem. The delivery goal is expressed as a constraint, and our objective is to improve user experience metrics while guaranteeing delivery goals. Inspired by the constraint policy optimization (RCPO) algorithm[11], we design a constrained reward R_t^c that takes delivery constraints into consideration.

The training process of the reinforcement learning agent is described in section 3.

2.2 Bipartite Graph Modeling

In guaranteed display advertising, we have a large number of forecast impressions together with a number of contracts. These contracts specify a demand as well as a target; we must deliver a number of impressions at least as large as the specified demand, and further, each impression must match the target specified by the contract. We model this as a bipartite graph. On one side are supply nodes, representing impressions, each impression is also an ad inventory. We will use supply node, impression and inventory interchangeably in the remainder of this paper. On the other side are demand nodes, representing contracts or ads. Similarly, we will use demand nodes, contracts and ads interchangeably. We add an arc from a given supply node to a given demand node if and only if the impression that the supply node represents is eligible (i.e. matches the target profile) for the contract represented by the demand node. We denote supply node by i and demand node by j, $\Gamma(j)$ is the neighborhood of j and likewise $\Gamma(i)$ is the neighborhood of i. Further, demand nodes are labeled with a demand d_j, which is precisely the amount of impressions required by the represented contract. In general, supply nodes will represent several impressions each, thus each supply node is labeled with a weight s_i, leading to a weighted graph (see [12][3][14] for more details). In real-world applications, the number of supply nodes is large due to the diversity in user attributes, but the number of contracts running in an ad server is usually small compared to the number of supply nodes. Figure 1 shows a simple example.

An optimal allocation must both be feasible and minimize some objective function. Here, our objective balances two goals: minimizing under delivery penalty, and maximizing long-term rewards. Each demand node/contract j has an associated penalty, p_j. Let

$\sum_{i\in\Gamma(j)} s_i x_{ij}$ be the delivered count, where s_i is supply weight in supply node i, x_{ij} is the allocation weight from supply node i to demand node j. u_j be the under delivery $u_j = max\{0, (d_j - \sum_{i\in\Gamma(j)} s_i x_{ij})\}$, i.e. the number of impressions delivered less than d_j. Then our total penalty is $\sum_j p_j u_j$. In this paper, without loss of generality, we assume $p_j = 1, \forall j$.

In order to represent the expected long-term reward R^l during ad allocation, we define an expected value function q^l_{ijt}, as the expected accumulated long-term reward within time window $[t, t + T_q]$, where T_q is a time window that remains fixed throughout the serving process. After selecting ad j at inventory i, t is the time step of inventory i. We assume each inventory has a unique time step t, therefore t is omitted for simplification, we also omit notation l for long-term reward:

$$q_{ij} = q^l_{ijt} = E\left[\sum_{\tau\in[t,t+T_q]} R^l(i,j,\tau)\right] \tag{4}$$

Note that here we assume the q_{ij} is known in advance, we introduce notation q_{ij} to better illustrate the relation of the action of MDP in 2.1 with the bipartite matching solution.

Given these goals, we may write our optimal allocation in terms of a convex optimization problem:

$$Minimize \quad \frac{1}{2}\sum_{i,j\in\Gamma} s_i \frac{V_j}{\theta_{ij}}(x_{ij} - \theta_{ij})^2 + \lambda_c \sum_{ij} x_{ij} q_{ij} \tag{5}$$

$$s.t. \quad \sum_{i\in\Gamma(j)} s_i x_{ij} \geq d_j \qquad \forall j \tag{6}$$

$$\sum_{j\in\Gamma(i)} x_{ij} \leq 1 \qquad \forall i \tag{7}$$

$$x_{ij} \geq 0 \qquad \forall i, j \tag{8}$$

Where V_j is the priority of contract j, λ_c is a parameter for balancing long-term reward and representatives measure $s_i \frac{V_j}{\theta_{ij}}(x_{ij} - \theta_{ij})^2$, this is determined according to business requirements. θ_{ij} is the average targeted ratio of each ad j:

$$\theta_{ij} = \frac{d_j}{|\Gamma(j)|} \tag{9}$$

Constraints 6 are called demand constraints. They guarantee that total delivery matches the delivery goal of contract j. Constraints 7 are supply constraints, and they specify that we serve no more than one ad for each impression. Constraints 8 are simply non-negativity constraints. The optimal allocation for the guaranteed display ad problem is the solution to the above problem.

2.3 Bipartite Graph Serving

In order to serve the bipartite graph model, we refer to the solution in [3][1][12]. Using the Karush-Kuhn-Tucker (KKT) conditions of the optimal solution:

$$\mathbb{L} = f(\mathbf{x}^*) - \sum_j \alpha_j\Big(\sum_{i\in\Gamma(j)} s_i x^*_{ij} - d_j\Big) + \sum_i \beta_i\Big(\sum_{j\in\Gamma(i)} s_i x^*_{ij} - s_i\Big) - \sum_{i,j\in\Gamma} \gamma_{ij} s_i x^*_{ij} \tag{10}$$

$$\forall j, \qquad \alpha_k\Big(\sum_{i\in\Gamma(j)} s_i x^*_{ij} - d_j\Big) = 0 \tag{11}$$

$$\forall i, \qquad \beta_i\Big(\sum_{j\in\Gamma(i)} s_i x^*_{ij} - s_i\Big) = 0 \tag{12}$$

$$\forall (i,j) \in E, \qquad \gamma_{ij} s_i x^*_{ij} = 0 \tag{13}$$

$$\forall (i,j) \in E, \qquad \alpha_k \geq 0, \beta_i \geq 0, \gamma_{ij} \geq 0 \tag{14}$$

Where $\mathbb{L}$ is the Lagrangian , $f(\mathbf{x}^*) = \frac{1}{2}\sum_{i,j\in\Gamma} s_i \frac{V_j}{\theta_{ij}}(x^*_{ij} - \theta_{ij})^2 + \lambda_c \sum_{ij} x^*_{ij} q_{ij}$

By solving the KKT condition, we can calculate selection probability based on dual variables α^*_j and β^*_i:

$$x^*_{ij} = max\left[0, \theta_{ij}(1 + \frac{\alpha^*_j + \lambda_c q_{ij} - \beta^*_i}{V_j})\right] \tag{15}$$

The proof of the solution is in [12]. q_{ij} is the user interest estimation for supply node i(e.g. short-term rewards and long-term rewards) and demand j according to [3]. During the online serving phase, an ad is randomly chosen according to x^*_{ij} from a list of qualified ads. The qualified ads is a subset of ads that target current supply node and also subject to business related constraints, such as ad impression frequency constraints.

In practice there are two main challenges to solve this problem and find a feasible solution. Firstly, the input bipartite graph represents the full set of contracts and the full set of impressions, this makes the graph too large to solve. There are some works solving sampled graphs and producing an approximated solution to the original problem[12][3][14][1]. There are also works using parallel computing to provide solutions at larger scale. Secondly, estimating q_{ij} is difficult since the estimated value function q_{ij} varies for different allocation plans. Moreover, q_{ij} is dependent on contextual information such as the show user is watching, hour of day of the user playback session, previous shows and ads the user had watched. These contextual information cannot be modeled with bipartite graph model easily.

To solve the above mentioned challenges, in section 2.2 we propose using reinforcement learning to model the problem.

2.4 Reinforcement Learning Serving

We can use the updated reinforcement learning agent to serve the guaranteed ads online. The output action of the reinforcement learning agent plays the role of dual variable α and reward value estimation q. Compared to directly outputing selection probability of each ad-inventory pair, this design limits the output action space and allows faster convergence.

Similar to 15, the selection probability for ad j in inventory i is given by:

$$x^*_{ij} = max\left[0, \theta_{ij}(1 + \hat{\alpha}^*_j + \lambda_c \hat{q}_{ij} - \beta^*_i)\right] \tag{16}$$

Where $a_t = [\hat{\alpha}_j, \hat{q}_{ij}]$ is the output of the reinforcement learning agent, λ_c is a parameter controlling the balance between long-term reward and delivery constraints. β behaves similar to the dual variable β in the bipartite graph matching solution[12]. For each inventory i, we found the minimum possible $\beta \geq 0$ that:

$$\sum_{j \in \Gamma(i)} x_{ij}^* \leq 1 \tag{17}$$

The reinforcement learning agent will be trained every T_q minutes. We can use sampled episodes to train the agent. The details of the serving process is described in section 3.

3 AGENT IMPLEMENTATION

In this section we describe the details of reinforcement learning agent training and how to use the agent in the production environment.

3.1 Agent Framework

The agent uses an continuous actor-critic framework[5] shown in figure 3. The actor network is deterministic policy network $\pi(s_t|w^\pi)$ that takes state s_t as input and outputs action a_t. We use a deep neural network to approximate the function $\pi(s_t|w^\pi)$, where w^π is the parameter for the actor. We train the actor with policy gradient from the critic (20).

The critic is a Deep Q Network (DQN) that learns a function $Q^\pi(a_t, s_t)$ that takes action a_t and state s_t at time t as input, and output is a action value estimation at this state. $Q^\pi(a_t, s_t)$ can be expressed as the expected return after taking an action a_t in state s_t and thereafter following policy π.

$$Q^\pi(a_t, s_t) = \mathbb{E}_{r_{i\geq t}, s_{i\geq t} \sim \mathcal{E}, a_{i\geq t} \sim \pi}(G_t|s_t, a_t) \tag{18}$$

According to Bellman equation, and because action is deterministic $a_t = \pi(s_t)$, equation 18 can be rewritten as:

$$Q^\pi(a_t, s_t) = \mathbb{E}_{r_{i\geq t}, s_{i\geq t} \sim \mathcal{E}}[R_t(s_t, a_t) + \gamma Q^\pi(s_{t+1}, \pi(s_{t+1}))] \tag{19}$$

Since the expectation in equation 19 is only related to environment $\mathcal{E}$, this means we can sample action a_t from other policies π_β to train the Q network. Our implementation uses experience replay, double Q network structure and Temporal Difference (TD) Learning as proposed in [5][6][7].

In order to train the actor network, we use the deterministic policy gradient algorithm proposed in [9]. We use critic DQN as the source of action value estimation:

$$\nabla_{w^\pi} J \approx \mathbb{E}_{s_t \sim \rho^\pi}[\nabla_a Q^\pi(a, s|w^Q)|_{s=s_t, a=\pi(s_t)} \nabla_{w^\pi} \pi(s|w^\pi)|_{s=s_t}] \tag{20}$$

Where s_t is collected by running policy π in the environment $\mathbb{E}$. ρ^π is the distribution of s_t. w^Q is the learnable parameters for the critic, w^π

We have a novel action design, for inventory i at time step t, action output a list of $\hat{\alpha}_j, \hat{q}_{ij}$ for each qualified ad j for inventory i. In order to reduce the parameter number of the system, the actor is a DNN model that outputs $\hat{\alpha}_j, \hat{q}_{ij}$ one at a time. This means the input to the actor includes both s_t and feature vector for ad_j, where ad_j is a qualified ad for inventory i.

$$a_t = [\hat{\alpha}_j, \hat{q}_{ij}], \qquad \forall j \in \Gamma(i) \tag{21}$$

At each time step, the system selects an ad ad_t from a list of qualified ads for inventory i at time step t. The reward R_t, state s_t and action a_t at time step t are collected and stored in a replay memory, then uniformly sampled to train the actor network and critic network. We will describe the reward calculation logic in the next section.

3.2 Generalized Reward

In this paper we propose a novel reward design to optimize long-term reward with delivery constraints. Inspired by the Reward Constrained Policy Optimization (RCPO) algorithm[11], this generalized reward is defined by equation 2. We use reward R_t^c to represent the demand constraints 6 for each contract j.

In this section, firstly we express our constraint as an expectation:

$$J_C^\pi(j) = E_{s \sim \rho^\pi}^\pi[C_j(s)] \leq constant \tag{22}$$

Then we can derive the constraint as reward in the reinforcement learning, since the reward in reinforcement learning is an expectation over accumulated returns. Lastly we show that this design can converge in the learning process of reinforcement learning.

According to equation 6, although in theory we want the ad inventory allocated to each ad is exactly each contracts' demand d_j, in practice we are more interested in the under delivery for each contract. Therefore we are expecting total delivery D_{actual} will be larger than demand d_j:

$$D_{actual}(j) = E_{i \sim \Psi_{inv}}^\pi \left(\sum_{i \in \Gamma(j)} x_{ij} \right) \tag{23}$$

$$= E_{i \sim \Psi_{inv}}^\pi \left(\sum_{i \in \Gamma(j)} max[0, \theta_j(1 + \alpha_j - \beta_i + \lambda_c q_{ij})] \right) \tag{24}$$

$$\geq d_j \tag{25}$$

where inventory i is randomly arrives at the system with a fixed and unknown distribution Ψ_{inv}. $\Gamma(j)$ is the inventory samples collected from Ψ_{inv} that are targeted by contract j. Equation 24 shows that we can calculate selection weight x_{ij} using estimated variables $\alpha_j, \beta_i, q_{ij}$. One thing to note here is that x_{ij} is actually a function of $\alpha_j, \beta_i, q_{ij}$, we will omit the notation in this paper. By moving expectation from left-hand side to right-hand side, of equation 23, we get a constraint in the same form proposed in [11]:

$$J_C^\pi(j) = E_{i \sim \Psi_{inv}}^\pi \left[C_j(i) \right] \tag{26}$$

$$= E_{i \sim \Psi_{inv}}^\pi \left[\sum_{i \in \Gamma(j)} \left(\frac{d_j}{|\Gamma(j)|} - x_{ij} \right) \right] \tag{27}$$

$$\leq 0 \tag{28}$$

Where $C_j(i)$ represents the sum of constraints for ad j, and $|\Gamma(j)|$ is the number of inventories for ad j in its lifetime. Similar to [11], we define the constrained reward at each time step that allows the

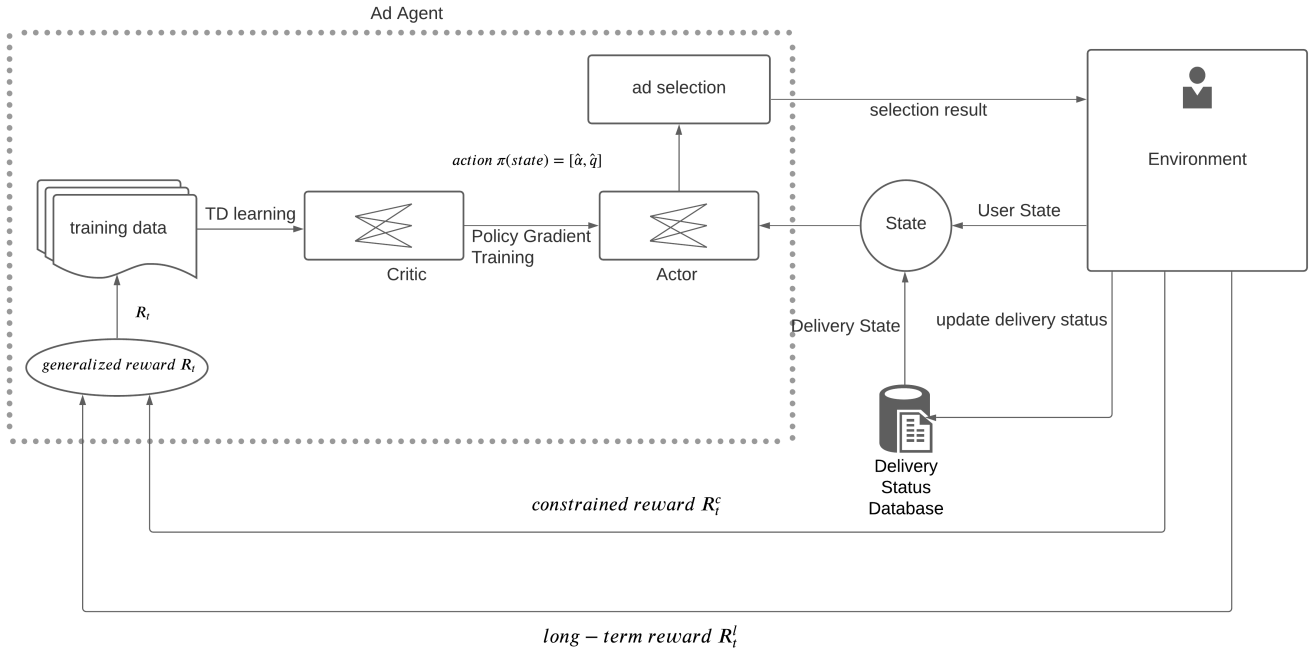

Figure 3: The actor-critic framework for guaranteed delivery advertising. We model the user as the environment $\mathbb{E}$ for reinforcement learning. During online serving stage, the input state consists of user state and pacing status collected from realtime delivery status database. The actor outputs action $a_t = [\hat{\alpha}, \hat{q}]$ as allocation plan. We can calculate selection weights from this allocation plan. Ads are selected randomly using selection weights. We update the delivery status after an ad is successfully delivered. We collect training data from production logs and calculate constrained reward R_t^c and long-term reward R_t^l according to section 3.2. We train the critic and the actor using Temporal Difference (TD) learning policy gradient, respectively.

agent to asymptotically converge to a feasible plan:

$$R_t^c = \sum_{j \in \Gamma(i)} -\lambda_p c_j(s_t, a_t) \tag{29}$$

$$= \lambda_p \sum_{j \in \Gamma(i)} (x_{ij} - \frac{d_j}{|\Gamma(j)|}) \tag{30}$$

Where $c_j(i) = c_j(s_t, a_t)$ is delivery constraint at each time step, s_t and a_t are the state and action of the MDP defined in section 2.1. We denote the penalty score at time step t as:

$$c_t = \sum_{j \in \Gamma(i)} c_j(i) \tag{31}$$

λ_p is a learnable parameter that acts similarly to the dual variable in convex optimization, it will increase when the under delivery is large, and lower otherwise. We will introduce the update of λ_p in section 3.3.

We can show that under some mild conditions the algorithm will converge our constrained reinforcement learning modeling that satisfies some assumptions. The detailed discussion is in appendix A.1.

3.3 Constrained Learning Algorithm

In this section we describe the learning algorithm for ad agents. The learning algorithm is a three-timescale actor-critic algorithm to learn the constrained reward proposed in section 3.2.

The output of the algorithm is a deterministic policy $\pi(s_t|w^\pi)$ that takes state s_t as input and outputs continuous action a_t defined in section 2.1. In this paper we use a DNN model to approximate the policy $\pi(s_t|w^\pi)$.

The algorithm is shown in algorithm 1.

3.4 Agent Online Serving

In order to simplify discussion, we define an episode for reinforcement learning as a time window $[t, t + T_q]$ at Hulu. The long-term reward is the total watch time in the window. The ad agent will take action at each ad break and select ads based on output action value.

In the online serving phase, the training of the model is scheduled every T_q minutes. The training data at time step $t \in [n \cdot T_q, (n+1) \cdot T_q]$ is collected using policy learned at the previous time window $t \in [(n-1) \cdot T_q, n \cdot T_q]$.

We use a random selection algorithm to choose ads. Given inventory i, we can derive M qualified ads $\{ad_1, ad_2, ..., ad_M\}$ from the ad server. Selection weights for each qualified ad is given by equation 15, where θ_{ij} is calculated by equation 9, $\hat{\alpha}_j$ and $\hat{q}_{ij}$ are the outputs

Algorithm 1: Constrained Actor Critic

Input:
penalty function $c(i)$, long-term reward weight λ_c and learning rates $\eta_1 > \eta_2 > \eta_3$
Output:
Learned actor $\pi(s_t|w^\pi)$ and critic $Q(s_t, a_t|w^Q)$.

Initialize actor $\pi(a_t|s_t w^\pi)$ and critic $Q(s_t, a_t|w^Q)$ with random weights
Initialize $\lambda_p = 0, t = 0, s_0 \sim \mu$
for $t \in$ *i-th time window* $[i \cdot T_q, (i+1) \cdot T_q]$ **do**
  Reset gradients $dw^Q \leftarrow 0$, $dw^\pi \leftarrow 0$ and $d\lambda_p \leftarrow 0$
  $C = 0$
  while s_t *not terminal* **do**
    Perform a_t according to policy $\pi(s_t|w^\pi)$
    Receive generalized reward R_t, next state s_{t+1}, penalty c_t in equation 31
    $t \leftarrow t + 1$
  end
  for $\tau \in t-1, t-2, ..., T_{q_i}$ **do**
    $\hat{R} \leftarrow R_\tau + \gamma \hat{R}$
    $dw^\pi \leftarrow dw^\pi + \nabla_{w^\pi} log\pi(s_\tau|w^\pi)(\hat{R} - Q(s_\tau, a_\tau|w^Q))$
    $dw^Q = dw^Q + \partial(\hat{R} - Q(s_\tau, a_\tau|w^Q))^2/\partial w^Q$
    $C = C + c_\tau$
  end
  if s_t *is terminal state* **then**
    $d\lambda_p \leftarrow -C$
    $t \leftarrow 0$
    $s_0 \sim \mu$
  end
  Update w^Q, w^π, λ_p by learning rate η_1, η_2, η_3
  Set $\lambda_p = max(0, \lambda_p)$
end

Table 1: Statistics of Hulu Dataset

Sessions	Ad Impressions	Contracts	Users
2, 622	21, 244	1, 916	2, 572

of current policy $\pi(s_t|w^\pi)$. β_i is calculated from equation 15 and constraint 17, where we use a binary search to find the minimum possible β_i to satisfy constraint 17, according to [12][3][14].

Delivered ad ad_j and watch time feedback at each time step are stored to calculate reward at t. For example, if we select ad_j at t and the user has finished watching ad_j, we will increase the delivery counter for ad_j by one. We will use the watched time from the current time t as a long-term reward.

4 EXPERIMENTS

In this section we conduct experiments to show the effectiveness of the proposed algorithm compared to bipartite matching baseline. We run the experiment on display advertising data at Hulu.

4.1 Dataset

In this paper we use user watch history and ad delivery history at Hulu. The dataset is sampled from historical logs collected from October 1, 2020 to October 31, 2020.

There are some key concepts in this dataset:

- **Watch Session** Users come to Hulu to watch shows, we define watch sessions as a collection of events, which starts when the user starts watching a show and ends when the user leaves the show.
- **Ad Impression** The ad server delivers ads during a user watch session. There will be a series of ads in a single watch session.

The detail of the dataset is shown in table 1. We will provide empirical experiment results on this dataset.

We consider following metrics in experiment:

- **Under delivery ratio** The ratio of under delivery count over delivery goal: $\mathbb{U} = \frac{\sum_j max[0, (\sum_{i \in \Gamma(j)} x_{ij} - d_j)]}{\sum_j d_j}$
- **Long-term Reward** The sum of long-term reward in the experiment: $\sum_t R_t^l$
- **Execution time** The execution time of the experiment.

In this experiment we build a simulated reward model that can provide simulated watch minute estimation. The reward model is a three layer neural network for regression that has 64, 64 and 1 neurons at each layer. The training data for this reward model is the production logs with watch minute as ground truth in the same time range of the experiment dataset. Both the baseline method and the proposed method have access to the reward model during the test. We then compare the sum of rewards generated by the reward model between the baseline method and the proposed method.

4.2 Baseline Algorithm

We use a simplified bipartite algorithm[12] as baseline to compare delivery constraint metrics and long-term reward metrics.

The objective for the baseline bipartite matching is:

$$\underset{x_{ij}}{Minimize}: \sum_{i,j} \frac{1}{2}(s_i x_{ij} - \frac{d_j}{|\Gamma(j)|})^2 + s_i r_{ij} x_{ij} \tag{32}$$

$$s.t. \quad \sum_{i \in \Gamma(j)} s_i x_{ij} \geq d_j \qquad \forall j \tag{33}$$

$$\sum_{j \in \Gamma(i)} x_{ij} \leq 1 \qquad \forall i \tag{34}$$

$$x_{ij} \geq 0 \qquad \forall i, j \tag{35}$$

Where r_{ij} is the long-term reward estimation based only on user and ad features (without contextual features such as the current content).

We model each ad impression as a supply node, each contract as a demand node. Since solving the bipartite matching needs inventory forecasting, we randomly sample 10% of the ad impressions and build a bipartite graph to approximate the original graph.

The solution of the bipartite matching is a list of dual variables α_j for each contract, we serve the bipartite matching allocation as proposed in section 2.3.

Table 2: Experiment result on Hulu dataset

Method	Under Delivery Ratio	Long-term reward (minutes)	Execution Time (minutes)
bipartite baseline	15.8 %	53469.3	5.0
proposed method	**15.5 %**	**53494.7**	10.0

4.3 Result

We compare the results of baseline and proposed algorithm in a simulated environment, where the ad request is from the Hulu production dataset as shown in table 1, and the long-term reward is given by a DNN model that is fitted on production data and predicts long-term rewards in the experiment.

Here we list the experiment result of the proposed reinforcement learning method and the bipartite matching baseline method in table 2.

4.4 Convergence Analysis

In order to evaluate the convergence of penalty parameter λ_p, we also plot the value of λ_p at each iteration of the experiment in figure 4. We set an upper bound on λ_p to avoid λ_p increasing without limit. In this experiment we set $max(\lambda_p) = 1.0$.

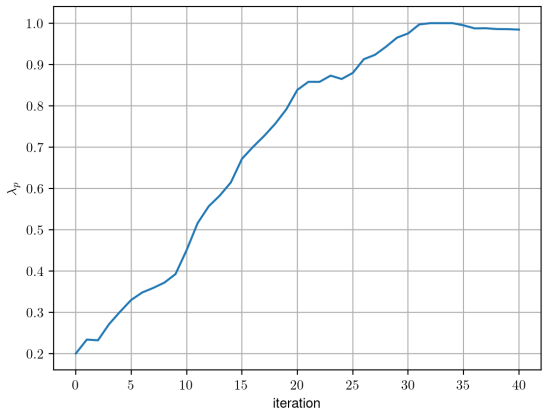

Figure 4: The convergence analysis of λ_p

As we can see from figure 4, λ_p increases at first of the experiment because the under delivery constraint is positive, and converges at the end of the experiment.

5 CONCLUSION

In this paper we proposed a multiple timescale reinforcement learning algorithm for guaranteed delivery advertising systems. This algorithm is proven to have comparable under delivery ratio metric with the state-of-the-art bipartite matching algorithm, while at the same time improves long-term reward performances. We are the first to propose using reinforcement learning to solve online ad allocation problems, this work allows the guaranteed ad delivery systems to adopt the recent advance techniques in the reinforcement learning field.

REFERENCES

[1] 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 *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*. 1195–1203.

[2] Xinshi Chen, Shuang Li, Hui Li, Shaohua Jiang, Yuan Qi, and Le Song. 2019. Generative adversarial user model for reinforcement learning based recommendation system. In *International Conference on Machine Learning*. PMLR, 1052–1061.

[3] Zhen Fang, Yang Li, Chuanren Liu, Wenxiang Zhu, Yu Zheng, and Wenjun Zhou. 2019. Large-Scale Personalized Delivery for Guaranteed Display Advertising with Real-Time Pacing. In *2019 IEEE International Conference on Data Mining (ICDM)*. IEEE, 190–199.

[4] Matthew Hausknecht and Peter Stone. 2015. Deep recurrent q-learning for partially observable mdps. *arXiv preprint arXiv:1507.06527* (2015).

[5] 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).

[6] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* (2013).

[7] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *nature* 518, 7540 (2015), 529–533.

[8] Alessandro Nuara, Francesco Trovo, Nicola Gatti, and Marcello Restelli. 2018. A combinatorial-bandit algorithm for the online joint bid/budget optimization of pay-per-click advertising campaigns. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 32.

[9] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. 2014. Deterministic policy gradient algorithms. In *International conference on machine learning*. PMLR, 387–395.

[10] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. 2017. Mastering the game of go without human knowledge. *nature* 550, 7676 (2017), 354–359.

[11] Chen Tessler, Daniel J Mankowitz, and Shie Mannor. 2018. Reward constrained policy optimization. *arXiv preprint arXiv:1805.11074* (2018).

[12] 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.

[13] Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. 2019. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature* 575, 7782 (2019), 350–354.

[14] Hong Zhang, Lan Zhang, Lan Xu, Xiaoyang Ma, Zhengtao Wu, Cong Tang, Wei Xu, and Yiguo Yang. 2020. A Request-level Guaranteed Delivery Advertising Planning: Forecasting and Allocation. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2980–2988.

[15] Weinan Zhang, Xiangyu Zhao, Li Zhao, Dawei Yin, Grace Hui Yang, and Alex Beutel. 2020. Deep Reinforcement Learning for Information Retrieval: Fundamentals and Advances. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 2468–2471.

[16] Xiangyu Zhao, Changsheng Gu, Haoshenglun Zhang, Xiaobing Liu, Xiwang Yang, and Jiliang Tang. 2019. Deep Reinforcement Learning for Online Advertising in Recommender Systems. *arXiv preprint arXiv:1909.03602* (2019).

[17] Xiangyu Zhao, Long Xia, Jiliang Tang, and Dawei Yin. 2019. " Deep reinforcement learning for search, recommendation, and online advertising: a survey" by Xiangyu Zhao, Long Xia, Jiliang Tang, and Dawei Yin with Martin Vesely as coordinator. *ACM SIGWEB Newsletter* Spring (2019), 1–15.

[18] Xiangyu Zhao, Xudong Zheng, Xiwang Yang, Xiaobing Liu, and Jiliang Tang. 2020. Jointly learning to recommend and advertise. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 3319–3327.

[19] Guanjie Zheng, Fuzheng Zhang, Zihan Zheng, Yang Xiang, Nicholas Jing Yuan, Xing Xie, and Zhenhui Li. 2018. DRN: A deep reinforcement learning framework for news recommendation. In *Proceedings of the 2018 World Wide Web Conference*.

167–176.
[20] Lixin Zou, Long Xia, Yulong Gu, Xiangyu Zhao, Weidong Liu, Jimmy Xiangji Huang, and Dawei Yin. 2020. Neural Interactive Collaborative Filtering. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 749–758.

A APPENDIX

A.1 Constrained Reward Assumptions

In this section we prove the convergence of proposed reward R_t. According to [11], The convergence of constrained reward is based on following assumptions:

- **a.1** Value function in **??** is bounded for all policies.
- **a.2** Every local minima of J_C^π is a feasible solution.
- **a.3** The learning rate of the reinforcement learning algorithm η_1 is faster than the learning rate η_2 of constrained reward λ_p.

Firstly, we can show the equation **??** is bounded. Since R_t at each time step is bounded by definition, and G_t is a sum over a finite time window, therefore **??** is bounded. Assumption **a.1** holds.

Secondly, assumption **a.2** requires every local minima of J_C^π is a feasible solution. Since our implementation randomly selects an ad based one the selection weights, this can be viewed as exploration around the local minima. Since a small value of J_C^π indicates small under delivery, therefore we can assume in practice the local minima of J_C^π lead to a small enough under delivery value, and assumption **a.2** holds.

Thirdly, assumption **a.3** holds when:

$$\sum_k \eta_1(k) = \infty \tag{36}$$

$$\sum_k \eta_2(k) = \infty \tag{37}$$

$$\sum_k (\eta_1(k)^2 + \eta_2(k)^2) = 0 \tag{38}$$

$$\frac{\eta_2(k)}{\eta_1(k)} \to 0 \tag{39}$$

We can design learning rates η_1, η_2 to meet these requirements. For example:

$$\eta_1(k) = 1/k^{0.5} \tag{40}$$

$$\eta_2(k) = 1/k \tag{41}$$

Therefore assumption **a.3** holds. □