Ad Exchange Optimization Algorithms on Advertising Networks

Luis Miralles Pechuán[1,2], Claudia Sánchez Gómez[1], and Lourdes Martínez Villaseñor[1]

[1] Facultad de Ingeniería, Universidad Panamericana, DF, Mexico

[2] Departamento de Ingeniería y Tecnología de Computadores, University of Murcia, Spain

{lmiralles,cnsanchez,lmartinez}@up.edu.mx

Abstract. Online advertising has seen great growth over the past few years. Advertisers have gotten better results with campaigns targeted at more specific audiences. Ad networks with few visits are unable to create such campaigns and hence are moving forward towards a new model, consisting of a huge global ad exchange market. In this market millions of advertisers compete for the ad space so that their ad will be shown to users upon visiting a page. In selecting the best candidate from all possibilities algorithms able to process advertiser's requirements in tenths of seconds are needed. To face this problem we have developed algorithms using techniques such as threads, AVL trees with hash, multiple node trees or Hadoop technology. Throughout this article we will show the results gained from each algorithm, a comparative performance analysis and some conclusions. We have also proposed some future lines of work.

Keywords: Ad exchange, online advertising algorithms, parallelism, AVL trees, multi-node trees, Hadoop, fuzzy logic.

1 Introduction

Online advertising offers advertisers great advantages when it comes to orienting campaigns to a particularly specified audience or making real-time edits. This explains why more advertisers are choosing to pay for publicity online [1]. Ad networks allow advertisers to post their ads on editor's pages. Editors are the ones with at least one web page and rent space for banners or other such adverts in return for commission.

As time goes on advertisers have been becoming more demanding with the requirements needed to reach an ever more specific audience. Advertisers segment their audiences using various attributes such as city, time, gender, keywords, device or operating system. This is known as microtargeting [2] and reduces the number of visits which can comply with their requirements, but au contraire advertisers pay a

higher price. Micro-targeting consists of segmenting an audience in line with various attributes inn order to be directed towards a small group with the same interest.

Doing this ensures adverts are only shown to users complying with the advertisers requirements and hence are more likely to buy the product. Small ad networks cannot offer such specific campaigns given the fact they do not receive enough visits, and that only a small part actually complies with advertisers requirements. It must also be mentioned that many visitors are not shown a single advert as they do not comply with the set requirements.

That is why it has become vital for small networks to work together to create one large global Ad Exchange Market. Each network is composed of a group of advertisers and a group of editors. In order to manage the exchange we have to take on some tasks such as invoice delivery, private policy [3] and fraud prevention [4] but without a doubt the most important task and what we are going to focus on here is selecting the best candidate from all possible candidates and doing it in the shortest time possible.

Some studies deal with the various factors that should cover the ads-exchange algorithms to assign a value to the Quality Score of each campaign. This research estimates this parameter depending on the performance obtained when other factors were shown [5]. There are also reports that aim to optimize optical price or the best candidate based on a number of parameters by the use of complex mathematical formulas [6]. We will focus on comparing the computational costs of several algorithms whose objective is to select the best candidate in the best possible time. We assigned random values to the Quality Score as assuming that they have already been calculated.

To solve this problem we have developed various solutions. Firstly, we applied parallelism through threads in the C# programming environment; this allows multiple radiuses to be run simultaneously.

We also added fuzzy logic to show adverts to visits that do not exactly comply with advertiser's requirements.

We then proposed other solutions where a tree structure is created in order to reduce the number of comparisons. To do so we used hash coded AVL trees and multiple node trees. These structures make it possible to create tree branches, and hence improve algorithm efficiency.

The final algorithm was developed using the language Pig Latin, from Apache Hadoop. This platform has the necessary tools to simply and efficiently solve Big Data problems.

For each algorithm we created a table of results and then compared them. Finally, we came to some conclusions regarding what we consider the best solution in terms of parameters and then proposed a series of improvements for the future.

2 Description of the Problem to be Solved

Due to the fact there are millions of advertisers and of which each and every one is simultaneously creating multiple campaigns, selecting an ad to be shown is a rather complex task.

To select the best advertiser we have to take them all into account, and give an answer in less than 0.1 seconds [6], so it is of the utmost importance to really design efficient algorithms.

The problem we are trying to solve requires selecting the most adequate campaign for each visit in the shortest time possible. To do this the requirements of each and every campaign need to be analyzed.

Should there be various advertisers who comply with the said requirements; the one with the highest Ad Rank is selected. The Ad Rank is a parameter aiming at better profits for the network but at the same time showing quality ads. The Ad Rank formula is:

$$\text{Ad Rank} = \text{CPC} \times \text{Quality Score}$$

Each platform uses its own method to calculate the Quality Score value, for example Google has never revealed how they calculate theirs.

The advertiser's parameter format is shown in table 1 below, it will be the same for the visit's parameter only with the Quality Score and CPC values removed.

Table 1. Advertisers selected parameter values.

Hour	Browser	Browser Version	OS	OS Version	Parameter N …	Quality Score	CPC
3	Chrome	20.0.1132.47	Macintosh	Intel 10.5	…	0,634	1,695
14	Chrome	22.0.1229.94	Windows	XP	…	0,982	6,088
15	I. Explorer	8.0	Windows	XP	…	0,796	9,370
1	I. Explorer	7.0	Windows	XP	…	0,730	6,856
7	Chrome	22.0.1201.0	Windows	Vista	…	0,545	1,704

The values of each column represented in table 1 are as follows:

1. Hour: Refers to the time of the day the visit was made.
2. Browser: Refers to the browser the visit came from, most commonly Internet Explorer or Chrome.
3. Browser version: This parameter refers to concrete browser version. Browsers are constantly getting faster and more secure version updates.
4. Operating System: The most common ones are Windows, Mac or Linux.
5. OS Version: Just like browsers, OS's have their version e.g. Windows 7, 8 or Mac OS X Lion.
6. Flash version: Some browsers have flash installed. Some versions include 11.3 r31, 10.0 r32 and 10.2 r153.
7. Has flash? : Indicates whether or not parameter uses flash or not.
8. Screen bitrate: This indicates the number of bits needed to show a pixel, usually 32 bits.
9. Screen resolution: Number of pixels by width and height of the on screen image.
10. Country: We can know the country using the users IP number.

11. City: As well as seeing the country of visit origin we can also see the specific city.
12. Language: This indicates the OS language, for example: en, en-us etc….
13. Network address: This refers to the ISP url the user is visiting from.
14. Network name: This refers to the name of the network being used by the user.
15. Access page: The access page is page visited previous to the visit. Most come from search engines but they can also be directly accessed, or through a link.
16. Visit type: User visit types can be direct or referrals using a search engine or any other page type.
17. CPC: Cost per Click. The maximum value an advertiser is willing to pay for an ad to be shown
18. Quality Score: This indicates the ad quality and is calculated based upon many factors such as the number of click per view.

In table 2 we can see configuration options. The numbers in each column represent the advertisers selected parameters; each number corresponds to the parameters described above.

Table 2. Option parameter configuration

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Option 1		X		X						X						
Option 2		X		X						X	X	X			X	
Option 3	X	X	X	X	X			X	X	X	X	X			X	
Option 4	X	X	X	X	X			X	X	X	X	X	X	X	X	X

Google uses supercomputers to solve complex algorithms in a tenth of a second. For example, when we search for "Brazil" at google.com we are given 3,230,000,000 results thanks to the efficient algorithms run on such supercomputers. To solve the problem we have used an Intel(R) Core (TM) i5-2400 CPU @ 3.10 GHz with 16Gb RAM running Windows 7 Pro Service Pack 1 64 bit.

By using this hardware we are going to solve the problem in different ways. We have developed thread code and AVL trees as well as Multi-Node trees within the Microsoft Visual C# 2010, C# Express environment. We have also tried Pig Latin using technology developed by Yahoo called Hadoop. To run Hadoop we have a virtual machine called Hortonworks Sandbox 2.1 on the OS Red Hat, which running Oracle Virtual Box, 4.3.14 r95030 and the aforementioned hardware.

3 Application of Parallelism and Fuzzy Logic within Ad Exchanges

3.1 Applying Parallelism to Ad Exchanges

In today's world we have the power of multi-core processors allowing huge volumes of information to be analyzed. Parallel computing is a processing method using

various instructions at once, as the name suggests, in parallel. This is based on the principle that large scale problems can be divided into smaller ones and hence be resolved simultaneously.

Threads are used in parallel computing, these are tasks that can be done at the same time as others. Different execution threads share a series of resources such as memory space, files or authentication keys.

Parallelism allows various advertisers to be simultaneously compared rather than each visit having to be compared one at a time.

The program we have developed creates 100.000 threads, and provided there is no time limit, can be executed in parallel. Every 100.000 threads can compare 100 campaigns that have been stored in a file along with the visit, making a total of 1.000.000 campaigns. Upon finishing the program, it writes the best solution in a file. We have repeated this step for 1000 visits.

The algorithm we have developed runs using three variables: The option, the number of seconds and the threshold of the degree of similarity. The variable "Option" indicates the number of parameters the advertiser has chosen such as those shown in table 2. For example, option 1 shows the chosen browser, OS and country. The variable "Seconds" indicates the maximum time to calculate a solution and then the variable "Threshold" represents the minimum similarity a visit must have in relation to the advertiser's requirements in order to be shown the ad.

The program pseudo-code is:

```
Begin_Main_Program
From visit = 1 to 1.000
  from j=1 to 100.000
    Create_thread(j);
  from k=1 to 100.000
    run_thread(k);

  While (thread_finish)
    Wait();
  Save_best_solution();
End_Main_Program

Run_Thread_Function
  Read_advertisers_from file(k);
  Compare_advertisers_with_visits();
  if(solution > global_solution)
    global_solution = solution;
  if(Last_thread())
    Write_solution_file(global_solution);
End_Run_Thread_Function
```

3.2 Application of Fuzzy Logic within Ad Exchanges

One of the biggest problems that come up when advertisers configure many a parameter is that very few visits comply with their requirements and hence an ad receives very few views. To increase ad coverage we can apply fuzzy logic. Using this, ads that are very similar but not entirely alike can still be accepted, and hence viewed.

Fuzzy or Heuristic logic is an extension of traditional logic using concepts similar to those of human thought. While traditional logic uses strict boundaries to determine where certain sets belong, for example, "a person is old if they are older than 70", however should a person be 69, they can still be classed as old.

Fuzzy Logic allows us to better adapt ourselves to the real world, and understand such expressions like "It's not very cold" or "You're very young". When it comes to understanding the quantifiers of expressions like "much", "very" or "a few" we use belonging functions to indicate to what extent the element is part of the set. Similarity Matrixes are also used to establish a degree of similarity amongst various elements in a set.

In order to establish the degree of similarity applied to our problem, we have created a series of matrixes that represent the grade of similarity between visit value and configuration value within a campaign on a scale from 0 to 1.

In table 3 we can see the degree of similarity that exists amongst the main languages of visits received by a webpage, given the webpage is in Spanish most visits are from Spanish speakers. In this table we can see the degree of similarity among Spanish speaking countries is very high. In total we have created 12 tables, one for each parameter, on which we wish to apply fuzzy logic.

Table 3. Similarity matrix for OS language parameter.

Language	Ca	En	En-Gb	En-Us	Es	Es-419
Ca	1	X	X	X	X	X
En	0	1	X	X	X	X
En-Gb	0	0.9	1	X	X	X
En-Us	0	0.8	0.7	1	X	X
Es	0	0	0	0	1	X
Es-419	0	0	0	0	0.9	1

3.3 Results Obtained from Fuzzy Logic and Parallelism

With a threshold value of 1 and an option value of 2, the algorithm took a total of 147,3 seconds to compare just one visit with 1.000.000 million advertiser campaigns. If we take the maximum established time of 0.1 seconds into account, we realize that this algorithm is unusable, but we have to take into account the fact that this algorithm is run on a supercomputer with optimized access to files or allows them to be held in memory, so such an algorithm may be viable.

Due to very high times, we have established a maximum number of seconds from which no more threads will be processed. Logically, the higher the number, the more threads can be run, and hence bring a better a solution. With a lower threshold more visits comply with advertisers requirements, giving better results. The results shown in table 4 show the average Ad Rank value. The higher the value, the better the quality of ads displayed.

Looking at table 5 the number of comparisons increases when using fuzzy logic. This is due to the fact that all similarity matrixes have to be run through. Firstly, we look at the lines comparing them with visit parameters and then we look at the columns comparing them with campaign values. Should the results be 3 and 5, the matrix cell [3, 5] will receive a grade of similarity between the two values.

Table 4. Results of the algorithm for parallelization by time using fuzzy logic.

	Threshold	1 Sec	2 Sec	3 Sec	5 Sec	10 Sec	15 Sec	25 Sec
Option 1	0.7	8,77	8,93	9,00	9,05	9,11	9,16	9,38
	0.8	8,78	8,93	9,00	9,05	9,10	9,16	9,40
	0.9	8,78	8,93	8,98	9,06	9,11	9,19	9,37
	1	8,80	8,94	8,99	9,06	9,11	9,17	9,37
Option 2	0.7	8,45	8,62	8,72	8,80	8,88	9,01	9,17
	0.8	8,28	8,54	8,64	8,72	8,85	8,92	9,09
	0.9	6,28	6,89	7,10	7,35	7,65	7,81	7,98
	1	5,01	5,59	6,03	6,26	6,95	7,11	7,54
Option 3	0.7	8,50	8,70	8,79	8,88	8,91	9,05	9,17
	0.8	7,50	7,92	8,11	8,29	8,49	8,57	8,70
	0.9	4,50	5,18	5,55	5,96	6,48	6,71	7,04
	1	0,08	0,19	0,25	0,38	0,51	0,74	1,00
Option 4	0.7	8,10	8,35	8,49	8,64	8,75	8,83	9,01
	0.8	6,58	7,05	7,34	7,63	7,97	8,09	8,25
	0.9	3,58	4,16	4,64	4,93	5,66	5,97	6,29
	1	0,02	0,06	0,05	0,09	0,17	0,19	0,35

Table 5. Number of comparisons for 1.000.000 campaigns using and not using fuzzy logic.

Option	Comparisons Using Fuzzy	Comparisons not using Fuzzy
1	107.600.000	107.400.000
2	1.212.300.000	117.273.810
3	2.035.700.000	134.354.215
4	2.434.900.000	144.191.923

Both "Using fuzzy" and "Not using fuzzy" comparisons are results of comparing visit parameters with 1.000.000 campaign parameters. More time is spent accessing files.

4 Using AVL Trees to Optimize Ad Exchanges

4.1 Developing Algorithms using AVL Trees

To improve computing costs of such algorithms we have employed AVL trees and hash code. AVL trees take their name from the first letter of the surname of its

inventors Adelson-Velskii and Landis. There are binary search trees that satisfy the condition that they are always balanced, so that for each node, the height of the left branch will never differ by more than one unit of the height of the right branch or vice versa.

A binary search tree is a data structure allowing the organization of attribute information; each tree node must comply with the following characteristics: Lower Nodes to the left of a particular node must contain lower values; lower nodes to the right must contain higher values.

For example, let's say the advertiser has decided to configure the following parameters with the following values: Time=21, Browser= Firefox, Browser Version = 14.0.1, OS = Windows, Country = Spain and City = Pamplona. In such a case the chain value will be: "21Firefox14.0.1WindowsXPSpainPamplona". When the hash function is applied to the chain the value becomes: "2C1ECBEA35C21B712410CE7F7D0BB".

Via a hash our algorithm codes the field values of each one of 1.000.000 advertiser's campaigns and then adds them to the AVL tree as nodes. Each node uses an alphanumeric keychain generated by the hash function representing the parameter combination and an attribute with Ad Rank value. A tree must then be created for each of the options, in our case we have four options and hence have created four trees.

4.2 Results Obtained with AVL Trees

The time needed to process 100.000 visits with 1.000.000 advertisers with this algorithm is 1.66 seconds, meaning that the algorithm runs around 9,2 million (9.206.250 to be precise) times faster than the threads. This is due to the algorithm not needing to access any files as the tree can be loaded from memory, and the number of comparisons per visit for option 2 has reduced to 1.172.738.107 with the "Using fuzzy" thread option at only 51.03 with AVL trees.

Table 6. Results obtained from AVL algorithm for 100.000 visits.

Option	Seconds	Average Ad Rank	Average comparisons
1	1,36	9,89	16,65
2	1,55	8,45	30,42
3	1,84	1,24	49,74
4	1,92	0,44	51,03

The average number of comparisons is calculated as the average 100.000 visits. The results are the best possible, given that each and every one of 1.000.000 advertisers has been compared. However, when using threads we had to limit the number by the amount of time taken and hence, the results were not the best possible achieved.

5 Using Multi-Node trees to improve ad performance

5.1 Developing algorithms using Multi-Node trees

Each first level node has a number of children representing possible values a campaign can achieve. Thus if the first campaign parameter has 29 different values, the first level will have 29 children. Should node number 7 on the first level have a second parameter of 12 values, then the node shall have 12 children and so forth for all parameters. The final tree level will contain the Ad Rank value, and just as with the AVL trees we have had to create four Multi-Node trees, one for each configuration option.

To solve the algorithm we tried three different solutions:

1. Unordered trees: These unordered trees consume the least as they do not as they do not have any operations to order. The trees are formed from selected parameters in advertiser's campaigns. The possible values are added to the tree as campaigns are processed
2. Ordered Trees: With ordered trees we applied the same process as with the unordered trees, though we then ordered them alphabetically by parameter name. This was done so that the descending binary search can be used right from the tree root to the leaves to obtain the Ad Rank value.
3. Ordered trees by frequency: In this case, we do the same as the first however we order the trees using the frequency with which an advertiser demands a parameter. If most advertisers configure the time as 13:00 then the most left hand side thread will have this value, and a comparison will be made using this node.

5.2 Results Obtained by Multi-Node Trees

Table 7. Results from Multi-Node trees from 100.000 visits.

		Not ordered		Ordered and using binary search		Ordered by frequency	
Option	Result	Seconds	Average comparisons	Seconds	Average comparisons	Seconds	Average comparisons
1	9,89	0,58	20,59	0,78	37,95	0,53	18,52
2	8,45	1,18	50,90	1,17	75,68	1,03	43,59
3	1,24	1,81	73,25	1,86	99,75	1,69	64,93
4	0,44	1,61	74,98	1,65	102,42	1,65	66,64

As we can see by the number of comparisons the best option is ordering by frequency, the second best are the unordered trees and the third best are the ones ordered by parameter name and then having a binary search applied.

Many tree nodes can be formed by four or five nodes so doing a binary search doesn't make much sense, we can also discard the order by frequency option as the

algorithm has 0.07% less comparisons and hence ordering them into a tree every time a campaign is added would be unjustified.

6 Hadoop Optimizing Ad Exchanges through Apache Hadoop

One of the simplest ways to solve the problem and we can safely assume one of the less puzzling, uses Hadoop, which was famously developed by a Yahoo employee. Apache Hadoop is a framework oriented towards finding solutions to Big Data problems, such is the case in point and the fact it also solves our problem in just a few lines.

This language is oriented to take advantage of the clusters and supercomputers of large companies such as Yahoo, Amazon and Google. These companies use this kind of structure because they run algorithms processing huge amounts of data.

This platform uses two programming languages, Hive and Pig Latin. To solve our problem we used Pig Latin, although it is not as efficient as the trees as it has an additional computing cost 151.2 times higher than AVL trees and 205.9 times higher than Multi-Node trees ordered by frequency, however it allows the problem to be solved in just 10 lines. The code is explained in the program comments below:

```
-- ADVERTISERS
-- Load advertisers table from memory
Anun0 = Load 'default.anunciantes2' USING
org.apache.hcatalog.pig.HCatLoader();
-- For each line we select the columns that interest us
Anun1 = Foreach Anun0 Generate $2, $4, $8, $9, $10, $11, $12, $15,
$19*$20;
-- Then group the lines together to later select the max Ad Rank
Anun2 = Group Anun1 by ($0,$1,$2,$3,$4,$5,$6,$7);
-- Remove groups
Anun4 = For each Anun3 Generate FLATTEN($0),$1;

-- VISITS
-- Load advertisers table from memory
Visitas0 = Load 'default.visitas' USING
org.apache.hcatalog.pig.HCatLoader();
-- For each cell we select the columns that interest us
Visitas1 = For each Visit0 Generate $2, $4, $8, $9, $10, $11, $12, $15;

-- JOINING VISITS AND ADVERTISERS
-- We create a table to coincide with both visit and advertise fields
Visitas2 = Join Visitas1 by ($0,$1,$2,$3,$4,$5,$6,$7), Anun4 by
($0,$1,$2,$3,$4,$5,$6,$7);
-- We then select the columns from those tables that interest us
Res = foreach Visitas2 generate $0,$1,$2,$3,$4,$5,$6,$7,$16;

-- Save answer
store Res into 'Respuestas';
```

In table 8 we can see the results obtained as well as the times needed to obtain them. Tests were done with 100.000 visits and 1.000.000 ad campaigns, and the results obtained are the same as the ones from the AVL and Multi-Node trees. Time is expressed in minutes and seconds, rather than solely seconds as for AVL and Multi-Node trees.

Table 8. Results obtained from the algorithm using Hadoop's Pig Latin

Option	Seconds	Results
1	214	9,89
2	226	8,6
3	260	1,24
4	309	0,44

7 Conclusions

According to the results, the thread option cannot be considered appropriate due to the enormous amount of time required to run the algorithm. One of the reasons behind the elevated time scale is the fact the program takes a long time accessing the 10.000 files used to save advertiser's campaigns. The number of total comparisons is the number for each thread multiplied by the number of threads, coming to a total of 24.349.000.000 comparisons, while using trees it does not exceed 103.

Via Hash, AVL trees give the best results, although they do have two disadvantages, firstly the tree needs to be modified for every single campaign, taking up a lot of time; secondly, these trees are inadequate for Fuzzy Logic use given that upon applying the hash function it gets difficult to compare attributes and establish a degree of similarity as they are coded. In order to implement this kind of logic all possible parameter relations would have to be hash coded with all possible combinations, bringing us to the conclusion that this is an unviable option as it will exponentially increase the number of tree nodes.

Another disadvantage affecting both AVL trees and Multi-Nodes is the computational cost of creating the tree, though this is not really anything to worry about as it can be done offline. That is to say it is not created at the time of a user visit, and hence is not a critical computational cost.

Multi-Node trees have the advantage over AVLs that they can use Fuzzy Logic. This can be done using a simple backtracking algorithm that traverses the tree and changes route should the similarity threshold be overcome. Taking these results into account, it seems that ordering is not a great advantage as looking at the results obtained there is no big difference between the number of neither comparisons nor time consumed.

Finally, if we take the capacity of some of the supercomputers used by technology companies into account, Pig Latin is the best option as its algorithm development code can be summarized in just ten lines. This has the advantage that errors are highly unlikely as well as Fuzzy logic being able to be implemented easily via UDF (User Defined Functions), which are user language implementation methods for both Python and Java programming languages.

8 Future Work

One possible improvement to this algorithm could be adding fuzzy logic; to do this we must make a translation table. If we take into consideration the fact that the three browsers in the table are similar we can then code them with the same code.

To improve comparison block searches upon fuzzy logic application a value could be assigned to each parameter. E.G., I. Explorer 8.0 shall be 7 and I. Explorer 8.0 shall be 9, meaning the similarity between the two will be [7, 9] of the browser matrix. With this a number of comparisons per search will be saved, this can be calculated using the formula: Comparisons: (Lines/2 + Columns/2), assuming the probability of coincidence for all values is the same.

Another improvement could be to keep the ordered by frequency algorithm and instead of ordering it per new campaign, an ordering algorithm shall be applied once per 1,000 new campaigns.

With such formula we can create a tree and then compare each visit with the advertisers formed tree. This in turn would make the program run much faster even though it would also require more lines of code.

References

1. IAB internet advertising revenue report. Obtained from http://www.iab.net/about_the_iab/recent_press_releases/press_release_archive/press_release/pr-122313 (2012)
2. Moe, W. W.: Targeting Display Advertising. London, UK: Advanced Database Marketing: Innovative Methodologies & Applications for Managing Customer Relationships (2013)
3. Neslin (Eds.): Advanced Database Marketing: Innovative Methodologies & Applications for Managing Customer Relationships, Gower Publishing, London (United Kingdom) (2013).
4. G. Johnson: The Impact of Privacy Policy on the Auction Market for Online Display Advertising. Simon School Working Paper No. FR 13-26 (2013)
5. B. Stone-Gross, R. Stevens, A. Zarras, R. Kemmerer, C. Kruegel, and G. Vigna.: Understanding Fraudulent Activities in Online Ad Exchanges. In ACM SIGCOMM Conference on Internet Measurement (IMC) (2011).
6. Y. Chen, P. Berkhin, B. Anderson, and N. R. Devanur: Real-time bidding algorithms for performance-based display ad allocation. KDD (2011)
7. W. Zhang, S. Yuan, and J. Wang: Optimal Real-Time Bidding for Display Advertising. In Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining (2014)