Dynamic Ad Network Ordering Method Using Reinforcement Learning

International Journal of Computational Intelligence Systems (2022) 15:27
https://doi.org/10.1007/s44196-022-00077-6

RESEARCH ARTICLE

Dynamic Ad Network Ordering Method Using Reinforcement Learning

Reza Refaei Afshar[1] · **Yingqian Zhang**[1] · **Uzay Kaymak**[1]

Received: 2 September 2021 / Accepted: 21 March 2022

Abstract
Real time bidding is one of the most popular ways of selling impressions in online advertising, where online ad publishers allocate some blocks in their websites to sell in online auctions. In real time bidding, ad networks connect publishers and advertisers. There are many available ad networks for publishers to choose from. A possible approach for selecting ad networks and sending ad requests is called Waterfall Strategy, in which ad networks are selected sequentially. The ordering of the ad networks is very important for publishers, and finding the ordering that will provide maximum revenue is a hard problem due to the highly dynamic environment. In this paper, we propose a dynamic ad network ordering method to find the best ordering of ad networks for publishers that opt for Waterfall Strategy to select ad networks. This method consists of two steps. The first step is a prediction model that is trained on real time bidding historical data and provides an estimation of revenue for each impression. These estimations are used as initial values for the Q-table in the second step. The second step is based on Reinforcement Learning and improves the output of the prediction model. By calculating the revenue of our method and comparing that with the revenue of a fixed and predefined ordering method, we show that our proposed dynamic ad network ordering method increases publishers' revenue.

Keywords Reinforcement learning · Prediction model · Waterfall strategy · Ad network ordering · Real time bidding

Abbreviations

RTB	Real time bidding
ads	Advertisements
SSP	Supply side platform
DSP	Demand side platform
HB	Header bidding
RL	Reinforcement learning
MDP	Markov decision process
GDPR	General data protection regulation
DAO	Dynamic ad network ordering

1 Introduction

Online advertising is one of the most important sources of income for website owners, and its turnover is growing every year [11]. The process of online advertising consists of placing blocks like iframe in a website and selling them to the advertisers of services or products [31]. These blocks are called ad slots and they generate impressions upon the webpage is viewed by an end user [16]. Advertising revenue is important for publishers, and increasing this revenue is an important task of publishers.

The traditional approach for a publisher to sell impressions is to directly interact with advertisers through guaranteed contracts. This approach is not efficient because a publisher has to put extra effort into searching and finding appropriate advertisers, and the number of available advertisers is large. Programmatic Advertising is another way that computer programs perform the process of advertising for a publisher. In this way, Artificial Intelligence and Machine Learning methods help to develop data driven approaches for decision making tasks, and the process of selling ad slots is performed in a few milliseconds [5].

Real Time Bidding (RTB) is a programmatic advertising approach in which an ad network is responsible

✉ Reza Refaei Afshar
r.refaei.afshar@tue.nl

Yingqian Zhang
YQZhang@tue.nl

Uzay Kaymak
U.Kaymak@tue.nl

[1] Industrial Engineering Department, Eindhoven University of Technology, Eindhoven, The Netherlands

Published online: 19 April 2022

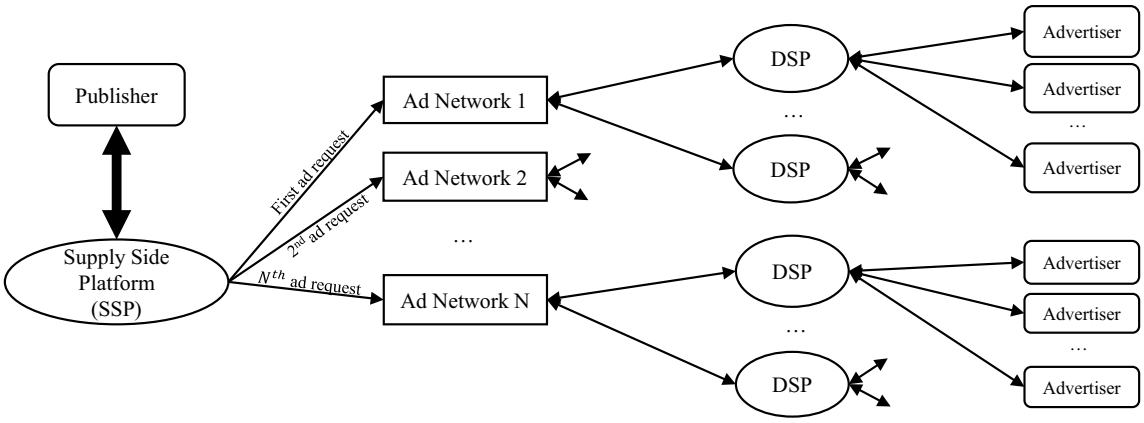


Fig. 1 The structure of the Waterfall Strategy from the publisher's point of view

to connect publishers and advertisers [41]. Ad networks run auctions to sell publishers' ad slots to the advertisers [15]. Ad networks receive the ad requests containing impression information from the publishers and perform online auctions in which the advertisers place their bids for the impressions. The advertiser with the highest bid is selected to show its advertisement (ads) on the publisher's website [38]. Basically, Demand Side Platforms (DSPs) and Supply Side Platforms (SSP) assist advertisers and publishers, respectively, to participate in the auctions [43]. Ad networks mostly run a second price auction with reserve price to sell the ad impressions [5]. The reserve price (also called the floor price) determines the minimum amount of money that the publisher expects to gain by selling the impression [43]. If the highest bid is less than the floor price, the impression remains unsold. Otherwise, the highest bidder is the winner, and it pays as much as the maximum of second highest bid and the floor price. If the auction of the current ad network has a winner, the process finishes, and the ad slot is filled with the winner's advertisement.

Different ad networks are available for a publisher to run auctions and sell a particular impression. Each ad network is connected to many advertisers with different preferences in buying an impression. Waterfall Strategy and Header Bidding (HB) are possible approaches for selecting ad networks. In Waterfall Strategy, the ad networks are selected sequentially, and in Header Bidding, all the ad requests are sent simultaneously. Although Header Bidding has become popular in the last few years, Waterfall Strategy is still used as an advertising approach for many websites because its latency can be significantly lower than HB [24]. According to [21], around 25% of websites in the US still use Waterfall Strategy as a standard advertising approach. Besides, in some recent RTB systems, HB is combined with Waterfall Strategy, and it is not completely replaced by HB [3]. We focus on the Waterfall Strategy to improve its performance as an existing approach, and the purpose of this paper is not to compare Waterfall Strategy and HB.

Waterfall Strategy is illustrated in Fig. 1. In the Waterfall Strategy, when a website is visited, an ad request is generated and is sent to the first ad network [9]. If the response of the first ad network is not successful, i.e., it cannot find an advertisement, the publisher should initialize and send another ad request to the second ad network. This process continues until selling the impression or consuming all of the ad networks and reaching a timeout [2].

This fixed ordering is not efficient in terms of revenue. The revenue comes from the winning bidder. This bidder is not necessarily the bidder with the maximum valuation for the impression because the possible higher bidders have never been approached. Therefore, the ordering of ad networks is very important for a publisher [23].

The results presented in this paper improve and extend upon earlier work by the same authors [1, 2]. In this paper, we develop a decision support system based on Reinforcement Learning (RL) to help publishers in selecting the ad networks in the Waterfall Strategy. Typical Waterfall Strategy relies on a predefined and fixed ordering of ad networks. We modify this framework and present a new strategy in which the ordering of ad networks is determined dynamically per impression. Our proposed method consists of a prediction model based on a supervised learning idea that is inspired from [33], in which a Deep Neural Network is initialized by a prediction model and further trained by RL. Unlike [33], in our method, first, the prediction model is trained using historical data and then initial value of each state action pair is calculated by the output of the prediction. The initialization step solves the sparsity of the reward function, because all combinations of ad requests and ad networks could not be observed in the Waterfall Strategy. In terms of latency, our method trains offline and provides an ad network using impression information only by looking at table Q-table. Therefore, our proposed method would not

take more than few milliseconds in theory. In particular, our contributions are as follows:

- Modeling the ad network selection procedure as a Markov Decision Problem (MDP) and developing an ad network ordering approach by integrating supervised learning and RL to solve the sparsity of the reward function.
- Introducing new benchmarks, including a set of predefined ordering and the output of the prediction model as a decision support tool, and demonstrating the effectiveness of our approach by comparing with these benchmarks.

In [1] we presented the preliminary work that contained the basic RL idea. In [2], the prediction model is added to the method as a separate step, and we showed that the RL framework improved the decision making. In this paper, we extend these works by introducing new benchmarks including predefined orderings and two ordering approaches based on the output of the prediction model, elaborating the experiments by analyzing the data to show the distribution of successful ad requests for each ad network, considering a new dataset to show the stability of the method in different time periods, and formally presenting the methodology that can be easily implemented and followed. Furthermore, the process of deriving the sequences and episodes is elaborated in detail. Therefore, this work is a substantial extension of the previous works.

2 Related Work

The literature contains extensive studies on revenue maximization for ad publishers. This section reviews literature related to RTB. Various methods have been developed for determining the floor price. Xie et al. introduce a method to set the floor price dynamically without any information about the bids [40]. In this method, a family of classifiers is used to predict whether the bid of a particular impression is high. The next level of prediction predicts the difference between the highest bid and second highest bid, and the floor price is set if this difference is high. In [4], a model parameter vector is learned by gradient descent, and the floor price is obtained by the inner product of this parameter vector and a feature vector containing auction information. In [20], the authors set the floor price in multi-channel real time bidding markets using separate mathematical models for setting up the floor price in offline and online channels. The online channel is the RTB auctions, and the offline channel is direct contracts with the advertisers. The methods introduced in [28–30] are useful in non-stationary environments. The price setting is based on considering the gap between the top bid and the second bid in the second price auction. In [42] proposes a pricing strategy by modeling the real time bidding environment as a dynamic game. In [32, 37], the contracts between publishers and advertisers are considered, which determine how many impressions should be sold offline and online. In our work, the reserve price is set using a fixed strategy derived from historical data, and this value is decreased constantly after each unsuccessful attempt.

RL has highly attracted researchers' attention in recent years, and it is also used in the context of real time bidding. In [36] an RL method is proposed to support the sellers in the dynamic pricing decisions where the auctions are not real time auctions. In this method, a learning algorithm tunes the parameters of a seller's dynamic pricing policy. In [6] an RL modeling of RTB is introduced to help advertisers in setting their bid price. In [39], the authors formulate the budget constraint bidding as a Markov Decision Process and propose a model-free RL framework to derive the optimal bidding strategy for the advertisers. In [18], the RTB environment is modeled as a multi-agent RL problem. Based on this modeling, the bidders learn how to act by considering other bidders as competitors. Unlike these previous approaches, in this paper, we model the problem of ad network ordering as an RL problem from the seller's side. Most of the work in this direction models the environment from advertisers' point of view, which is different from ours.

Despite its importance, the ad network ordering problem has gained less attention in recent years in comparison to dynamic pricing. In [19], the problems of ad network ordering and pricing are modeled as a multi-armed bandit problem, and a variant of the Upper Confidence Bound algorithm is used to derive the optimal strategy. In this modeling, a joint action consisting of a floor price and ad network ordering is considered in the bandit modeling. Our proposed method is different because we utilize contextual data, and the action is ad network ordering. Based on [14] the revenue is the most important factor when a publisher wants to select one ad network. However, there is no single ad network that can work well for all impressions. Therefore, dynamic ordering is necessary. In our preliminary works, we proposed two versions of the ad network ordering method based on RL [1, 2]. In this paper, the idea is elaborated, and complete experimentation is presented.

3 Problem Definition

As mentioned before, the typical Waterfall Strategy relies on the predefined and fixed ordering of ad networks, and this fixed ordering is not efficient in terms of revenue. Upon loading a webpage of a website containing an ad slot, the website owner or the publisher deals with a decision making problem. The problem is selecting an ad network at each

Table 1 Features of jth ad request of ith sequence

Field name	Notation	Definition	Type
Event state	e_{ij}	The result of attempt: 0: fail, 1: success	Binary
Timestamp	τ_{ij}	time of ad request (hour of a day)	Numerical
Country code	κ_{ij}	A code specify country of the user visiting publisher's website.	Nominal
Ad tag id	ϕ_{ij}	A unique string corresponds to an advertisement slot	Nominal
Ad network id	a	Id of each ad network (Ad exchange, AdSense, AOL,...)	Nominal
Page URL	Υ_{ij}	URL of the webpage containing the ad slot	Nominal
User properties (OS, browser, device, etc)	χ_{ij}	Information related to the end user	Nominal
Floor price	f_{ij}	The amount of floor price (reserve price)	Numerical
Request order	o_{ij}	Order of current attempt in a sequence of attempts	Numerical

decision moment among a list of possible ad networks to send an ad request. This ad network runs an auction to sell the ad slot, and the response is either successful or unsuccessful. Based on the response, the next decision moment occurs, or the process finishes. In our formulation, n is the number of ad networks, and $a = 1, \dots, n$ is an ad network. In order to develop the dynamic ad network ordering method, a set of ad requests and their responses are used in training and testing. Dataset D contains the set of all ad requests that are divided into p sequences. Each sequence contains the set of ad requests for filling a certain ad slot. l_i is the number of ad requests in the ith sequence. D is the union of three disjoint datasets D_1, D_2 and D_3 where D_1 is used in the first step of the proposed method, D_2 is used in the second step, and D_3 is the test dataset. Each ad request is denoted by $x_{ij}, i = 1, \dots, p\ ,\ j = 1, \dots, l_i$ which is the jth ad request of the i^{th} sequence and it contains a certain number of features. Let e_{ij} be a binary value that determines whether an ad request is successful. Using this notation, $p(e_{ij} = 1|x_{ij}) = p(e_{ij}|x_{i,j})$ is the success probability of sending x_{ij} to ad network a. Likewise, $p(e_{ij} = 0|x_{ij}) = 1 - p(e_{ij} = 1|x_{ij})$ is the probability of receiving an unsuccessful response by sending x_{ij} to a. Considering these features, each ad request x_{ij} is formally defined as follows:

$$x_{ij} = (\tau_{ij}, \kappa_{ij}, \phi_{ij}, a, \Upsilon_{ij}, \chi_{ij}, f_{ij}, o_{ij}),\quad i = 1, \dots, p\ \ j = 1, \dots, l_i \tag{1}$$

where, the descriptions of the features are elaborated in Table 1.

The problem of selecting ad networks in Waterfall Strategy is a sequential decision making problem [26]. At each attempt, the publisher decides to send an ad request to a particular ad network, and then a response is received. Based on this response, the next state is determined, and the publisher knows whether it should send another request to a new ad network or not. This setting follows the Markov property in which the next state and received reward are independent of all previous states [34, 35]. Therefore, we model the problem as an RL problem and try to learn state action values using initial state action values provided by the prediction model.

Our method is based on tabular RL. Tabular RL is used for solving ad network ordering problem, which is explained in Sect. 4 because the states and actions representations are finite and countable. In other words, we propose a modeling of dynamic ordering problem as an RL problem in which the states space and actions space are discrete and finite. Furthermore, the problem is episodic, and there is no prior information about the model of the environment. Hence, the Monte Carlo algorithm as a model-free RL algorithm is used for learning the state-action values. In the RL modeling of the ad network ordering problem, deriving a state representation is hard because not all features in an ad request are useful for representing the states. For instance, it has been observed that features like page URL are almost independent of the revenue, and it is better to be removed from the states. Furthermore, as shown in Sect. 4.1, features like floor price are highly important in the success of an ad request and the revenue. In Sect. 4.2.1, we will select a subset of features which represent the ad requests well and retain their important properties. The selected features are determined by using the feature importance listed in Table 2.

The historical ad requests that are obtained by following Waterfall Strategy are not sufficient for learning because, for many pairs of ad requests and ad networks, there is no information in the historical data. In other words, the data is sparse in terms of ad requests and ad networks pairs. In order to solve the sparsity, a prediction model is developed on a part of the data, and the output of the prediction model gives us the initial state-action values that will later be updated by using another part of the data. This prediction model is the first step of our proposed method, and the Monte Carlo algorithm is the second step. Figure 2 illustrates an overview of the method.

Our goal is to utilize the historical data to develop a decision support tool for ad publishers that follow Waterfall

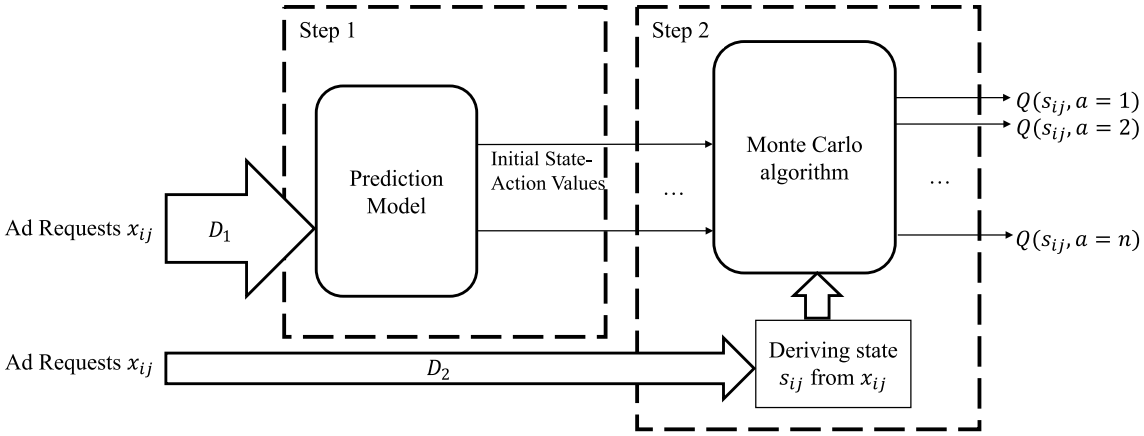


Fig. 2 An overview of our proposed method

Table 2 Top seven most important features

Rank	Feature	Importance
1	f_{ij}	0.303459
2	a	0.200992
3	o_{ij}	0.184083
4	Υ_{ij}	0.060835
5	χ_{ij}	0.059338
6	ϕ_{ij}	0.052920
7	τ_{ij}	0.027170

Strategy. Generally, our method is trained on a set of ad requests, and after training, it could be used as a decision support system. For each coming ad request, the model decides the best ad network that can provide the maximum revenue.

4 Dynamic Ad Network Ordering Method

In this section, we present our method for dynamically ordering the ad networks in the Waterfall Strategy. We are aiming to obtain an ordering by choosing the ad network at each decision moment that maximizes the estimated revenue. Our method consists of two steps. The first step is a prediction model, and it provides a lower bound of the estimated revenue for each combination of ad requests and ad networks. The second step is an RL modeling of the problem to improve the initial values obtained from the prediction model.

The first step of our proposed method is an offline process. Once this step is finished, the lower bound of the revenue is used as an initial value for the second step, which is the Monte Carlo algorithm. The second step could be performed either online or offline. In the online version, the value function is continuously updated for each coming ad request. In other words, before sending an ad request, the proper ad network is selected based on the value function, and upon receiving the response, the same value is updated. In the offline version, a set of ad requests are used for deriving the Q values, and this Q table is used as a lookup table for each coming ad request to decide the best ad network. The two steps are explained in the following subsections.

4.1 Prediction Model

The prediction model is a binary classifier that receives ad request x_{ij} and determines whether it is successful in finding advertisements or not. Vector x_{ij} is the feature vector shown in (1). The event state is the target feature that is used as the true label in supervised learning. We construct the prediction model such that it outputs probability values for the successful case, i.e. $p(e_{ij}|x_{ij})$.

The success probability has not the same unit as the revenue. Since the main goal is to increase the revenue, we estimate the revenue by multiplying the floor price and the success probability. Based on the second price auction, the revenue of each ad impression is obtained by (2).

$$R(x_{ij}) = \begin{cases} 0 & \text{if } f_{ij} > b_{ij}^{(1)} \\ \max\{f_{ij}, b_{ij}^{(2)}\} & \text{if } f_{ij} \leq b_{ij}^{(1)} \end{cases} \quad (2)$$

where, $b^{(1)}$ and $b^{(2)}$ are the highest bid and the second highest bid, respectively. Normally, the highest and the second highest bids for ad request x_{ij} and ad network a are not observable for the publisher. Hence, the actual revenue is unknown, and we need to estimate that. To find a value proportional to the actual revenue, we use the multiplication of the success probability and the floor price. This value is the estimated lower bound of revenue of x_{ij} when it is sent to a. Equation 3 shows this estimated revenue which is the output of the first step.

$$\mathbb{E}[\underline{R}(x_{ij})] = p(e_{ij}|x_{ij})f_{ij} \tag{3}$$

where $\mathbb{E}[\underline{R}(x_{ij})]$ is the estimated lower bound of revenue when x_{ij} is sent to a which is a feature of x_{ij}.

Different features of x_{ij} have different impacts on the prediction performance. In other words, the importance of the features is different, and the most important features determine the predicted class. The feature importances are computed as the mean deviation of accumulation of the impurity decrease within each tree of a random forest classifier[1] [13, 25]. Table 2 illustrates top seven important features with their importance values. This table shows that the total importance of floor price, ad network, and request order is around 0.7, and this value means that these three features are highly informative for learning. As it is explained in Sect. 4.2.1, the state representation of each ad request is obtained by considering this feature importance table. These values are provided by the prediction model [12].

4.2 Modeling with Reinforcement Learning Framework

In the following sub-sections, the components of a Markov Decision Process (MDP), including states, actions, and rewards, are defined. The agent in this setting is the publisher, and the environment contains the RTB participants from the publisher's point of view. In our case, the environment contains publishers, ad networks, and their interactions.

4.2.1 States

In our modeling, states are derived from ad requests. The combination of user information and impression properties construct ad request x_{ij}. The ad request x_{ij} also contains an integer number, namely Request Order (o_{ij}) that determines the number of unsuccessful attempts to fill the current ad slot. In other words, o_{ij} shows the number of ad networks that have been tested so far and cannot provide an advertisement. This information in x_{ij} is used to define the states.

A straightforward approach to define states is to use the combination of all features shown in (1). Since there are around 500,000 ad requests per day in our study case and no two ad requests are the same because each ad request has a specific time, defining states in this way results in large state space. Two possible solutions for coping with this state-space are removing the time feature or reducing the unique values of this feature by grouping them. However, after performing both approaches, the number of states is still large. User characteristics (Υ_{ij}) are removed due to GDPR[2] and the values of other features are grouped to reduce the size of state space. Among these features, the country code is irrelevant because the data is specifically for the The Netherlands. Timestamp induces slight separation according to Table 2. Page URL is initially included in the state representation; however, this entailed a large state space that the agent cannot be trained properly using the current configuration. This modeling requires function approximation methods such as REINFORCE, where the data is insufficient for that. Hence, the selection of the features is managed by removing these features from the state representation. Finally, the combination of the floor price, request order, and ad tag id provides a state space. This state space is not only small enough but also contains important information of the ad requests because they are selected based on the importance of the features illustrated in Table 2.

Table 2 shows the seven most important features. Floor Price and Request Order are selected from the top three. As it is explained in Sect. 4.2.2, ad networks are actions, and ad tag id (ϕ_{ij}) is the unique identifier of each ad slot which is important in determining the states. The sum of the importance of the selected features shows that they are the most informative features. We use the information of Table 2 to select the features for modeling states and actions. The total importance of the selected features for states and actions is around 0.7. The other features have many unique values, and they increase the size of state space, while the importance is less than 0.05 for each of them. Since the number of unique values of other features is large and they are not very important, we discard them to reduce the state space.

To further reduce the size of the space, the values of request order and floor price are converted to binary values. This technique is usually used for converting images to binary images [22]. For this purpose, two thresholds named θ_f and θ_r are defined. These two values are the medians over the values of the floor price and request order in the dataset D_2. We use medians because they balance the number of values in each group. This process reduces the number of unique states from 5×10^5 to 1245. Let F_{ij} and O_{ij} be the new binary values corresponding to the floor price and the request order, and they are shown in (4) and (5) respectively. The states are defined in (6).

$$F_{ij} = \begin{cases} 0 & \text{if } f_{ij} < \theta_f \\ 1 & \text{if } f_{ij} \geq \theta_f \end{cases} \tag{4}$$

$$O_{ij} = \begin{cases} 0 & \text{if } o_{ij} < \theta_r \\ 1 & \text{if } o_{ij} \geq \theta_r \end{cases} \tag{5}$$

[1] sklearn.ensemble.RandomForestClassifier

[2] General Data Protection Regulation

$$s_{ij} = (F_{ij}, O_{ij}, \phi_{ij}) \tag{6}$$

where s_{ij} is the corresponding state of x_{ij}.

4.2.2 Actions

At each decision moment, an ad network is selected to send an ad request. This ad network is an action in our modeling that is denoted by an integer $a \in \{1, \ldots, n\}$, where n is the number of available ad networks. The set of actions contains available ad networks that are modeled by integer numbers for simplicity. By sending an ad request and receiving the response, the environment transitions to a next state whose floor price is reduced, its ad tag id is the same, and its request order is incremented. In this way, the transition function modifies the request order and the floor price based on the response of the auction performed in the selected ad network.

4.2.3 Rewards

Two possible responses could be received by sending an ad request to an ad network. When the ad network is successful in finding a bidder for the ad slot, it is filled, and the publisher earns revenue that is at least equal to the floor price. Otherwise, the publisher needs to resend another request to another ad network. In order to reduce the number of unsuccessful attempts, a penalty is necessary for the algorithm to avoid unsuccessful attempts. Hence, a minus reward is chosen for the unsuccessful responses. The average of all floor prices of successful ad requests is around one, and a publisher will lose this value on average when it fails to find an advertiser. Therefore, for each x_{ij} we define f_{ij} as the reward of successful responses and -1 for the reward of unsuccessful responses. In other words, the rewards of all transitions in an episode except the last one are -1, and the reward of transiting to a terminal state is f_{ij}. Equation (7) shows the reward in our modeling.

$$r(s_{ij}, a) = \begin{cases} f_{ij} & \text{if } e_{ij} = 1 \\ -1 & \text{if } e_{ij} = 0 \end{cases} \tag{7}$$

where $r(s_{ij}, a)$ is the instant reward when the publisher selects a as an action for the state s_{ij} and e_{ij} is the event state of x_{ij}. The Return of s_{ij} is the cumulative instant rewards obtained from s_{ij} to the end of sequence i, i.e. given by (8).

$$G(s_{ij}, a) = \Sigma_{k=j}^{l_i} r(s_{ik}, a) = -(l_i - j) + f_{i,l_i} \tag{8}$$

where, l_i is the length of current sequence, $G(s_{ij}, a)$ is the return that follows s_{ij} and $a \in \{1, 2, \ldots, n\}$ is the observed ad network of kth ad request of the sequence i.

If the publisher gets an advertisement in a certain state, this state is called a terminal state, and the ad network selection is completed. Otherwise, the next state is a new state with different values of request order and floor price. Normally, when publishers follow a Waterfall Strategy and face unsuccessful responses, they decrease the floor price and resend the request to the following ad network. The new request order is one more than the previous value, and this is a property of the environment in state transition.

4.2.4 Learning Action Values

An episode in our modeling is the sequence of ad requests to fill a certain ad slot. Usually, there is no explicit information about the episodes in the RTB data. As mentioned in Sect. 3, the Monte Carlo method is selected for learning the action values because the length of each episode is short, and exact returns of episodes could be obtained easily. Sect. 5 explains how to obtain the episodes.

Equation (9) is the updating rule of the Monte Carlo algorithm [34]. In a typical Monte Carlo algorithm, the initial values are set to zero. We modify this initialization and use the values provided by the prediction model as the initial state-action values. The sample averaging of the Monte Carlo algorithm is a weighted average over all ad requests that are used in both steps. The first step processes the ad requests, and the second step deals with the states. They should be consistent in the updating rule.

The output of the first step is the estimated revenue of an ad request, and the second step expects to receive states-action initial values as input. In order to make them consistent, the request-based estimated revenue should be mapped to a state-based revenue. For this purpose, for each set of ad requests that are mapped to a single state, the average estimated revenue is calculated. This average is considered as the estimated revenue of the single state, as shown in (10).

$$Q(s_{ij}, a) = \frac{\sum_{q=1}^{n^2_{s_{ij},a}} G_q(s_{ij}, a) + \rho(s_{ij}, a) n^1_{s_{ij},a}}{n^1_{s_{ij},a} + n^2_{s_{ij},a}} \tag{9}$$

$$\rho(s_{ij}, a) = \frac{\sum_{s_{ij}=(F_{ij}, O_{ij}, \phi_{ij})} \mathbb{E}[\underline{R}(x_{ij})]}{n^1_{s_{ij},a}} \tag{10}$$

where $n^1_{s_{ij},a}$ is the number of ad requests used in the first step which corresponding state and action are s_{ij} and a. $n^2_{s_{ij},a}$ is the number of ad requests used in the second step which corresponding state and action are s_{ij} and a respectively. Let $G_q(s_{ij}, a)$ be the qth element of the list of returns that are observed when a is selected for s_{ij}, and $\rho(s_{ij}, a)$ be the

Table 3 Datasets that are used in each step of the dynamic ordering method

Dataset	Ad requests of	Dataset	Ad requests of	Purpose
D	20–26 November 2017	D'	01–06 July 2018	All the historical data
D_1	20–22 November 2017	D'_1	01–03 July 2018	Training the prediction model
D_2	23–25 November 2017	D'_2	04–05 July 2018	Training the RL part
D_3	26 November 2017	D'_3	06 July 2018	Evaluating the method

average of all estimated lower bound of revenues for those ad requests x_{ij} where $s(x_{ij}) = s_{ij}$. In order to compute $Q(s_{ij}, a)$ using (9) more efficiently, (12) shows the incremental version of (9). This equation can be easily obtained by modifying (9) in the way that is explained in [34].

Algorithm 1: Ad Network Ordering

Data: Two sets of Ad Requests, D_1 and D_2
Result: Q values for each ad request and ad network.

```
/* Step 1:                                          */
```
Train a Prediction Model with D_1 ;
Calculate $\mathbb{E}[\underline{R}(x_{ij})]$ using (3) for all x_{ij} of D_2 and all a;
Calculate $\rho(s_{ij}, a)$ using (10);
Find $n^1_{s_{ij},a}$ for each (s_{ij}, a) of D_1 ;
```
/* Step 2:                                          */
```
Initialize $Q(s_{ij}, a)$ with $\rho(s_{ij}, a)$ for each (s_{ij}, a) of D_2;
$iteration = 0$;
repeat
  foreach *Episode i in* D_2 **do**
    foreach *Ad Request* x_{ij} *in episode i* **do**
      Convert x_{ij} to s_{ij} and compute $G(s_{ij}, a)$ using (8);
      Update $Q(s_{ij}, a)$ based on (12) ;
      $n^2_{s_{ij},a} \leftarrow n^2_{s_{ij},a} + 1$;
    end
  end
  $iteration = iteration + 1$
until *iteration* $<$*a large number e.g.* 10^6;
return Q

$$Q_0(s_{ij}, a) = \mathbb{E}[\underline{R}(x_{ij})] \quad \forall i, j, a \tag{11}$$

$$Q_{t+1}(s_{ij}, a) = Q_t(s_{ij}, a) + \frac{1}{n^1_{s_{ij},a} + n^2_{s_{ij},a} + 1}(G(s_{ij}, a) - Q_t(s_{ij}, a)) \tag{12}$$

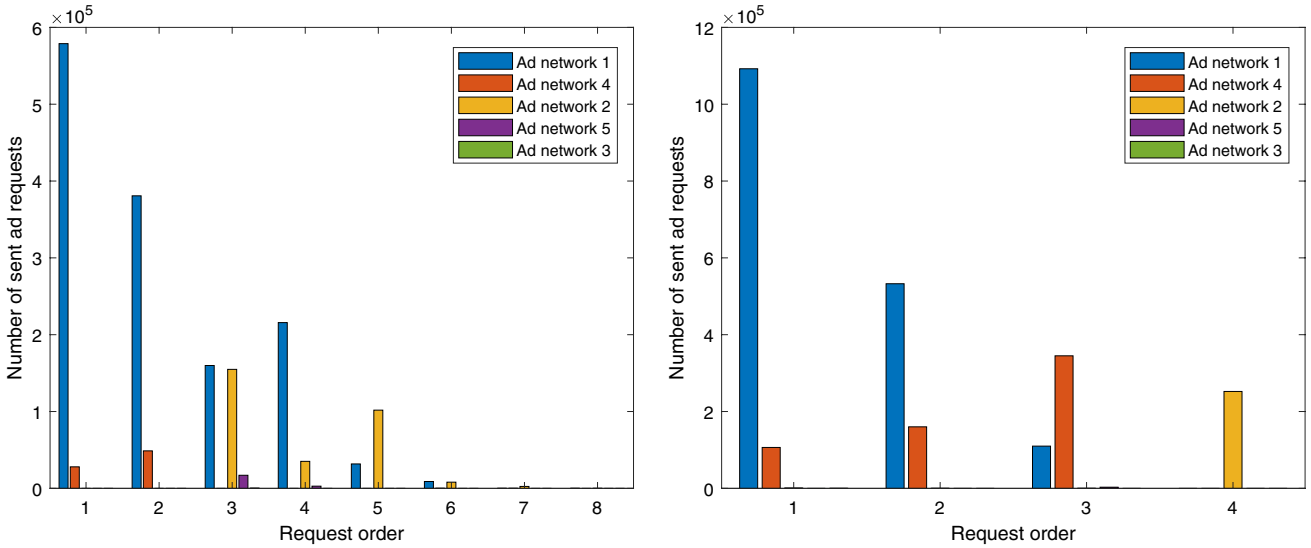
where t is the timestep. At each iteration, this value denotes the number of times that s_{ij} and a are observed so far in the second step. Now, we are ready to present our two-step algorithm. The pseudo-code of our proposed method is listed in Algorithm 1.

Algorithm 1 could be used in two different ways. The first one is an offline method. In this method, the final outputs of the algorithm are the Q values to be used with a greedy policy. If we have another dataset named D_3 containing new ad requests, we can determine the best ad network for each ad request by looking at the corresponding state in Q table and finding the action that corresponds to its maximized revenue. The second one is an online method. In this approach, the first step is performed separately, and the second step is run in a real environment, and it continuously updates the Q table for each coming ad request. In this version, while receiving a new ad request, first, the publisher decides the best ad network based on Q values. Then, the response is received, and the corresponding Q value is updated accordingly. We leave the online approach to future publications, and we focus on the offline method in this paper.

5 Data Description

In our experiments, we use two weeks of historical data. We define two datasets $D = D_1 \cup D_2 \cup D_3$ and $D' = D'_1 \cup D'_2 \cup D'_3$ that each corresponds to a particular period. D consists of the ad requests from 20th to 26th of November 2017 and D' contains the ad requests of the first six days of July 2018. Table 3 shows the properties of each dataset. The features in a typical ad request are shown in Table 1 [1]. These two datasets are used to evaluate the dynamic ad network ordering method. The purpose of using these two datasets is to show that although the properties of an RTB system are subject to change over time, the dynamic ad network ordering method stays helpful in maximizing revenue.

As data pre-processing, we perform data cleaning and sequence extraction. The ad requests of D and D' are grouped into p sequences, where each sequence consists of the ad requests to sell a particular impression. The sequences are very important in our modeling because they are considered as episodes in the Monte Carlo algorithm. However, normally there is no explicit information about the sequences



(a) Ad requests of D **(b)** Ad requests of D'

Fig. 3 The number of ad requests that are sent to each ad network for each request order and for the ad requests of **a** dataset D and **b** dataset D'. The predefined ordering of ad networks can be observed from these figures, and the majority of ad requests at each request order are sent to a certain ad network

in the RTB data. We assume that for two ad requests that aim to fill a single ad slot, features like user characteristics, ad tag id, and URL are the same. Furthermore, the time difference between these two ad requests may not be more than a few milliseconds. Our sequence extraction algorithm iterates over all ad requests of both D and D' once, and the output is the set of detected sequences. At each step, an ad request x is selected, and it is compared with the last ad request of all sequences that have been found so far. If there is an ad request that its request order is one less than the request order of x, and other features except time and floor price are the same, the current ad request is appended to the corresponding sequence. The pseudo-code is illustrated in Algorithm 2.

There are two types of sequences. A complete sequence ends with an ad request whose request order is one. It means that the sequence is successful in finding an advertisement for a certain ad slot. The other type is incomplete sequences. This kind of sequence cannot find an advertisement due to various possible reasons like a timeout. Since it is not clear why a sequence is incomplete, the incomplete sequences are not helpful in predicting the event state. Therefore, all of these sequences are removed from the first step. The next necessary pre-processing step is dealing with nominal features.

The last column of Table 1 shows the type of features in an ad request. Some of the features are nominal, and they should be converted to numerical values. We use One Hot Encoding to convert each value of each feature to a binary value [17]. One hot encoder assigns a column for each feature value. For example, if a feature contains 1000 unique values, the corresponding one hot feature vector contains 1000 columns. Since the number of unique values of features like URL is very large, using all of these values in the feature vector results in a large vector which is hard to manage. To solve this problem, all feature values with more than 100 unique values are sorted in descending order based on their frequency. The top 100 values are selected to be converted to numerical features. All the other values with lower frequencies are aggregated into a single feature value. Therefore, for each feature with more than 100 unique values, 101 feature columns are assigned in one hot encoding. We used 100 because the total frequency of the top 100 unique values is more than 90% of all values for each feature. Finally, the feature vector contains 666 features.

Since the number of unsuccessful ad requests is far more than the number of successful ones, the dataset is not balanced. Oversampling methods like SMOTE are not suitable for our problem because they may produce incorrect data samples [8]. For example, it is possible that SMOTE produces a data sample with a float value for request order. We use the random under-sampling method to balance the dataset because the successful ad requests are more important, and we try to keep them intact [7]. For this purpose, a subset of unsuccessful ad requests is selected, which its size is equal to the number of successful ad requests. Based on this process, the dataset is balanced, and it has an equal number of data instances for each target class.

Preliminary analysis on the relations between features like floor price and the revenue are discussed in [27]. We

extend this analysis by focusing on the predefined orderings and the number of ad requests for each ordering. In order to show the predefined orderings that are observed in the dataset, for each request order, the frequencies of observed ad networks are calculated, and they are shown in Fig. 3a, b. As shown in these figures, ad network 1 is the most popular, and the publisher prefers to test it again after an unsuccessful attempt. The frequencies of other ad networks are very low, and they are hardly observable in the figures.

Algorithm 2: Find sequences of ad requests

Data: Ad requests set D and D'
Result: Sequences of ad requests
Initialize empty sets S and S_{open} to store sequences;
foreach *Ad request* $x = (\tau, \kappa, \phi, a, \Upsilon, \chi, f, o, e)$ **do**
  foreach *Open sequence i in* S_{open} **do**
    Let x_{il_i} be the last ad request of i ;
    if $(\kappa = \kappa_{il_i})$ & $(\phi = \phi_{il_i})$ & $(\Upsilon = \Upsilon_{il_i})$ & $(\chi = \chi_{il_i})$ & $(o = o_{il_i} + 1)$ & $(|\tau - \tau_{il_i}| < 200ms)$ **then**
      Append x to sequence i and set $l_i = l_i + 1$
    end
    if $e == 1$ **then**
      Add i to S and remove it from S_{open}
    end
  end
  if *No open sequence is found for x* **then**
    Add x to a new sequence i' and add this sequence to S_{open}. set $l_{i'} = 1$
  end
end
Add all remaining open sequences in S_{open} to S ;
return S

6 Experiments

In this section, the performance of the proposed method is discussed. First, the prediction model is evaluated in terms of how well it can predict the revenue of each ad request. Then the estimated revenue of using the two-steps method is compared with the real revenue to show the benefits of our method. In order to understand the added value of each step, we select ad networks based on the output of the first step. These experiments are performed by using historical RTB data.

6.1 Evaluation Metrics

To evaluate the prediction model, we employ ROC Curve and area under ROC curve (AUC) as they demonstrate the quality of the binary classification models well. Since the Waterfall Strategy relies on the predefined ordering of ad networks, we calculate the revenue of different permutations of ad networks as a baseline to show that no predefined ordering of ad networks can provide more revenue. Thus, the second step and the combined model are evaluated by comparing the total revenue with the baselines, including the first step and the predefined ordering.

6.2 Evaluation of the Prediction Model

In order to build the prediction model, different classification algorithms, i.e. Decision Tree, Support Vector Machine, Gaussian Naive Bayes, and Random Forest, are tested. The hyper-parameters of these algorithms, like the number of trees in Random Forest, are found by random search. We used ROC Curve to compare the performance [10]. The ad requests of D are used for identifying the proper classifier. The classifier is trained on the ad requests of D_1, and the obtained prediction model is tested on the ad requests of D_3. Figure 4 compares the classification algorithms. As it is illustrated in this figure, Random Forest works best in terms of AUC. Therefore, the random forest method with a maximum depth of 10 and 100 trees is selected as the prediction model.

Although the distributions of ad requests of varying days are different, the classifiers are not biased to a certain day. In order to show that the prediction model works almost the same for different days, the ad requests of each day of $D_2 \cup D_3$ and $D'_2 \cup D'_3$ are used to test the two prediction models, respectively. Figure 5a, b illustrate the ROC Curves for each prediction model. The prediction model could be trained once and is useful for predicting the success probability of future ad requests without any retraining. However, it is observed that the prediction model that is trained on D does not work well for D'. Figure 5a, b show that by training the prediction model on a set of ad requests, we can predict the success probability of the ad requests of coming days with a good performance.

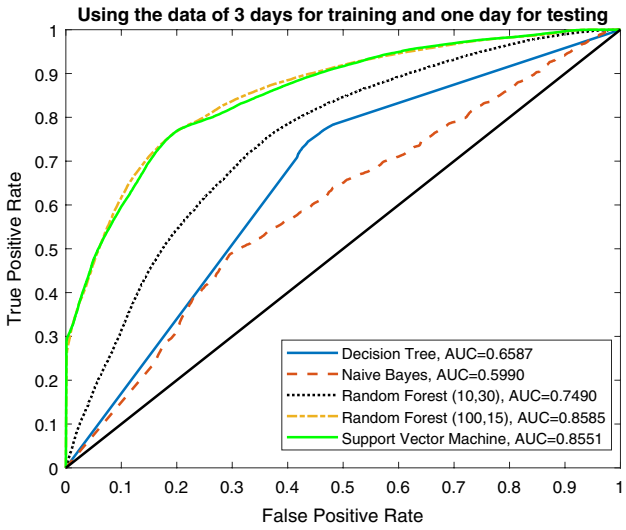


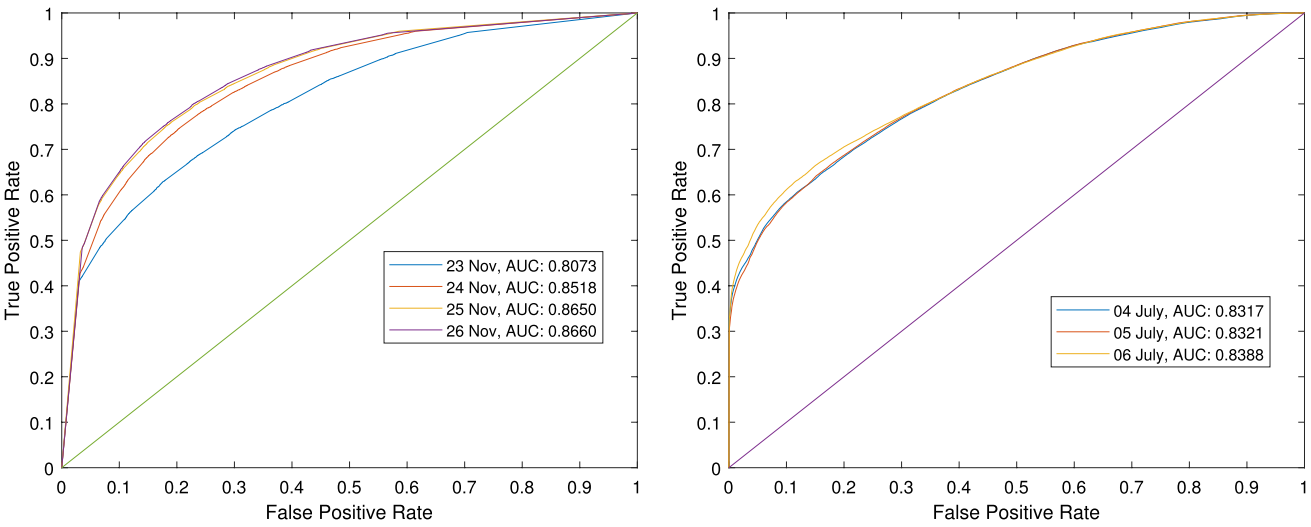
Fig. 4 A comparison between different classification methods. The prediction model is trained using D_1 and tested on D_3. The numbers in the parentheses of Random Forest indicate the number of trees and the maximum depth of each tree respectively

6.3 Evaluation of Dynamic Ad Network Ordering Method

The output of the proposed two-steps method is an ordering of ad networks that aims to provide the maximum revenue for a publisher. In order to evaluate this method, two concepts are defined: Real Revenue and Estimated Revenue. The real revenue is the sum of floor price of those x_{ij} where e_{ij} is one. Similarly, the estimated revenue is the revenue that is obtained by applying the dynamic ad network ordering method. This revenue is the sum of the maximum Q value of the first ad request of each sequence. In other words, for the first ad request of all sequences, the Q values corresponding to the ad network that provides the maximum revenue are summed. By summing these values, an estimation of the revenue is obtained. The other ad requests of each sequence are ignored because the Q values are the average of episode returns, and if the first ad request is successful, the process finishes. Therefore, the Q values of the first ad requests contain the revenue of other ad requests in this sequence.

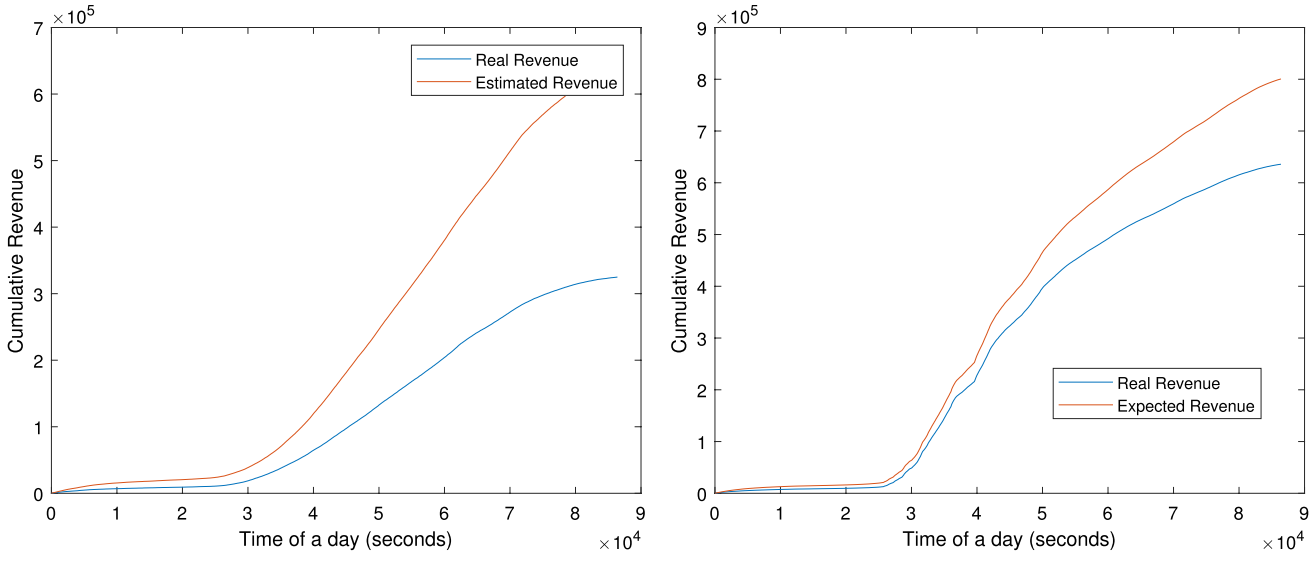
As illustrated in Table 3, two datasets are used for evaluating the method. The dynamic ad network ordering method is tested on D and D' separately. D_1 and D'_1 are used for training the prediction model. D_2 and D'_2 are used in RL step. The output of the Monte Carlo algorithm is a $Q(s, a)$ for each possible s and a. The estimated revenue is obtained by applying the method on D_3 and D'_3. The cumulative revenue curves that are obtained from the data (real revenue) and from applying the method (estimated revenue) are illustrated in Fig. 6a, b. Based on these figures, we can conclude that using the two-steps method can increase the revenue drastically.

By comparing Figs. 6a, b in terms of the difference between the real revenue and the estimated revenue of our method, one can see that this difference is reduced in D'_3. In other words, the real revenue is closer to the estimated revenue in the recent dataset. We explored the reason and found out that in the newer dataset, the ordering of the ad networks is not completely fixed. The strategy is still waterfall. However, the policy of selecting an ad network after



(a) Training on D_1 **(b)** Training on D'_1

Fig. 5 ROC Curves for evaluating the prediction models in case that how well they can predict the success probability of the ad requests of two datasets. **a** Is trained on D_1 and is test on each day of $D_2 \cup D_3$. **b** Is trained on D'_1 and is tested on $D'_2 \cup D'_3$



(a) Testing on D_3 **(b)** Testing on D'_3

Fig. 6 Real revenue vs. estimated revenue

Table 4 Frequencies of the orderings of ad networks in D and D'

Observed in D	Frequency	Observed D'	Frequency
1	737, 601	1	1, 306, 297
1, 1	542, 038	1, 1	757, 304
1, 1, 2	341, 503	1, 1, 5	580, 019
1, 1, 1	112, 088	1, 1, 5, 2	451, 638
1, 1, 1, 1	95, 418	1, 5	246, 286
1, 1, 2, 1	68, 799	1, 5, 1	159, 646
1, 1, 1, 1, 2	44, 609	1, 5, 1, 2	141, 567
1, 1, 2, 1, 2	38, 037	5	87, 715
1, 5	14, 804	1, 5, 5	48, 196
5	13, 073	1, 5, 5, 2	46, 142

More than 98% of observed orderings in D start with 1 and then 2. However, D' has more different predefined orderings and this is the reason that higher revenue is obtained in D' than D

getting an unsuccessful response is different. Table 4 shows the topmost frequent orderings and their frequencies. As we can see from this table, the policy of deciding the ad networks has been changed in the new dataset. In D, the first ad network is the main one, and the publisher prefers it as the second attempt for almost 90% of impressions. However, in D', the first ad network is chosen to send the second ad request for less than 80%. This is the reason behind the difference in the total revenue because the system does not follow the predefined ordering for around 20% of ad requests, and this leads to increase the revenue. The dynamic ordering method in this paper aims to completely replace the predefined ordering.

A question that can be arisen here is why the publisher sends the next ad request to the first ad network when it was already unsuccessful. Actually, each ad network sets the floor price and runs an auction. If the ad network cannot find a bidder, one possible way is to reduce the floor price and run the auction again. That is why we see sequences that contain several attempts to a certain ad network. Apparently, the first ad network is the most popular for the publisher, and they reduce the floor price before sending the next ad request to the same ad network.

6.4 Prediction Model as an Ordering Method

In order to find out the most important step in dynamic ad network ordering method, we test each step for decision making. If the power of the method is due to the prediction model, we can use it as an ordering method, and there is no need for any improvement by the RL step. This question is answered by using the output of the prediction as an ordering strategy.

For evaluating this method, two different approaches are followed. The first one is to consider the first ad request of each sequence. This definition is similar to the way that is followed for finding the total revenue in Sect. 6.3. However, it does not make sense here to just keep the first ad request and ignore the rest because the revenue that is obtained from the prediction model is not the return of the episode. The estimated revenue is the sum of $\mathbb{E}[\underline{R}(x_{ij})]$ for the first ad request of all sequences. The blue curve in Fig. 7 shows this revenue. Based on this figure, this ordering method does not

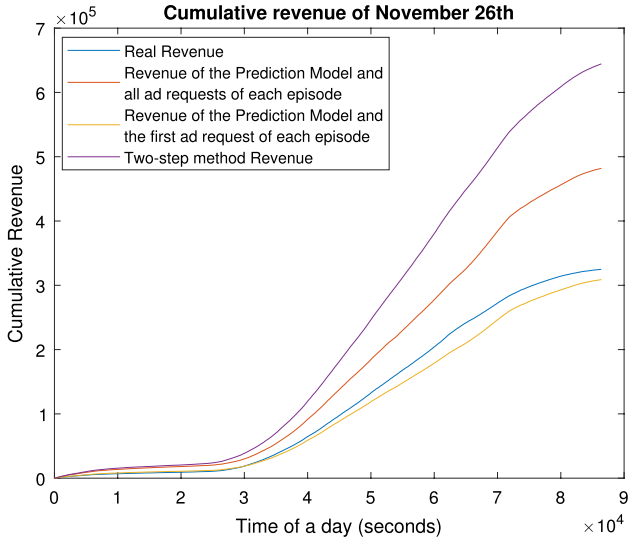


Fig. 7 Comparison between the revenue obtained from different ordering methods

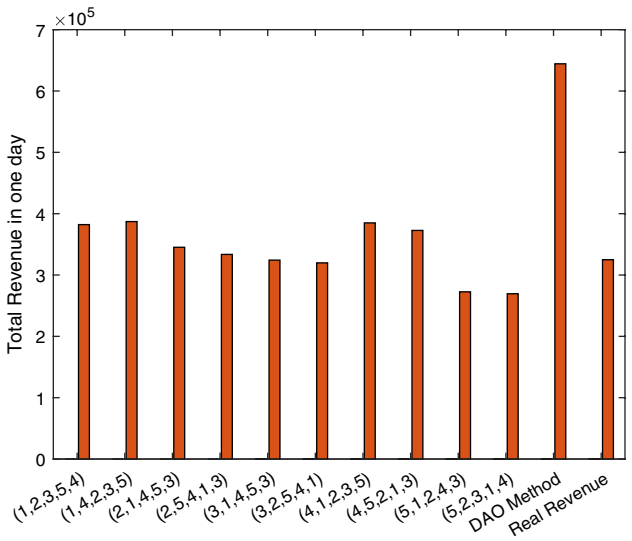


Fig. 8 Comparing the highest revenue obtained from predefined ordering with the revenue of dynamic ad network ordering (DAO) method. Since there are five ad networks, 120 different predefined orderings are possible. For each ad network a, we found the top two orderings that start with a and provide the maximum revenue. The revenue of these ten orderings and also the real revenue and the estimated revenue of our proposed method are shown as 12 bars

outperform the predefined ordering. Nevertheless, it can be a good estimation of the real revenue.

The second approach for defining the estimated revenue is to find the estimated revenue of the sequences. As mentioned in Sect. 3, $p(e_{ij}|x_{ij})$ is the success probability of sending x_{ij} to a and the revenue is found by (3). With the probability of $1-p(e_{ij}|x_{ij})$, a cannot provide an advertisement and x_{ij} should be sent to another ad network. An estimated revenue can be found in this way. Equation (13) shows this estimated revenue.

$$\begin{aligned} E(i) =& p(e_{ij}|x_{i0})f_{i0} + (1-p(e_{ij}|x_{i0}))p(e_{ij}|x_{i1})f_{i1} \\ &+ \cdots + (\prod_{j=1}^{l_i-1}(1-p(e_{ij}|x_{ij})))p(e_{ij}|x_{il_i})f_{il_i}, \end{aligned} \tag{13}$$

where i is an episode or sequence of ad requests to fill a certain ad slot and x_{ij} is the j^{th} ad request of sequence i. f_{ij} is the floor price of x_{ij}. The red curve in Fig. 7 illustrates this revenue.

The revenue obtained based on different strategies is illustrated in Fig. 7. Using the output of the prediction model for selecting ad networks provides less revenue in comparison to the two-step method. Therefore, the RL part really improves the decision approach and increases the revenue.

6.5 Comparing Different Predefined Orderings

In the dataset, there are five different ad networks. To identify them easily, we assign integer numbers starting from 1 to these ad networks. Therefore, there are 120 different predefined orderings. The cumulative revenue of all of them is computed and then sorted based on descending ordering of the total revenue. In order to show that none of these predefined orderings can do better than our method, the top two orderings that are started with each ad network are selected and illustrated in Fig. 8. Hence, the figure shows twelve different orderings. In other words, these ad network selection policies work as a benchmark to clarify the superiority of our proposed method. The 1, 4, 2, 3, 5 sequence is the predefined ordering with the maximum revenue among others. As it is clear from this figure, the maximum revenue that can be obtained from the predefined ordering is almost half of the revenue of our two-step method.

7 Conclusion

Since online advertising is growing rapidly and its turnover becomes larger and larger every year, website owners and online publishers find it suitable for increasing their revenue. For websites that provide free services, online advertising is a substantial source of income. Therefore, revenue maximization in online advertising is of great importance for online publishers. As mentioned before, ad networks are responsible for running ad auctions, and they serve as interfaces between publishers and advertisers. In order to connect with the ad networks, Waterfall Strategy is a common way in which the ad requests are sent to a set of ad networks sequentially until an advertisement is acquired. Typically,

the ordering of ad networks is predefined and fixed, which affects the revenue of ad publishers. Hence, deriving a policy to decide the best ad network ordering in Waterfall Strategy is crucial for ad publishers.

This paper proposed a two-step method that can be used as a decision support system for ad publishers who decide to participate in the auction via a Waterfall Strategy. We considered the ad network selection procedure as a sequential decision making problem and utilized RL to derive the most profitable ordering of ad networks. Our method might help publishers to find the best ad network for each impression. The selected ad network could provide a higher estimated revenue in comparison with other ad networks. If the first ad network is not successful in finding an advertisement, our method recommends the second best ad network. Therefore, the output of the two-step method for each impression is an ordering of ad networks that is dynamically provided per impression.

In recent research, the applications of RL in the RTB environment have mainly considered the advertisers as primary agents. Although significant, few researchers have focused on the system from the publisher's point of view. Our proposed method is a decision support tool for ad publishers to decide the ordering of ad networks for each generated impression. Using historical data for evaluating our proposed method, the revenue of ad publishers is significantly increased in theory. Although latency in real-time environment may lead to different performance than the theoretical result, our proposed method does not take more than few milliseconds for deciding an ad network because the ad network is obtained by looking at Q-table which can be performed in constant time.

The main limitation of this work is using the available data. The data is acquired from a waterfall strategy system based on predefined ordering, and that limits exploration because the reward function is sparse. For this reason, we employed tabular RL and handled this issue by using a prediction model; however, a dataset with fewer unseen state-action pairs would improve the quality of the model. Besides, accessing an actual RTB environment would allow the agent to explore, and we did not have this access for this work. Another limitation of this work is induced by the highly dynamic environment, which makes the methods less effective after some time. In this case, we have to retrain our approach frequently to adapt to the new environment properties.

Acknowledgements The authors would like to thank Azerion and Triodor for sharing the data used in this research, and Surfsara - the Dutch national High-Performance Computing and e- Science support centre, for providing high-performance systems.

Author Contributions Modeling the ad network selection procedure as a Markov Decision Problem (MDP) and use Reinforcement Learning to solve it; Introducing new benchmarks including a set of predefined ordering and the output of the prediction model as a decision support tool, and demonstrating the effectiveness of our approach by comparing to these benchmarks. Providing initial values for Q-table using prediction model.

Funding This work was supported EU EUROSTARS (Project E! 11582).

Availability of data and material The data is uploaded along with the manuscript.

Declaration

Conflict of interest The author(s) declare(s) that they have no competing interests.

References

1. Afshar, R.R., Zhang, Y., Firat, M., Kaymak, U.: A reinforcement learning method to select ad networks in waterfall strategy. In: Proceedings of the 11th International Conference on Agents and Artificial Intelligence. SCITEPRESS-Science and Technology Publications (2018)
2. Afshar, R.R., Zhang, Y., Firat, M., Kaymak, U.: A decision support method to increase the revenue of ad publishers in waterfall strategy. In: IEEE Conference on Computational Intelligence for Financial Engineering and Economics (CIFEr) (2019)
3. Afshar, R.R., Zhang, Y., Firat, M., Kaymak, U., Metin, A.I., Tarakçıoğlu, G.S., Baş, C.: Reserve price optimization with header bidding and ad exchange. In: 2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC). IEEE, pp. 830–835 (2020)
4. Austin, D., Seljan, S., Monello, J., Tzeng, S.: Reserve price optimization at scale. In: 2016 IEEE 3rd International Conference on Data Science and Advanced Analytics (DSAA). IEEE, pp. 528–536 (2016)
5. Busch, O.: The Programmatic Advertising Principle, pp. 3–15. Springer, Cham (2016)
6. Cai, H., Ren, K., Zhang, W., Malialis, K., Wang, J., Yu, Y., Guo, D.: Real-time bidding by reinforcement learning in display advertising. In: Proceedings of the Tenth ACM International Conference on Web Search and Data Mining. ACM, pp. 661–670 (2017)
7. Chawla, N.V.: Data mining for imbalanced datasets: an overview. In: In Data Mining and Knowledge Discovery Handbook, pp. 875–886. Springer, Berlin (2009)

8. Chawla, N.V., Bowyer, K.W., Hall, L.O., Kegelmeyer, W.P.: Smote: synthetic minority over-sampling technique. J. Artif. Intell. Res. **16**, 321–357 (2002)
9. Dinodia, P.: Header bidding vs waterfall: How the two revenue optimisation hacks differ (2017)
10. Fawcett, T.: An introduction to roc analysis. Pattern Recogn. Lett. **27**(8), 861–874 (2006)
11. Fuchs, C.: The Online Advertising Tax as the Foundation of a Public Service Internet. University of Westminster Press, London (2018)
12. Garreta, R., Moncecchi, G.: Learning Scikit-Learn: Machine Learning in Python. Packt Publishing Ltd, Birmingham (2013)
13. Géron, A.: Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems. O'Reilly Media, Inc., Newton (2017)
14. Ghosh, A., McAfee, P., Papineni, K., Vassilvitskii, S.: Bidding for representative allocations for display advertising. In: International Workshop on Internet and Network Economics. Springer, Berlin, pp. 208–219 (2009)
15. Graham, R.: A brief history of digital ad buying and selling (2010)
16. Ha, L.: Online advertising research in advertising journals: a review. J. Curr. Issues Res. Advert. **30**(1), 31–48 (2008)
17. Harris, D., Harris, S.: Digital Design and Computer Architecture. Morgan Kaufmann, Burlington (2010)
18. Jin, J., Song, C., Li, H., Gai, K., Wang, J., Zhang, W.: Real-time bidding with multi-agent reinforcement learning in display advertising. In: Proceedings of the 27th ACM International Conference on Information and Knowledge Management. ACM, pp. 2193–2201 (2018)
19. Kveton, B., Mahdian, S., Muthukrishnan, S., Wen, Z., Xian, Y.: Waterfall bandits: learning to sell ads online (2019). arXiv preprint arXiv:1904.09404
20. Li, J., Ni, X., Yuan, Y.: The reserve price of ad impressions in multi-channel real-time bidding markets. IEEE Trans. Comput. Soc. Syst. **5**(2), 583–592 (2018)
21. Loebbecke, C., Cremer, S., Richter, M.: Header bidding as smart service for selling ads in the digital era. J. Inf. Syst. Eng. Manag. **5**(4), em0123 (2020)
22. McAndrew, A.: An introduction to digital image processing with Matlab notes for scm2511 image processing. Sch. Comput. Sci. Math. Vic. Univ. Technol. **264**(1), 1–264 (2004)
23. Muthukrishnan, S.: Ad exchanges: research issues. In: WINE '09 Proceedings of the 5th International Workshop on Internet and Network Economics, pp. 1–12 (2009)
24. Pachilakis, M., Papadopoulos, P., Markatos, E.P., Kourtellis, N.: No more chasing waterfalls: a measurement study of the header bidding ad-ecosystem. In: Proceedings of the Internet Measurement Conference on—IMC '19, pp. 280–293 (2019)
25. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al.: Scikit-learn: machine learning in python. J. Mach. Learn. Res. **12**, 2825–2830 (2011)
26. Puterman, M.L.: Markov Decision Processes: Discrete Stochastic Dynamic Programming. Wiley, New York (2014)
27. Refaei Afshar, R., Zhang, Y., Firat, M., Kaymak, U.: Reinforcement learning method for ad networks ordering in real-time bidding. In: van den Herik, J., Paula Rocha, A., Steels, L. (eds.) Agents and Artificial Intelligence, pp. 16–36. Springer, Cham (2019)
28. Rhuggenaath, J., Akcay, A., Zhang, Y., Kaymak, U.: A PSO-based algorithm for reserve price optimization in online ad auctions. In: 2019 IEEE Congress on Evolutionary Computation (IEEE CEC) (2019)
29. Rhuggenaath, J., Akcay, A., Zhang, Y., Kaymak, U.: Fuzzy logic based pricing combined with adaptive search for reserve price optimization in online ad auctions. In: IEEE International Conference on Fuzzy Systems (FUZZ-IEEE) (2019)
30. Rhuggenaath, J., Akcay, A., Zhang, Y., Kaymak, U.: Optimizing reserve prices for publishers in online ad auctions. In: 2019 IEEE Conference on Computational Intelligence for Financial Engineering and Economics (CIFEr) (2019)
31. Ryan, K.M., Graham, R.S.: Digital Display Advertising, pp. 85–100. Palgrave Macmillan US, New York (2014)
32. Sayedi, A.: Real-time bidding in online display advertising. Mark. Sci. **37**(4), 553–568 (2018)
33. Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al.: Mastering the game of go with deep neural networks and tree search. Nature **529**(7587), 484 (2016)
34. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT, Cambridge (2018)
35. Szepesvári, C.: Algorithms for reinforcement learning. Synth. Lect. Artif. Intell. Mach. Learn. **4**(1), 1–103 (2010)
36. Vengerov, D.: A gradient-based reinforcement learning approach to dynamic pricing in partially-observable environments. Futur. Gener. Comput. Syst. **24**(7), 687–693 (2008)
37. Wang, J., Chen, B.: Selling futures online advertising slots via option contracts. In: Proceedings of the 21st International Conference on World Wide Web. ACM, pp. 627–628 (2012)
38. Wang, J., Zhang, W., Yuan, S., et al.: Display advertising with real-time bidding (rtb) and behavioural targeting. Found. Trends® Inf. Retriev. **11**(4–5), 297–435 (2017)
39. Wu, D., Chen, X., Yang, X., Wang, H., Tan, Q., Zhang, X., Xu, J., Gai, K.: Budget constrained bidding by model-free reinforcement learning in display advertising. In: Proceedings of the 27th ACM International Conference on Information and Knowledge Management. ACM, pp. 1443–1451 (2018)
40. Xie, Z., Lee, K.C., Wang, L.: Optimal reserve price for online ads trading based on inventory identification. In: Proceedings of the ADKDD'17. ACM, pp. 6 (2017)
41. Yuan, S., Wang, J., Zhao, X.: Real-time bidding for online advertising: measurement and analysis. In: Proceedings of the Seventh International Workshop on Data Mining for Online Advertising. ACM, p. 3 (2013)
42. Yuan, S., Wang, J., Chen, B., Mason, P., Seljan, S.: An empirical study of reserve price optimisation in real-time bidding. In: Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, pp. 1897–1906 (2014)
43. Zhang, W., Yuan, S., Wang, J.: Optimal real-time bidding for display advertising. In: Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, pp. 1077–1086 (2014)