FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Assignment of bachelor's thesis

Title:	Analysis and implementation of planning strategies for guaranteed advertisement selection
Student:	Jakub Dvořák
Supervisor:	Ing. Josef Bouška
Study program:	Informatics
Branch / specialization:	Web and Software Engineering, specialization Software Engineering
Department:	Department of Software Engineering
Validity:	until the end of summer semester 2020/2021

Instructions

Guaranteed advertisement is a system of orders for the required sum of impressions over time for multiple clients.

1) Describe how existing guaranteed advertisement system works and what are its advantages and disadvantages of using a primitive selective algorithm.
2) Improve strategy of advertisement distribution and fulfilling orders over the existing solution and prove the choice on a prototype.
3) Analyze advertisement selecting algorithms based on predicted requests (i.e. Shale [1]) and algorithms working only with feedback in real-time (i.e. Prop-alloc [2]). Specific algorithms are only suggested.
4) Implement proofs-of-concept of given algorithms. It is not necessary to implement the whole system for advertisement delivery. The programming language is not restricted.
5) Test the chosen solution performance in context with the existing system by simulating on artificial data. Evaluate the results of testing and recommend how to improve the existing system.

–

[1] https://arxiv.org/pdf/1203.3619.pdf
[2] https://storage.googleapis.com/pub-tools-public-publication-data/pdf/942f3ec628f9d40b289a5ccf9b750e4...

Electronically approved by Ing. Michal Valenta, Ph.D. on 31 January 2020 in Prague.

Bachelor's thesis

ANALYSIS AND IMPLEMENTATION OF PLANNING STRATEGIES FOR GUARANTEED ADVERTISEMENT SELECTION

Jakub Dvořák

Fakulta informačních technologií ČVUT v Praze
Department of Software Engineering
Vedoucí: Ing. Josef Bouška
June 27, 2021

České vysoké učení technické v Praze
Fakulta informačních technologií
© 2020 Jakub Dvořák. Všechna práva vyhrazena.
Tato práce vznikla jako školní díla na Českém vysokém učení technické v Praze, Fakultě informačních technologií. Práce je chráněna právními předpisy a mezinárodními úmluvami o právu autorském a právech souvisejících s právem autorským. K jejímu užití, s výjimkou bez uplatněných zákonných licencí nad rámec oprávnění uvedených v Prohlášení je nezbytný souhlas autora.

Odkaz na tuto práci: Jakub Dvořák. *Analysis and implementation of planning strategies for guaranteed advertisement selection.* Bachelor's thesis. České vysoké učení technické v Praze, Fakulta informačních technologií, 2021.

Contents

List of Figures

List of Tables

I would like to thank Seznam.cz a.s., and especially my supervisor Ing. Josef Bouška for supporting me and allowing me to carry out a part of my bachelor's thesis research as my work assignment.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended. In accordance with Article 46(6) of the Act, I hereby grant a nonexclusive authorization (license) to utilize this thesis, including any and all computer programs incorporated therein or attached thereto and all corresponding documentation (hereinafter collectively referred to as the "Work"), to any and all persons that wish to utilize the Work. Such persons are entitled to use the Work in any way (including for-profit purposes) that does not detract from its value. This authorization is not limited in terms of time, location and quantity.

In Prague on 27th June 2021 .....

Abstrakt

Problém garantované reklamy je problém online stochastického výběru vhodného kandidáta v bipartitním grafu. Tato práce se zaměřuje na existující algoritmy řešící tento problém, konkrétně SHALE, PropAlloc a Advert. V práci tyto algoritmy podrobně popíšeme a vysvětlíme, jak u každého algoritmu proces výběru probíhá. V práci také implementujeme framework pro porovnání algoritmů na výběr garantované reklamy ve srovnatelných podmínkách na nasimulovaných datech. U implementace tohoto frameworku se zaměřujeme na možnost rozšíření o další algoritmy a o možnost vývoje algoritmů. V této práci také implementujeme všechny výše uvedené algoritmy do vytvořeného frameworku a porovnáme je v několika scénářích na základě obchodních požadavků specifikovaných společností Seznam.cz. Z práce vyplývá, že celkově nejlepším algoritmem je algoritmus Advert, ale v některých kategoriích vycházejí lépe ostatní algoritmy.

Klíčová slova garantovaná reklama, porovnávající framework, SHALE, PropAlloc, Seznam.cz, Python

Abstract

Guaranteed display advertisement serving is a problem of online stochastic matching in bipartite graphs. This thesis takes a closer look at existing algorithms solving this problem, namely SHALE, PropAlloc, and Advert. We go on to describe these algorithms in detail, explaining how their individual matching process works. We also implement a bench-marking framework to compare these algorithms in a level playing field with simulated data, with main focus on scalability of this solution to other algorithms, and algorithm development. We implement the before mentioned algorithms into the bench-marking framework, and compare their performance in multiple scenarios based on business requirements specified by Seznam.cz. In the end we determine Advert to be the best performer, however, we also point out the positives of every algorithm.

Keywords guaranteed advertisement, comparison framework, SHALE, PropAlloc, Seznam.cz, Python

Acronyms

ARIMA	Autoregressive integrated moving average
CSV	Comma-separated values
DSP	Demand-side platform
JSON	JavaScript Object Notation
PyPI	Python Package Index
SSP	Supply-side platform
SVG	Scalable Vector Graphics

Chapter 1
Introduction

1.1 Motivation

Advertising is what powers the internet in today's day and age. It is a lucrative business that is a main source of income for internet giants Google and Facebook, and locally even for Seznam.cz. What puts them in a position of making profits on advertising is that they provide SSP.

SSP, or Supply-side platform, is a service that works as an advertisement provider for web pages that want to monetize their content. Once the web page gets SSP to serve advertisements on their website, they are making money, the DSP is making money, and the SSP as the provider takes a cut. The SSP is, in its essence, an advertisement marketplace, matching demand for advertisements from web pages to supply from DSPs.

DSP, or Demand-side platform, is the source of the advertisements. Under each SSP there are usually many DSPs. When the web page asks for an advertisement, the SSP carries out an auction, decides which advertisement wins, and as such, how will the site make the most money. The DSPs take part in the auction based on many targeting parameters which the SSP provides. Targeted advertising is a popular form of online advertising because it allows specification of target demographic, limiting displaying the advertisements only to them. These SSP targeting parameters describe the type of advertisement needed, the place where it will be shown, as well as description of the user that visited given website. This information is obtained from the web page, or from the SSP itself which usually keeps track of identity of the visitors.

That is in general how display advertising works - as a DSP you get a specific request and you either match it to your advertisement, put a price on it and take part in the auction, or you just do not take part at all. Your client only pays if his advertisement is shown, and can set up limits on the amount he wants to spend and when he wants the advertisement to be shown.

Guaranteed advertisement is different from the normal display advertisement because as a DSP you are committing to serving a specific amount of impressions in the future. You are guessing that said amount of people matching the targeting will visit the targeted site in the specified time frame.

The problem of guaranteed advertisement is that it relies on the DSP knowing that it is going to be able to display advertisement certain number of times in the future. So the DSP is naturally starting to be dependent on knowing the future, or at least predicting it with a reasonable accuracy. At the same time, its also necessary to be able to compute these predictions within such a time frame, that by the time the next forecast finishes, the old predictions are not completely out of date. Adding more options of targeting the user, and giving more options of restricting the supply to the advertisers through methods like capping[1] causes the complexity of

[1] the same advertisement will not be shown to the same user in the time interval selected by the advertiser

computing these predictions to rise exponentially. This means we have to find a balance between not relying on completely accurate predictions while at the same time having the advertisements evenly distributed.

1.2 Aim of the Thesis

As the serving of guaranteed advertisement is a problem troubling multiple internet companies, some of them have brought out research papers describing possible algorithms for the guaranteed advertisement distribution. However, there is not a direct comparison between these, because they are tailor made solutions for the requirements of the company the algorithm was developed for. The goal for this work is to provide that comparison, and more importantly to provide a platform for comparing such algorithms in a level playing field. Any future algorithms can then be plugged into this framework and compared on multiple business-critical parameters against other algorithms. This work will not be focusing on computing predictions, nor will it be focusing on the SSP side of advertisement transaction. Both will however be described for context and considered in comparison.

1.3 Structure of the Thesis

Chapter 2 shows what the general algorithm should do, as well as describing the algorithms that were chosen to implement. They are namely PropAlloc[2], SHALE[3] and a legacy Seznam.cz algorithm used in previous guaranteed advertisement solution.

Chapter 3 describes the framework created for testing these algorithms, as well as how it simulates real circumstances and what are our business-critical metrics for production implementation.

Chapter 4 will present the findings from experiments using framework from chapter 3, and carefully crafted data exploring real world scenarios.

The thesis will conclude in Chapter 5, where we explore possible future work on expanding the framework as well as testing new algorithms.

[2] courtesy of Google and Columbia University
[3] courtesy of Yahoo

Chapter 2

Algorithm for choosing advertisement

2.1 Introduction to guaranteed advertisement selection

In internet advertisement, there is demand and supply.

Demand is representing advertisements themselves. There are many choices of targeting the advertisement to fit the perfect web page visitor for a product, as well as measures to moderate the amount of exposure given to the same visitors. It is possible to target peoples gender, age, geographical location, interests, the device they are using, as well as the site they are on, even a specific article on a news site. It is also possible to then narrow the selection down even further, by selecting which hours of the day the advertiser wants to show the ad, or utilizing capping. The concept of capping is that every visitor has a unique identification, and the advertiser can limit how often to show the advertisement to them.

The other side is supply. Supply is represented by impressions. Impression is the feedback DSP gets after serving the advertisement, usually a call to a specific URL confirming that the advertisement was actually rendered into the targeted site. This is the definite metric that has a business value. If the DSP does not receive the confirmation of serving an advertisement in form of impression, it cannot bill their client for it. Each impression is unique, and carries information about the visitor who has seen the advertisement. This information is used to select the advertisement based on targeting on the user. Some impressions will not have all of the targeting data available because of the recent GDPR legislation, giving the visitors control over the data that websites collect about them. All of the targeting selection is of course supplied to the DSP before choosing the advertisement, however, impressions are the actual product that the DSP is selling, so that is why we consider them to be the supply. Even though the supply is actually visitors of the website.

While the targeting selection allows for a combinatorial explosion, in normal advertising that is a non-issue. The DSP simply runs the currently running advertisements through the specified filters, and selects the best candidate in the end - the one that makes the most money.

Where this starts getting complicated is when the DSP guarantees to serve certain number of impressions for given targeting. In this scenario, the DSP is selling something that will happen in the future, and cannot quite know if the requested impressions will actually arrive. So it has to turn to predicting, or forecasting the future. It is not difficult to approximate future data points in a time series based on historical data. There are many models out there to select from. For example, the current production solution in Seznam.cz features implementation of Hannan-Rissanen [13] algorithm for ARIMA models by Workday, Inc.[21]. The problem is that when

Table 2.1 Order for three advertisements from Example 2.1.

AdvertisementId	Interest	Location	Quantity
1	I1	L1 or L2	100
2	I2	N/A	200
3	I2	L2 or L3	300

$I1$ $I2$

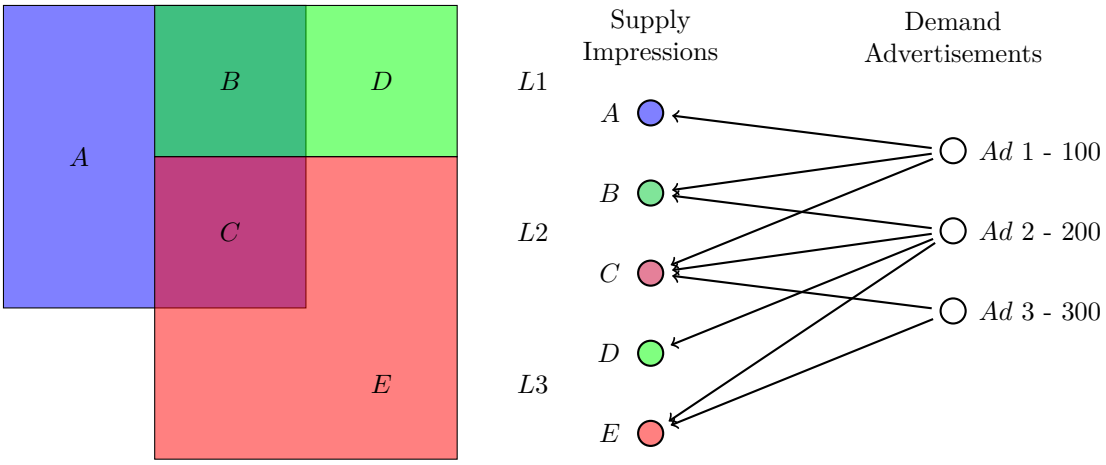

Figure 2.1 Transformation of the example offer from Table 2.1 to bipartite graph.

more advertisements are added into the equation the targeting gets fragmented even further, and as such increases the number of series that the algorithm has to forecast.

In order to be able to compute this forecast of impressions, while keeping the number of series minimal, they are grouped together by matching targeting in the largest possible *disjunct* sets. Creating such sets allows to transform this problem to a bipartite graph matching, where one side of nodes represents supply, the other demand, and edges represent whether or not is given advertisement matching the adjacent impression set. The process of this transformation is shown in the example 2.1.

Transforming this problem into stochastic matching in bipartite graphs, allows us to solve it as there already exists a lot of research on that topic.

The problem of solving matching in graphs has been popular in computer science for a very long time, and as such there exists considerable amount work on those algorithms over the years.[12, 5, 19, 15, 16] Ran Duan and Seth Pettie went into great depths to describe the history and complexity of Cardinality and Maximum Weight Matching algorithms, and improve the efficiency by approximating the result.[4]

Focusing on online matching in advertisement, there is a plethora of work, some of which serves as a base for the algorithms that we chose for implementation.[6, 11, 20, 18, 3]

The algorithms that we chose to describe, implement and benchmark are SHALE, PropAlloc and Advert. All of them have a single purpose - when they receive a request that can be matched into a supply node on the bipartite graph, they have to select what advertisement, if any, to return as a winner.

▶ **Example 2.1.** With the given three advertisements from Table 2.1 the targetings form a Venn diagram where each overlapping targeting forms a new data series to forecast. In total we have 5 final series that need to be predicted in this example. These series are displayed

in the bipartite graph connected to their respective advertisements. We can see that each of the advertisements has its quantity specified. The targeting data series nodes will have the quantity attached once it is forecast. That will help certain algorithms decide in selection of advertisement, other algorithms are able to function with just this bipartite graph. E.g. when a request matching supply node B arrives, the algorithm has to decide whether to pick *Ad* 1, or *Ad* 2.

Some of these series can be predicted by simply applying the selected forecast algorithm to specific series - e.g. B or C in Figure 2.1. Other series have to be predicted by forecasting multiple series and adding or subtracting them - e.g. series A has to be computed the following way:

- Add series Interest I1, Location L1
- Add series Interest I1, Location L2
- Subtract series Interest I1 & I2, Location L1 - was already predicted as series B
- Subtract series Interest I1 & I2, Location L2 - was already predicted as series C

2.2 SHALE

This section will describe the practical application of the SHALE algorithm.[2] The algorithm consists of 2 stages - offline and online, and the offline stage is divided into 2 phases. The offline stages needs forecast predictions to give weight to the supply side of the bipartite graph. It works on the basis of approximating the best possible weights of edges of the bipartite graph for stochastic matching by computing them from coefficients assigned to every node. Those coefficients are adjusted in every round of offline phase. The algorithm can be stopped at any point and give usable edge weights, however, the more rounds that are run the closer the weights are to the perfect solution.

2.2.1 Glossary

- i - index of targeting set/supply
- S_i - forecast size of targeting set
- j - index of advertisement/demand
- D_j - ordered number of impressions
- x_{ij} - weight of given edge, value is used for selecting advertisement,

$$x_{ij} = max(0, \sum_j (\theta_j) * (1 + (\alpha_j - \beta))$$

- θ_j - ordered demand divided by possible supply for its targeting,

$$\forall j \theta_j = \frac{D_j}{\sum_{i'}(S_{i'})}, i' \in \text{(edges connected to j)}$$

- α_j - demand constraint, $\forall j \sum_{i'}(S_{i'} * x_{i'j}) >= D_j, i' \in$ (edges connected to j)
- β_i - supply constraint, $\forall i \sum_{j'}(x_{ij'}), j' \in$ (edges connected to i)

The original algorithm also includes penalties for failure to deliver, importance of presence of advertisement in impressions, and mechanisms to finish unfinished orders. All of these do not make business sense for us, so we will forgo using them by substituting them by neutral values not affecting the algorithm. We will also forgo the zeta computation, which serves as α_j with receding demand. This decision was taken due to the architecture of our distributed server solution.

2.2.2 First phase

Computing α_j for groups and β_i for targetings. This phase is where the approximation takes place. The more iterations it goes through, the better the coefficients should be. However, there is no minimal number of iterations. The algorithm should provide usable data when stopped at any point.

α_j is initiated to 0 for all advertisements.

2.2.2.1 Computing Betas

β_i is computed from solving following equation $\sum_j(x_{ij}) = 1$. However, since x_{ij} uses max function we cannot simply equate β_i to:

$$\beta_i = \frac{(1 + \sum_j(\theta_j))\sum_j(\theta_j * \alpha_j)}{\sum_j(\theta_j)}$$

since for some impression set

$$\theta_j * (1 + \alpha_j - \beta_i)$$

could equal less than 0. Negative value should be substituted by 0, and as such not take part in computation of β_i value. We will instead compute $\beta_i' = \alpha_j + 1$ which gives us the solution

$$x_{ij} = max(0, \sum_j(\theta_j) * (1 + (\alpha_j - (\alpha_j + 1)))) = max(0, \sum_j(\theta_j) * 0)$$

This new variable therefore gives us the primitive solution for $x_{ij} = 0$. The way we use it is by ordering this targeting's edges by this value in ascending manner. That way we can compute β_i in the fashion specified above, however, if the β_i value is negative, we can simply set β_i as 0 and continue onto the next targeting set. If the β_i value is higher than β_i', we can eliminate the edge with lowest β_i' and recalculate the β_i again. If the β_i has a positive value lower than β_i' we can actually use it. Otherwise, we just set β_i as 0. We follow this pattern until we get the β_i matching the requirements.

This process is then repeated for each targeting set.

2.2.2.2 Computing Alphas

In a similar fashion α_j are computed by solving $\sum_i(S_i * x_{ij}) = D_j$ for each advertisement. But once again, the max function does not allow us to compute α_j straight forward with the following equation:

$$\alpha_j = D_j * \sum_i(\theta_j * \beta_i) - \sum_i(\theta_j)$$

since the result could be negative. Much like with β_i, we will select $\alpha_j' = \beta_i - 1$. The edges are then sorted in descending order by this value. Now we can again use this equation for finding α_j. If the found α_j is greater than α_j' we can select is as a result, however, limited to the max value of 1. If we do not match that requirement, we remove the edge from the α_j computation

and try again until we get a valid α_j value, or we end up with 0 edges and therefore 0 as α_j value as well.

2.2.3 Second phase

Since we have eliminated the mechanisms to deal with finishing unfinished orders, and receding demand, this phase only consists of one more round of recalculating betas.

2.2.4 Online phase

In this phase the original SHALE algorithm computes values of x_{ij} for advertisements that match the incoming impression, however, due to our architecture, we are lacking the computed zeta. Therefore, we are missing the need to compute this value online. The value is computed for each edge of the bipartite graph in the second phase of offline stage.

2.2.4.1 Computing x

For all advertisements matching a single targeting set, the sum of their x values will be less than or equal to 1. This means that there is still a possibility to not select a single advertisement, if the space is under-allocated.

2.3 PropAlloc

Compared to the SHALE algorithm, the PropAlloc algorithm is a lot more simple, and its major selling point for our application is its lesser dependency on predictions.

The research paper references two versions of this algorithm - PropAlloc and PropAlloc+ - with the latter one allowing additional parameters for setting weight to graph edges while maximizing a combination of weight and entropy matching.[1]

Since that has no added business value for our case, we will focus solely on PropAlloc, which is the same algorithm set up with above mentioned parameters in such a way that they are no longer present.

This algorithm does not have an offline stage, and does not need predictions.

2.3.1 Glossary

- C_a - ordered number of impressions
- $\epsilon \in (0, 1)$ - parameter affecting the scale by which betas are adjusted each round
- β_a - priority score for given advertisement node
- x_{ia} - priority score for given graph edge,

$$x_{ia} = \frac{\beta_a}{\sum_{a' \in N_i}(\beta_{a'})}, \forall a \in N_i$$

- $Alloc_a$ - sum of assigned allocations of given advertisement,

$$Alloc_a = \sum_{i \in N_a}(x_{ia})$$

2.3.2 Algorithm

While the original algorithm has two stages - first one for establishing beta values, and second one for the rest of the serving process - that is not possible for us due to the business requirements. The algorithm expects graph to be immutable, and thus that after $O(\frac{\log n/\epsilon}{\epsilon^2})$ rounds the betas would settle to their optimal values. Here n represents number of advertisers. In our implementation it is however possible for the edges to change nodes during the run of the algorithm, therefore we have to carry on with the first stage of establishing betas until the end.

Our implementation of the algorithm initializes all the betas to 1 at the beginning, and then starts updating them for all impressions that arrive by the following rules:

$$Alloc_a \leq \frac{C_a}{(1+\epsilon)} \implies \beta_a = (1+\epsilon) * \beta_a$$

$$Alloc_a \geq (1+\epsilon) * C_a \implies \beta_a = \frac{\beta_a}{(1+\epsilon)}$$

and for advertisements with $Alloc_a > C_a$

$$x_{ia} = \frac{C_a}{Alloc_a} * \frac{\beta_a}{\sum_{a' \in N_i}(\beta_{a'})}, \forall i \in N_a$$

until the very end of the run.

2.4 Legacy Seznam.cz solution - Advert

This algorithm is not described in any research paper, as it is a custom solution that was tailor made more than a decade ago by the research and development team at Seznam.cz. The main attractions of this algorithm are that it was created with the servers and entity structure in mind, as well as the possible volatility of ordered advertisements. That means that it is quite resistant to the supply and demand changing. At the same time however, that comes with drawbacks.

This algorithm is similar to PropAlloc by the fact that it does not need supply predictions to operate, however, that is where the similarities end.

While there are predictions calculated in the system, it is only for the purpose of entering advertisements into the system. They are only forecast to see if it should be allowed to sell given number of impressions. The online algorithm itself operates independently.

We can still separate it into an offline and online phase.

2.4.1 Offline phase

Once the advertisement is entered, the plans for it are created. This is done by spreading the impressions equally through the ordered days, recognizing if some day is a national holiday, weekend, or weekday and adjusting the percentage of available impressions that day. Similar mechanic is then used in further segmenting these plans to hourly granularity. There is a static traffic curve that was created retrospectively from averaging traffic coming into the system. This process is shown in example 2.2.

Since the algorithm was written with the consideration that some of the impressions will be marked as fraudulent[1], and that it will most likely be unable to deliver the exact ordered amount, the whole plan is also inflated by a configurable variable called quantity cushion. It is recommended this variable is set to 1.05 meaning effectively that the plan for each hour is inflated by 5% for achieving the best results.

[1] Fraudulent impressions are considered multiple hits from the same IP address, hits from the internal IP range, or artificial traffic of web crawler.

Table 2.2 Order of one advertisement for Example 2.2.

ID	Impressions	Day From	Day To
1	2600000	4	6

Table 2.3 Calendar accompanying the order for Example 2.2.

Day	Type	Coefficient
1	weekday	1
2	weekday	1
3	weekday	1
4	weekday	1
5	holiday	0.8
6	weekend	0.8
7	weekend	0.8

▶ **Example 2.2.** Normal weekday will have coefficient of 1 while the weekend or holiday might have 0.8. The plan for the day would then be created by multiplying the total ordered impressions by the fraction of current coefficient to the sum of all coefficients in the plan. The order for this example is specified in Table 2.2. To create the plans for this advertisement we need to take a look into the calendar 2.3 to know what coefficients to apply. For each day we can then apply the static traffic curve from Table 2.4. Afterwards we are left with a complete plan for this ad, shown in Table 2.5.

2.4.2 Online phase

Working with the plan created in offline phase, this phase tries to serve the specified amount of advertisements by converting the demand into a linear function. We know how many impressions to serve in an hour, so at any point in the hour, we can say how many impressions should have been served by now. While not completely accurate due to real traffic spiking during the hour instead of having a linear course, it gives us a precise plan to follow, and more importantly, tells us at any point if we are ahead or behind of said plan.

This is used in a function computing each advertisement's property called fitness. This function takes two configuration parameters. Quantity tolerance describes how far ahead of the plan we want to get in the hour, while quantity exponent affects the exponents used in this function.

2.4.2.1 Glossary

- S_a - ordered number of impressions
- S'_a - current number of total dispatched impressions
- P^b_a - total number of impressions planned to be served by the beginning of current hour
- P^e_a - total number of impressions planned to be served by the end of current hour
- P^h_a - number of impressions planned for current hour
- P^c_a - total number of impressions planned to be served at current time
- t - number of seconds passed in current hour
- Q_t - quantity tolerance

Table 2.4 Hourly coefficients accompanying the order for Example 2.2.

Hour	Coefficient
0	0.010
1	0.005
2	0.005
3	0.005
4	0.005
5	0.015
6	0.030
7	0.055
8	0.065
9	0.055
10	0.055
11	0.055
12	0.060
13	0.055
14	0.055
15	0.050
16	0.050
17	0.050
18	0.050
19	0.060
20	0.065
21	0.065
22	0.050
23	0.030

Table 2.5 Resulting plan for Example 2.2.

Day	Hour	Impressions	Day	Hour	Impressions	Day	Hour	Impressions
4	0	10000	5	0	8000	6	0	8000
4	1	5000	5	1	4000	6	1	4000
4	2	5000	5	2	4000	6	2	4000
4	3	5000	5	3	4000	6	3	4000
4	4	5000	5	4	4000	6	4	4000
4	5	15000	5	5	12000	6	5	12000
4	6	30000	5	6	24000	6	6	24000
4	7	55000	5	7	44000	6	7	44000
4	8	65000	5	8	52000	6	8	52000
4	9	55000	5	9	44000	6	9	44000
4	10	55000	5	10	44000	6	10	44000
4	11	55000	5	11	44000	6	11	44000
4	12	60000	5	12	48000	6	12	48000
4	13	55000	5	13	44000	6	13	44000
4	14	55000	5	14	44000	6	14	44000
4	15	50000	5	15	40000	6	15	40000
4	16	50000	5	16	40000	6	16	40000
4	17	50000	5	17	40000	6	17	40000
4	18	50000	5	18	40000	6	18	40000
4	19	60000	5	19	48000	6	19	48000
4	20	65000	5	20	52000	6	20	52000
4	21	65000	5	21	52000	6	21	52000
4	22	50000	5	22	40000	6	22	40000
4	23	30000	5	23	24000	6	23	24000

- Q_e - quantity exponent

The fitness is then computed for each advertisement a like this:

$$P_a^h = P_a^e - P_a^b$$

$$P_a^c = P_a^b + (\frac{P_a^b * t}{3600})$$

$$\delta = max(0, P_a^c - S_a')$$

$$fitness = max((Q_t * P_a^h)^{1-Q_e} * \delta^{Q_e}, O)$$

With the fitness established we can sort advertisements in descending order and start picking winners. This is done by going down the list from highest to lowest and picking advertisements that can be served. Due to the nature of the server, not all targeting based filtering is done before this selection process. Should this part of the process be reworked, then the selection process is just picking the highest fitness.

Chapter 3

Framework for algorithm comparison

What we have in total is 3 different algorithms, with very different approaches. SHALE, PropAlloc and Advert are all online stochastic graph matching algorithms that find their use in guaranteed advertisement. Our task is to find out which one will work the best for Seznam.cz. All these algorithms were described in detail in Chapter 2. What they all have in common is that they all need ordered advertisements, and they all need incoming requests for advertisements. This is what we need to simulate.

In this chapter we will introduce a framework that will allow to just plug in any algorithm, possibly for the future even other algorithms then the ones mentioned. This framework produces a unified output and eliminates randomness where possible, while also replicating the real advertisement serving cycle as closely as possible. We will go over the decision making process that led to choosing the language for implementation as well as the the whole design of the framework.

3.1 Choosing language

When picking the language for this project, the decision was quite simple. Our requirements are that the finished product

- needs to be easy to run, understand, and even edit - it could in the future be used by the research team at Seznam.cz for testing solutions.
- has to be able to support wide scale of file formats (CSV, JSON, Parquet, possibly others).
- does not have to be high-performance. Speed is not our priority, nor is efficiency.

For all the reasons listed above, our language of choice is Python. Python is an interpreted language with enforcement of indentation, which - some would argue - adds code readability.[8] The fact that the language is interpreted makes it a great fit for proof of concept programming where we want to often modify the code without having to wait for compilation. Another thing that makes the case for Python even more convincing is the sheer amount of libraries available from their own repository PyPI.[10] Their package manager pip is automatically configured to use this repository and makes installation of required packages really frictionless. The downside of python is that even though most of its core libraries are written in C, the programs performance is in most use cases not comparable to low-level languages. However, for our use case the performance should be sufficient.

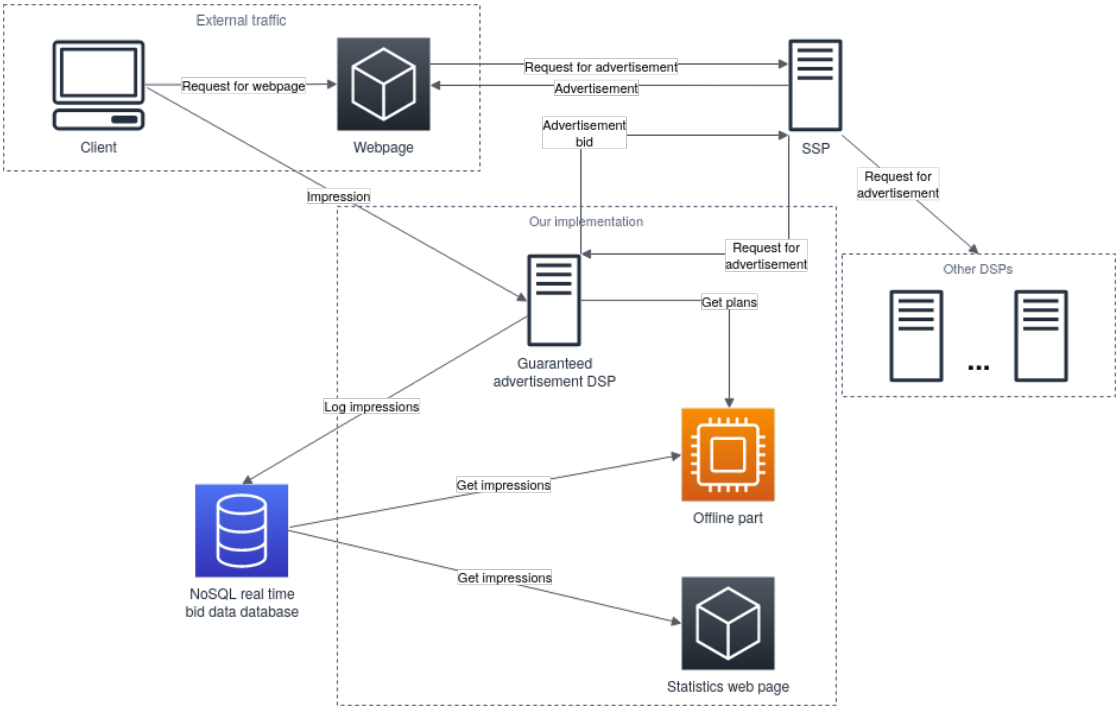

Figure 3.1 High level look at the production architecture. Client asks for a web page, and the web page asks for advertisements. SSP starts an auction and requests all DSPs to submit their candidates. From said candidates SSP picks a winner and returns it to web page, which in turn calls the winning DSP's impression URL to inform them of successfully displaying their advertisement. The page possibly also informs other DSPs about losing in the auction so that they can still accurately produce impression forecasts for their needs. The impression endpoint makes sure to write the data into our NoSQL database, so that the offline part of our algorithm - should our algorithm require one - can have access to this real time information. This data is also relayed to the statistics page where our clients can control the performance of their advertisements. All of this and more is done in under 500ms.

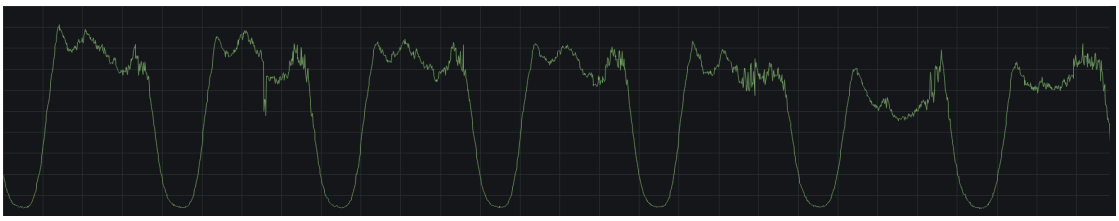

Figure 3.2 One week of traffic in real production environment. The timeline is from Monday to Sunday. The values represent requests per second.

3.2 Server and offline process

First, let us take a look at the production architecture. We will see how the requests go through the system, so that we know what to mock, and what to implement.

In Figure 3.1 we can see a clear separation of what the framework will implement, as well as all the other components that we either have to mock, or be conscientious of. We have to implement the server, the offline process, as well as a tool for visualizing resulting statistics.

While building a server would not be difficult, it would add unnecessary complexity to our benchmarks, as well as the code. It would mean we would have to run a server, while also running some process sending requests to that server. We can bypass that by simply emulating serving by reading requests from a file.

Such file will have to store a large amount of lines, and does not need to be human readable. For that reason the choice of file format is Apache Parquet, a very compact columnar storage format used in various Apache data processing frameworks.[7]

With the format selected, we need to fill the file with data. While it is possible to capture real requests in our production, we would have no input on what requests are coming in, when, and in what amount. We would also have to clear those requests with GDPR legislative. Therefore, the better solution here is to generate our own requests for bench-marking purposes, matching the real production request curve seen in Figure 3.2 to the best of our abilities.

Seeing how some algorithms need predictions, we need to have the data representing the forecast impressions closely resemble the actual requests. This opens up a possibility for us to actually generate the requests from forecast data which we will manually create. The predictions we need are going to have to be broken down to hourly granularity. Even though it is possible to create such data manually, once there is a couple of targetings, and a couple of days we want to predict, the number of data entries rises exponentially. We do however want this to be human readable, so that we can potentially find a specific row and edit it manually for simulating a specific scenario.

3.2.1 Prediction generation

With this in mind, we can create a program that will take as input

- list of targeting sets with specified daily impressions
- weekly coefficients specifying the weekly traffic curve
- hourly coefficients specifying the hourly traffic curve for each day

Using these variables, we can generate hourly plan for any amount of days from one line specifying one targeting set. As was previously specified, the output has to be human readable. Naturally so does the input, since that will be human created data for specific benchmarks. As such, we decided to use CSV. The reason for that is that the format is easily editable in any text editor of choice, but is also a recognized format by the popular spreadsheet software such as Microsoft Excel, Google Sheets and others. Also, Python offers native support of the CSV format, so working with it as far as reading and writing data goes is as simple as a few lines of code.[9]

Since CSV files are structured data files emulating table or spreadsheet we can specify the data in each entry for each file. The description of those files is provided in Tables 3.1, 3.2, and 3.3, as well as the output file in Table 3.4.

To make our requests closely resemble the real traffic, we will use actual production data to create hourly coefficients. We do this by taking the requests incoming only on the endpoint that serves advertisements, seen in Figure 3.2. We take a total number of requests received in an hour, for each hour for the duration of one week. Then we average the numbers for each hour of

Table 3.1 Data structure of daily predictions file.

Field name	Description	Type
ZoneId	Unique name of the spot on web page where advertisement will be drawn	string
TargetingId	Unique number identifying the disjunct targeting set	integer
Impressions	The number of impressions the user wants predicted on a normal weekday	integer

Table 3.2 Data structure of daily curve file. While this file is still technically in CSV format, it has only one column, and as such does not necessarily resemble a classic CSV file. However, it has other rules. The curve is meant to represent weekly traffic scaling, so it is expected to be exactly 7 lines long. One line for each day of the week, Monday to Sunday.

Field name	Description	Type
Daily coefficient	Multiplier of predicted impressions for given day in a week	double

the day, and create the coefficients by dividing the results over the sum of all averages. Doing so, we are left with coefficients in Table 3.5

When it comes to the daily coefficients, we will use the same coefficients we will be using in Advert implementation. Both hourly and daily coefficients will be used in our Advert mock configuration for creating plans so it makes sense to adopt that configuration. Advert views weekdays as having coefficient 1, and weekends or holidays as having coefficient 0.8.

The traffic curve has actually changed since the global pandemic started, and should we want the mock requests to resemble actual traffic closely, we might have to recalculate the coefficients once again. The curve is there mainly to demonstrate that given algorithm can deal with varying demand.

3.2.2 Requests generation

With the generated hourly predictions, it is possible to start generating requests. Our requirement is to be able to simulate inaccuracy of predictions, while keeping the general traffic curve. Because of the separation of generation of requests from the server mock, we can introduce random deviation from the generated prediction. Our generator will therefore take configuration parameters specifying the range in which it is allowed to sway from the predicted requests. Setting such deviation to the range from 0 to 0 will allow us to create requests shadowing predictions exactly and thus for example see if any prediction based algorithm is more dependent on accurate predictions than others.

As the input of this program, we will feed it the predictions generated by the program described in previous section. Format of this input is specified in Table 3.4. The format of the output will not be in CSV, but as earlier explained is Apache Parquet. Using Python gives us the advantage of having a library for working with Parquet at the ready. In this case the library in question is pandas, and more specifically their DataFrame implementation.[17]

Table 3.3 Data structure of hourly curve file. While this file is still technically in CSV format, it has only one column, and as such does not necessarily resemble a classic CSV file. However, it has other rules. The curve is meant to represent daily traffic scaling, so it is expected to be exactly 24 lines long. One line for each hour in a day.

Field name	Description	Type
Hourly coefficient	Multiplier of predicted impressions for given hour in a day	double

Table 3.4 Data structure of the hourly prediction file, the output of the prediction generation. While day and hour combination could be represented by one timestamp, we choose to separate them to improve the human readability. We can always later transform it to timestamp for internal use.

Field name	Description	Type
Day	Specifies the day this prediction belongs to	integer
Hour	Specifies the hour this prediction belongs to	integer
ZoneId	Unique name of the advertisement board on web page where advertisement will be rendered	string
TargetingId	Unique number identifying the disjunct targeting set	integer
Impressions	The number of impressions that are predicted for this combination of zone and targeting	integer

Table 3.5 The resulting hourly coefficients created from production requests.

Hour	Coefficient
0	0.011843233402519
1	0.00601335679856
2	0.004357084356617
3	0.004373363140177
4	0.007464276712183
5	0.015706790394239
6	0.0326317318052
7	0.054340796802974
8	0.067595979062075
9	0.058797537829832
10	0.057145142687474
11	0.058293895916321
12	0.060413383968527
13	0.0571500556121
14	0.052597896928031
15	0.050264638746891
16	0.047667498437786
17	0.047275345672189
18	0.050642395988726
19	0.058848387522874
20	0.06477167851259
21	0.062230362925086
22	0.044529477354308
23	0.025045689422719

Table 3.6 Description of Parquet file containing requests.

Field name	Description	Type
Timestamp	Custom timestamp implementation specifying the time of request	integer
ZoneId	Unique name of the spot on web page where advertisement will be drawn	string
TargetingId	Unique number identifying the disjunct targeting set	integer

As parquet is a columnar storage with strict types, we have to specify the output now even more so than with the CSV files. The resulting Parquet specification can be found in Table 3.6.

3.2.3 Server mock implementation

We have to create a general class describing a server and its interface, which can later be used to implement specific algorithms. The input will obviously read requests generated by the program from previous section, however it will also read ordered advertisements from a CSV file, which will be made available to specific implementations through class variables. That file format's specification is in Table 3.7. In this table is mentioned zone, which is the place in a website where the advertisement is rendered. Zone usually has a unique identification, as well as specification of the size of advertisements that it wants. The reasoning behind separating zone and targeting is the fact that an advertisement can only target a single zone, while there is no restrictions on other targetings. Because of that we have one parameter indicating the destination zone, and other single parameter summarizing all other possible targetings into a single identification number.

This server skeleton will keep the statistics of dispatched advertisements, as some algorithms need the feedback to work correctly, and of course as we need it to be able to compare the effectiveness of the algorithms. It will produce two output files.

One will be somewhat of a request log, a CSV file containing a request on each line, which can, much like on a real server, be used for debugging or simply identifying issues with given algorithm. Described in Table 3.8

The other file will be in JSON format and provide a more high level look on the statistics, showing the breakdown of impressions and misses down to each minute. Impression being the confirmation of seen advertisement, and miss being the confirmation of lost auction on SSP. The reason behind using the JSON format is that it very closely resembles the internal Python dictionary structure. That is a good thing because it allows us, with use of the native Python JSON library, very easily save and load the data into the program. Loading statistics created by a different run can be useful in simulating a situation where and advertiser changes targeting in the middle of its run.

The last functionality of this skeleton is simulating unrealized impressions. As was explained earlier in the thesis, the fact that the DSP returns an advertisement to SSP does not mean that it can count it as impression. The impression is only realized once the SSP selects the advertisement, the web page confirms the impressions, and the DSP clears it as not being fraudulent. This option therefore allows the user to declare a certain percentage of impressions as unrealized. The drawback here is that this introduces a randomness factor to the simulation and as such the benchmark using this option will need to be taken lightly in the evaluation.

3.2.3.1 SHALE

The implementation of SHALE algorithm, described in chapter 2, into the server mock skeleton is - much like any other algorithm - a case of instantiating the class described in previous section. We will add additional configuration file for this specific implementation, as SHALE needs to create a plan from predictions. We will therefore provide the implementation with the hourly

Table 3.7 Data structure of ordered advertisements file.

Field name	Description	Type
AdvertisementId	Unique identifier of advertisement. Is present to allow for pairing up with statistics present in a resumed simulation	integer
ZoneId	Unique name of the spot on web page where advertisement will be drawn	string
TargetingIds	One or more unique identifiers of the disjunct targeting sets	integer array
Day from	Day on which the advertisement starts	integer
Day to	Day on which the advertisement ends	integer
Hour planning	Specific hours when displaying advertisement is allowed	integer array
Impressions	Number of ordered impressions	integer

Table 3.8 Data structure of request log.

Field name	Description	Type
Timestamp	Custom timestamp implementation specifying the time of request	integer
ZoneId	Unique name of the spot on web page where advertisement will be drawn	string
TargetingId	Unique number identifying the disjunct targeting set	integer
AdvertisementId	Unique identifier of advertisement	optional integer

prediction file generated earlier in the pipeline, and specified in Table 3.4. Here we benefit greatly from the created system of generating requests from manually created predictions.

The offline phase that computes all coefficients will take place in the class initialization, preparing for the serving in online phase. The implementation will get possible candidates for selection as winners from the server mock, and based on their ID's and the targeting of given request will get their coefficients. The coefficients are then used for weighted random selection.

3.2.3.2 PropAlloc

The PropAlloc implementation itself does not need any additional information that is not already available from the server mock. Only thing needed is to initialize the default variables, and the selection mechanism itself.

While the coefficients are not created before the online phase, but rather adjusted as the serving goes on, the principle is similar to SHALE implementation: find the coefficients for given advertisements selected by server mock, and use them for weighted random selection of the winner.

3.2.3.3 Advert

As was mentioned chapter 2, the Advert algorithm uses plans created by applying daily and hourly coefficients to the ordered impressions. This process can be seen in example 2.2, and we can once again re-purpose the files created for generating predictions. Hourly and daily curve files are described in Tables 3.3 and 3.2 respectively.

The online part then has only responsibility of computing fitness for each candidate and selecting the highest ranking one as the winner.

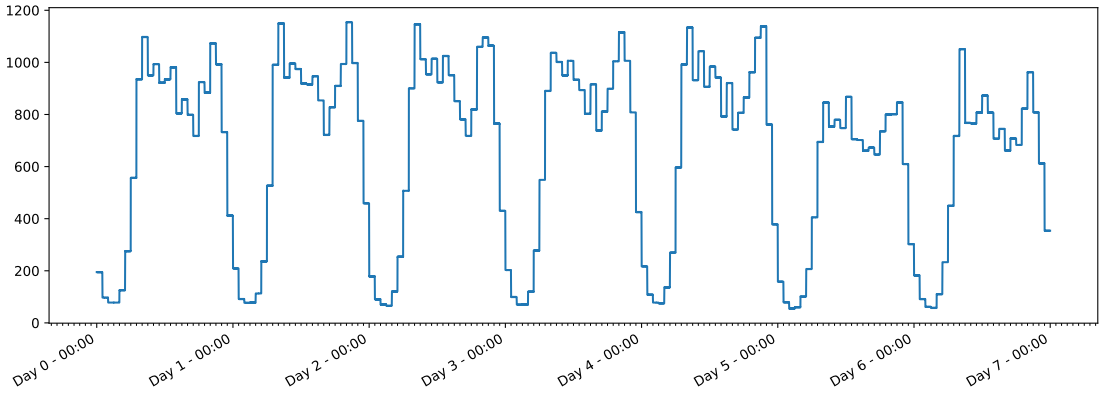

Figure 3.3 One week of traffic generated by the simulation. The timeline is from Monday to Sunday. The values represent requests per second.

3.3 Graph statistics

While the framework provides statistics in text form so that they can be analyzed in a way that suits the users, we have also implemented a way to visualize those results. That is why we have another Python program that reads the JSON statistics, and with the use of the matplotlib library visualizes them as a graph in the form of a SVG image.[14]

This program is designed in a way that allows specifying the time frame to graph, as well as setting the width and height of resulting image and the frequency of x-axis labels.

3.4 Summary

When used all together the programs create a pipeline that takes specific configuration and at the end produces graphs visualizing the benchmark results. This whole pipeline is illustrated in Figure 3.4 for better understanding.

The whole reason for creating this pipeline was to simulate the real traffic pictured in Figure 3.2. Taking a look at the result from running this pipeline with no advertisements and default parameters, pictured in Figure 3.3, we can see that the graph resembles the original quite closely.

While the steps in the graphs and the linear progression between hours does not shadow the reality exactly, the whole reasoning behind implementing the traffic curve in the first place was to examine the impact rapid supply change has on the given algorithm. For this purpose the approximation our pipeline creates will be sufficient.

We have, therefore, managed to implement a whole data pipeline for bench-marking online bipartite matching algorithms consisting of 6 different programs. Three of those are implementations of the algorithms instantiating the general server mock class, which was also implemented and designed by us to fit into the data pipeline.

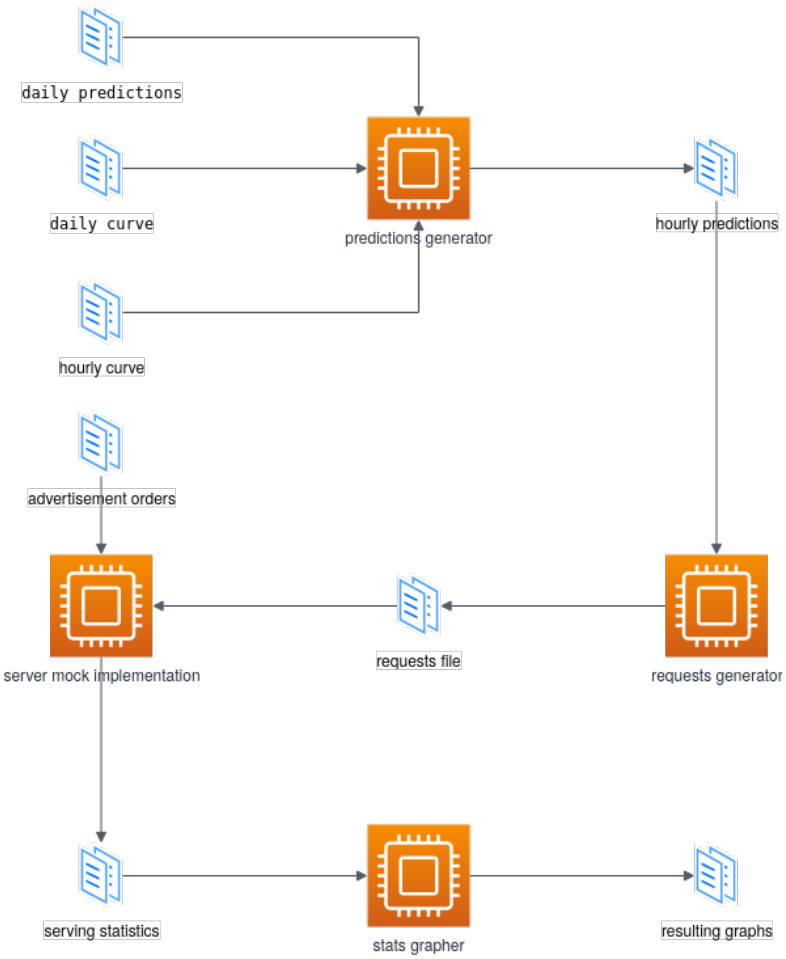

Figure 3.4 Diagram illustrating the data flow through the implemented system.

Chapter 4
Benchmarks

With the bench-marking framework implemented we can finally create specific benchmarks and see how well different algorithms combat the given scenarios. We will conduct 4 benchmarks, each one in 8 variants to help us decide which algorithms satisfy the business requirements the best.

4.1 Business requirements

First, we need to specify what are the important business qualities that we want to find in these algorithms.

The very obvious one is that all of the ordered impressions have to be served. This is a basic requirement, as when we do not deliver what was ordered, we do not get paid for it.

Next there is the importance of spreading out the impressions equally throughout the duration of the advertisement. E.g. we have an order for 7 days for a total of 700 impressions. We would ideally want to dispatch 100 impressions each day, even if the first day can service all of the 700 impressions.

Another requirement is that should there not be enough supply to meet the demand of advertisements on given disjunct targeting set, all the advertisements should then fail to deliver in a similar manner. What is not wanted is to give preference to a single advertisement to the detriment of others.

Our last requirement is that the algorithm needs to be able to deal with the fact that SSP can deny the impression in favor of another advertisement that will end up making more money for the web page as well as SSP.

With this set, we can move onto the benchmarks.

4.2 Benchmark variants

Each benchmark will be conducted in exactly 8 variants. To see the effects that unreliable predictions will have on given algorithm, we will run the benchmarks on:

- requests matching the predicted requests exactly
- requests deviating from hourly predictions by 0% - 10%
- requests deviating from hourly predictions by -10% - 0%
- requests deviating from hourly predictions by -10% - 10%

Table 4.1 Benchmark 1 advertisement order.

AdvertisementId	ZoneId	TargetingIds	Day from	Day to	Hours	Impressions
0	zone1	1	0	7	0-23	3300000

Table 4.2 Benchmark 1 daily predictions.

ZoneId	TargetingId	Impressions
zone1	1	1000000

This will help us assess if the algorithm starts under performing when supplied with more or less requests than predicted respectively. Or when the hourly predictions do not match the requests, but the total request count is still very close to predicted total.

Each of these variants will be run in two additional variants, and that is utilizing unrealized impressions. This will simulate the fact that the auction for the spot on the target web site, that our selected advertisement will enter, might be won by other advertisements. As the guaranteed advertisements are more expensive then normal advertisements, this will happen rarely, but it is possible. For this reason, we will see the effect 5% of unrealized impressions will have on each variant, as well as keeping the baseline of 0%. As we will run these two modifications for each variant mentioned above, that brings us to a total of 8 variants for each benchmark.

4.3 Benchmark 1 - a single advertisement

4.3.1 Description

This will be a very simple benchmark. Single advertisement running for 7 whole days. This does not necessarily simulate that there is only a single advertisement in the whole system, just that there is only a single one in the specified disjunct targeting set.

The order that will be used in this benchmark is described in Table 4.1, and the daily targeting predictions in Table 4.2. From the tables we can gather that the advertisement is ordered for 3 300 000 impressions, and the targeting set has a base of 1 000 000 impressions. That means that with the weekend coefficients, there will be 6 600 000 impressions available for the advertisement. Seeing how the advertisement is planned for exactly the half of the available space, no algorithm here should have trouble with delivering the ordered impressions. What we will be therefore looking for more so on this benchmark is equal distribution over time.

4.3.2 Results

The total count of delivered impressions is shown in Table 4.3, and the graph for each of the algorithms with the configuration of requests matching predictions exactly and undelivered impressions making 5% is shown in Figure 4.1.

The first business value - delivering all ordered impressions was managed by both Advert and PropAlloc algorithms. Advert has managed to deliver more impressions then originally ordered, however that is down to the settings of the quantity cushion parameter of the algorithm. It is also better business practice to deliver more impressions than ordered, if the other option is delivering less. The costs connected to returning money to the clients are higher than a possible loss created by delivering extra impressions for free. The most important takeaway here is that both PropAlloc and Advert have managed to deliver the same amount of impressions no matter the circumstances.

The algorithm struggling in this category is SHALE. While the impressions for exact requests overflow the ordered amount by only 69 impressions, that is where the success of this algorithm

Table 4.3 Benchmark 1 results.

Algorithm	Requests deviation	Unrealized impression percentage	Clean impressions
Advert	0%	0%	3464997
Advert	0%	5%	3464997
Advert	0% - 10%	0%	3464997
Advert	0% - 10%	5%	3464997
Advert	-10% - 0%	0%	3464997
Advert	-10% - 0%	5%	3464997
Advert	-10% - 10%	0%	3464997
Advert	-10% - 10%	5%	3464997
PropAlloc	0%	0%	3300000
PropAlloc	0%	5%	3300000
PropAlloc	0% - 10%	0%	3300000
PropAlloc	0% - 10%	5%	3300000
PropAlloc	-10% - 0%	0%	3300000
PropAlloc	-10% - 0%	5%	3300000
PropAlloc	-10% - 10%	0%	3300000
PropAlloc	-10% - 10%	5%	3300000
SHALE	0%	0%	3300069
SHALE	0%	5%	3134989
SHALE	0% - 10%	0%	3461771
SHALE	0% - 10%	5%	3290060
SHALE	-10% - 0%	0%	3138065
SHALE	-10% - 0%	5%	2981616
SHALE	-10% - 10%	0%	3299993
SHALE	-10% - 10%	5%	3133643

ends. Every variant where the requests deviate in any way from the predictions, SHALE deviates from ordered impressions as well.

When we check with the graphs in Figure 4.1 however, we can see that the good results coming from PropAlloc were to the detriment of the other quality that we are looking for, which is equal distribution. PropAlloc has managed to serve all the impressions in first three and a half days, returning only misses for the rest of the campaign run. This category shows much better results for SHALE, however, the clear winner of this benchmark is the legacy algorithm of Advert.

4.4 Benchmark 2 - a single advertisement

4.4.1 Description

This will be a modification of benchmark 1. Single advertisement running for 7 whole days. The difference here being that the advertisement is only running from midnight to noon each day.

The order that will be used in this benchmark is described in Table 4.4, and the daily targeting predictions in Table 4.5. From the tables we can see that the advertisement is ordered for 3 300 000 impressions, and the targeting set has a base of 1 000 000 impressions. That means that with the weekend coefficients, there will be 6 600 000 impressions available for the advertisement. Since the advertisement runs for only half a day however, and the morning period provides less impressions, no algorithm will be able to deliver the full amount ordered. In this case it is important to deliver every impressions available, so that the possible penalty is the lowest.

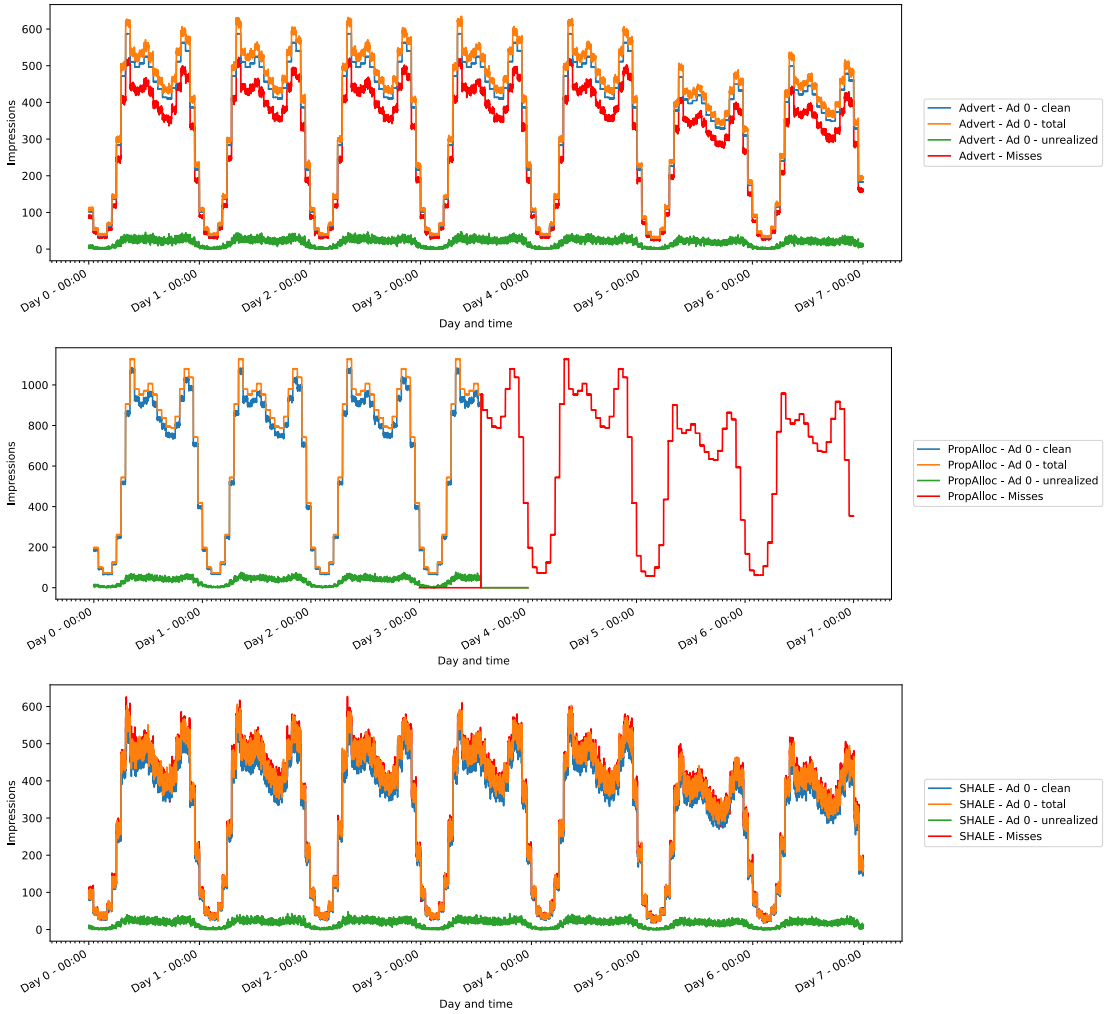

Figure 4.1 Benchmark 1 results.

Table 4.4 Benchmark 2 advertisement order.

AdvertisementId	ZoneId	TargetingIds	Day from	Day to	Hours	Impressions
0	zone1	1	0	7	0-11	3300000

Table 4.5 Benchmark 2 daily predictions.

ZoneId	TargetingId	Impressions
zone1	1	1000000

Table 4.6 Benchmark 2 results.

Algorithm	Requests deviation	Unrealized impression percentage	Clean impressions
Advert	0%	0%	2517481
Advert	0% - 10%	0%	2628953
Advert	-10% - 0%	0%	2392846
Advert	-10% - 10%	0%	2514346
PropAlloc	0%	0%	2517485
PropAlloc	0% - 10%	0%	2628957
PropAlloc	-10% - 0%	0%	2392850
PropAlloc	-10% - 10%	0%	2514350
SHALE	0%	0%	2517485
SHALE	0% - 10%	0%	2628957
SHALE	-10% - 0%	0%	2392850
SHALE	-10% - 10%	0%	2514350

Table 4.7 Benchmark 3 advertisement order.

AdvertisementId	ZoneId	TargetingIds	Day from	Day to	Hours	Impressions
0	zone1	1,4,5,7	0	7	0-23	4400000
1	zone1	2,4,6,7	0	7	0-23	4400000
2	zone1	3,5,6,7	0	7	0-23	4400000

4.4.2 Results

The total count of delivered impressions is shown in Table 4.6, and the graph for each of the algorithms with the configuration of requests matching predictions exactly is shown in Figure 4.2. For this benchmark only, the results containing unrealized impressions will be ignored. The reason for that is that the algorithms are already lacking impressions to use for delivery, so randomly lowering their supply gives no additional informational value.

From the total numbers of clean impressions we can see that Advert is possibly struggling in the first second of its serving cycle, as the fitness is 0, and therefore no advertisement is dispatched. This is something that can be taken to be improved. Other than that, SHALE and PropAlloc have managed to successfully use all impressions available. Needless to say, that all the graphs shown in Figure 4.2 look almost the same, as was always expected. It is still important to have the confirmation that all algorithms behave correctly in this scenario.

4.5 Benchmark 3 - three advertisements with overlapping targetings

4.5.1 Description

This benchmark will show how each algorithm can deal with multiple advertisements competing for the same impressions. Figure 4.3 shows the overlapping of targetings of the three ordered advertisements. Each one circle is representing one of the advertisements, and the numbers from 1-7 represent the different disjunct targeting sets. We formalize this graphic into the order specified in Table 4.7, and the daily targeting predictions in Table 4.8.

To better explain the figures in the targeting table, there is 1 300 000 impressions available for each of the 3 advertisements every standard day. However for 600 000 of those they will compete with one of the other two advertisements, and for 400 000 they will compete with both of them.

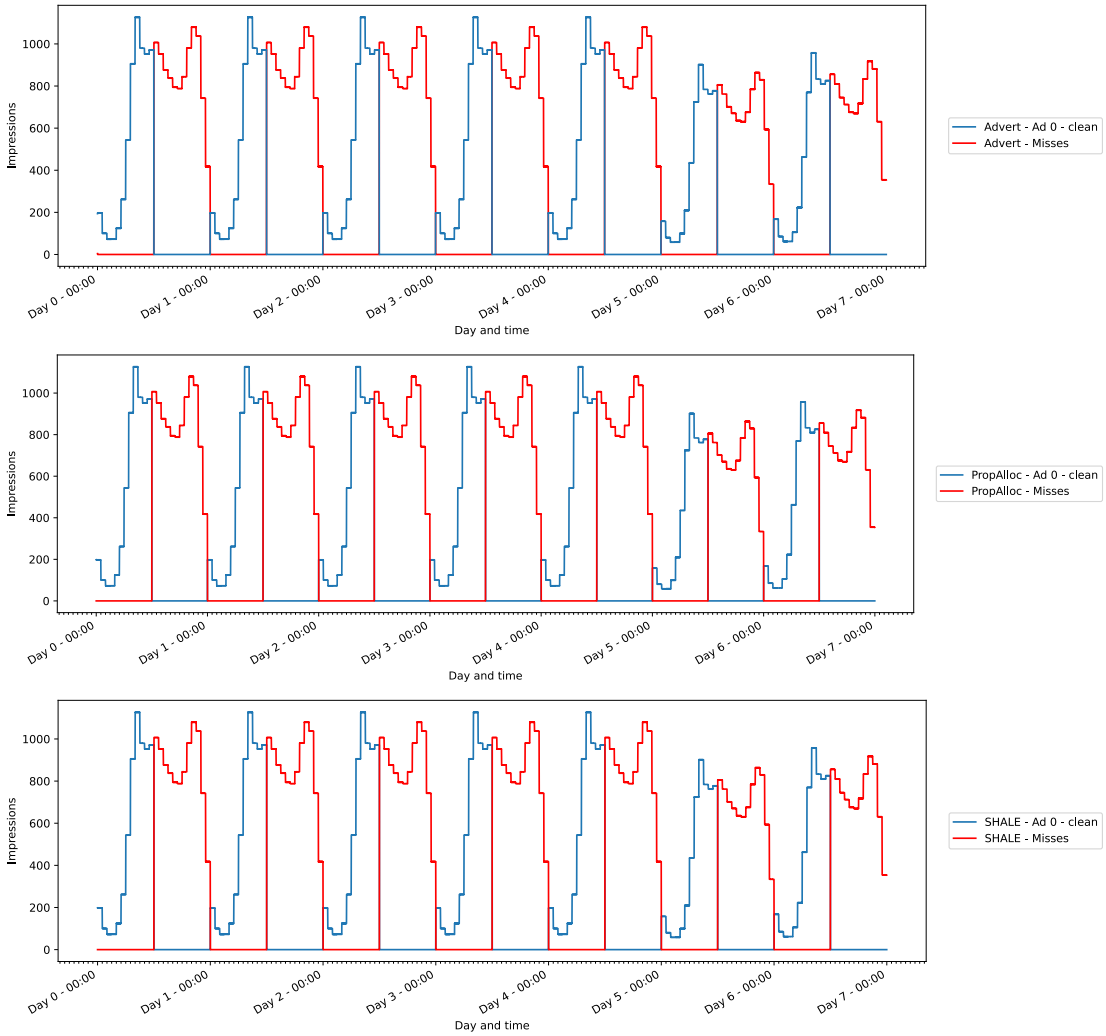

Figure 4.2 Benchmark 2 results.

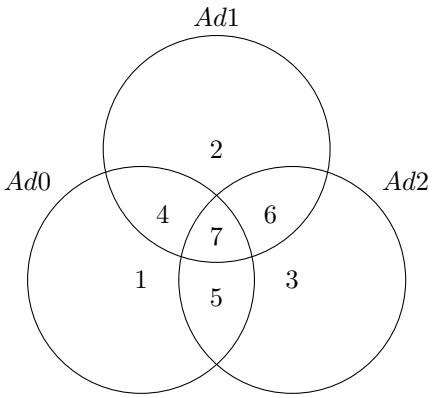

Figure 4.3 Benchmark 3 order.

Table 4.8 Benchmark 3 daily predictions.

ZoneId	TargetingId	Impressions
zone1	1	300000
zone1	2	300000
zone1	3	300000
zone1	4	300000
zone1	5	300000
zone1	6	300000
zone1	7	400000

4.5.2 Results

The total count of delivered impressions is shown in Table 4.9, and the graph for each of the algorithms with the configuration of requests matching predictions exactly and undelivered impressions making 5% is shown in Figure 4.4.

The absolute numbers in this benchmark look actually the worst for PropAlloc, because while it has managed to deliver the exact ordered amount in most cases, it has failed to deliver Ad 0 and Ad 1 while delivering Ad 3 in the case where the requests were lowered by up to 15% thus not providing enough supply to satisfy demand. This is bad practice because we do not want to play favoritism when it comes to the advertisers. This is further enforced by the graphs shown in Figure 4.4. While SHALE and Advert have all very fluent graph, the PropAlloc graph shows that during days 5 and 6 the algorithm starts updating the coefficients resulting in great volatility in the impression values for each advertisements.

SHALE is once again struggling when requests do not match the predictions, and the Advert algorithm is once again at the forefront.

4.6 Benchmark 4 - a single advertisement switching targeting

4.6.1 Description

This benchmark is a single advertisement running for 7 whole days. Or rather running for 4 days, after which the targeting is switched for the remaining 3 days.

The order is specified in Table 4.10 and the daily targeting predictions in Table 4.11. This means that there is total of 4 780 000 impressions available for an advertisement requesting 4 400 000. As the requests supply dramatically declines after 4 days, it is expected that most algorithms will fail to deliver the guaranteed sum.

4.6.2 Results

The total count of delivered impressions is shown in Table 4.12, and the graph for each of the algorithms with the configuration of requests matching predictions exactly and undelivered impressions making 5% is shown in Figure 4.5.

It is clear that due to the setup of the benchmark, the best performer here is PropAlloc. The reason behind that being the fact that the algorithm does not know anything about the supply that will come over time, so it forces all advertisements out immediately, only slowing down once the advertisement is nearing the total count. Advert all around presents better results then SHALE, but both of them fail in comparison to PropAlloc.

Table 4.9 Benchmark 3 results.

Algorithm	Requests deviation	Unrealized impression percentage	Clean impressions		
			Ad 0	Ad 1	Ad 2
Advert	0%	0%	4619995	4619995	4619994
Advert	0%	5%	4619994	4619993	4619993
Advert	0% - 10%	0%	4619995	4619995	4619995
Advert	0% - 10%	5%	4619995	4619995	4619995
Advert	-10% - 0%	0%	4616757	4616757	4616757
Advert	-10% - 0%	5%	4387812	4387811	4387811
Advert	-10% - 10%	0%	4619996	4619996	4619996
Advert	-10% - 10%	5%	4612068	4612068	4612068
PropAlloc	0%	0%	4400000	4400000	4400000
PropAlloc	0%	5%	4400000	4400000	4400000
PropAlloc	0% - 10%	0%	4400000	4400000	4400000
PropAlloc	0% - 10%	5%	4400000	4400000	4400000
PropAlloc	-10% - 0%	0%	4400000	4400000	4400000
PropAlloc	-10% - 0%	5%	4362905	4318744	4400000
PropAlloc	-10% - 10%	0%	4400000	4400000	4400000
PropAlloc	-10% - 10%	5%	4400000	4400000	4400000
SHALE	0%	0%	4401731	4399170	4400137
SHALE	0%	5%	4180239	4180125	4179472
SHALE	0% - 10%	0%	4617933	4620131	4622764
SHALE	0% - 10%	5%	4388269	4390910	4390291
SHALE	-10% - 0%	0%	4166025	4167883	4166453
SHALE	-10% - 0%	5%	3956857	3958369	3957992
SHALE	-10% - 10%	0%	4410626	4413276	4412036
SHALE	-10% - 10%	5%	4191072	4190493	4194259

Table 4.10 Benchmark 4 advertisement order.

AdvertisementId	ZoneId	TargetingIds	Day from	Day to	Hours	Impressions
0	zone1	1	0	4	0-23	4400000
AdvertisementId	ZoneId	TargetingIds	Day from	Day to	Hours	Impressions
0	zone1	2	4	7	0-23	4400000

Table 4.11 Benchmark 4 daily predictions.

ZoneId	TargetingId	Impressions
zone1	1	1000000
zone1	2	300000

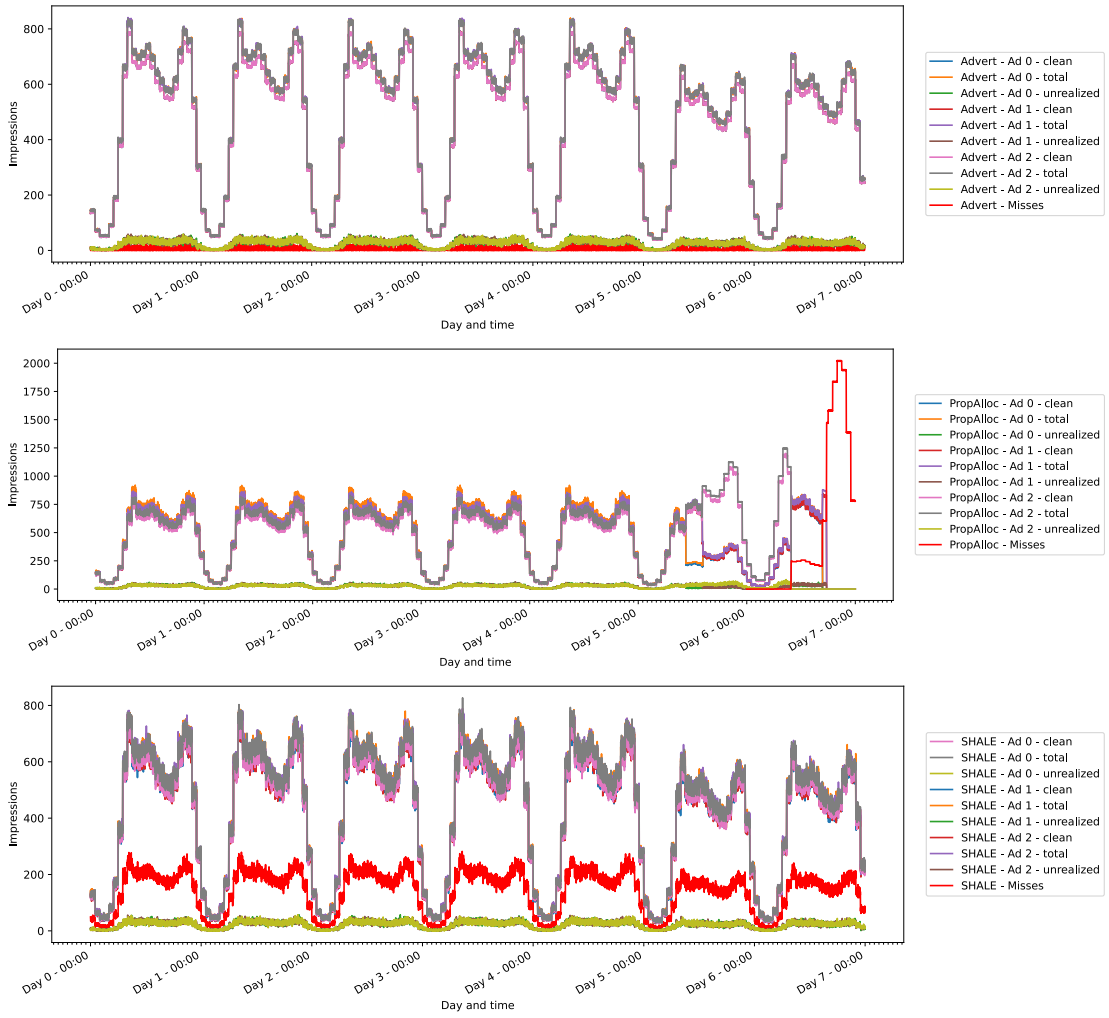

Figure 4.4 Benchmark 3 results.

Table 4.12 Benchmark 4 results.

Algorithm	Requests deviation	Unrealized impression percentage	Clean impressions
Advert	0%	0%	3574035
Advert	0%	5%	3534100
Advert	0% - 10%	0%	3614231
Advert	0% - 10%	5%	3572761
Advert	-10% - 0%	0%	3534700
Advert	-10% - 0%	5%	3497091
Advert	-10% - 10%	0%	3567889
Advert	-10% - 10%	5%	3528577
PropAlloc	0%	0%	4400000
PropAlloc	0%	5%	4400000
PropAlloc	0% - 10%	0%	4400000
PropAlloc	0% - 10%	5%	4400000
PropAlloc	-10% - 0%	0%	4400000
PropAlloc	-10% - 0%	5%	4346711
PropAlloc	-10% - 10%	0%	4400000
PropAlloc	-10% - 10%	5%	4400000
SHALE	0%	0%	3441584
SHALE	0%	5%	3269919
SHALE	0% - 10%	0%	3602674
SHALE	0% - 10%	5%	3422193
SHALE	-10% - 0%	0%	3281079
SHALE	-10% - 0%	5%	3119095
SHALE	-10% - 10%	0%	3424332
SHALE	-10% - 10%	5%	3252750

As shown in the Figure 4.5 Advert and SHALE are using the full supply once the targeting switch happens, trying to fulfill the order. PropAlloc meanwhile is done in the middle of day 6.

4.7 Summary

Throughout the benchmarks only one algorithm managed to keep up with all our business requirements - Advert. And it makes sense as the algorithm was adjusted over the years to adapt to all the scenarios that came up. It has evolved over the years to tackle the problems that came as feedback, and as such became a tailor made solution difficult to replace.

The SHALE algorithm however shows great potential in computing plans. Replacing the method creating plans in Advert with SHALE would allow for a more accurate serving as well as possible better distribution of advertisements into disjunct targeting sets.

PropAlloc matched the set business requirements the least, however, improving the algorithm by supplying hourly plans it should follow, instead of keeping multiple day plans, could provide better results.

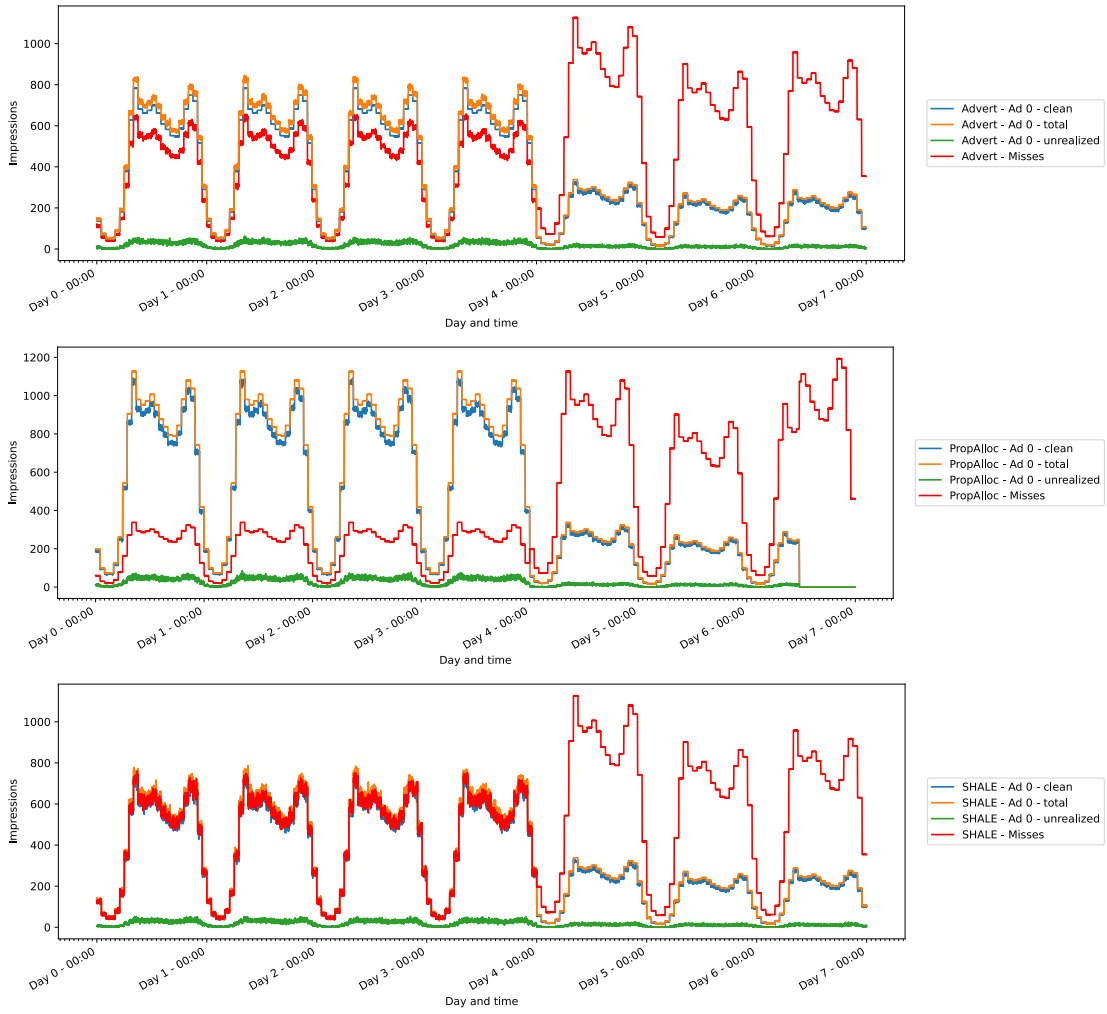

Figure 4.5 Benchmark 4 results.

Chapter 5
Conclusion

In this thesis we have familiarized ourselves with the problem that is guaranteed display advertising, the way that the advertisement makes its way to the web page visitor, and all the constraints related to generating predictions and taking part in auction. We have explored existing solutions to the problem, namely SHALE, PropAlloc and Advert, describing them in detail to see the different approaches.

To emphasize the differences between the algorithms we have implemented a bench-marking framework to compare them against each other. This consists of multiple Python scripts working in a data pipeline, producing multiple machine-readable statistics files as well as a visualization of given simulation in a graph. We have managed to successfully create data generators that simulate the request flow on production servers, which can be used in our benchmark framework.

We have bench-marked the algorithms against each other on 4 different scenarios, each one exploring some other aspect of what is required from these algorithms in production. The legacy Seznam.cz algorithm ended up as the algorithm that matched the business requirements the closest.

We have set out to understand, implement and evaluate alternatives to the current existing solution, and create a platform to compare any other algorithms for guaranteed display advertisement. That goal was achieved, however, there is still more work that can be considered.

5.1 Further work

There is definitely a case to be made to rework the architecture of the production environment to be able to accommodate for the algorithms in their full scale. The researcher feels that the need to stick to the current architecture has been a detriment to the implementation of the alternative algorithms.

As was suggested in the summary of Chapter 4 there is potential to improve the Advert algorithm by using SHALE to compute plans for the online phase of Advert. The dual constraints that SHALE operates with allow for a much better distribution than the current linear solution in Advert.

While the benchmarks are representative of the situations that the selected algorithm will be facing in production, there is always space for more edge cases, and more scenarios to be bench-marked.

Bibliography

[1] Shipra Agrawal, Vahab Mirrokni, and Morteza Zadimoghaddam. Proportional allocation: Simple, distributed, and diverse matching with high entropy, 2018. URL: http://proceedings.mlr.press/v80/agrawal18b/agrawal18b.pdf.

[2] Vijay Bharadwaj, Peiji Chen, Wenjing Ma, Chandrashekhar Nagarajan, John Tomlin, Sergei Vassilvitskii, Erik Vee, and Jian Yang. Shale: An efficient algorithm for allocation of guaranteed display advertising, 2012. arXiv:1203.3619.

[3] Peiji Chen, Wenjing Ma, Srinath Mandalapu, Chandrashekhar Nagarajan, Jayavel Shanmugasundaram, Sergei Vassilvitskii, Erik Vee, Manfai Yu, and Jason Zien. Ad serving using a compact allocation plan, 2012. arXiv:1203.3593.

[4] Ran Duan and Seth Pettie. Linear-time approximation for maximum weight matching. *Journal of the ACM*, 61(1), 2014. URL: https://web.eecs.umich.edu/~pettie/papers/ApproxMWM-JACM.pdf, doi:10.1145/2529989.

[5] Thomas E. Easterfield. A combinatorial algorithm. *Journal of the London Mathematical Society*, s1-21(3):219–226, 1946. URL: https://londmathsoc.onlinelibrary.wiley.com/doi/abs/10.1112/jlms/s1-21.3.219, doi:10.1112/jlms/s1-21.3.219.

[6] Jon Feldman, Aranyak Mehta, Vahab Mirrokni, and S. Muthukrishnan. Online stochastic matching: Beating 1-1/e, 2009. arXiv:0905.4100.

[7] Apache Software Foundation. Apache parquet. online. URL: https://parquet.apache.org/.

[8] Python Software Foundation. Python. online. URL: https://www.python.org/.

[9] Python Software Foundation. Python csv library. online. URL: https://docs.python.org/3/library/csv.html.

[10] Python Software Foundation. Python package index. online. URL: https://www.pypi.org/.

[11] Arpita Ghosh, Preston McAfee, Kishore Papineni, and Sergei Vassilvitskii. Bidding for representative allocations for display advertising, 2009. arXiv:0910.0880.

[12] Frank L. Hitchcock. The distribution of a product from several sources to numerous localities. *Journal of Mathematics and Physics*, 20(1-4):224–230, 1941. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/sapm1941201224, doi:10.1002/sapm1941201224.

[13] Dawei Huang and Lei Guo. Estimation of nonstationary armax models based on the hannan-rissanen method. *The Annals of Statistics*, 18(4), 1990. URL: https://www.jstor.org/stable/2241884.

[14] John D. Hunter. Python matplotlib library. online. URL: https://matplotlib.org/.

[15] Leonid Kantorovich. On the translocation of masses. *Management Science*, 5(1):1–4, 1958. URL: http://www.jstor.org/stable/2626967.

[16] Eugene L. Lawler. *Combinatorial Optimization: Networks And Matroids*. Holt, Rinehart & Winston, New York, 1974.

[17] Wes McKinney. Python pandas library. online. URL: https://pandas.pydata.org/docs/user_guide/dsintro.html#dataframe.

[18] Vahab S. Mirrokni, Shayan Oveis Gharan, and Morteza Zadimoghaddam. Simultaneous approximations for adversarial and stochastic online budgeted allocation, 2011. URL: https://homes.cs.washington.edu/~shayan/simultaneous.pdf.

[19] Robert L. Thorndike. The problem of classification of personnel. *Psychometrika*, 15(3):215–235, 1950. doi:10.1007/BF02289039.

[20] Erik Vee, Sergei Vassilvitskii, and Jayavel Shanmugasundaram. Optimal online assignment with forecasts, 2010. URL: http://theory.stanford.edu/~sergei/papers/ec10-lagrange.pdf.

[21] Inc. Workday. Timeseries-forecast. online. URL: https://github.com/Workday/timeseries-forecast.

Contents of attached medium