Bid Shading In First-Price Real-Time Bidding Auctions

Tuomo Tilli

Master's Thesis

Master of Engineering - Big Data Analytics

2019

MASTER'S THESIS	
Arcada	
Degree Programme:	Master of Engineering - Big Data Analytics
Identification number:	7253
Author:	Tuomo Tilli
Title:	Bid Shading In First-Price Real-Time Bidding Auctions
Supervisor (Arcada):	Leonardo Espinosa Leal
Commissioned by:	ReadPeak Oy

Abstract:
Online advertisements can be bought through a mechanism called real-time bidding (RTB). In RTB the ads are auctioned in real time on every page load. The ad auctions can be second-price or first-price auctions. In second-price auctions the one with the highest bid wins the auction, but they only pay the amount of the second highest bid. In this paper we focus on first-price auctions, where the buyer pays the amount that they bid. The buyer should bid more than others to win the impression, but only as little amount more as possible and at maximum what they consider the impression to be worth. This research will evaluate how multi-armed bandit strategies will work in optimizing the bid size in ReadPeak's first-price real-time bidding environments. ReadPeak is a demand-side platform (DSP) which buys inventory through ad exchanges. We analyze seven multi-armed bandit algorithms on offline data from the ReadPeak platform. Three algorithms are tested in ReadPeak's production environment. We discover that the multi-armed bandit algorithms reduce the bidding costs considerably compared to the baseline. This has potential to bring significant savings for the advertiser. More research is required to get a decisive result on which algorithm performs the best in the production environment.

Keywords:	bid shading, bid optimization, multi-armed bandits, ReadPeak
Number of pages:	53
Language:	English
Date of acceptance:	

CONTENTS

Figures

Tables

1 INTRODUCTION

1.1 Background

The way advertisements are bought in online medias is vastly different from how it is done in traditional print media. In print media, for instance a newspaper, an advertiser can pay to have their ad shown on the front page of the newspaper. This ad is then shown to all the readers of the newspaper. In online advertising, advertisers can buy, for example, a certain number of impressions in a publication for the ad. This means that the advertiser can decide and get a guarantee on the number of people who will see the ad.

In online advertising, a technique called programmatic ad buying (Gonzálvez et al., 2016) is getting more and more popular. In programmatic ad buying, computer software handles the purchasing of the ads, instead of people negotiating the deals. This makes the ad buying process efficient as humans do not need to be involved as much.

A specific type of programmatic ad buying is real-time bidding (Google., 2011). In real-time bidding (RTB) the ads are auctioned in real time on every page load. When a user loads a web page a bid request is triggered. The bid request is sent to an ad exchange which routes the request to the advertisers. The advertisers will respond with a bid of their choice. Often the bid size is chosen relative to the value the advertiser considers the impression to be.

The bid request may contain data about the publisher and the user who triggered the page load. This data can be used to determine the value of the specific impression. Whoever bids the highest will win the impression and the ad will be shown on the page. A single page may contain multiple ad slots thus the auction process may happen multiple times per page load.

Often publishers use supply-side platforms (SSPs) to sell their inventories. SSPs are used as intermediaries between the publisher, the ad exchange and the advertisers. SSPs enable publishers to open their inventories to multiple advertisers. This increases competition and in turn enables the publishers to maximize their profits. With SSPs publishers can also set a floor price under which they will not sell the impression and the publishers can control which advertisers are allowed to buy the inventory.

Publishers can use a technique called header bidding to connect to multiple SSPs or ad exchanges and reach even more advertisers simultaneously. Header bidding gets its name from the fact that the bid request is triggered in the web page header. The bid request may be sent to multiple SSPs or ad exchanges simultaneously. The ad exchanges and SSPs return the winning bid from their internal auction and the publisher will select the winner from the incoming bids. This gives more power to the publisher to control the ad shown on the site.

Advertisers in turn use demand-side platforms (DSP) to buy the advertisement inventory offered through the ad exchanges (Muthukrishnan, S., 2009). DSPs enable advertisers to use a single interface to connect to multiple publishers through multiple ad exchanges. With DSPs advertisers can select the publishers they want their ads to be shown at. Advertisers can also choose to target certain users and set the price they are willing to pay to reach the users.

Figure 1 describes the online advertising environment. The environment consists of different entities. Advertisers will use DSPs to buy inventory from the sources that they choose. DSPs use ad exchanges to connect to the inventory. The publishers provide the inventory through SSPs. The publishers use SSPs to set floor prices and select which advertisers may buy impressions from them. Finally, the SSPs connect to the ad exchanges to link the advertisers with the publishers.

In RTB, a bid request is created by a publisher, when a user visits a web page. The bid request is sent to an SSP or multiple SSPs. The bid request is then routed through an ad exchange into a DSP. The DSP then decides whether it wants to respond to the bid request with a bid on behalf of an advertiser.

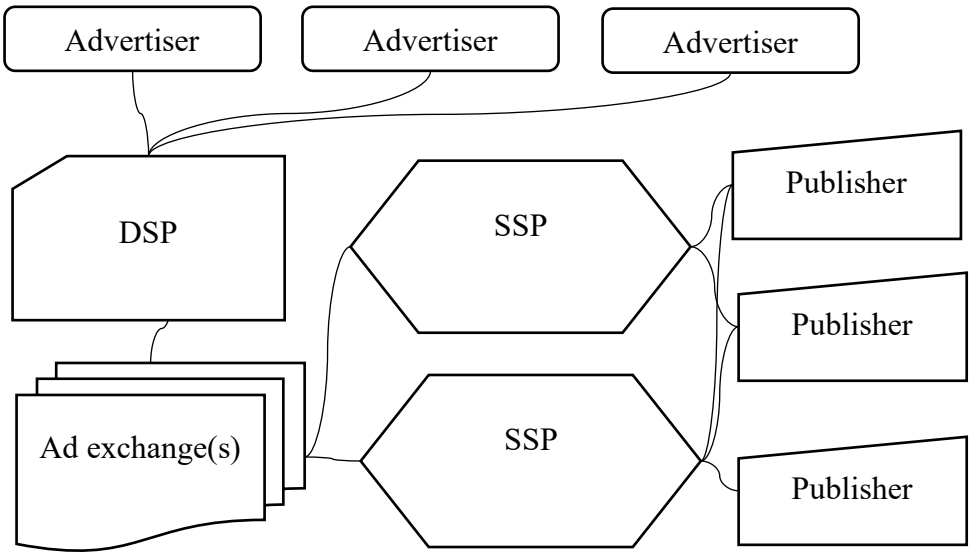

Figure 1. An illustration of the online advertising environment.

How can the advertisers determine the right amount to bid? Advertisers should have an idea what a user on their site is worth to them. This allows them to calculate the amount to bid. The bid is the amount the advertiser is willing to pay for the click. DSPs often bill advertisers on a cost per click (CPC) basis and publishers bill ad exchanges on a cost per mille (CPM) basis.

The Open RTB specification (IAB Real Time Bidding (RTB) Project) specifies that the ad auction may be a second-price (Edelman et al., 2007) or a first-price auction (Levin, J., 2004). Generally, RTB auctions have been second-price auctions. In second-price auctions the one with the highest bid wins the auction, but they only pay the amount of the second highest bid. In second-price auctions it is a weakly dominant strategy to bid the amount one values the win at (Levin, J., 2004).

The issue with second-price auctions is that malicious publishers or intermediaries could modify the actual price of the winning bid. As long as the cost is lower than the actual bid, the advertiser would have no way of knowing if they are paying the correct price.

With the rise of header bidding the auctions are moving towards first-price auctions. First-price auctions make the transactions more transparent as the buyers know exactly what they will pay for the impressions. On the other hand, first-price auctions are problematic for the buyer as they pose a risk of overbidding. The buyer should bid more than others

to get the impression, but only as little amount more as possible and at maximum what they consider the impression to be worth. The process of working out the right amount to bid is called bid shading (Sluis, S., 2017). Figuring out the right amount to bid could mean life or death to a DSP, as bidding too much could result in unhappy customers who will move elsewhere.

1.2 Motivation and Aim of the Study

Working out the best amount to bid in first-price ad auctions has potential to save a lot of money for the buyer. When there are tens or hundreds of millions of auctions being processed, saving just a little bit of money per auction quickly builds up. However, when the number of other participants in the auctions is unknown, it may be hard to figure out the right amount to bid. The number of the other participants varies and the participants will have their own bidding strategies. This results in a complex environment where there are many unknown factors.

This research will investigate how to optimize the bid size in first-price auctions. The study will focus on finding out the least amount that will win the auction in a specific situation. The amount may change depending on different factors like weekdays or times of the day. The goal is to find a way to be able to estimate the winning price as close as possible. The estimate can be used in determining the right amount for an advertiser to bid or whether they should bid at all (although an advertiser should always bid even if they value the impression less than the estimate, as they have nothing to lose).

1.3 Data and Methods

This thesis will use data from real OpenRTB auctions happening through ReadPeak's (https://www.readpeak.com/) Demand Side Platform (DSP). ReadPeak is a platform that enables advertisers to promote their content online. ReadPeak makes it easy for advertisers to buy inventory from a publisher that they choose.

The auction data will contain the auction timestamp as well as information on the bid request like the requesting publisher and the ad placement on the page. The data will also contain the bid sizes that were bid and the information whether the bid won the auction

or not. The data will only contain the winning bid amounts for the bids that were won by a bid from the ReadPeak DSP. The other bid sizes will be unknown.

This research will evaluate how multi-armed bandit strategies will work in optimizing the bid size in ReadPeak's first-price real-time bidding environments. The amount to bid will be given a range and the range will be divided into slots where each slot acts as a bandit's arm. Different bandit algorithms will be evaluated against each other to find the best performing one. The bandit algorithms will be compared with the base line of always bidding the maximum value of the impression. The best algorithms will also be evaluated in the production environment.

This study will research only multi-armed bandit algorithms and whether they are suitable in solving this type of problems. This study will not go into other machine learning or reinforcement learning methods.

1.4 Definitions

Ad exchange - A software platform used to connect advertisers and publishers. Ad exchanges hold bidding auctions. DSPs and SSPs are connected through ad exchanges.

Bid shading – The process of working out the amount to bid in an auction, where the amount is less than what the buyer thinks the impression is worth.

Click-through rate (CTR) - Clicks divided by impressions.

Cost Per Click (CPC) - Price of one click.

Cost Per Mille (CPM) - Price of a thousand impressions.

Demand-side platform (DSP) - A software platform used by advertisers to buy inventory from publishers. Advertisers can select which publishers to buy from and set the price they are willing to pay.

Effective Cost Per Mille (eCPM) - A value used to compare different pricing options. For campaigns charged on CPM, eCPM equals CPM. For campaigns charged on CPC, eCPM is calculated with the following formula: eCPM = CPC * CTR * 1000.

First-price auction - A type of auction where the one with the highest bid wins and and the winner pays the price they have bid with.

Header bidding - A type of programmatic technique where ad auctions are triggered from the header of the page when the page loads. Bid requests may be sent to multiple ad exchanges to reach as many advertisers as possible. Publisher holds a first price auction using the bids returned by the ad exchanges to determine the ad to show.

Programmatic ad buying - Ad buying done by computer software. Ads may be bought through real time auctions or advertisers may buy a guaranteed number of impressions from a certain publisher.

Real-time bidding (RTB) - A type of programmatic ad buying where the ads are sold through auctions held in real time when a web page loads.

Second-price auction - A type of auction where the one with the highest bid wins, but the winner pays only the amount of the second highest bid.

Supply-side platform (SSP) - A software platform used by publishers to sell inventory to advertisers. Publishers may select which advertisers are allowed to advertise on their site and may set a floor price under which they will not sell the inventory.

1.5 Structure of the Thesis

The rest of the thesis is divided into four chapters. Chapter two presents the related work in the area. Chapter three describes the methods used in solving the problem. In chapter four the results of the experiments are presented. Finally, chapter five provides discussion of the results, concludes the thesis and provides options for future work.

2 RELATED WORK

Bid optimization in real-time bidding has been studied extensively. In Ghosh et al. (2009), the authors consider the problem of acquiring a given number of impressions with a given budget. They take into account both the case where all of the winning bids are known and the case where only the bids won by the bidder are known. In header bidding environments only the winner knows the bid price of the winning bid per auction. In their approach the algorithm learns the distribution of the other bidders in the exploration phase and then bids against that. They consider the bid distribution of the other bidders to remain the same, which may not be true in modern header bidding environments.

Another approach to device a bidding strategy is to try to forecast the bid distribution of the advertising campaigns, as is done by the authors in Cui et al. (2011). They use historical bidding data to build a gradient boosting decision tree to forecast the bid distribution. They aim to estimate how many impressions an advertiser will win by bidding a certain amount. This approach may work to some extent. However, it does not take into account the non-stationary nature of the bid landscape.

Most of the previous work on bid optimization has focused on second-price auctions, as is the case in the studies described above. The authors in Perlich et al. (2012) present a bidding strategy to optimize the bidding price. They compare three different bidding price strategies. Strategy one is to multiply the bid with the score ratio of the particular inventory. Strategy two is to bid 0, if the score ratio is less than 0.8, bid 1 if score ratio is between 0.8 and 1.2. Then for ratios over 1.2 bid twice the base price. The baseline is to always bid the amount set by the advertiser. The paper is more focused on estimating the probability of a conversion for the inventory than optimizing the actual bid. The bidding strategies themselves are quite simple.

A lot of the bid optimization research has focused on search-based auctions. The authors in Amin et al. (2012) consider the bid optimization problem faced by an advertiser where they want to get as much value (clicks) for the budget. They cast the problem as a Markov Decision Process (Bellman, R., 1957) with censored observations. This means that they know the winning price only on the bids that they have won. They propose a learning method based on the Kaplan-Meier method (Kaplan, E. L., & Meier, P., 1958). The Kaplan-Meier method has the advantage that it performs also on censored data. Their

method works by dividing the campaign time into periods. The algorithm tries to maximize the number of clicks purchased within the period with the budget. Although they also consider second-price auctions, this is the same goal that we want to achieve in our research.

Repeated auction theory has been researched from the sellers' perspectives also. The researchers in Amin et al. (2013) propose an algorithm for the seller to optimize the floor price against strategic (non-truthful) bidders. The floor price is the minimum amount the seller is willing to sell the good for and non-truthful bidders are bidders who do not always bid the amount they value the impression at. In second-price auctions it makes sense to bid the impression value, but in first-price auctions, which are becoming more and more common, the buyer should strategically modify the bid size to be as low as possible. The authors view the problem as a bandit problem. However, they note that a traditional bandit algorithm like Upper Confidence Bound (UCB) (Sutton & Barto, 2017) or EXP3 (Auer et al., 2002) may not perform well as the buyer may strategically adjust the bidding once the seller's algorithm reaches a price it considers optimal. This is the same kind of an issue that we need to consider in our research, as the bidders we are competing against may adjust their strategies based on our bidding behavior.

The authors in Weed et al. (2015) propose models to combat these kinds of situations where the reward structure may change. They propose a model for the stochastic and adversarial cases. In stochastic models the rewards are randomly distributed around the value of the goods. In adversarial model the rewards may change. For the stochastic model they propose an UCB-type algorithm. The adversarial bandit problems are tricky for algorithms like UCB as they will not be able to react very quickly to the changing rewards. The authors propose an algorithm based on EXP3 as the standard EXP3 algorithm is designed for a finite number of actions. In their auction setup the number of actions is unbounded as is the case in our research also.

In Perlich et al. (2012) the authors suggested that the best bidding strategy was linearly related to the predicted CTR of the ad. The authors in Zhang et al. (2014) were one of the first ones to discover the non-linear relationship between the optimal bid and the impression evaluation. Their research suggests that one should try to bid more on low price impressions than focus on high valued impressions. They do not consider the best

strategy to be to bid the true value of the impression. They suggest that other aspects should be taken into account including the total budget of the campaign, time remaining for the campaign and the expected amount of impressions to be won with the bid. This way they can optimize the performance of the campaign as well as possible. They show that their method performs better than bidding linearly related to the predicted CTR, although the linear method performs relatively well on big budgets. The authors in Zhang et al. (2014a) also show that the linear method preforms well in second-price auctions. These methods cannot be used directly in our case though as they focus on second-price auctions where bidding high is not punished as much as in first price auctions.

The issue with censored data in real-time bidding environments is approached in Wu et al. (2015) and Zhang et al. (2016). In Wu et al. (2015) the authors show that training regression models based on only the winning price would be overfitting to the winning bids which are generally higher than the losing bids. Their solution is to build a censored regression model on the losing bid data and combining it with a linear regression model trained on the winning data. They show that the combined model works better than either one of the models separately. In Zhang et al. (2016) the authors are also able to tackle the issue with bias in learning using censored data. However, the issue with both of these methods where the model is trained on the historical data is that strategies based on the model cannot adapt to changes in the competing bidders bid strategies.

Low regret learning algorithms for figuring out the right amount to bid are presented in Heidari et al. (2016). They consider a setting where a seller (publisher) chooses the buyer (ad exchange) in advance based on the historical bidding data. The difference with this setting and the header bidding environment is that in header bidding environments the publisher receives all of the bids for an auction and selects the highest bid.

The authors in Heidari et al. (2016) compare their algorithms with the EXP3 algorithm. They do not expect the EXP3 algorithm to perform well in their setting as they expect the publisher to also play a no-regret algorithm. They do not experiment with the algorithms in a setting where the seller receives all of the bids and chooses the highest one. This setting would be interesting to us.

According to the authors in Wang et al. (2017), they propose the first deep reinforcement learning agent for optimizing the bidding. They use the agent on JD.com. At first JD.com

used a bidding approach based on eCPM where they predict the CTR. They noticed it did not work so they added the possibility for human experts to modify the bid coefficient. This was inefficient. At last they developed an agent based on a variant of Deep Q Network (DQN). They encode each auction request data (user data, product IDs and ads) into plain text. They one-hot encode the text and feed it to a deep convolutional neural network. The model works well without elaborate feature engineering.

The bidding agent works by defining each day as an episode. Net profit of auction is the reward. Actions are the bids where the bid is chosen from values [0, 0.01, 0.02,..., C] where C is the bid ceiling. The state of the agent is defined as the data they have about the user, product IDs and ads.

It seems like the latest research in bid optimization is focusing on reinforcement learning to optimize the bidding behavior. In Cai et al. (2018) the authors build a Markov Decision Process framework to figure out the best bidding strategy. As in Wang et al. (2017), they divide the flow of auctions into episodes. The bidding agent considers three main pieces of data: the amount of auctions remaining in the episode, the budget amount left and a vector of data containing information such as the city and day of the week. Based on the data the agent determines the amount to bid. If the bid wins, the agent can observe the market price and user response later. If the bid loses, the agent learns nothing. When an episode runs out, the auction number and budget are reset and a new episode starts. Their method also takes into account the pacing of the bidding so that the campaign runs smoothly over the whole campaign period. Pacing the bidding is an issue that needs to be taken into account in our research also. The bid size should reflect how important it is to win the bid, so that the whole campaign budget will be spent.

In Jin et al. (2018) the authors implement a multi-agent bidding solution where the agents cooperate to find the optimal bidding strategies that will benefit all of the bidders. Their solution is designed for the Taobao e-commerce platform where the merchants advertise and sell their products. One of the main goals of the method is to find an equilibrium where all of the advertisers benefit, instead of increasing the competition. This is different from our goal where we want to beat the other bidders in the header bidding auction.

The study most close to our research is presented in Jauvion et al. (2018). Their objective is to maximize a SSP's revenue in competing with other bidders in first-price header

bidding auctions. They do not necessarily want to win most of the auctions, but they want to minimize the amount they pay for the auctions that they win. They present the problem as a stochastic contextual bandit problem. The method uses a version of Thompson Sampling combined with particle filtering. The particle filtering enables the algorithm to adapt to the changing bid sizes and bidding strategies of the other participants in the auctions. Their method is designed to minimize the overspending in first-price auctions, which none of the other previous studies focused on. The algorithm performs fast enough so that it suits our need where we need to come up with the bid price in a few milliseconds. In our study we compare their bidding strategy with other bandit strategies designed for non-stationary environments, e.g. policies which forget the past observations at a certain rate.

3 RESEARCH METHODOLOGY

3.1 Introduction

This section describes the methods analyzed in this research. Three different types of multi-armed bandit algorithms are analyzed. The algorithms chosen are Epsilon-Greedy (E-Greedy) (Bubeck & Cesa-Bianchi, 2012), Upper Confidence Bound (UCB) (Bubeck & Cesa-Bianchi, 2012) (Auer et al., 2012) and Thompson Sampling (Bubeck & Cesa-Bianchi, 2012). To increase the performance of the algorithms in non-stationary environments, we use different kinds of variants of the algorithms. For UCB we test a Sliding Window (Moulines & Garivier, 2008) as well as an Exponentially Decaying (Cohen & Strauss, 2003) and an Exponentially Decaying Sliding Window variant of the algorithm, that we propose here. For Thompson Sampling, we experiment with a dynamic version of the algorithm named Dynamic Thompson Sampling (Granmo & Agrawala, 2011). These algorithms are described in more detail below.

3.2 Methods

Multi-armed bandit algorithms get their name from the problem where a gambler tries to maximize their winnings when playing slot machines or one-armed bandits. Each slot machine returns a reward that follows an unknown distribution specific to that machine.

The reward returns between the machines are different, but the player does not know how. The player tries to figure out the best way to play the machines to gain the most reward. To do this they must try out the different machines, but still exploit the machine they think provides the best return. The multi-armed bandit algorithms try to solve this problem of how much one should explore the different options versus how much should they exploit the one they think is the best option.

We face the dilemma of exploration-exploitation tradeoff all the time in our lives. For example, should we choose a restaurant we know is pretty good or should we try out a new one that could potentially be very good. We want to eat the best food possible, but by trying out a new restaurant we risk getting worse food than by choosing an option we know is pretty good already.

The problem of figuring out the right amount to bid in consecutive real-time bidding auctions can be thought of as an exploration-exploitation dilemma. We might have knowledge of an amount that wins the auction, but do we get the best reward (the lowest price) by bidding that amount. We can describe the bidding problem as a multi-armed problem by dividing the bid amounts into the arms and trying to find out the best way to pull the arms to maximize the total reward, or in this case minimize the total amount that is paid to win the auctions. This research aims to study, if multi-armed bandits can provide a good solution to the bidding problem.

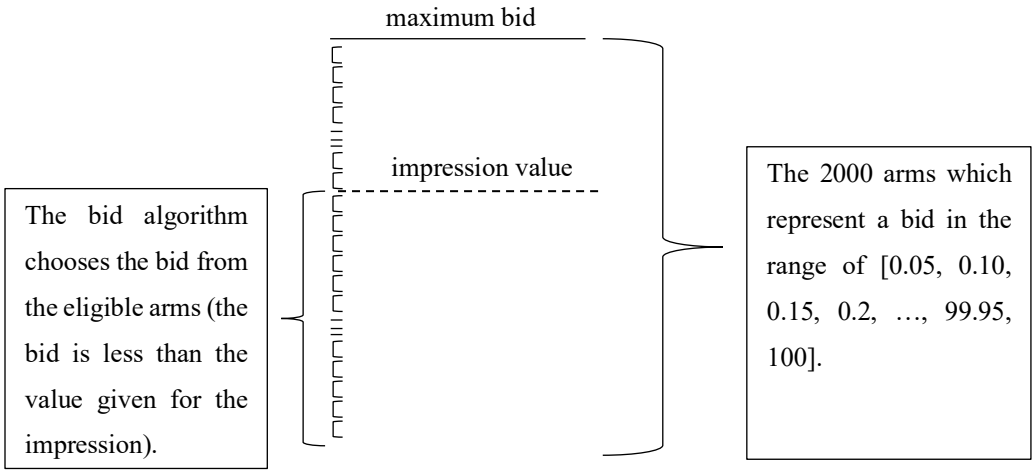

Figure 2. Description of how the multi-armed bandit algorithms are used in generating the bid amount.

Figure 2 describes how the multi-armed bandit algorithms are used in the bidding process. The number of arms may affect the performance of the algorithms. In this research we

choose the number of arms to be 2000. We set the maximum amount to ever bid to 100, which means that each arm represents a value in the range of [0.05, 0.10, 0.15, 0.2, …, 99.95, 100]. The bidding algorithms are limited, at most, to bidding the maximum value given for the impression, as bidding more would not give us any value.

3.2.1 The Epsilon-Greedy Algorithm

One of the simplest and easiest multi-armed bandit algorithms to understand is the epsilon-greedy strategy. In the E-Greedy algorithm, the strategy is to exploit the best (based on observed data) option most of the time and explore the other options randomly a part of the time. For instance, the best option could be chosen 90 % of the time and the other options are chosen randomly 10 % of the time. The parameter ε is used to determine percentages of exploration versus exploitation, so that the best option is exploited 1 - ε times and the other options are explored ε times.

The E-Greedy algorithm can be described by the following formula:

ε ($0 < \varepsilon < 1$) times choose randomly an arm

1 - ε times choose the best arm

The best arm is determined by the avg. reward gained by the arm so far. The reward for the arm on each round is normalized and is calculated by the following formula:

If the bid wins the auction, the reward is $(\max_{bid} - bid) / \max_{bid}$.

If the auction is lost, the reward is 0,

where $\max_{bid}$ is the maximum amount we will ever bid. In these experiments $\max_{bid} = 100$.

3.2.2 The Upper Confidence Bound Algorithm (UCB)

The Upper Confidence Bound algorithm (Sutton & Barto, 2017) works by the optimism in the face of uncertainty principle. This means that that the algorithm expects each arm to perform as well as the observed data dictates, even if there is not much data and the confidence is low. In the best case, the chosen arm is actually the best one and the algorithm acts optimally. In the worst case, the data will prove that another arm may in

fact be better. By using this strategy, the algorithm will eventually learn the actual rewards for each arm.

The UCB algorithm is implemented as follows:

1. First select an arm to play randomly

2. Then on each round select the arm that has the maximum performance calculated by the formula:

$$avg.reward + \sqrt{(c * \frac{\ln n}{n_A})},$$

where n is the number of rounds played, n_A (> 0) is the number of rounds played by the arm and c > 0 controls the degree of exploration. The avg. reward is calculated in the same way as for the E-Greedy algorithm.

The part added to the average reward is called the exploration term. This ensures that there is some exploration of arms that have not been played much. Looking at the formula, we can conclude that the reward for arms that have not been played increases. This is because on each round the dividend within the square root increases and the divisor does not increase for arms that do not get played. This ensures that the algorithm chooses arms, that have not been played much, every once in a while, even if their average reward has not been so good. We may have gotten very unlucky with some arms in the beginning, thus it is good that the arms will be tested again. This is also good in our case, because the reward for each arm will change in time.

One thing to note here is that our implementation of the UCB algorithm differs slightly from the UCB algorithm described in Sutton & Barto, (2017). The difference is that in the beginning, instead of playing each arm once, we choose an arm randomly. This is because we cannot choose among all of the arms as the arms to play are limited by the impression value.

3.2.3 The Sliding Window UCB Algorithm

The Sliding Window UCB algorithm (SW-UCB) (Moulines & Garivier, 2008) works just like the regular UCB algorithm described above except that it considers only the last N

rewards played by the algorithm. For each arm, the average reward is calculated by taking the rewards received by the arm within the N last rounds.

The SW-UCB works as follows:

1. First select an arm to play randomly

2. Then on each round select the arm that has the maximum performance calculated by the formula:

$$avg.reward\ in\ window + \sqrt{(c * \frac{\ln(\min(n,w))}{n_w}}),$$

where n is the number of rounds played, w is the window size, n_w (> 0) is the number of rounds played by the arm within the window and c > 0 controls the degree of exploration. Average reward is calculated by taking the average of the rewards within the window for the arm.

3.2.4 The Exponentially Decaying UCB Algorithm

As is the case with the SW-UCB algorithm, the Exponentially Decaying UCB (ED-UCB) algorithm differs from the UCB algorithm only by the way the average rewards for the arms are calculated. With ED-UCB, we only need to maintain the exponentially decaying weighted average in a single counter for each arm. This is done with the following formula (Cohen & Strauss 2003):

$$C \leftarrow (1 - d)x + dC$$

where 0 < d < 1 is the rate of decay and x is the reward received. Otherwise the ED-UCB works just the same way as the UCB algorithm:

1. First select an arm to play randomly

2. Then on each round select the arm that has the maximum performance calculated by the formula:

$$C + \sqrt{(c * \frac{\ln n}{n_t}}),$$

where C is the exponentially decaying weighted average, n is the number of rounds played, $n_t > 0$ is the number of rounds played by the arm and c > 0 controls the degree of exploration.

3.2.5 The Exponentially Decaying Sliding Window UCB Algorithm

The Exponentially Decaying Sliding Window UCB (EDSW-UCB) algorithm combines the methods used in the SW-UCB and ED-UCB algorithms. As far as we know, this algorithm has not been tested before. In EDSW-UCB the exponentially decaying weighted average is calculated over the rewards in the window. The EDSW-UCB algorithm works as follows:

1. First select an arm to play randomly

2. Then on each round select the arm that has the maximum performance calculated by the formula:

$$exp.decaying\ avg.reward\ in\ window + \sqrt{(c * \frac{\ln(\min(n,w))}{n_w}}),$$

where n is the number of rounds played, w is the window size, $n_w > 0$ is the number of rounds played by the arm within the window and c > 0 controls the degree of exploration. The exponentially decaying average in the window is calculated as follows:

Loop over the rewards in the window starting from the oldest value and calculate:

$$R_{n+1} * d + R_n * (1 - d),$$

Where R_n is the previous value after the loop (initially the oldest value), R_{n+1} is the next value in the window and d is the decay rate.

3.2.6 The Thompson Sampling Algorithm

Thompson sampling was first introduced in 1933 by William R. Thompson (Thompson 1933). The algorithm did not receive much attention until recently. Nowadays it is used in many applications ranging from online advertising to arcade games (Russo et al. 2017).

With the Thompson Sampling (TS) algorithm, each arm uses a beta distribution to model the arm's win rate. This means that we can use the algorithm to target a win rate that we want. This is very useful as most often we do not want to win all of the impressions, but only a certain percentage of them. In these experiments we target a 90 % win rate.

The TS algorithm works in the following way:

1. Each arm is set a uniform prior beta distribution, with alpha and beta equal to 1.

2. A random value is drawn from the distribution of each arm.

3. The arm which value is closest to the targeted win rate is selected.

4. The distributions are updated based on the received rewards. A win increments the arm's alpha parameter by one and a loss increments the arm's beta parameter by one.

The prior distributions could be defined using estimates on the arms' win rate, which would enable the algorithm to find the optimal arms faster. However, as the number of impressions is so vast and the algorithms will be running indefinitely, this would not really improve the performance of the algorithm.

3.2.7 The Dynamic Thompson Sampling Algorithm

With the regular Thompson Sampling algorithm defined above, the number of wins and losses will get larger with every round. This means that the arms' distributions will strongly converge toward the win rates of that moment and the amount of exploration will get smaller. However, the real-time bidding environment is non-stationary and will change over time. As the number of wins and/or losses for each arm is very large, the arms' distributions will adapt quite slowly to the changes in the environment. To combat this, we will use a dynamic version of the Thompson Sampling algorithm.

The Dynamic Thompson Sampling (D-TS) algorithm (Granmo & Agrawala 2011) works similarly to the plain TS algorithm. The exception is how the alpha and beta parameters are defined. The D-TS algorithm defines a threshold value C, which determines how fast the old alpha and beta value will decay. The alpha and beta values are updated as follows:

If the round number is less than C, then the alpha and beta values are updated as in the plain TS algorithm, where a win increments the alpha parameter by one and a loss increments the beta parameter by one.

If the round number is larger than C, then the parameters are updated by:

$$A_n = (A_{n-1} + reward) * \frac{C}{C+1}$$

$$B_n = (B_{n-1} + 1 - reward) * \frac{C}{C+1},$$

where the reward is 1 if the arm won, and 0 if it lost.

Using the above formula, we can ensure that the algorithm keeps exploring and will adapt to changes in the environment.

3.3 Data

The algorithms will be tested using data collected from the ReadPeak advertising platform. The data is gathered from the internal auctions happening within the ReadPeak platform as well as the data received from the external ad exchange. The platform collects data of every auction, bid response and impression notification going through the platform. The data is sent to a data lake from which the data can be fetched for analysis.

We first run an internal auction to determine the ad that will be sent to the external ad exchange. Each advertiser has set a bid they are willing to pay for a click. The bid and the CTR of the ad are used to calculate the actual bid amount used in the auction. In these auctions, all the participants and their bid amounts are known. However, the other participants in the external ad exchange's auctions are not known and the win price is known only, if our ad wins the impression. This should not matter as we still get a good representation of the winning prices for the exchange and the data will simulate the non-stationarity of the winning bid price. The external exchange's auctions are also second-price auctions. This means that the winning price is in fact the amount that needs to be bid at least to win the impression.

The algorithms will use the bid amount of the winner of the internal auction and the winning price of the exchanges auction. The winning price of the exchange's auction is

the minimum that needs to be bid to win the auction. Our algorithm should try to set the bid amount as close to this as possible, but not lower. The maximum amount the algorithm may bid, is the bid amount of the winner of the internal auction. The difference between the winning price and the bid amount, that the algorithm bids, can be used to calculate the regret for each algorithm.

We will run experiments on two separate datasets. The two datasets contain data from two different publishers located in the UK. For the first dataset the bidding is done through the Taboola (https://www.taboola.com/) platform and the auctions are second price auctions. The first dataset contains data from December 16th to December 18th, 2018. The dataset contains 708 265 auctions. Figure 3 shows the data for dataset 1. The win prices are shown in red and the maximum amount the algorithm will can bid is shown in blue.

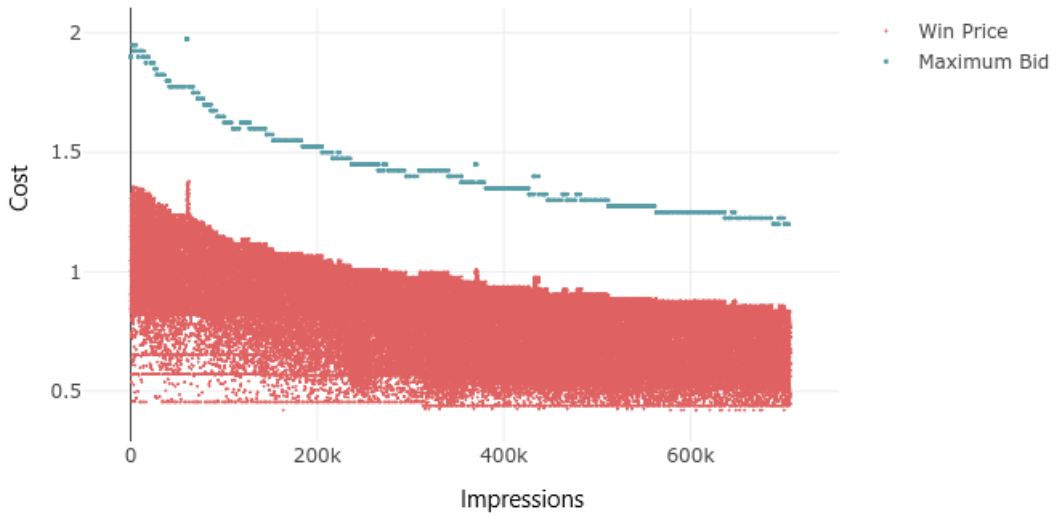

Figure 3 The winning prices and the maximum bids in dataset 1.

The second dataset contains data for bids done through the AppNexus (https://www.appnexus.com/) marketplace. The auctions are second price auctions and the dataset includes 348 875 auctions from January 20th to January 26th, 2019. Figure 4 describes the data of dataset 2. Like for dataset 1, the winning price is shown in red and the maximum bid is shown in blue.

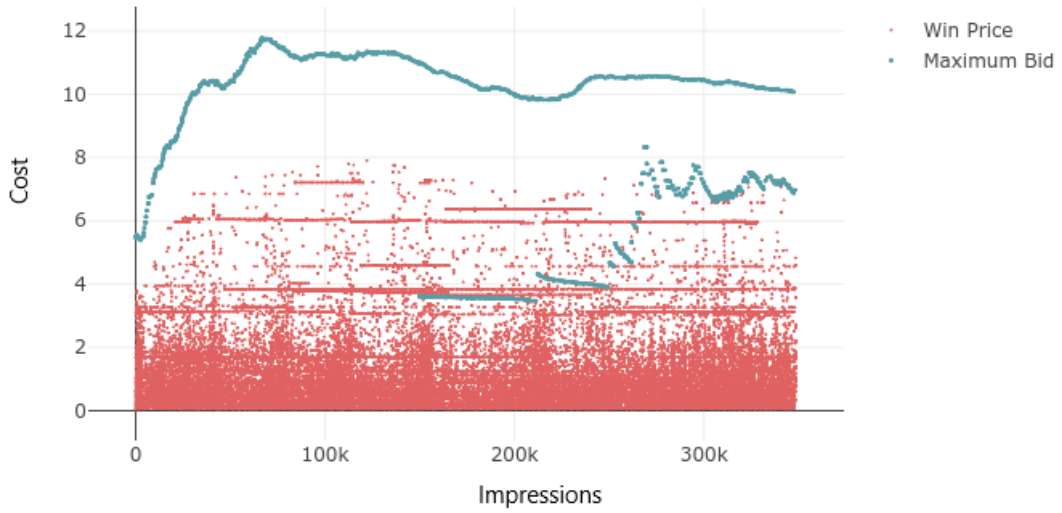

Figure 4. The winning prices and the maximum bids in dataset 2.

In dataset 1, the winning and maximum bid sizes are relatively small compared to dataset 2. In dataset 2, there are larger differences also with the maximum bid and the winning price than in dataset 1. If the maximum amount than can be bid is much larger than the win price, the algorithm has more room to experiment with different bid sizes.

3.3.1 Ethical Considerations

A major ethical issue in online advertising is the use of personal information to enhance the performance of the advertisements. The personal information is often used in the targeting of the advertisements. In this research the data is completely anonymous by nature. No user identifying data is included in the datasets. The impressions bought or bids performed cannot be tied to individual users or advertisers.

3.4 Research Design

This study will research the performance of the algorithms described in section 3.2 within the real-time bidding environment. We will test the algorithms individually using the auction data. For each algorithm we will calculate the cumulative average cost per

impression as well as the win percentage. The lower the cost per impression and the higher the win percentage, the better.

The auction environment will be simulated by using the collected auction data. In each round, we will check how the bid, chosen by the algorithm, performs against the winning price and keep track of the cumulative average cost per impression and the win percent. The bid chosen by the algorithm will be capped at the bid amount of the winner of the internal auction. The auctions will be run in order using the timestamps of the data.

We will also look at the run time of the algorithms. A rough requirement for us is that the algorithm should run in less than a millisecond. This is because we have about 200 milliseconds to provide the bid response to the bid request. The 200 millisecond time includes the network latency, the ad selection process from the internal auction as well as any other calculations.

Lastly, we will select the three best performing algorithms to be tested in the production environment. The algorithms will be tested by running them simultaneously alongside a no algorithm baseline. The no algorithm baseline always bids the eCPM amount of the ad that won the internal auction.

4 RESULTS

4.1 Simulated Auctions

We ran simulated auctions on the two datasets. First, for each algorithm, we tested different parameters to find out the best parameters to use. Then we ran the algorithms alongside each other to see how their performances compare to each other.

When analyzing the performance of the algorithms, we look at two values: the win rate and the average cost per impression. The higher the win rate and the lower the average cost per impression is, the better the algorithm. In some case it may be important to get impressions for as low cost as possible, while the win rate does not matter as much. Sometimes it may be important to get the impressions as fast as possible and the cost is not as important.

There is some randomness with all of the algorithms. With E-Greedy, some of the time an arm is picked randomly. With UCB, the first arm is picked randomly and with Thompson Sampling values are drawn from the beta distribution. This means that the algorithms may perform a bit differently on each round. The differences are quite small though, but the results should not be used as facts. Instead, the results should be used for pointing us in the right direction in choosing the right algorithms for further analysis in the production environment. It is also not unambiguous to determine which algorithm is the best. We look at the combination of the win rates and the average impression costs to determine the best algorithms, but which combination is the best, is not always clear.

4.1.1 The Epsilon-Greedy Algorithm

The epsilon greedy algorithm works by exploiting the best arm most of the time and exploring other arms the rest of the time. The epsilon parameter defines how much to explore versus exploit. Table 1 shows how the different values of the epsilon parameter affect the win rate on auctions ran on datasets 1 and 2.

Epsilon	Win Rate (dataset 1)	Win Rate (dataset 2)
0.001	99.9 %	98.9 %
0.01	99.5 %	98.0 %
0.05	97.3 %	97.3 %
0.1	94.5 %	96.3 %
0.2	89.0%	93.6 %

Table 1. The E-Greedy algorithm win rates for different values of epsilon on the two datasets.

Figure 5 shows the cumulative average cost per impression for the E-Greedy algorithms on dataset 1. The averages are quite close to each for the all of the algorithms.

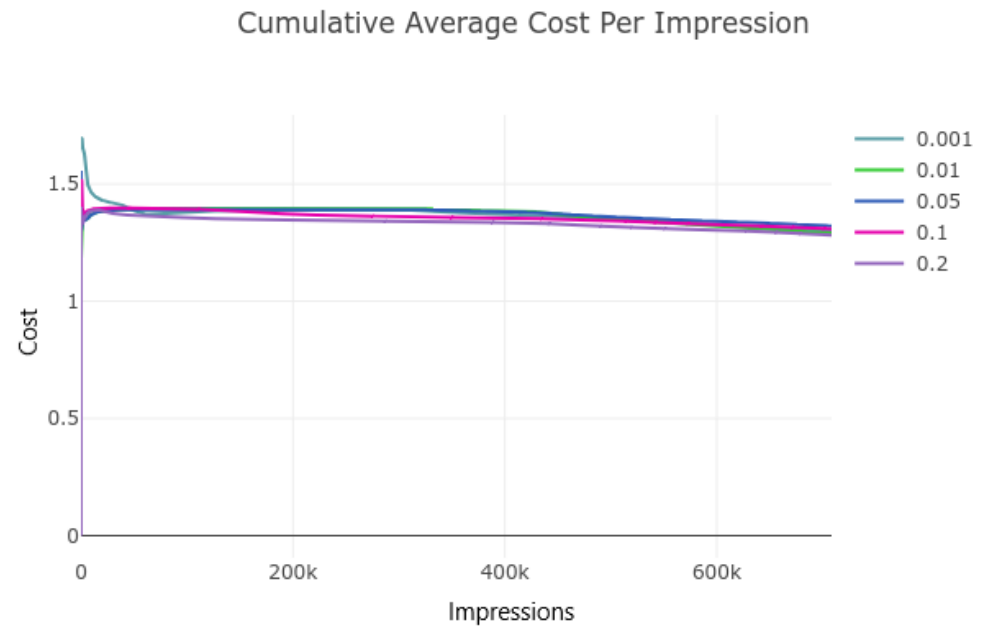

Figure 5. The cumulative average cost per impression for the E-Greedy algorithms with different values of epsilon on dataset 1.

Figure 6 shows the cumulative average cost per impression for the algorithms on dataset 2. The averages are a bit more spread out that for dataset 1.

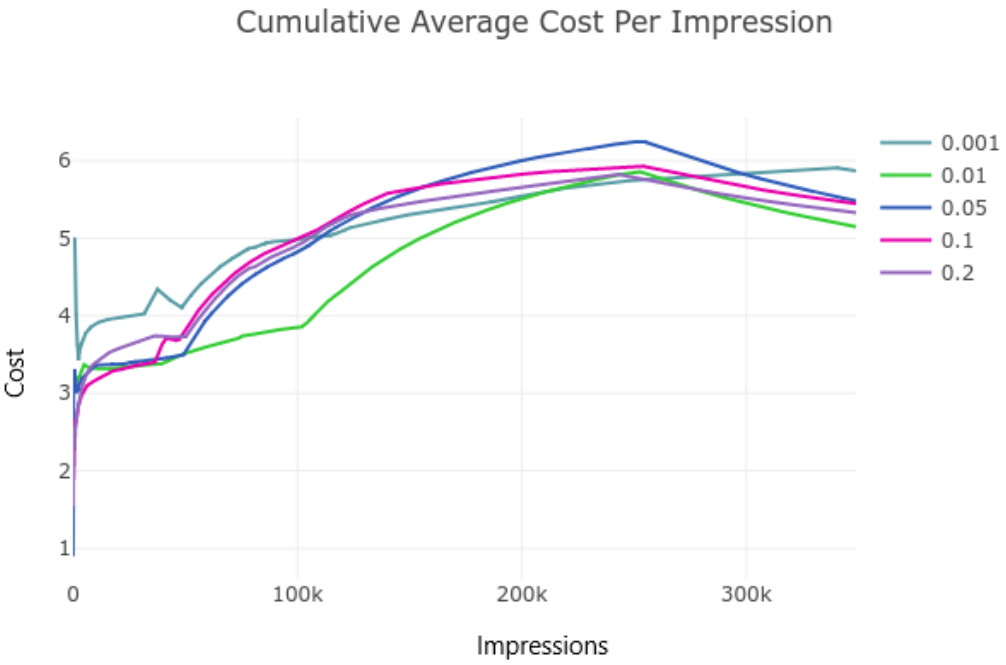

Figure 6. The cumulative average cost per impression for the E-Greedy algorithms with different values of epsilon on dataset 2.

For dataset 1, the average costs per impression are very close to each other. However, the win percentages differ some. With dataset 2, the difference in the impression cost is larger. The algorithm with the epsilon parameter 0.01 (explores 1% of the time) seems to perform the best when taking into account the winning percentages as well as the cumulative average cost per impression.

4.1.2 The Sliding Window UCB Algorithm

The sliding window UCB algorithm contains two parameters, the window size and the exploration multiplier. The exploration multiplier defines how large the exploration term is relative to the average reward. Testing showed that a value of 0.2 was good and that value is used in the experiments. Table 2 shows the win rates for the tested window sizes on the two datasets.

Window Size	Win Rate (dataset 1)	Win Rate (dataset 2)
400	99.6 %	79.6 %
500	96.9 %	82.1 %
600	96.6 %	82.4 %
700	94.9 %	84.5 %
800	95.1 %	86.3 %
900	95.6 %	87.7 %
1000	96.0 %	87.6 %

Table 2. The Sliding Window UCB algorithm win rates for different window sizes on datasets 1 and 2.

Figure 7 shows how the different window sizes perform relative to the costs per impression.

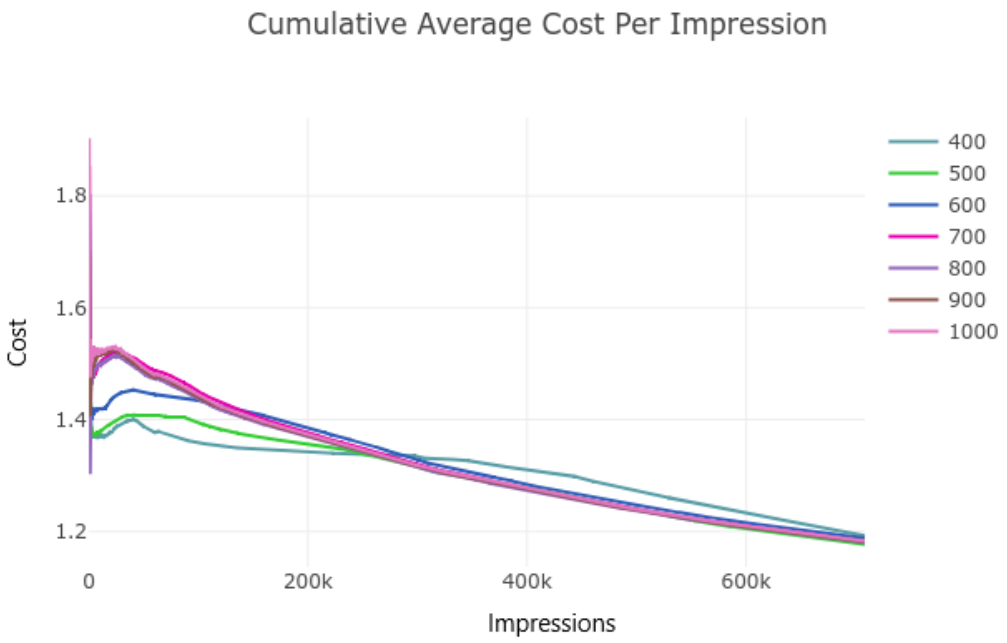

Figure 7. The cumulative average cost per impression for the SW-UCB algorithms with different window sizes on dataset 1.

Figure 8 shows the cumulative average costs per impression for the SW-UCB algorithms with different window sizes. Here the differences are larger than with dataset 1.

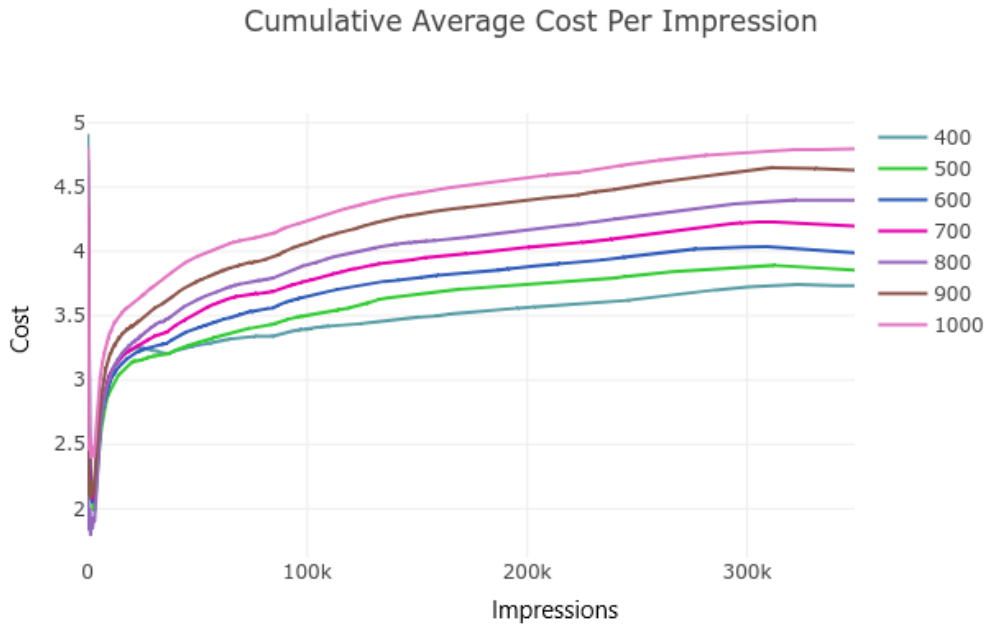

Figure 8. The cumulative average cost per impression for the SW-UCB algorithms with different window sizes on dataset 2.

On dataset 1 the differences with average costs per impression are quite small. Interestingly, the smaller window sizes look to have better win rates although the differences are small. On dataset 2 the spread with the average cost per impression is larger than for dataset 1. Window size 500 seems to generate a good mix of win percent and costs per impression.

4.1.3 The Exponentially Decaying UCB Algorithm

The Exponentially Decaying UCB algorithm contains a parameter which indicates how fast the old values decay. The smaller the value, the faster the old values are forgotten. Table 3 shows the win rates for the ED-UCB algorithm with different decay rates on datasets 1 and 2.

Decay Rate	Win Rate (dataset 1)	Win Rate (dataset 2)
0.9	99.9 %	99.7 %
0.99	99.7 %	94.2 %
0.999	99.9 %	93.3 %
0.9999	99.5 %	93.7 %
0.99999	97.7 %	85.0 %

Table 3. The Exponentially Decaying UCB algorithm win rates for different decay rates on datasets 1 and 2.

Figure 9 shows the cumulative average cost per impression for the different variations of the ED-UCB algorithm on dataset 1.

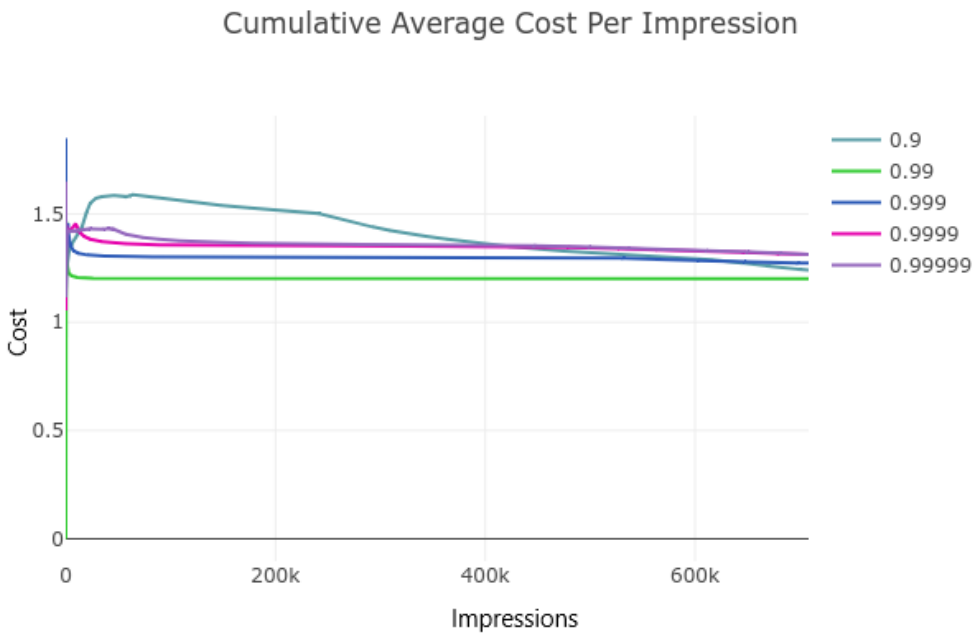

Figure 9. The cumulative average cost per impression for the ED-UCB algorithms with different decay rates on dataset 1.

Figure 10 contains the ED-UCB performance on dataset 2.

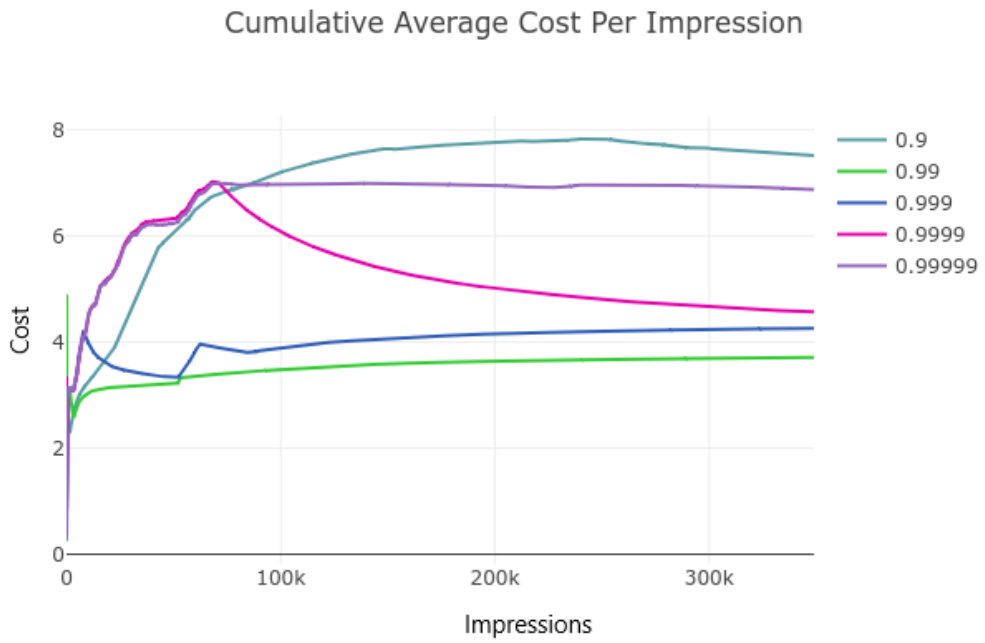

Figure 10. The cumulative average cost per impression for the ED-UCB algorithms with different decay rates on dataset 2.

On dataset 1 the differences with the average impression costs are quite small, although the algorithm with decay rate of 0.99 looks to stand out. On dataset 2 the differences in the impression costs are larger than on dataset 1. The algorithm using the decay rate of 0.99 looks to be a clear winner on both datasets.

4.1.4 The Exponentially Decaying Sliding Window UCB Algorithm

The Exponentially Decaying Sliding Window UCB (EDSW-UCB) algorithm contains two parameters which can be used to control how fast the algorithm adapts to changes in the environment. The parameters are the window size and the decay rate. Preliminary tests indicate that decay rate of 0.99 seem to perform the best. Table 4 shows how different window sizes affect the win rate on dataset 1 and 2 with the EDSW-UCB algorithm using decay rate of 0.99.

Window Size	Win Rate (dataset 1)	Win Rate (dataset 2)
2500	98.1 %	86.7 %
5000	99.0 %	90.0 %
8000	99.0 %	89.0 %
10000	99.1 %	88.0 %
12000	99.2 %	88.5 %
15000	99.2 %	85.7 %

Table 4. The Exponentially Decaying Sliding Window UCB algorithm win rates for different window sizes on datasets 1 and 2 using the decay rate of 0.99.

Figure 11 contains the EDSW-UCB algorithms average impression costs on dataset 1.

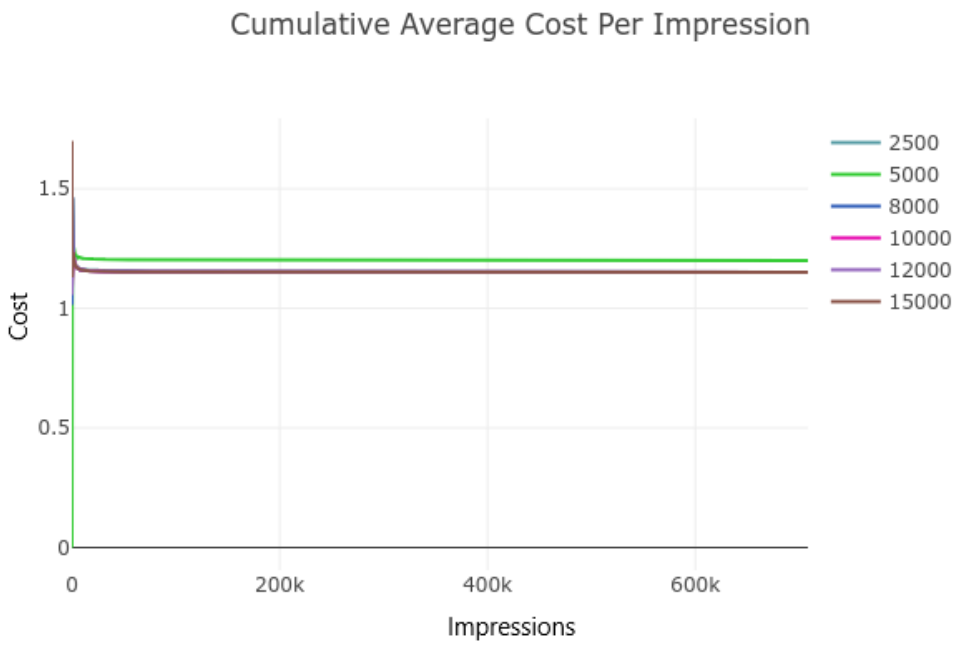

Figure 11. The cumulative average cost per impression for the EDSW-UCB algorithms with decay rate of 0.99 and different window sizes on dataset 1.

Figure 12 contains average impression cost the data for EDSW-UCB on dataset 2.

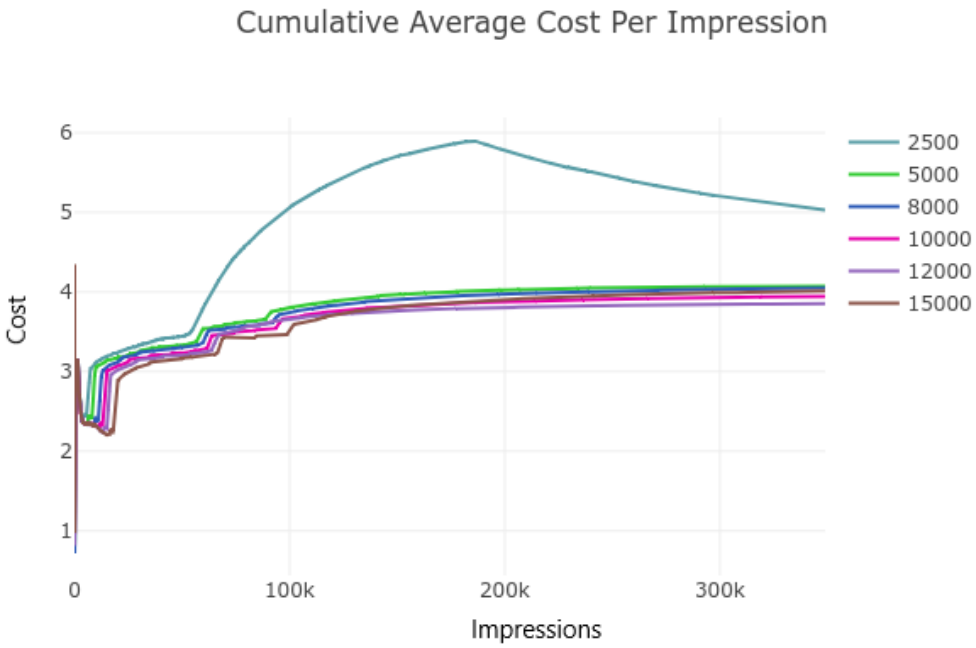

Figure 12. The cumulative average cost per impression for the EDSW-UCB algorithms with decay rate of 0.99 and different window sizes on dataset 2.

The differences between the algorithms are very small on both datasets. Only the smallest window size of 2500 looks to perform clearly the worst on dataset 2. The window size of 12000 seems to perform the best.

4.1.5 The Thompson Sampling Algorithm

The Thompson Sampling algorithm differs from the previous analyzed algorithms in the sense that with Thompson Sampling it is possible to target a certain win rate. Each arm uses a beta distribution to model a specific win rate. This means that it is possible to dynamically choose the win rate for each bid. For example, a campaign may not be in any hurry to accumulate impressions, thus we can choose to target a lower win rate and get the impressions at a lower price than with a higher win rate.

Table 5 shows how modifying the targeted win rate parameter affects the actual win rate on datasets 1 and 2. The algorithm is able to follow the targeted win rate quite well. With dataset 2 the actual win rate seems to follow the target win rate even better than for dataset 1.

Target Win Rate	Win Rate (dataset 1)	Win Rate (dataset 2)
10 %	14.8 %	10.7 %
20 %	25.2 %	25.0 %
30 %	41.4 %	29.6 %
40 %	46.3 %	40.2 %
50 %	54.6 %	46.5 %
60 %	68.9 %	57.8 %
70 %	87.6 %	77.0 %
80 %	93.2 %	88.8 %
90 %	96.5 %	94.0 %
99 %	99.9 %	99.1 %

Table 5. Thompson Sampling target win rate vs actual win rate on datasets 1 and 2.

Figure 13 shows how targeting different win rates affects the average cost per impression. As the win rate gets lower, also the cost per impression gets lower.

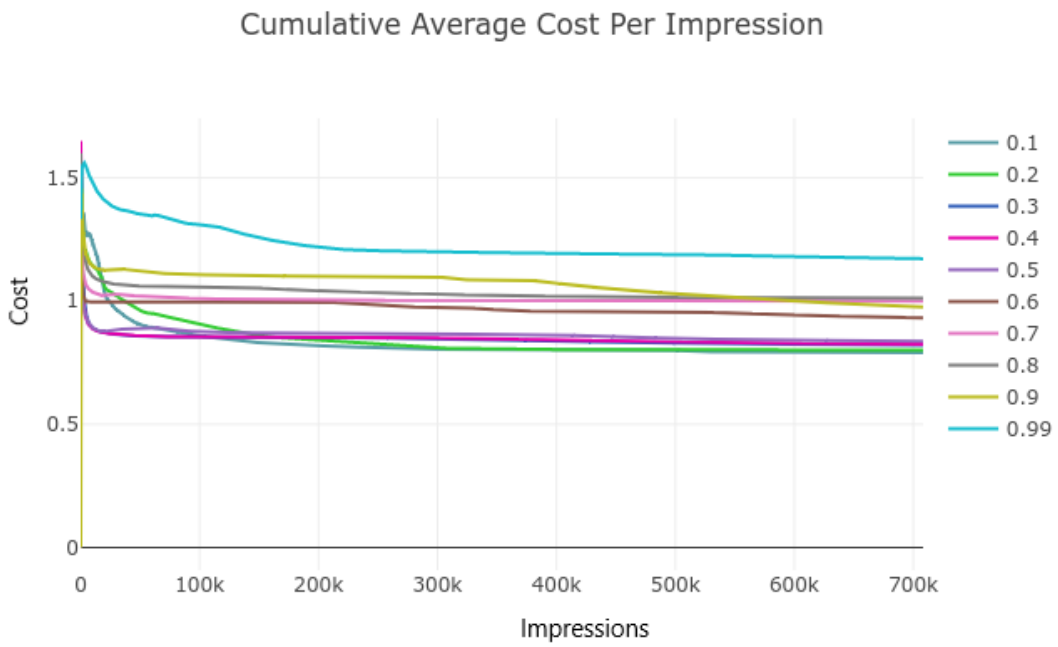

Figure 13. Thompson Sampling cumulative average cost per impression for the different target win rates on dataset 1.

Figure 14 shows the cumulative average cost per impression for different target win rates on dataset 2. Here it is even more clear that the lower the win rate, the cheaper it is to win the impressions.

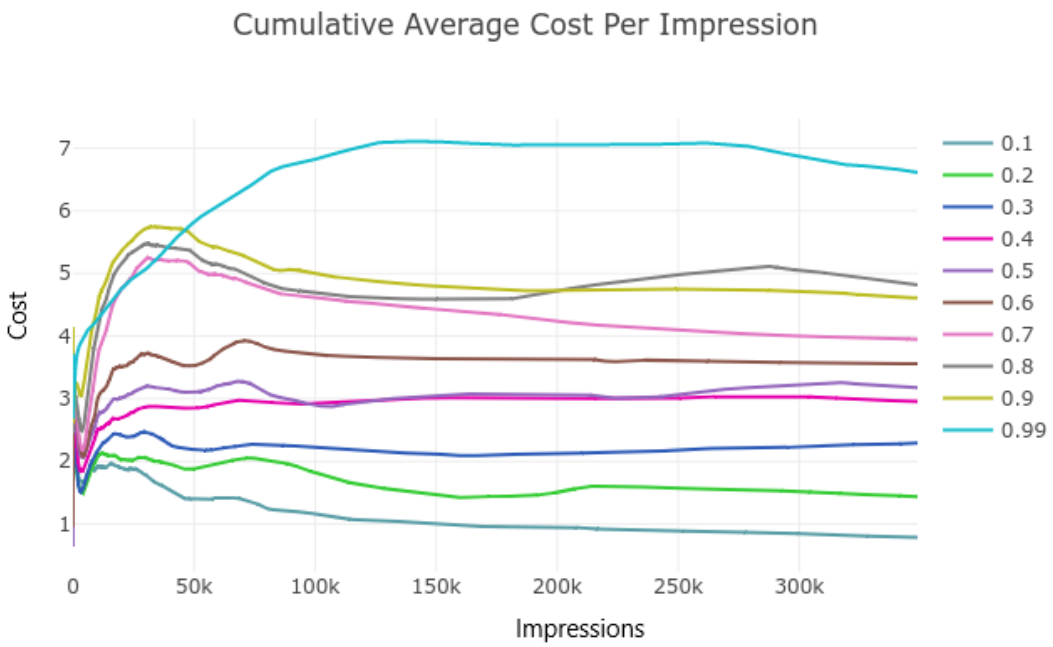

Figure 14. Thompson Sampling cumulative average cost per impression for the different target win rates on dataset 2.

Figure 13 and Figure 14 show how it may make sense to lower the win rate to get impressions for a cheaper price than with a high win rate. This will make sense, if the campaign is not in any hurry to accumulate impressions.

The Thompson Sampling algorithm will explore less and less as each arm's distribution gets more data. For this reason and due to the non-stationary nature of the bidding environment, the dynamic version of the Thompson Sampling is used in the experiments.

4.1.6 The Dynamic Thompson Sampling Algorithm

Most of the previous algorithms' win rates are close to 90 %. We choose 90 % as the target win rate for the Dynamic Thompson Sampling (D-TS) algorithms also. In practice the target win rate can be dynamically changed depending on the state in the bidding environment.

The Dynamic Thompson Sampling algorithm uses a threshold parameter to state how fast the alpha and beta values decay. Table 6 shows how the threshold parameter affects the win rate on the two datasets.

Threshold	Win Rate (dataset 1)	Win Rate (dataset 2)
100	97.7 %	94.8 %
500	96.9 %	94.2 %
1000	97.0 %	94.2 %
5000	96.6 %	94.0 %
7500	98.0 %	93.7 %
10000	98.2 %	94.0 %
15000	97.3 %	94.4 %

Table 6. The Dynamic Thompson Sampling algorithm win rates for different threshold values on dataset 1 using the target win rate of 0.9.

Figure 15 shows the D-TS algorithms' average impression costs on dataset 1.

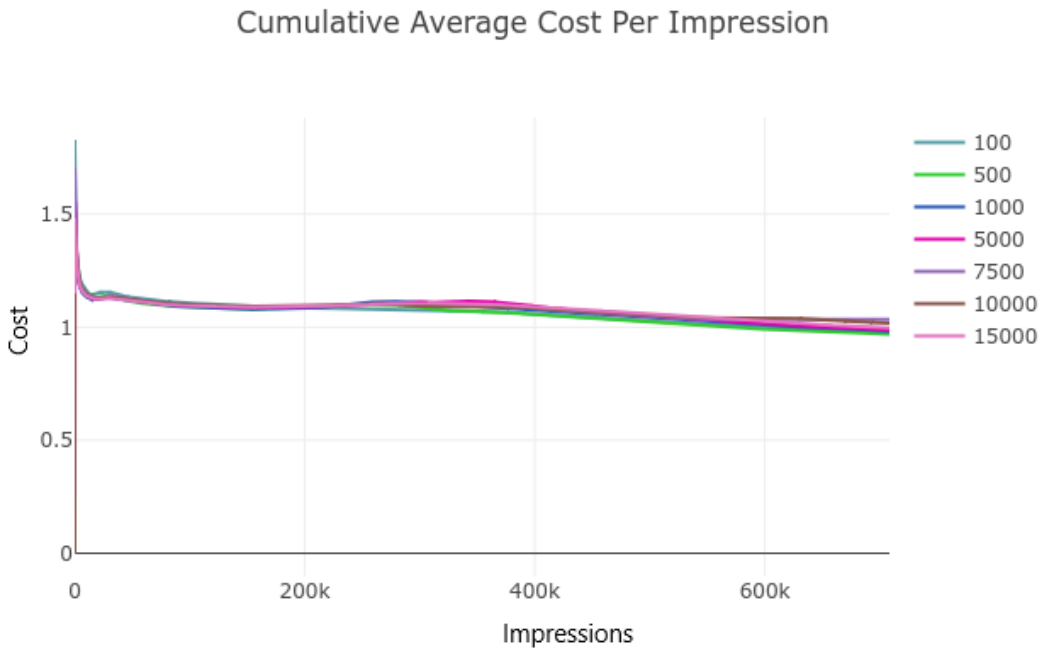

Figure 15. The cumulative average cost per impression for the D-TS algorithms with target win rate of 0.9 and different threshold values on dataset 1.

The D-TS algorithms' average impression costs on dataset 2 can be seen in Figure 16.

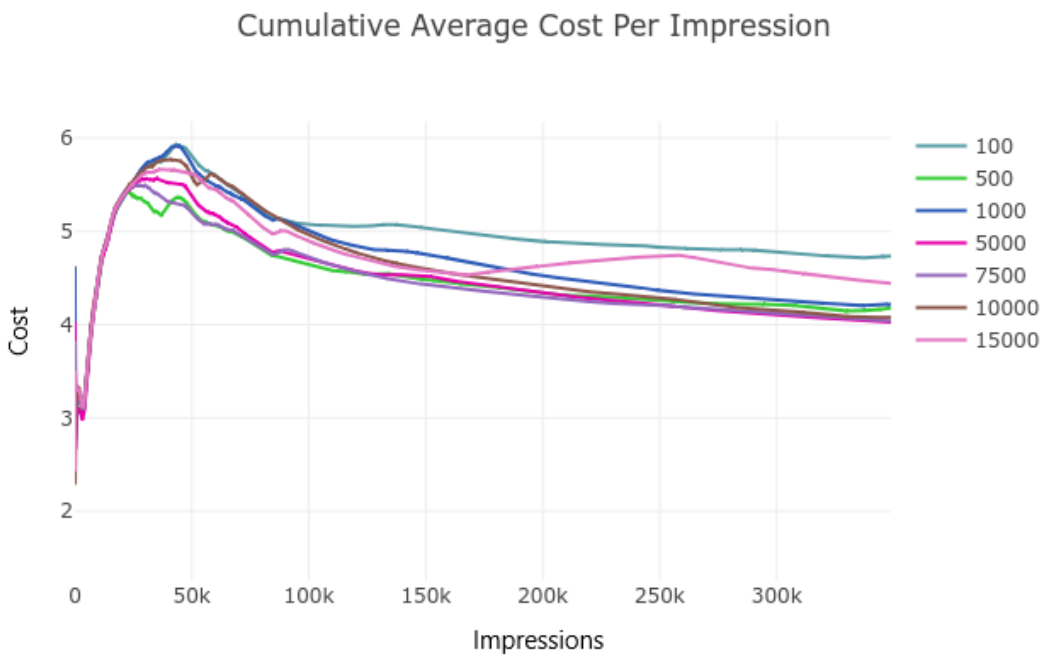

Figure 16. The cumulative average cost per impression for the D-TS algorithms with target win rate of 0.9 and different threshold values on dataset 2.

The differences between the algorithms with different threshold are quite small on dataset 1. On dataset 2 the differences can be seen more clearly, although the best performing algorithms are all very close to each other. Overall, the best performing algorithm looks to be the one with a threshold value of 5000. However, closer analysis should be done in the production environment with the different threshold values as well as different target win rates.

4.1.7 Comparison

To see how the algorithms perform compared to each other, we ran the best versions of each algorithm at the same time. Table 7 shows the win rates for each algorithm on datasets 1 and 2.

Algorithm	Win Rate (dataset 1)	Win Rate (dataset 2)
E-Greedy (e=0.01)	99.5 %	97.9 %
SW-UCB (ws=500)	97.2 %	81.6 %
ED-UCB (dr=0.99)	99.7 %	94.9 %
EDSW-UCB (ws=12000, dr=0.99)	99.4 %	87.2 %
D-TS (targ. win rate=0.9, th=5000)	96.4 %	94.5 %

Table 7. The win rates for the best versions of each algorithm on the two datasets.

Figure 17 shows how the algorithms perform compared to each other on dataset 2.

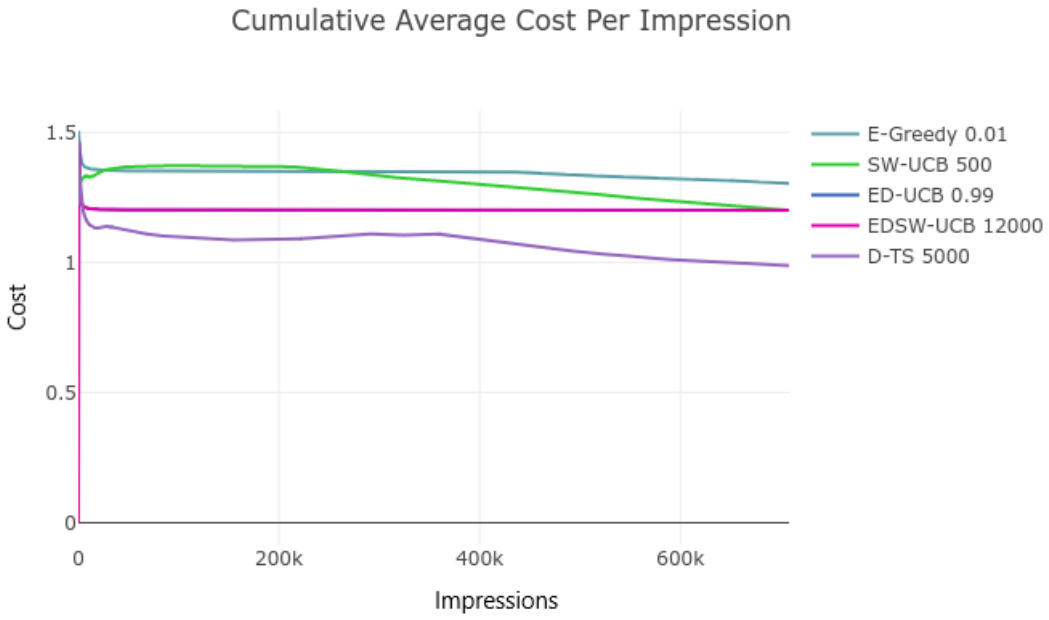

Figure 17. The cumulative average cost per impression for the best versions of each algorithm on dataset 1. The EDSW-UCB algorithm overlaps quite closely the ED-UCB algorithm, which makes it hard to see the ED-UCB algorithm line.

The cumulative average impression costs for the algorithms on dataset 2 are described in Figure 18.

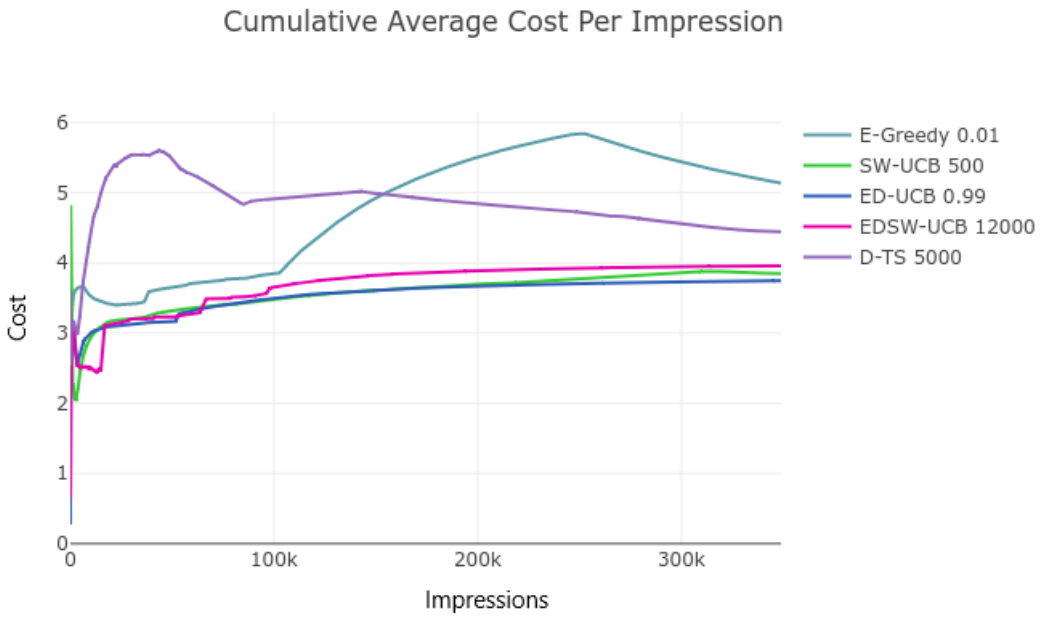

Figure 18. The cumulative average cost per impression for the best versions of each algorithm on dataset 2.

The D-TS performs the best on dataset 1 and the ED-UCB performs the best on dataset 2. On dataset 1, regarding the average cost per impression, the D-TS is a clear winner. However, the win rate is a bit lower than for the other algorithms. The ED-UCB performs the second best on dataset 1. On dataset 2 it can be seen how the D-TS algorithm explores unplayed arms much more than the other algorithms. This explains why the D-TS bids higher than the other algorithms in the beginning on dataset 2, where the maximum bid is much higher than the winning price. The E-Greedy, ED-UCB and D-TS algorithms have much better win rates than the SW-UCB and EDWS-UCB algorithms on dataset 2.

The algorithms selected to be tested in production are the E-Greedy (e=0.01), ED-UCB (decay rate=0.99) and the D-TS (target win rate=0.9, threshold=5000). We pick the E-Greedy algorithm, because it consistently produces high win rates. It is also the simplest algorithm which makes it interesting to see how it performs in the production environment. The ED-UCB algorithm performs well overall. It is also simple and efficient to implement. The D-TS algorithm looks to produce good results regarding the win rate and the average cost per impression. With the D-TS algorithm, it is easy to dynamically change the target win rate depending on the situation. This is very useful for us.

4.2 Run Times

The run time of each algorithm was recorded within the simulated auctions. The algorithms were run in a NodeJS environment on an Intel Core i5-6600K CPU @ 3.50 GHz with 16 GB of RAM. The run times are listed in Table 8. The run time is the average of all the rounds and the different versions of the algorithms. For some of the algorithms the run time differs depending on the parameters. For instance, for the SW-UCB and EDSW-UCB algorithms, the run times went up when the window size got bigger. This makes sense as it is required to do more calculations the larger the window size is.

Algorithm	Average Run Time (milliseconds)
E-Greedy	0.03
SW-UCB	0.05
ED-UCB	0.03
EDSW-UCB	0.23
TS	0.05
D-TS	0.05

Table 8. Algorithm run times.

The EDSW-UCB algorithm is clearly the slowest of the algorithms. All of the algorithms run fast enough for our use case as the limit is one millisecond. If the difference between the performance of the algorithms is close, it may make sense to select a faster algorithm as the computation needed is larger the slower the algorithm and this costs money.

4.3 Production Results

We have tested the three chosen algorithms D-TS (target win rate=0.9, threshold=5000), ED-UCB (decay rate=0.99) and E-Greedy (e=0.01) against a no algorithm baseline in production. The no algorithm baseline always bids the eCPM amount of the ad. Within one publisher there may be multiple ad placements. We run the algorithms separately for each placement.

We ran experiments on the algorithms on an ad placement on a Norwegian publisher's site for the time period of 9.4.2019-23.4.2019. We ran 4434483 bids which resulted in 540093 impressions. The placement uses first price auctions and the bidding is done through the AppNexus platform.

Table 9 shows the win rates for the algorithms running in the placement. It should be noted that the eCPM bid for the ads is quite low on the placement, which means that the win rates will be low compared to the offline results. Further research is required on analyzing how changing the target win rate on the D-TS algorithm will affect the win rate in this case.

Algorithm	Win %
Baseline	12.929
D-TS	11.974
E-Greedy	11.953
ED-UCB	11.859

Table 9. The win rates for the algorithms for bids done through the ReadPeak platform for a placement on a Norwegian publisher.

Figure 19 shows the average bids per day for the algorithms bidding in the ad placement. There is a clear rise in the baseline bid amount on the 16th. This may happen due to a new campaign starting or an old campaign stopping from running in the placement. Also, a running campaign may value the impressions differently as the campaign progresses.

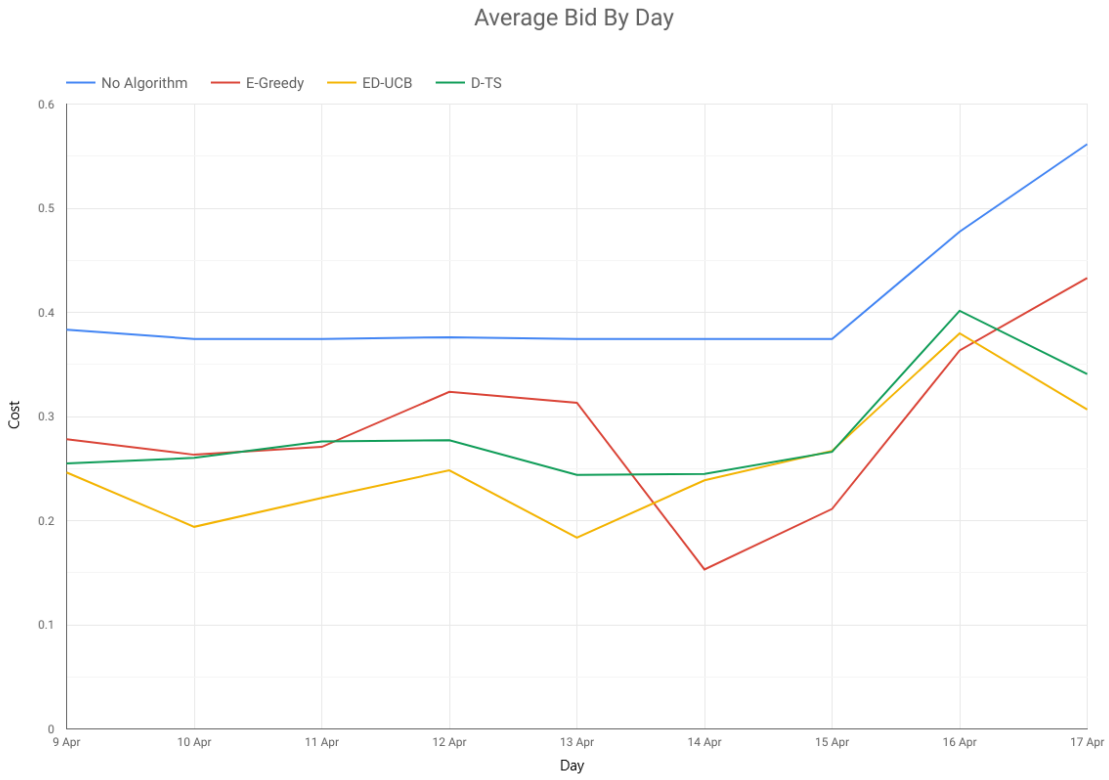

Figure 19. The average bids by day for the algorithms done through the ReadPeak platform for a placement on a Norwegian publisher.

Figure 20 shows the average impression cost by day for the placement.

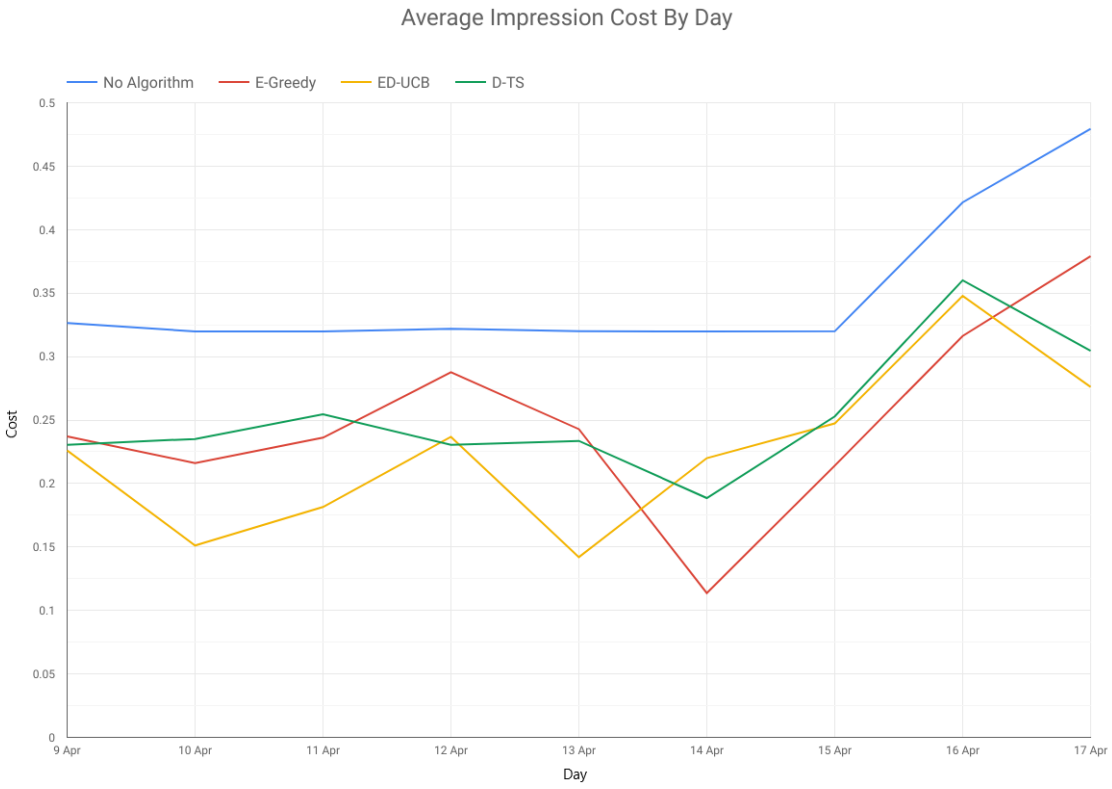

Figure 20. The average impression cost by day for the algorithms done through the ReadPeak platform for a placement on a Norwegian publisher.

Table 10 shows the total average impression costs for the algorithms running in the placement.

Algorithm	Avg. Impression Cost
Baseline	0.370
D-TS	0.273
E-Greedy	0.262
ED-UCB	0.252

Table 10. The total average impression costs for the algorithms for bids done through the ReadPeak platform for a placement on a Norwegian publisher.

The win percentage for the algorithms decreased 7.4 % to 8.3 % from the baseline. However, the average impression cost decreased as much as 25.9 % to 31.9 %. This indicates that the benefit of using the bidding algorithms in first-price auctions is substantial.

It is clear from figures Figure 19 and Figure 20, that the bid algorithms do a good job of lowering the bidding costs. The performance of the algorithms seems to fluctuate on

different days compared to each other, thus it is hard to come to a definite conclusion on the best algorithm.

5 CONCLUSION

5.1 Summary

In this research we discussed the programmatic ad buying and described how the real-time bidding ecosystem works. We examined how multi-armed bandit algorithms perform in optimizing the bid amounts in real-time bidding environments. Seven multi-armed bandit algorithms were introduced and their performance was analyzed on offline data. Three of the best performing algorithms (E-Greedy, ED-UCB and D-TS) were selected for closer analysis in a real world environment. We ran the three algorithms in production and discovered that the algorithms reduced the bidding costs considerably compared to the baseline. Which algorithm performs the best is inconclusive.

5.2 Discussion

All of the three algorithms tested in production suffered from a cold start problem. The cold start problem is caused by the fact that there is no data about the environment when the algorithms are initialized. This means that every time the algorithms are initialized, each arm is on the same level and the algorithms will need to learn again which arms are best for the specific environment. New instances of the algorithms are spawned several times a day, thus this has surely had an effect on the production results.

It is hard to say which algorithm has suffered the most from the cold start problem. Likely, this has not affected the E-Greedy nor the ED-UCB algorithms as much as the D-TS algorithm, as they do not need as much data to perform optimally. The D-TS algorithm needs at least the threshold amount of hits for each arm before it actually starts decaying the values. Since a large amount of the bids have occured by instances of the algorithm that have not reached this limit yet, the results do not necessarily represent the actual performance of the D-TS algorithm. Also, the D-TS algorithm explores the unplayed arms quite aggressively, so the cold start issue has likely had a negative effect on the

performance of the algorithm. To counter this issue, new instances of the algorithms should utilize the data gathered by the old instances.

5.3 Future Work

In the future, more research is needed in determining which algorithm is the best overall. We will run experiments with the Dynamic Thompson Sampling algorithm using different values of the decaying threshold as well as different values of the target win rate. We will also use the data already gathered by the algorithms when initializing the new instances of the algorithms to avoid the cold start problem. Also, determining the optimal number of arms to use will need more investigation. The five cent slot per arm may not be optimal as the bid can be set on a one cent accuracy. We also plan on dynamically changing the target win rate depending on the value we give for the impression. In addition, it is worth it to opt out of bidding on requests we determine that we will not have a chance of winning, with the value we give for the impression. This will save resources and money.

REFERENCES

Amin, K., Kearns, M., Key, P., & Schwaighofer, A. (2012). Budget Optimization for Sponsored Search: Censored Learning in MDPs. *Uai*, 54–63. https://doi.org/10.1021/acs.chemmater.5b00119

Amin, K., Rostamizadeh, A., & Syed, U. (2013). Learning Prices for Repeated Auctions with Strategic Buyers. https://doi.org/10.1111/j.1532-950X.2013.01099.x

Auer, P., Cesa-Bianchi, N., & Fischer, P. (2012). Finite-time Analysis of the Multiarmed Bandit Problem. *Machine Learning*, *47*(2), 235–256. https://doi.org/10.1023/A:1013689704352

Auer, P., Cesa-Bianchi, N., Freund, Y., & Schapire, R. E. (2002). The Nonstochastic Multiarmed Bandit Problem. *SIAM Journal on Computing*, *32*(1), 48–77. https://doi.org/10.1137/S0097539701398375

Bellman, R. (1957). A Markovian decision process. Journal of Mathematics and Mechanics, 679-684.

Bubeck, S., & Cesa-Bianchi, N. (2012). Regret Analysis of Stochastic and Nonstochastic Multi-armed Bandit Problems. *CoRR*, *5*(1), 1–122. https://doi.org/10.1561/2200000024

Cai, H., Ren, K., Zhang, W., Malialis, K., Wang, J., Yu, Y., & Guo, D. (2018). Real-Time Bidding by Reinforcement Learning in Display Advertising. https://doi.org/10.1145/3269206.3272021

Cohen, E., & Strauss, M. (2003). Maintaining Time-Decaying Stream Aggregates. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.92.5614&rep=rep1&type=pdf

Cui, Y., Zhang, R., Li, W., & Mao, J. (2011). Bid Landscape Forecasting in Online Ad Exchange Marketplace. *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '11*, 265. https://doi.org/10.1145/2020408.2020454

Edelman, B., Ostrovsky, M., & Schwarz, M. (2007). Internet Advertising and the Generalized Second-Price Auction : Selling Billions of Dollars Worth of Keywords. *The American Economic Review*. https://www.benedelman.org/publications/gsp-060801.pdf

Ghosh, A., Rubinstein, B. I. P., Vassilvitskii, S., & Zinkevich, M. (2009). Adaptive bidding for display advertising. *Proceedings of the 18th International Conference on World Wide Web - WWW '09*, 251. https://doi.org/10.1145/1526709.1526744

Gonzálvez, J. C., & Mochón, F. (2016). Operating an Advertising Programmatic Buying Platform : A Case Study. https://doi.org/10.9781/ijimai.2016.361

Google. (2011). The Arrival of Real-Time Bidding and What it Means for Media Buyers. https://www.thatwhitepaperguy.com/downloads/Google-White-Paper-The-Arrival-of-Real-Time-Bidding-July-2011.pdf

Granmo, O., & Agrawala, A. (2011). Thompson Sampling for Dynamic Multi-Armed Bandits. https://doi.org/10.1109/ICMLA.2011.144

Heidari, H., Mahdian, M., Syed, U., Vassilvitskii, S., & Yazdanbod, S. (2016). Pricing a Low-regret Seller. *Proceedings of The 33rd International Conference on Machine Learning*, *48*, 2559–2567. http://proceedings.mlr.press/v48/heidari16.html

IAB Real Time Bidding (RTB) Project. OpenRTB API Specification Version 2.5. https://www.iab.com/wp-content/uploads/2016/03/OpenRTB-API-Specification-Version-2-5-FINAL.pdf

Jauvion, G., Grislain, N., Dkengne, P. S., Garivier, A., & Gerchinovitz, S. (2018). Optimization of a SSP's Header Bidding Strategy using Thompson Sampling. https://doi.org/10.1145/3219819.3219917

Jin, J., Song, C., Li, H., Gai, K., Wang, J., & Zhang, W. (2018). Real-Time Bidding with Multi-Agent Reinforcement Learning in Display Advertising. https://doi.org/10.1145/3269206.3272021

Kaplan, E. L., & Meier, P. (1958). Nonparametric estimation from incomplete observations. Journal of the American statistical association, 53(282), 457-481.

Levin, J. (2004). Auction Theory. https://web.stanford.edu/~jdlevin/Econ%20286/Auctions.pdf

Moulines, E., & Garivier, A. On Upper-Confidence Bound Policies for Non-Stationary Bandit Problems, (2008). https://arxiv.org/pdf/0805.3415.pdf

Muthukrishnan, S. (2009). Ad Exchanges : Research Issues. *Internet and Network Economics, Lecture Notes in Computer Science*, 1–12.

Perlich, C., Dalessandro, B., Hook, R., Stitelman, O., Raeder, T., & Provost, F. (2012). Bid Optimizing and Inventory Scoring in Targeted Online Advertising. *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '12*, 804. https://doi.org/10.1145/2339530.2339655

Russo, D., Van Roy, B., Kazerouni, A., Osband, I., & Wen, Z. (2017). A Tutorial on Thompson Sampling. https://doi.org/10.1561/2200000070

Sluis, S. (2017). Everything You Need To Know About Bid Shading (2019). https://adexchanger.com/online-advertising/everything-you-need-to-know-about-bid-shading/ Accessed 18.4.2019.

Sutton, R., & Barto, A. (2017). *Reinforcement Learning: An Introduction*. https://doi.org/10.1016/S1364-6613(99)01331-5

Thompson, W. R. (1933). On the Likelihood That One Unknown Probability Exceeds Another In View of the Evidence of Two Samples. *Biometrika*, *25,* (3/4), 285–294. https://doi.org/10.1080/14789940701474889

Wang, Y., Liu, J., Liu, Y., Hao, J., He, Y., Hu, J., … Li, M. (2017). LADDER: A Human-Level Bidding Agent for Large-Scale Real-Time Online Auctions. Retrieved from http://arxiv.org/abs/1708.05565

Weed, J., Perchet, V., & Rigollet, P. (2015). Online learning in repeated auctions. https://doi.org/10.1016/j.sbspro.2011.10.005

Wu, W. C.-H., Yeh, M.-Y., & Chen, M.-S. (2015). Predicting Winning Price in Real Time Bidding with Censored Data. *Proceedings of the 21th ACM SIGKDD International*

Conference on Knowledge Discovery and Data Mining - KDD '15, 1305–1314. https://doi.org/10.1145/2783258.2783276

Zhang, W., Yuan, S., & Wang, J. (2014). Optimal real-time bidding for display advertising. *Proceedings of the 20th ACM SIGKDD*, 1077–1086. http://dl.acm.org/citation.cfm?id=2623633

Zhang, W., Yuan, S., Wang, J., & Shen, X. (2014). Real-Time Bidding Benchmarking with iPinYou Dataset. http://arxiv.org/abs/1407.7073

Zhang, W., Zhou, T., Wang, J., & Xu, J. (2016). Bid-aware Gradient Descent for Unbiased Learning with Censored Data in Display Advertising. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining - KDD '16*, 665–674. https://doi.org/10.1145/2939672.2939713