Ad Delivery Optimization

K R I S T I N A  N Y L A N D E R

Master of Science Thesis
Stockholm, Sweden 2012
Ad Delivery Optimization

K R I S T I N A N Y L A N D E R

2D1021, Master’s Thesis in Computer Science (30 ECTS credits)
Degree Progr. in Computer Science and Engineering 300 credits
Royal Institute of Technology year 2012
Supervisor at CSC was Stefan Nilsson
Examiner was Stefan Arnborg

TRITA-CSC-E 2012:017
ISRN-KTH/CSC/E--12/017--SE
ISSN-1653-5715

Royal Institute of Technology
School of Computer Science and Communication
KTH CSC
SE-100 44 Stockholm, Sweden
URL: www.kth.se/csc
Abstract

This thesis is done at Spotify, a music streaming service company. For Spotify, income from the ad market becomes more and more important. Thus, to increase revenue, under-deliveries of campaigns need to be minimized.

To do this, mathematical optimization methods have been used, together with already available software in form of a simulator for ad delivery.

There is no obvious way to express the goal function analytically, and this introduces problems. To get a value of the goal function, a simulation of the ad delivery needs to be performed. This results in that the value is noisy due to re-sampling of the user base.

Two different methods have been tried; the simple approach of only updating the priorities of the campaigns that are under-delivering, and the classic optimization method gradient descent. The hope was to be able to approximate the gradient for gradient descent, and various methods for that have been tried.

The results indicate that the simple approach worked best. The income lost could be lowered by one percentage unit in simulations. With gradient descent no improvements at all could be seen. But further tests should be done to confirm the result.
Referat

Optimering av annonsleverans

Det här examensarbetet är utfört på Spotify, som tillhandahåller en tjänst för att strömma musik. Allt eftersom Spotify växer i Europa blir inkomster från annonser mer och mer viktiga. För att öka inkomsten behöver underleveransen av annonskampanjer minskas.

Här kommer det att göras försök att minska underleveransen av kampanjer genom att använda matematiska optimeringsmetoder, tillsammans med befintlig mjukvara i form av en simulator för annonsleverans.


Med den enkla metoden har den uteblivna inkomsten minskat med en procentenhet, medan gradient descent inte har lett till synbara förbättringar. För att vara riktigt säker på att resultatet håller bör fler test göras.
Acknowledgments

I would like to thank Marcus Isaksson, my supervisor at Spotify, for taking the time to help me with this project. Without his feedback and knowledge of mathematics and statistics this would have been nearly impossible. I would also like to thank Spotify for giving me the opportunity to do this, and Erik Bernhardsson for coming up with the idea for this thesis. For proofreading I would like to thank Marie-Sofie Karlsson, she went over this report with a very red pen! Last but not least, I would like to thank Tobias Evert for his constant support and encouragement, and my parents for always believing in me.
Contents

I  The ad delivery optimization problem in short  
   1  The ad optimization problem  
      1.1  Introduction  
      1.2  Outline of the report  
      1.3  Definitions  
         1.3.1  Ads  
         1.3.2  Mathematical optimization  
      1.4  Problem definition  
      1.5  Limitations  
      1.6  Important results  
      1.7  Recommendations and future work  

II  Technical details  
   2  Introduction to optimization  
      2.1  Characteristics of optimization  
         2.1.1  Optimization with constraints  
   3  Optimization methods  
      3.1  Naive method  
      3.2  Evolutionary algorithms  
         3.2.1  How evolutionary algorithms work  
         3.2.2  Different variants  
         3.2.3  When to use an evolutionary algorithm  
      3.3  Gradient descent  
         3.3.1  Method  
         3.3.2  Step length  
      3.4  Simulated annealing  
         3.4.1  Details of simulated annealing  
         3.4.2  Properties of simulated annealing  
         3.4.3  Making simulated annealing faster  

4  Description of the Ad System
4.1 Ad chooser .................................................. 16
4.2 Advis - Ad visualization servlet ........................ 16

5 Implementation details 19
5.1 Ad-Opt from an optimization perspective ............. 19
5.2 Naive method .............................................. 21
  5.2.1 Update all that under-deliver ....................... 21
  5.2.2 Update all that under-deliver using probabilities instead of priorities .......................... 21
5.3 Gradient descent ......................................... 21
  5.3.1 Gradient calculation ................................ 21
  5.3.2 Step length .......................................... 25
  5.3.3 Normalizing ......................................... 25
5.4 Test data .................................................. 25
5.5 Finding out the maximal income ......................... 26

6 Result 30
6.1 Naive method ............................................ 30
6.2 Gradient descent ....................................... 31

7 Conclusions 35

8 Future work 36
  8.1 Naive method .......................................... 36
  8.2 Gradient descent .................................... 36
  8.3 Flow algorithm ....................................... 36
    8.3.1 Improving the flow algorithm .................. 37
    8.3.2 Further uses of the flow algorithm .......... 37
  8.4 Confirm result in other countries .................. 38

Bibliography 39
Part I

The ad delivery optimization problem in short
Chapter 1

The ad optimization problem

1.1 Introduction

Spotify is a major music streaming service in Europe. Users can sign up for the free service with ads, or pay for the unlimited or premium service, without ads. To this day, Spotify is one of the major markets for online ads in Sweden, and still growing. As the size of the ad inventory increases, inefficiencies in the ad delivery system have a larger impact on delivery.

The revenue from campaigns is based on the number of impressions a campaign delivers. Every campaign has a fixed number of impressions to deliver. Right now, the revenue is far from optimal due to under-delivery. In this thesis I will try to minimize under-deliveries of campaigns, and thus maximize the revenue. To do this I will implement several optimization methods with the help of an existing simulation system called Advis (see Section 4.2).

1.2 Outline of the report

The first part consists of a non-technical summary, which introduces the problem, presents the most important results and gives suggestions to what can be done in the future.

The second part of the report is more technical. An introduction to mathematical optimization is given in Chapter 2. After that, various optimization methods are presented in Chapter 3. Before going through how they were implemented in Chapter 5 an introduction to the ad system used at Spotify is given in Chapter 4. The results are presented more thoroughly in Chapter 6, and conclusions are made in Chapter 7. Finally, future recommendations are given in Chapter 8.
1.3 Definitions

1.3.1 Ads

**Campaign** An ad campaign consists of one or more ads that are run during a specific time period. The campaign has certain constraints on which days and hours it might be played and the kind of user it targets. There are also constraints on how often an ad may be presented to a user. The campaign also has a fixed number of impressions it should deliver.

**Impression** An impression is when an ad is played or showed to a user once.

**CPM** CPM is short for *cost per mille*, that is, the amount of money paid (by the advertiser, to Spotify) for a thousand impressions.

**Under-delivery** A campaign under-delivers if it cannot deliver all of its desired number of impressions.

**Over-delivery** A campaign over-delivers if it delivers more than its desired number of impressions.

**Priority** Priority will be used to describe the parameter that decides in what order ads should be given to the client software (from now on the term *client* will be used to denote the software used by the user). The relation between CPM and priority is that the initial value of the priority is decided by the CPM.

1.3.2 Mathematical optimization

**Goal function** The goal function is the function to maximize/minimize. In this report the goal function is the total income, and should be maximized.

1.4 Problem definition

The ad delivery system used at the moment allocates ads to users by using a greedy algorithm. This algorithm chooses ads based on their CPM. As can be seen in the simple example in Table 1.1, this is not optimal.

Instead of using CPM as priority for the ads, it would be good if priorities could be partially based on which campaigns are most at risk of being under-delivered.

The idea is to look at the revenue as a function of each campaign’s priority, and use mathematical optimization methods to find priorities that maximize the revenue. Since the campaign delivery depends on user behaviour, it is very hard to calculate beforehand. A forecasting system will be used to simulate a set of campaigns, with user data from the past, to get an approximate value of the revenue.
CHAPTER 1. THE AD OPTIMIZATION PROBLEM

Campaign A - 100,000 impressions, 20 CPM, last day is Saturday
Campaign B - 20,000 impressions, 10 CPM, last day is Monday
An average of 20,000 impressions per day

Allocation of campaigns:

<table>
<thead>
<tr>
<th>Strategy</th>
<th>Mon</th>
<th>Tue</th>
<th>Wed</th>
<th>Thu</th>
<th>Fri</th>
<th>Sat</th>
<th>Sun</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Greedy</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2. Optimal</td>
<td>B</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td></td>
</tr>
</tbody>
</table>

Result
1: 100,000 impressions · 20 CPM = 2,000
2: 100,000 impressions · 20 CPM + 20,000 impressions · 10 CPM = 2,200

Table 1.1. An example of where a greedy strategy fails. Allocation 1 allocates ads with a greedy strategy and thus choses A before B on Monday. The effect is that B will not get all its impressions delivered. Note: This example is heavily simplified compared to reality.

1.5 Limitations

The ad system contains several different ad types. This project will be limited to audio ads, which are ads played between songs. It should be possible to extend the application to contain other ads as well in the future. Different countries can easily be tested, but testing will be limited to Sweden.

1.6 Important results

Some improvements have been seen. When the priorities for the campaigns that under-deliver are updated, the lost income, relative to the optimal income, decreases with approximately one percentage unit. Different implementations of gradient descent have also been tried, but with no increase in revenue.

For a more thorough presentation and analysis, see Chapter 6 and 7.

1.7 Recommendations and future work

Before considering using this system, I would recommend more thorough tests to be done. Especially repeated tests for the same period and method, and for other countries should be done. The results here are promising, but not basis for a decision.

As mentioned in Chapter 8, an algorithm used for approximating the maximum possible income for a period, can be investigated further and maybe be used for optimization instead.
Part II

Technical details
In this part of the report, a more technical introduction to the problem will be given. Chapter 2 starts out with a general introduction to mathematical optimization. The main optimization methods considered will be described in Chapter 3. To perform the optimization, an already present forecasting system has been used. That, and the rest of the ad-system, will be presented in Chapter 4. In Chapter 5, details of the implemented optimization methods will be given. And finally, in Chapter 6, there will be a thorough presentation of the results with conclusions in Chapter 7, and recommendations for the future in Chapter 8.

If you are familiar with optimization methods, both Chapter 2 and Chapter 3 can be skipped. To understand what has been made, Chapter 5 should be read though.
Chapter 2

Introduction to optimization

Mathematical optimization is the theory of finding a global or local optimum of a function. The optimum can be a minimum or a maximum, depending on the application. The following notation is used in [NW06]:

\[
\min f(\bar{x}) \quad \text{subject to} \quad c_i(\bar{x}) = 0 \quad i \in \mathcal{E} \\
\quad d_j(\bar{x}) \geq 0 \quad j \in \mathcal{I}.
\]

Here \( f \) is the function to be optimized, the vector \( \bar{x} \) is its arguments, and \( \mathcal{E} \) and \( \mathcal{I} \) are the set of indices for the equality and inequality constraints, \( c_i \) and \( d_i \) are arbitrary constraint functions. This notation will be used throughout the rest of the report.

**Feasible set**

The feasible set is the set of all possible values of \( f(\bar{x}) \).

**State space**

The state space \( S \) is the set of all possible arguments \( \bar{x} \) to \( f \).

### 2.1 Characteristics of optimization

In this section different characteristics of optimization problems and methods will be presented.

**Discrete and continuous optimization**

In discrete optimization all arguments are integers, while in continuous optimization they are real values. This implies that for continuous problems the feasible set is infinite, while for discrete problems it is finite [NW06] (in practice). Despite this, continuous problems are often easier to handle because in most cases the functions are smooth. The goal function of discrete problems can change dramatically when moving from one point to another even though the points are considered to be close.
Global and local optimization methods

A global optimization method is a method that finds a global optimum of a function. If the optimization is a minimization, the following $\bar{x}_i$ is sought for:

$$\bar{x}_i \text{ s.t. } f(\bar{x}_i) < f(\bar{x}_j), \quad \forall i \neq j.$$ 

A global optimum means that no other value in the feasible set is lower. A local optimization method only cares about a close neighbourhood. The following is a local minimum:

$$\bar{x}_i \text{ s.t. } f(\bar{x}_i) < f(\bar{x}_j) \quad \forall j \in N$$

where $N$ is a neighbourhood around $i$. This means that there might be other minima that are lower.

Stochastic and deterministic optimization

A stochastic optimization algorithm optimizes the expected value of the function when it is not possible to fully describe it [NW06]. A stochastic optimization algorithm also uses random elements, either in the input data or in the algorithm [Wik11f]. An example of a stochastic optimization method is Simulated Annealing.

A deterministic optimization method considers the values of the goal function to be exact, and the result will thus be the same every time the method is run with the same data [NW06, Wik11f].

2.1.1 Optimization with constraints

Since I do not want priorities to just keep increasing (due to practical reasons), a constraint is added to the optimization. A simple way of normalizing the priorities is saying that the sum of all priorities should be constant:

$$\sum_{i=1}^{N} x_i = c$$

where $x_i$ is the priority for campaign $i$, $N$ is the total number of campaigns and $c$ is the sum of all campaigns’ CPM.
Chapter 3

Optimization methods

The topic of optimization methods is vast. Here a brief overview will be given of the main methods considered for this application.

Since the goal is to find an improvement, there is no requirement on the methods chosen that they need to find the global optimum. One might argue that there is no point in finding the global optimum here since the goal function will be approximated by a simulation. Thus this global optimum might not be the global optimum when the solution is used.

In this chapter four iterative methods will be described. The first method in Section 3.1 is just a naive approach to the problem. In Section 3.2 evolutionary algorithms are described. Gradient descent in Section 3.3 is a method commonly used in the past. Here it is chosen for its simplicity. Simulated annealing in Section 3.4 is an algorithm using stochasticity. It is relatively simple to implement, but it can be hard to get the parameters right for the application.

There exists many more algorithms for solving optimization problems. The ones mentioned above have been chosen because they are common, popular, easy to implement and/or has historical value. In the end, the ones I had time to implement were the naive method and gradient descent.

3.1 Naive method

The basic idea behind this method is to be quick instead of being exact. Two naive approaches to solving the problem can be:

- For every iteration, update the priority of one campaign which under-delivered, if there is no improvement, try to update another campaign.

- For every iteration, update the priority of all campaigns which under-delivered.
3.2 Evolutionary algorithms

In this section evolutionary algorithms will be presented. Evolutionary algorithms (EA) are a class of algorithms inspired by biological mechanisms. They are based on a population of parameter sets that are iteratively enhanced.

3.2.1 How evolutionary algorithms work

A simple description of how an EA works [Wei09, ES98]:

1. A set of individuals (an individual is $\bar{x}$) is chosen as the initial population

2. Evaluate the fitness of the population. The fitness is a value of how “good” an individual is.

3. Select individuals from the population based on their fitness value.

4. Create a new generation using crossovers/recombination (combining parts of different individuals) and/or mutations (changing single parameters of an individual).

5. Repeat from 2 until a good enough solution is found.

In the selection step, candidates can be selected not only based on their fitness but also their age (the number of iterations they have been present) [ES98]. Exactly how the individuals are chosen differs from implementation to implementation.

3.2.2 Different variants

There exists several different kinds of evolutionary algorithms. They differ in how the state space they operate on is defined and what they are trying to evolve. The following are common variants:

**Genetic Algorithm** Uses bit strings as state space [Wei09, ES98]. Favors recombination over mutation [Whi01].

**Evolution Strategies** Uses real vectors as state space [Wei09, ES98]. Favors mutation over recombination and uses quite small populations (1-20 individuals) [Whi01].

**Genetic Programming** Uses trees as state space [Wei09, ES98]. Is used in machine learning to optimize a population of computer programs [Wik11b] and automatic programming [Whi01]
3.3. GRADIENT DESCENT

3.2.3 When to use an evolutionary algorithm

According to Whitley [Whi01] EAs are known as weak methods. A weak method is a method that does not use domain specific knowledge. Therefore there are a lot of other optimization methods that might give a better result when domain specific knowledge can be used.

EAs are on the other hand well suited for parallelization since a lot of individuals can be evaluated in parallel.

A problem with EA in this application is the population size. If a large population of solutions is used, more individuals needs to be evaluated in every step and fewer steps can be taken due to time constraints.

3.3 Gradient descent

Gradient descent (GD), also called steepest descent, is considered a simple and reliable method by Güler [Gül10]. It is a globally convergent algorithm, meaning that the solution will reach a local minimum from an arbitrary initial position. Though its convergence is slow [Gül10], and there is no guarantee that the global minimum will be found, it might be a local one. Which minimum it finds is highly dependent on the initial position [Kel99]. An illustration of the method can be seen in Fig. 3.1.

3.3.1 Method

The general idea is to move in the direction where the gradient decreases, to find a minimum as soon as possible. How much to move is decided by a line search method.
3.3.2 Step length

An important part of GD is choosing the right step length for updating the solution. This is done by the line search method. An improperly chosen step length can result in a method that does not converge [Gül10].

**Exact minimization rule** When using the exact minimization rule the step length $t_k$ is chosen such that

$$f(x_k + t_k d_k) \leq f(x_k + t d_k) \quad \forall t \geq 0.$$ 

Here $x_k$ and $d_k$ are the position and gradient before the k:th step. This means that the next value will be the minimum on a line from $x_k$ in the direction of $d_k$. To find this $t_k$ can be impossible and/or very expensive, as can easily be understood.

Another version is the **Limited minimization rule**. Instead of requiring $t_k$ such that the minimum is reached, $t_k$ is only searched for in some range $0 \leq t_k \leq s$.

**Constant step-length rule** With the Constant step-length rule, $t$ is a constant predetermined value.

**Armijo’s rule** The idea with Armijo’s rule is that the decrease in function value should be “sufficiently large” when a step is taken. This is expressed by the following equations [Gül10]:

$$t = \beta^m, \quad \beta \in (0, 1), \quad m \in \mathbb{N}$$

$$f(x_{curr} - t\nabla f) - f_{curr} < -\alpha t \|\nabla f\|^2,$$

where $t$ is the step length and $\alpha \in (0, 1)$ is a parameter often chosen as $\alpha = 10^{-4}$ according to Kelley [Kel99]. The idea is to find the smallest possible $m$ such that the equation hold, which means that the improvement gotten with the step length $t$, is within a factor $\alpha$ of the predicted improvement. The first equation is mainly there to guide when choosing a suitable $t$: a $\beta$ is chosen, then larger and larger values of $m$ are tried until the equation holds. For more details about the Armijo rule see [Kel99, Gül10].

3.4 Simulated annealing

Simulated annealing (SA) is a method inspired by the process of annealing in metallurgy. Annealing is a process consisting of heating and cooling a material to improve its properties. During the annealing process the atoms move around, through states of higher internal energy, until they finally rests in the state with the (hopefully) lowest energy.

SA is a stochastic method that closely tries to imitate the physical process. An initial temperature is set from the beginning, and an annealing schedule describes how the temperature decreases. The acceptance probability function decides if a new
3.4. SIMULATED ANNEALING

state is accepted or not, and new states are generated by the candidate generator. In the beginning when the temperature is higher, uphill moves will be permitted. But as the temperature decreases they will become more and more rare.

3.4.1 Details of simulated annealing

State space - The state space $S$ consists of all possible states for the arguments $\bar{x}$ to the function $f$.

Goal function: $f()$ - The goal function is the function to optimize.

Candidate generator: $\text{neighbor}()$ - The candidate generator function generates the new states. A heuristic used as a general rule [Wik11e] tries to generate new states with an energy that is similar to the current state’s energy. This tends to exclude both exceptionally good and exceptionally bad solutions. Exactly how the candidate states are generated differs a lot from application to application.

Acceptance probability function: $P(\Delta f)$ - This describes the probability that a new state is accepted given the old state. The original function used is the Bolzmann probability factor:

$$P(\Delta f) = \begin{cases} e^{-\frac{\Delta f}{k_b T}} & \text{if } \Delta f > 0 \\ 1 & \text{otherwise} \end{cases}$$

where $k_b$ is the Bolzmann constant, $\Delta f = f_{k+1} - f_k$ and $T$ is the temperature.

Annealing schedule: $T(k)$ - The annealing schedule is a description of how the temperature should decrease. It is important to choose this wisely, as can be seen in Section 3.4.2.

Initial temperature, $T_0$ - The initial temperature is the temperature in the beginning of the annealing. It can affect the result and properties greatly, which can be seen in Section 3.4.3.

3.4.2 Properties of simulated annealing

Guarantee of global optimum SA will converge to the global optimum if $t \to \infty$ iterations are performed and the correct annealing schedule is used [Wei09]. The annealing schedule $T(k)$ should not be faster than

$$T(k) = \frac{T_0}{\log k},$$

if $T_0$ is “high enough” [Ing93]. Otherwise it is turned into a simulated quenching (see Section 3.4.3) algorithm instead, and convergence is not guaranteed.
Disadvantages

- If a logarithmic annealing schedule is used, so that convergence is guaranteed, SA can be a very slow method [Wei09, SGSBT09].
- Adjusting the parameters of the SA for a specific problem is practically impossible [Ing93, SGSBT09].
- SA loses the guarantee of convergence to a global optimum if an incorrect annealing schedule is used [Ing93].

Advantages Some advantages to SA according to Ingber [Ing93]:

- Easy to implement compared to other non-linear optimization methods.
- Finding the global optimum is guaranteed (in theory) if a correct annealing schedule is used (but very hard to do in practice).
- It does not matter if the goal function is non-linear, discontinuous or stochastic. It will work anyway.
- Arbitrary boundary conditions and constraints are no problem for the method.

3.4.3 Making simulated annealing faster

There exists many techniques for making SA perform better. Some of them are ergodic, meaning that they still have the statistic guarantee of finding the optimum - others lose that guarantee.

Simulated quenching

When the annealing schedule is chosen to be faster, the guarantee of finding a global optimum disappears and SA turns into simulated quenching (SQ) instead.

The reason for choosing a faster schedule is performance. The loss of the ergodic property is not always a large problem. Reaching that optimum can take a long time, and sometimes “good enough” is sufficient.

Choosing the initial temperature

Shakouri et. al. [SGSBT09] have investigated the choice of initial temperature in SA. A common strategy they mention is setting $T_0$ high enough that almost all states are possible in the beginning. That way the risk of getting stuck in a local optimum is lower.

Instead of doing this, they propose to set $T_0$ much lower, maybe low enough that the rate between acceptable states and the total number of tried states is 10% of what it is for higher temperatures (called a mushy state). A higher initial temperature means that a slower annealing schedule must be used since the “atoms”
3.4. SIMULATED ANNEALING

need time to arrange themselves into a lower energy state. With a lower initial temperature the annealing schedule can be faster and thus the SA becomes faster.

To be able to start with a lower initial temperature from this mushy state the SA needs to be initialized by some other method to lower the energy.
Chapter 4

Description of the Ad System

This chapter gives a brief introduction to the ad system used at Spotify, and Advis, the simulator used to forecast ad delivery. Both are already present systems at Spotify.

The Ad Chooser is a backend system that is responsible for allocating ads to the client (the software used by the user to listen to Spotify). The client then shows ads to the user.

4.1 Ad chooser

This is a simplified view of how the Ad Chooser and the client interacts with each other.

The Ad Chooser is responsible for allocating ads to the client. In Fig. 4.1 the ad request process is described with a sequence diagram. The client requests ads from the Ad Chooser and supplies a list of ads that have been played recently and should be avoided. The Ad Chooser compiles a list of ads that match the constraints (i.e. female/male, age) given by the client, and sends them back to the client. In this list each ad has a priority, which is set from its CPM, with some modifications.

When it is time for an ad break, the client goes through this list, from the ad with the highest priority to the one with the lowest priority. Fig. 4.2 contains an illustration of this. For every ad the client goes through, it checks that the ad has not expired and that it still fulfills the rules set up for that campaign. If it does, that ad is chosen. If it does not, the next ad is considered. The ad chosen might or might not be the optimal choice. In the figure, ad nr. 4 has a lower priority, but that campaign ends tomorrow, so maybe that one would be the optimal ad to choose.

4.2 Advis - Ad visualization servlet

The Advis system was created to be able to forecast the result of a campaign. Given a campaign, it simulates that campaign together with other campaigns active during
4.2. ADVIS - AD VISUALIZATION SERVLET

Figure 4.1. UML sequence diagram for when the client requests ads from the Ad Chooser.

Advis samples a given number of users from the total user base for the past four weeks. For every user, the user’s usage patterns is fetched for that period. To decrease the amount of data only the usage pattern for a week is saved. For the other weeks, the only thing saved is if the user was active or not. When the system wants to know when the user was listening a certain day, the activity that day is checked. If the user was active that day, the listening data is fetched from the same time period. It returns the number of impressions delivered by your campaign, the number delivered by campaigns with higher CPM and campaigns with lower CPM, among other things.

Figure 4.2. Time for an ad break in the client. The client goes through the list of ads in descending order of priority. For every ad, it checks that the ad has not expired and that the ad still fulfills the rules set up for the campaign. If the ad is ok, it is chosen.
the corresponding day of the saved week. The idea is that the day of week is what matters most when it comes to usage patterns.

The campaigns are fetched from the campaign database. All campaigns that run during the simulated time period are chosen, the rest are discarded.

The simulation is carried out in smaller steps. The current time is increased by a fixed amount, and then ads are played for all users in that interval. For every user, the overlapping activity-events are processed, and “ads” are played. For every ad-break, all the campaigns matching that user that week are processed, and the one with the highest priority is chosen (of the currently active ads).

To decrease the noise in the results, a change to the user sampling was tried. Instead of sampling 10,000 users, care was taken to sample users until every campaign had 50 matching users. After that, more users were sampled, until a total of 10,000 users were reached. To not introduce bias, for every set of users, a factor is saved to remember how many “real” user the set represents.
Chapter 5

Implementation details

In this chapter, details of the implemented methods will be described. Different versions of the Naive method is described in Section 5.2, and in Section 5.3, Gradient Descent is discussed. But first some properties of this optimization problem will be discussed.

5.1 Ad-Opt from an optimization perspective

Here, five properties, or problems, with this optimization problem, are discussed. Since the goal function is approximated with a simulation, the resulting value is both noisy and expensive to calculate. The inherent properties of the problem also makes it discontinuous, and organizational issues complicate things since all information is not available when needed.

Noise

Since the user base is re-sampled every iteration of the optimization, two evaluations with the same arguments will not return the same value. If the user base was the same for all iterations, the result would be much less noisy. On the other hand, the result would also be a lot less general, since only a small sample of the users would be used. Introducing a bias like that is not an option, thus the number of users has been increased instead. In the beginning, a sample of 5,000 users was used; now a sample of 10,000 users is used. If more users were chosen, the simulation would take too long.

Expensive function

The goal function is approximated by quite an expensive simulation. The time complexity for the simulation is $O(nmk)$, where $n$ is the number of users sampled, $m$ is the number of campaigns (approx. 400 – 700), and $k$ is the number of smaller intervals the simulated range is divided in (3h intervals mostly). A six month simulation with a sample of 10,000 users can take up to seven minutes to perform.
Most of the time is spent loading user data that is needed for the simulation. This, in combination with the wish to perform the optimization on a daily basis, limits what methods that can be used.

**Discontinuous function**

As can be seen in Fig. 5.1, the number of impressions a campaign gets (and thus the revenue) increases suddenly when the campaign gets a certain priority. The same behaviour can be seen when looking at the total revenue for all campaigns while changing the priority of a single campaign. This is because the only thing that matters is the order of the campaigns, with respect to priority. The exact value of the priorities are irrelevant.

![Figure 5.1](image_url)  
*Figure 5.1.* Plot describing how the number of impressions changes with the change in priority, for a campaign.

**Dimensionality**

The dimensions of this optimization problem are the number of campaigns that are active in the simulated time range. There can be over 400 campaigns for a six month simulation. Due to this fact, methods that depend on sampling are unsuitable here.

**Lack of information**

Another problem for the optimization is the lack of information about campaigns. Most of the campaign data is entered into the database the day the campaign starts. Because of this, it is hard to run the optimization on a long period into the future.
5.2 Naive method

5.2.1 Update all that under-deliver

The first approach tried was to update the priority of all campaigns that under-deliver. All under-delivering campaigns are updated with a constant amount.

The idea of updating one campaign at a time was discarded since it would take a lot of evaluations of the goal function, and thus a very long time.

5.2.2 Update all that under-deliver using probabilities instead of priorities

For reasons explained in Section 5.3.1, the priorities were exchanged for probabilities. This means that instead of trying to take the ad with the highest priority, the one with the highest priority has the highest probability of being chosen. The probability is calculated as follows:

\[ P(\text{Campaign } i \text{ wins } h) = \frac{w_i}{W_h}, \]

\( w_i \) is the priority for campaign \( i \) and \( W_h \) is the total weight for all the campaigns competing about \( h \) (one impression).

5.3 Gradient descent

There are two parts to GD that are important here. First the gradient should be calculated, and then the step length needs to be chosen.

5.3.1 Gradient calculation

The value to optimize is the total revenue. With that in mind, the function to optimize will look like this

\[ y(\bar{x}) = \text{total revenue}, \quad \bar{x} = (x_1, \ldots, x_n), \quad x_i > 0 \]

where \( x_i \) is the priority for campaign \( i \). To do this with GD, the gradient,

\[ \nabla y = \left( \frac{\partial y}{\partial x_1}, \ldots, \frac{\partial y}{\partial x_n} \right) \]

needs to be calculated. Since this problem cannot be expressed analytically, the gradient has to be approximated.

Gradient based on the expected revenue

One idea is to, for every impression that a campaign did not get (due to someone else having a higher priority), save the priority of the winning campaign if it is
within a certain amount $\Delta x$. With that, the expected revenue for a priority $P_i$ and campaign $i$ ($E_i[P_i]$) is calculated. The gradient then becomes

$$\nabla y = \left( \frac{\partial y}{\partial x_1}, \ldots, \frac{\partial y}{\partial x_n} \right)$$

with

$$\frac{\partial y}{\partial x_i} \approx \frac{E_i[P_i + \Delta x] - E_i[P_i]}{\Delta x},$$

where $N$ is the number of campaigns, $\Delta x$ is the amount added to the priority (decided experimentally), and $E_i[P_i + \Delta x]$ is the number of impressions campaign $i$ could get with priority $P_i + \Delta x$. The problem here, though, is that with the discontinuous goal function, $E_i[P_i + \Delta x] - E_i[P_i]$ can be zero since $\Delta x$ is not enough to make the campaign “beat” another campaign to an impression. To get rid of this problem, noise can be added to the goal function, to make it more smooth. By adding a normal distributed noise with $\mu = \text{prio}$ and $\sigma^2 \approx 10$ (around 10% of the priority), changes in priority should make the changes in revenue less sudden, as shown in Fig. 5.2. Every time the priority of a campaign is accessed, the random noise is added. Now $E_i[P_i]$ can be calculated in a different way.

For every impression that a campaign gets or misses, the campaign that won that impression is saved. If it was the campaign itself that won the impression, number two is saved instead.

Here we assume that $C_i$ and $C_j$ are two independent normal distributed random variables. $C_i$ is the random variable that describes the priority for campaign $i$ (the one we’re interested in), and $C_j$ describes the priority for another campaign.

Now we can form a new random variable $X_j = C_i - C_j$. Since $C_i$ and $C_j$ are independent normal distributed, $X_j$ will also be normal distributed with $\mu_{X_j} = \mu_{C_i} - \mu_{C_j}$ and $\sigma^2_{X_j} = \sigma^2_{C_i} + \sigma^2_{C_j}$. To simplify the calculations I only consider
5.3. GRADIENT DESCENT

two campaigns competing with each other here, but in reality the number is far greater. If \( X_j > 0 \), campaign \( C_i \) has a greater priority than \( C_j \) and wins that impression.

With the cumulative distribution function for the normal distribution:

\[
F(x, \mu, \sigma) = \phi \left( \frac{x - \mu}{\sigma} \right) = \frac{1}{2} \left[ 1 - \text{erf} \left( \frac{x - \mu}{\sigma \sqrt{2}} \right) \right]
\]

where \( \phi(x) \) is the cumulative distribution function for the standard normal distribution, the expected number of impressions for campaign \( C_i \) can now be calculated as

\[
E_i[P_i] = \sum_{j=1, j \neq i}^{N} P(X_j > 0) \cdot a_j = \sum_{j=1, j \neq i}^{N} (1 - F(0, \mu_{X_j}, \sigma_{X_j})) \cdot a_j
\]

where \( a_j \) is the number of times campaign \( j \) won an impression over campaign \( i \) (or ended up as number two), and erf is the Gauss error function[Wiki11a]. To get a value of this, \( F(0, \mu, \sigma) \) needs to be calculated. This is not trivial, since it contains the error function, but it can be approximated[Wiki11c]. We also need to calculate \( E_i[P_i + \Delta x] \), this is done by defining \( X_j' = (C_i + \Delta x) - C_j \) and consequently \( \mu_{X_j'} = \mu_{C_i} + \Delta x - \mu_{C_j} \) and \( \sigma_{X_j'}^2 = \sigma_{X_j}^2 \).

\[
E_i[P_i + \Delta x] = \sum_{j=1, j \neq i}^{N} P(X_j' > 0) \cdot a_j = \sum_{j=1, j \neq i}^{N} (1 - F(0, \mu_{X_j'}, \sigma_{X_j'})) \cdot a_j
\]

Now when we have \( E_i[P_i] \) and \( E_i[P_i + \Delta x] \), the gradient can be approximated:

\[
\frac{\partial y}{\partial x_i} \approx \frac{E_i[P_i + \Delta x] - E_i[P_i]}{\Delta x}
\]

This method is very inaccurate, since no consideration is taken to the fact that more campaigns can influence the current one. Neither is any consideration taken to the fact that ads can only be played so often, and given to the same user a certain amount of times.

This method will never get a negative partial derivative, since the loss for other campaigns are not calculated.

Gradient calculation when using probabilities instead of priorities

Calculating the exact expected revenue is far from easy when using priorities. Here, the priorities have been exchanged for probabilities in an attempt to simplify the calculations. As was explained in Section 5.2.2, the campaign with the highest priority has the highest probability of being chosen.

To calculate the expected revenue, the set \( \mathcal{H} \) is defined as “all requests for an ad from the client”. Then the following becomes the expected number of impressions
for one campaign:

\[ H = \{ \text{all ad requests} \} \]

\[ Y_i = \text{number of impressions for campaign } p \]

\[ E_i[Y_i]_{org} = \sum_{h \in H} P(\text{Campaign } i \text{ wins } h) = \sum_{h \in H} \frac{w_i}{W_h}. \]

\( w_i \) is the weight for campaign \( i \), and \( W_h \) is the total weight for all the campaigns in \( h \). \( E_i[Y_i]_{org} \) now represents the expected number of impressions that campaign \( i \) would get with its original priority.

If we sum over all campaigns, and multiply with each campaign’s CPM, we get the approximated total expected income with original priorities:

\[ f_{org}(\bar{w}) \approx \sum_{i \in C} E_i[Y_i]_{org} \cdot cpm_i = \text{total income}. \]

To approximate the gradient, we need the total expected income with changed priorities as well. For this, the following can be defined:

\[ E_i[Y_i]_{inc} = \text{Expected impressions for campaign } i \text{ when priority for campaign } i \text{ is increased} \]

\[ E_i[Y_i]_{dec} = \text{Expected impressions for campaign } i \text{ when the priority for another campaign is increased} \]

We can calculate them like this

\[ E_i[Y_i]_{inc} = \sum_{h \in H} \frac{w_i + \Delta x}{W_h + \Delta x} \]

\[ E_i[Y_i]_{dec} = \sum_{h \in H} \frac{w_i}{W_h + \Delta x}. \]

To calculate \( \frac{\partial y}{\partial x_i} \), we need to know the change in income when the priority of campaign \( i \) is changed:

\[ \frac{\partial y}{\partial x_i} \approx \frac{f_{new}(\bar{w}) - f_{org}(\bar{w})}{\Delta x} \]

where \( f_{new} \) is the approximated total income when the priority for campaign \( i \) is increased with \( \Delta x \):

\[ f_{new}(\bar{w}) \approx \left( \sum_{j=1, j \neq i}^{N} E_j[Y_j]_{dec} \cdot cpm_j \right) + E_i[Y_i]_{inc} \cdot cpm_i \]

and \( N \) is the total number of campaigns.

How other campaigns are affected if the priority of one campaign is increased is actually taken into account here. The probability that other campaigns would get an impression, if the probability of one campaign is increased, decreases, and thus their total number of expected impressions decreases.

24
Problems with this method is that even though how other campaigns affects the goal function when the priority of one is increased, it’s still only an approximation of the effect. All other campaigns are considered, even though only a few of them might be competing with the current one.

5.3.2 Step length

The step length used is the constant step length in Section 3.3.2.

The Armijo rule, and the exact minimization rule from Section 3.3.2 are not considered, because having to evaluate the function to find a proper step size is not feasible here.

A constant step length will be used in the beginning. Exactly what the constant step length should be, will be determined experimentally.

A very long step length might result in that some campaigns are given a too high priority and take too many impressions from other campaigns. A step length that is very short might result in that the optimization becomes too slow.

5.3.3 Normalizing

To keep the priorities of some campaigns from hitting the roof the priorities need to be normalized. This will be done by adding a simple constraint to the optimization, as described in Section 2.1.1.

This is enforced by multiplying all priorities by a normalizing constant $C$:

$$w_i = w_i \cdot \frac{C}{\sum_i w_i}.$$

This will not affect the result of the goal function, since the relative order of the priorities will be the same.

5.4 Test data

Test data has been divided into four sets. Dataset 1 (2010-01-01 – 2010-07-01) has been used as day-to-day test data. Occasionally dataset 2 (2010-07-01 – 2011-01-01) has been used to check if results are consistent. When the algorithms were considered to be ready, two datasets from 2011 have been used to confirm the results. The reason for dividing 2011 into two datasets, is the functionality introduced in Spotify 2011-05-01. From that date, all users using the free service have a hard cap on how many times they can listen to a song, and how much they can listen every month. This most probably introduces a different user behaviour, and it will be harder to allocate ads since the amount of music played will probably be less.

The tests are done using 10,000 users in Sweden.
5.5 Finding out the maximal income

Maybe it is not possible to get a better income than what was achieved in the initial tests. This would of course be great to confirm. But how do you do such a thing?

What needed to be done was to match the users’ listening time to campaigns. Matching problems can be solved by reducing the problem to a flow problem, and this was done here. The flow in this graph would consist of impressions. But since the revenue should be maximized, the algorithm should consider that. The max flow algorithm [GT02] cannot be used, since that would only find the matching that results in the maximum number of impressions. Instead, the minimum-cost flow algorithm [GT02] will be used.

Create graph

A graph was created from the users and campaigns. The flow should go from the source, through the user, to a campaign, and then to the target. One unit of flow means that the user listens to the campaign once, as can be seen in Fig. 5.3. Here we have one node for every user, and one node for every campaign. There are the following edges:

- From the source to every user, with cost 0 and capacity equal to the maximum impressions for that user.
- From every user to every campaign the user matches, with cost 0 and capacity inf.
- From every campaign to the target, with cost $cpm_{\text{max}} - cpm$ and capacity equal to the desired number of impressions for that campaign.

![Figure 5.3. The basic layout of the flow graph with only one user and one campaign.](image)
The capacity is on top of the edges, and the cost, associated with the flow of one unit through that edge, is below.

Since the delivery of campaigns should be evenly distributed (one campaign should not deliver all its impressions in one day if it’s active several days), all user nodes and campaign nodes are divided per day, and we now get the graph pictured in Fig. 5.4. Instead of one node for every user, there will now be one node for every user and day, and one node for every campaign and day. The edges will be the following:

- From the source to every user and day, with cost 0 and capacity equal to the maximum impressions for that user, divided by the total number of days.
5.5. FINDING OUT THE MAXIMAL INCOME

- From every user and day, to every campaign and day the user matches, with cost 0 and capacity inf.

- From every campaign and day, to a node representing the whole campaign. The cost is 0 and the capacity is equal to the desired number of impressions for that campaign, divided by the total number of days.

- From every campaign to the target, with cost $cpm_{\text{max}} - cpm$ and capacity equal to the desired number of impressions for that campaign.

![Figure 5.4. The layout of the graph after dividing campaign and user nodes per day. Capacity of the edges is on top, cost is below.](image)

To get an as accurate value as possible, we want to have as many users as possible in the graph. This would however create a huge graph. To decrease the size of the graph users will be clustered together. In Fig. 5.5, there is an example of how the graph can look before and after user clustering. The only difference for the nodes, is that the user and day-nodes will turn into usergroup and day-nodes. The edges will be the following:

- From the source to every usergroup and day, with cost 0 and capacity equal to the maximum impressions for all users in that group, divided by the total number of days.

- From every usergroup and day, to every campaign and day the usergroup matches, with cost 0 and capacity inf.

- From every campaign and day, to a node representing the whole campaign. The cost is 0 and the capacity is equal to the desired number of impressions for that campaign, divided by the total number of days.

- From every campaign to the target, with cost $cpm_{\text{max}} - cpm$ and capacity equal to the desired number of impressions for that campaign.

The users that are clustered together are the ones that match the same set of campaigns, the same day. In Fig. 5.5, user 1 and 3 both match campaign 1 day 2, thus they can be merged into one node. The same applies to user 2 and 3 who match campaign 1 day 1.
The minimum-cost flow algorithm minimizes the cost of sending a given amount of flow through a graph. In this application, the revenue should be maximized, and the cost on the edges from campaigns to target will thus be $\text{cost}_i = \text{cpm}_{\text{max}} - \text{cpm}_i$, as noted above. The campaigns with the highest cpm will then get the lowest costs. The “given amount of flow” part of the algorithm also needs some attention. It is not clear beforehand what the flow passing through the graph will be. To solve this another edge is created, running between the source and the target. The capacity of this edge will be the total number of impressions for all campaigns, and the cost will be $\text{cost} = \text{cpm}_{\text{max}} + 10$. This means that it will have the highest cost of all edges, and thus the algorithm will only choose it, if there is no other way from the

Figure 5.5. The graph before and after user clustering.
5.5. FINDING OUT THE MAXIMAL INCOME

source to the target. An example of the final graph with the “dummy edge” added can be seen in Fig. 5.6.

Figure 5.6. Graph after user clustering, with dummy edge added.

Run flow

To calculate the flow with the minimum cost, the min-cost flow algorithm from the LEMON graph library [LEM11] is used. It is implemented with the network simplex linear programming method.
Chapter 6

Result

In this chapter, the results will be presented and discussed. In Section 6.1, results from the naive method presented in Section 5.2, will be shown. In Section 6.2, results from the gradient descent method in Section 5.3, will be shown.

To mimic how the system would be used later, these test are done with a procedure that iteratively repeats optimization runs and simulation runs. The idea is to run the optimization during the night, get new priorities for the campaigns, and use them the next day. Since tests cannot be run on the real ad system, the day-part is simulated.

The optimization can be run on periods with different lengths, 14 days and 28 days have been tried here. The optimization and simulation have been run on different user sets to avoid optimizing for, and testing, the same set of users.

All simulations have been repeated 20 times, and the number used is the average over that period.

6.1 Naive method

In Fig. 6.1, results from 2010-07-01 to 2011-01-01 can be seen. 0 days is the reference point, when no optimization have been done. After that two different lengths of the optimization period have been tried, 14 and 28 days. There seems to be an improvement, at least for the optimization period of 14 days.

In Fig. 6.2, results from 2011-01-01 to 2011-05-01 can be seen. 0 days is the reference point, when no optimization have been done. There seems to be an improvement by at least one percentage point, when the optimization period is 14 days. The strange thing is that, when probabilities is used instead of priorities, the result seems to deteriorate. A longer optimization period means that more information is available, and thus one would think that the result should get better.

In Fig. 6.3, results from 2011-05-01 to 2011-05-21 can be seen. The optimization started 2011-04-01, but only results from 2011-05-01 were counted.

This period is interesting, since this is when the track cap was applied. As for the previous figure, two different lengths of the optimization period have been
6.2. GRADIENT DESCENT

Figure 6.1. Second half of 2010, naive method. This is a plot describing the lost income in percent of the total income. Data is from 2010-07-01 to 2011-01-01.

used, and 0 days is when no optimization have been done. Here no improvement at all can be seen, and the loss is much higher than before. The larger amount of under-delivery is most probably an effect of the track cap.

6.2 Gradient descent

In Fig. 6.4, results from 2011-01-01 to 2011-05-01 can be seen, and in Fig. 6.5, results from 2011-05-01 to 2011-05-21 can be seen. For the results from May, the optimization was started 2011-04-01, but only results from 2011-05-01 were counted.

No improvement is seen. This is conclusive with earlier indications.
Figure 6.2. First half of 2011, naive method. This is a plot describing the lost income in percent of the total income. Data is from 2011-01-01 to 2011-05-01.
6.2. GRADIENT DESCENT

Figure 6.3. Three weeks in May 2011, naive method. This is a plot describing the lost income in percent of the total income. Data is from 2011-05-01 to 2011-05-21. The optimization was started 2011-04-01, but only results from 2011-05-01 were counted.
Figure 6.4. **First half of 2011, gradient descent.** This is a plot describing the lost income in percent of the total income. Data is from 2011-01-01 to 2011-05-01. The optimization started 2011-04-01, but only results from 2011-05-01 were counted. The method used for calculating the gradient is the one based on the probability of choosing a campaign, in Section 5.3.1.

Figure 6.5. **Three weeks in May 2011, gradient descent.** This is a plot describing the lost income in percent of the total income. Data is from 2011-05-01 to 2011-05-21. The optimization started 2011-04-01, but only results from 2011-05-01 were counted.
Chapter 7

Conclusions

It is hard to draw any conclusions when there is insufficient data to do correct estimation of errors or confidence intervals. What can be said though, is that the experiments with gradient descent has failed so far. Only updating those that under-deliver is the best option. The naive method is promising, but further investigations needs to be done to get a definitive answer if there really is an improvement. At least one other longer period should be tried, and another country.

The effects of the track cap should be considered as well. With a smaller inventory than before (fewer possible impressions from users), and no change in the amount of ads to deliver, the best strategy might be the current one, and there is no gains to be made by trying to optimize priorities.
Chapter 8

Future work

Here ideas for future improvements will be presented.

8.1 Naive method

The naive method is very simple, but there are always things that could have been done better. In this case, only updating a (random) subset of the campaigns might be a good idea. When fewer campaigns are changed at a time, the optimization will be more fine tuned.

8.2 Gradient descent

One problem when probabilities is used to approximate the gradient, is that there is a maximum number of impressions a campaign can get (its desired number of impressions), and that is not considered at all. It can be take care of by defining $E[Y_i]$ like this (the same can be done for $E[Y_i]_{inc}$ and $E[Y_i]_{dec}$):

$$E[Y_i]_{org} = \min(\text{desired}_i, \sum_{h \in H} w_i / W_h).$$

The min-function is non-differentiable. To solve that, it should be possible to replace the min-function with arctan or another sigmoid function [Wik11d].

8.3 Flow algorithm

In the flow algorithm there are several things that can be improved. Right now, to calculate the flow graph, a few approximations are done:

- A campaign and a user is considered to be matching if they are active on the same day, even if they are not active during the same time.
8.3. FLOW ALGORITHM

- To calculate how many impressions a user group can get, we first calculate the total time for ads for that group. That is then divided by a mean value of ad lengths, to get approximately how many impressions are available.

- There are constraints on how often a user can see a certain ad. This is not enforced in the flow graph.

8.3.1 Improving the flow algorithm

To get rid of the problem of one flow unit being one impression, and the inherent approximation there, the flow unit can be changed to seconds instead of impressions. The total ad time for the user group would be the capacity of the source-user group edge, and for campaigns, the capacity would be \( \text{impressions} \cdot \text{ad.time} \). The effect will be that flow can be fractions of impressions, but the effect of that should be negligible.

To solve the problem of users and campaigns matching, even though they do not have any active time in common, a more detailed check can be made. I decided that the time this will take probably will not be worth it, since most of the campaigns run during all hours.

Constraints on how often a user can see a certain ad will be hard to implement. It might be possible to approximate it by limiting the time for that ad depending on the constraint and the interval used in the graph.

8.3.2 Further uses of the flow algorithm

The flow algorithm could actually be used in the optimization as well. There are two ideas on how to proceed at the moment. The first one is quite simple. When a flow with minimal cost is found, the information about how large the flow between a specific user node and campaign is, could be used as a base for the priority. In Fig. 8.1 the final flow for one user group from the min-cost flow algorithm can be seen. The probabilities for the campaigns here would be:

\[
\begin{align*}
c_1 &= \frac{10}{10 + 5 + 15} = \frac{1}{3} \\
c_1 &= \frac{5}{10 + 5 + 15} = \frac{1}{6} \\
c_1 &= \frac{15}{10 + 5 + 15} = \frac{1}{2}.
\end{align*}
\]

The other idea builds on updating the flow often. The campaigns that are being under-delivered are prioritized in the ad system. Here the algorithm most probably needs to be closely integrated in the ad system, since the capacities of the campaigns need to be continuously updated.
8.4 Confirm result in other countries

This system will only be useful if it can be used in all countries where Spotify operates. To make sure that is the case, tests should be done in other countries as well.

Figure 8.1. Flow from user group to campaigns.
Bibliography


<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
</table>