Budget Optimisation in Sponsored Search

by

Giannis Nikolentzos

A dissertation submitted in partial fulfilment of the degree of
MSc in Artificial Intelligence
by examination and dissertation

Supervisor: Dr Alex Rogers
Examiner: Dr Srinandan Dasmahapatra

January 20, 2017
Sponsored search has gained increasing commercial importance over the last years. Search engines use ad auctions to sell advertising slots corresponding to a set of keywords. These advertisements are placed alongside organic search engine results when a user submits a query containing the corresponding keywords. In this environment, both search engines and advertisers have to make a series of important decisions. In this work, we consider the budget optimisation problem faced by advertisers participating in this type of auctions. The problem of how to allocate a limited advertising budget is very complex. Advertisers need to choose what queries to bid on and how much money to bid in order to maximise their profit given their limited budget. Moreover, their expected profit depends on many uncertain factors which have to be taken into account making the decision problem even more difficult. As a result, the development of effective bidding strategies is of high importance to advertisers as they can increase their profits to a great extent and survive the competition in the online environment. Due to the high complexity that governs the bidding task, advertisers often use software robots (i.e., agents) that can autonomously take intelligent bidding decisions in real time. As part of this work, we present a set of practical bidding strategies and we design and implement a software agent that incorporates these strategies and is able to place bids on sponsored search auctions on behalf of its owner. In order to test its efficiency, the Trading Agent Competition Ad Auctions game (TAC/AA) is used as our testbed.
Contents

Acknowledgements vii

1 Introduction 1
   1.1 Research Aims 4
   1.2 Research Objectives 5
   1.3 Contribution 5
   1.4 Dissertation Structure 5

2 The Game 7
   2.1 General Information 7
   2.2 Publisher 8
   2.3 Advertisers 9
   2.4 Search Users 10
   2.5 Summary 12

3 Background and Related Work 13
   3.1 Auction Mechanisms 13
   3.2 Bidding Strategies in Sponsored Search Auctions 14
      3.2.1 General Strategies 14
      3.2.2 TAC/AA Strategies 17
   3.3 Techniques Used 20
      3.3.1 Multiple Choice Knapsack Problem 20
      3.3.2 Particle Filtering 22
      3.3.3 Hill Climbing 25
   3.4 Summary 26

4 Bidding Agents for Sponsored Search Auctions 27
   4.1 stuBID_Version1 27
   4.2 stuBID_Version2 31
   4.3 stuBID_Version3 33
   4.4 stuBID_Version4 37
   4.5 Summary 38

5 Experiments 39
   5.1 Experimental Setting 39
   5.2 General Performance 39
   5.3 Particle Filters 41
   5.4 Estimators 43
CONTENTS

5.5 Distribution Capacity ................................................. 46
5.6 Performance after Applying Modifications .......................... 47
  5.6.1 Modified Estimation Window of Bids ......................... 47
  5.6.2 Modified Budget .................................................. 48
  5.6.3 Re-exploration of the Bidding Space ......................... 49
  5.6.4 Addition of Hill Climber in the First Two Versions ......... 50
5.7 Summary .................................................................... 50

6 Conclusions and Future Work ................................. 51
Acknowledgements

I would like to thank my parents for their support and the education they gave me throughout my life. I also offer my regards to my supervisor Dr. Alex Rogers, who gave me the opportunity to work on this exciting and challenging project. Finally, I would like to express gratitude to the PhD student Lampros Stavrogiannis for offering me his support, assistance and guidance in critical stages of this project.
Chapter 1

Introduction

Over the past years, search engines have gained increasing popularity among Internet users. Their main task is to guide users to the web pages that contain the information they are looking for. Search engines realised that their role as information gateways for millions of users provided them an excellent opportunity to increase their profits and they introduced a form of advertising known as Internet advertising in search engine or sponsored search [1–5]. When a user submits a query, the search engine returns the regular search results, that is, the links it has deemed relevant to the search, together with paid advertisements. A query is a set of words, known as keywords, that a user enters into a search engine to satisfy his or her information needs. Figure 1.1 illustrates the results that Google returned after a search for a specific query.

Figure 1.1: Results that Google returned after performing a search for the query “vacation greece”. The advertisements (results on the right and first three results on the left) appear along with the algorithmic search results (all other results on the left).
Sponsored search shares several common characteristics with traditional advertising, but there are also many differences between them. Due to the fact that the advertisements that are displayed depend on the users’ queries, advertisers can precisely target their advertisements based on the users’ search terms. They can also get information about the number of clicks each keyword is generating and which clicks are getting converted to purchases. From the users’ perspective, this approach offers a great benefit: it is more likely that sponsored search advertisements will be quite relevant to their interests as the displayed results are tailored to their queries.

With the spread of the Internet all over the world and the development of new web technologies, many advertisers realised the exciting marketing opportunities that sponsored search could provide them. As a result, they started advertising themselves through search engines in order to increase their profits. Hence, it is not surprising that sponsored search has flourished since its first appearance and that it has become a primary online advertising format. This justifies the fact that it is one of the major sources of revenue for many search engines such as Google, Yahoo! and Bing. For example, Google reported total revenue of over $43.6 billion for the year of 2012 [6]. According to statistics from IAB, sponsored search revenues in the United States reached $16.9 billion for the year of 2012 which accounted for 46.3% of the total Internet advertising revenues in the United States for the same year [7]. Internet advertising in search engine results is commonly acknowledged as a promising investment opportunity. In Figure 1.2 we can see the historical annual revenue trends in the United States. The

![Annual revenue 2003-2012 ($ billions)](image)

Figure 1.2: Historical annual revenue trends over the 10-year period from 2003 to 2012 in the United States (IAB Internet Advertising Revenue, Report 2012 Full Year Results [7]).

The popularity of sponsored search will continue to grow as the number of Internet users and the number of online transactions continue to increase. Besides the opportunities that sponsored search created for advertisers, it also increased the competition between
them. As a result, many advertisers are forced to advertise their products or services simultaneously across several search engines in order to survive stiff competition.

Most search engines use auctions to sell their advertising space inventory \([8, 9]\). Each advertiser selects a set of keywords and places a bid on each one of them. These bids state the amount the advertiser is willing to pay for a click on its corresponding advertisement. On a typical search page, for each query, the search engine makes available to advertisers a limited number of advertising slots (i.e., positions for advertisements in the results page). The slot that an advertiser wins affects the chances that its advertisement will be clicked on and thus these advertising slots have varying desirability from the perspective of advertisers. When a user submits a query to the search engine, an auction is run among the advertisers who have placed bids on a keyword or combination of keywords matching the user’s query to determine which of these advertisers are allocated advertising slots. The slot that an advertiser wins depends on his rank in the auction.

There are several different types of auction mechanisms that could be used to allocate these slots and determining which mechanism is the best is still an open problem. In the case where there are no budget constraints, the Vickrey-Clarke-Groves (VCG) mechanism \([10–12]\) is the most appropriate auction that can be used. The VCG is a truthful auction that has the property that the most efficient strategy of the participating advertisers is to bid their true valuation of a click. Despite this appealing property of the VCG mechanism, search engines do not prefer to use it. The main reason for this is that the VCG mechanism is complex and vulnerable to collusion and it may lead to low revenue for the auctioneer, that is, the search engine. An auction mechanism that was used in the early stages of sponsored search was the Generalized First Price (GFP) auction mechanism. However, the GFP mechanism was unstable due to the dynamic nature of the environment and was soon replaced by the Generalised Second Price (GSP) auction \([13]\). The GSP is the most commonly used auction mechanism at the time of writing this report. It is a non-truthful auction mechanism and as a result, the participating advertisers have to undertake the complicated task of choosing a bidding strategy. The advertisers pay only if a user clicks on their advertisement. For example, if a hotel owner bids on the search term “hotel Southampton” and wins a slot, when a user submits the query “hotel Southampton”, he will be shown a link to the hotel’s web page. If the user clicks on this link, he is transferred to the webpage and the advertiser receives a potential customer. For each such click, the advertiser pays a specific amount of money which depends on the type of the auction mechanism used by the search engine.

One of the most challenging problems faced by advertisers participating in sponsored search auctions is the budget optimisation problem: advertisers have to determine a set of keywords they are interested in and to set the bids for each one of them in order to maximise their profit given their limited budget. The fact that the expected profit depends on many uncertain factors which have to be taken into account makes the decision problem even more difficult. As a result, the development of effective bidding strategies is of high importance to advertisers as they can increase their profits to a great extent and survive the competition in the online environment. In order to be able to make the appropriate decisions, the advertiser has to understand the auction mechanism and to be able to strategise appropriately. The advertiser’s job is facilitated
by the search engines which in most cases, provide statistics and information about
the performance of the keywords on which he has already placed bids. Due to the
high complexity that governs the bidding task, many advertisers hire consultants or
resort to companies that undertake to do the bidding for them. These consultants and
companies usually charge them large amounts of money.

Given the challenging, dynamic environment of sponsored search, it is often impossible
for advertisers to manually tune their bidding strategies online. Hence, advertisers
often use software that can autonomously take intelligent bidding decisions in real
time. Autonomous agents have emerged as a new paradigm for computing and are
proving to be very effective in a variety of fields. An agent is a software program
that exhibits aspects of intelligent behaviour and is capable of independent action on
behalf of its owner [14]. The main point here is that agents act independently. They
are given a number of objectives they need to satisfy and are responsible for making
the appropriate decisions that will lead them to their fulfillment. Recent years have
witnessed a booming growth in the number of agents that participate in electronic
marketplaces. It is believed that autonomous trading agents can prove successful in
solving the budget optimisation problem that was described previously. There are a
number of researchers who have tried to implement software that solves the problem
and is able to act automatically without human intervention [15]. However, there was
no realistic simulator available to test their implementations. The Trading Agent Com-
petition Ad Auctions (TAC/AA) game\footnote{http://aa.tradingagents.org/} was developed as a solution to this problem.
The game provides a complex environment in which eight agents compete against
each other. The successful performance of agents in scenarios like the one described
here, shows that the future of agents in electronic marketplaces and electronic auctions
is very promising.

1.1 Research Aims

As mentioned above, the budget optimisation problem faced by advertisers participat-
ing in sponsored search auctions is a very challenging problem. Advertisers seek to
maximise their profit and as a result, the development of bidding strategies that allow
them to achieve their goal is very important. The increasing popularity of sponsored
search makes the development of efficient strategies more essential than ever. The aim
of this work is to provide a solution to the budget optimisation problem. Specifically,
our goal is to present a set of practical bidding strategies and to design and implement
a bidding agent that employs them and can efficiently solve the optimisation prob-
lem faced by its owner. The agent is able to act autonomously in the Trading Agent
Competition Ad Auctions game which is used as our testbed in order to analyse the
performance of the developed strategies.
1.2 Research Objectives

To achieve the aims of this research we need to be able to address the two main subproblems that compose the original problem provided by the Trading Agent Competition. The first subproblem is the prediction of some parameters of the game as well as the modelling of the user populations. The second subproblem is an optimisation problem: the agent has to determine on which keywords to bid, the amount of money it will bid on each one of them and how to profitably set a budget. As regards the first subproblem, we have used particle filters [16] to model the states of the user populations. We have also used information about past events to approximate hidden game parameters. For the second subproblem, in order to determine the positions of the auctions that have the potential to lead to profit maximization, we have modeled the problem as a multiple choice knapsack problem (MCKP) [17] and solved it with the help of a greedy algorithm. In order to estimate the amount of money we have to bid to get our advertisement shown at the desired positions of the auctions, we have used an exponentially weighted moving average that takes into account the whole bidding history of the game. Our budget is set based on the number of conversions (i.e., sales) that have to be carried out on the next day and this number is calculated using a hill climbing algorithm.

1.3 Contribution

The contribution of this work to addressing the budget optimisation problem faced by advertisers that choose sponsored search as the medium to promote their services and products is mainly associated with the design of a software agent which is able to completely undertake the advertising campaign of its owner. The agent was evaluated in the realistic environment provided by the Trading Agent Competition. Although several such agents have been implemented, each agent combines different techniques in order to manage its bidding tasks and take the appropriate decisions concerning other critical parts of the problem. In the Trading Agent Competition Ad Auctions game, advertisers do not suffer from liquidity issues and therefore, are not restricted by budget constraints. However, budgets are ubiquitous in real sponsored search auctions. For the purposes of our study, we considered the Trading Agent Competition Ad Auctions problem as a budget optimisation problem where we define a specific budget and we attempt to maximise the profit given this limited budget. The beneficial role that setting a budget and organising the bidding strategy based on it can play in increasing an advertiser’s profit was examined as part of this work.

1.4 Dissertation Structure

The remainder of this report is organised as follows:

In Chapter 2, we provide a description of the Trading Agent Competition Ad Auctions game. We give a general overview of the game and we present its specifications and the key features that have to be taken into consideration while designing an agent.
Chapter 3 summarises previous works on bidding strategies for sponsored search along with techniques we have used in our strategy.

In Chapter 4, we present the four versions of our agent. We first describe the main features of the model-free agent that was initially implemented. These features are maintained in the improved versions of our agent that are described in detail later in this Chapter.

In Chapter 5, we give details about the experiments that were conducted in order to test different parts of the agents, we present the results that emerged and we provide an analysis in an attempt to interpret the data that these experiments generated giving a deeper justification about them.

Finally, in Chapter 5, we conclude and propose directions for future work.
Chapter 2

The Game

In this Chapter, we describe the Trading Agent Competition Ad Auctions (TAC/AA) game which was used as a testbed for the evaluation of the agents that were implemented as part of this study. More specifically, we initially give a general description of the game and subsequently, we provide a more detailed description of the tasks that are carried out by the three entities that operate in the TAC/AA scenario.

2.1 General Information

The TAC/AA game [18, 19] is one of the games that are organised annually by the Trading Agent Competition. The need for a game that would allow the development of automated bidding techniques in the domain of Internet advertising through sponsored search was evident [20]. As a result, the TAC/AA game was introduced in 2009 and since then, many researchers in the field have turned their attention to the competition. In TAC/AA, three types of agents interact with each other: users, advertisers and publishers. These entities and their corresponding actions are shown in Figure 2.1. The user and publisher agents are controlled by the server, while the advertiser agents correspond to the agents that are implemented by the participants. In the TAC/AA game, the users are interested in buying components of a home entertainment system. There are three manufacturers (Flat, Lioneer and PG) and each manufacturer produces three components (TV, Audio and DVD) leading to a total of nine distinct products. The manufacturers supply advertisers with these products. The advertisers use the ad auctions to make the products available to users through sponsored search advertising. In each game, eight advertisers compete with each other to sell their products to users. The game lasts 60 virtual days, each lasting 10 seconds. At the end of the game, the advertisers are ranked according to their profits. In the following sections we provide more information about the three types of agents of the TAC/AA scenario.
Chapter 2 The Game

2.2 Publisher

The publisher receives the bids that are submitted by the advertisers and has to make a decision on which advertisements to display and the position that these are going to be displayed. In TAC/AA, there are two regions that advertisements can appear: regular and promoted slots. Advertisements in promoted slots are likely to receive more clicks than advertisements in regular slots. The total number of slots and the number of promoted slots for each query are revealed to the advertisers at the beginning of the game.

For each query, the publisher runs an auction among the advertisers who have placed bids on that query to determine which of these advertisers are allocated advertising slots. In the TAC/AA game, the queries that are submitted by the simulated users consist of a manufacturer (Flat, Lioneer, PG) and a component (TV, Audio, DVD). However, in a submitted query, one of these fields or both of them can be null leading to 16 distinct query types. The ranking of the advertisers’ bids is carried out using the ranking method proposed by Lahaie and Pennock [21]. The two researchers examined which is preferable, to rank advertisers’ bids directly or to adjust these offers by estimated click probabilities and they proposed a method that interpolates between these extremes using a (squashing) parameter $\chi$. More specifically, each bid is assigned a score equal to $(e_q)^{\chi}b_q$ where $e_q$ is the estimated click probability for query $q$ and $b_q$ is the advertiser’s bid for query $q$. The value of the squashing parameter $\chi$ is revealed to the advertisers at the start of the game. The advertisements are ranked according to these scores. The publisher may impose reserve scores, that is, minimum bid scores, for both regular and promoted slots for each query type. An advertiser that wins a slot eligible for promotion will only be promoted if its score is not lower than the promotion reserve score. Advertisers with scores lower than the regular reserve score do not win any slots.

The auction mechanism that is used is a Generalised Second Price (GSP) auction [13]. As a result, when one advertisement is clicked, its owner pays the minimum amount
it could have bid while still beating the score of the advertiser ranked below him or the reserve score if the score of the advertiser ranked below him is lower than that. Specifically, the cost-per-click price is given by:

\[
cpc = \begin{cases} 
\frac{\rho_q}{e_{q,p}} b_{q,p} & \text{if } e_{q,p} b_{q,p} \geq \rho_q \geq e_{q,p+1} b_{q,p+1} \\
\frac{e_{q,p+1} b_{q,p+1}}{e_{q,p}} & \text{otherwise}
\end{cases}
\]

where \(e_{q,p}\) is the estimated click probability of the advertiser ranked in the \(p^{th}\) position for the query \(q\), \(b_{q,p}\) is the bid that the advertiser placed on the query and \(\rho_q\) is the promoted or regular reserve score for the query.

### 2.3 Advertisers

Each simulated day, for each of the 16 query types, a keyword auction is run. For each auction, the advertisers have to submit a bid, an advertisement for display and an optional spend limit (i.e., budget). The bid refers to the amount of money the advertiser is willing to pay when a user clicks on his advertisement. Every advertiser sells every product. However, each one of them specializes in a particular manufacturer and a particular component which are announced at the beginning of the game. Advertisements can be targeted or generic. Targeted advertisements relate to a specific product, while generic do not. If an advertiser reaches its spend limit, its advertisement is removed and the advertisements that are located below his advertisement are moved one position up.

The number of sales (called conversions) depends on various factors. It is easier for the advertiser to sell a product if the user’s preference matches its component specialty. Another factor that affects the sales is the distribution constraint effect. Each advertiser is assigned a capacity from three discrete capacity levels \(\{HIGH(600), MED(450), LOW(300)\}\) and when the number of products sold in the last four days plus the current day exceeds this capacity, it has to put items on backorder. As a result, users are less eager to buy and conversions are decreased. Moreover, the advertiser’s income is greater when it sells a product that matches its manufacturer specialty. When it sells a product, it receives \(USP\) (\(USP = $10\) in the game). When it sells a product that matches its manufacturer specialty, it receives \((USP + USP \cdot MSB)\) (\(MSB = 0.4\) in the game).

The advertisers have to take their decisions based on the limited information about customers and competitors they are provided. Specifically, each day the advertiser receives three reports summarizing events from the previous day. The first report from the publisher includes information for both the advertiser and its competitors. More specifically, for each query, the advertiser receives information about the number of clicks on the advertisement it placed, the number of impressions (i.e., user views in results page), the average cost-per-click, the type of advertisement he displayed (targeted or generic) and the average position (over 10 randomly selected samples) of his advertisement. Moreover, the report includes the type of advertisements displayed by the other advertisers and the average positions of these advertisements (over 10 randomly selected samples). The second report comes from the bank and informs the
Table 2.1: Query types and their focus levels

<table>
<thead>
<tr>
<th>Manufacturer</th>
<th>Component</th>
<th>Focus Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>-</td>
<td>F0</td>
</tr>
<tr>
<td>Flat</td>
<td>-</td>
<td>F1</td>
</tr>
<tr>
<td>Lioneer</td>
<td>-</td>
<td>F1</td>
</tr>
<tr>
<td>PG</td>
<td>-</td>
<td>F1</td>
</tr>
<tr>
<td>-</td>
<td>Audio</td>
<td>F1</td>
</tr>
<tr>
<td>-</td>
<td>DVD</td>
<td>F1</td>
</tr>
<tr>
<td>-</td>
<td>TV</td>
<td>F1</td>
</tr>
<tr>
<td>Flat</td>
<td>Audio</td>
<td>F2</td>
</tr>
<tr>
<td>Flat</td>
<td>DVD</td>
<td>F2</td>
</tr>
<tr>
<td>Flat</td>
<td>TV</td>
<td>F2</td>
</tr>
<tr>
<td>Lioneer</td>
<td>Audio</td>
<td>F2</td>
</tr>
<tr>
<td>Lioneer</td>
<td>DVD</td>
<td>F2</td>
</tr>
<tr>
<td>Lioneer</td>
<td>TV</td>
<td>F2</td>
</tr>
<tr>
<td>PG</td>
<td>Audio</td>
<td>F2</td>
</tr>
<tr>
<td>PG</td>
<td>DVD</td>
<td>F2</td>
</tr>
<tr>
<td>PG</td>
<td>TV</td>
<td>F2</td>
</tr>
</tbody>
</table>

advertiser about its current score. The last report from the sales analyst enumerates
the conversions for each of the 16 types of queries. These reports do not provide to
the advertiser the necessary information to model its environment accurately and he
can only make predictions and estimates about it. Moreover, the fact that the received
reports present events from the previous day and the submitted bids are for the next
day does not allow the advertisers to take into account the current day’s events and
adds more uncertainty to the existing making the advertisers’ problem even more
difficult.

2.4 Search Users

In the TAC/AA scenario, there are 90,000 simulated users and each of them has a
preference for one of the nine distinct products. The users search for their preferred
product and then navigate available advertisements and possibly transact. In order to
search for products, the users submit queries to the search engine. Each one of the 16
distinct query types belongs to a focus level. If the query does not contain anything,
its focus level is F0. If it contains a manufacturer or a component, it is F1. If it contains
both a manufacturer and a component, the focus level is F2. The 16 query types
and their focus levels can be seen in Table 2.1. Each user will only buy his preferred
product. In the game, there are nine products and as a result, the population can be
divided in nine subpopulations. Each user subpopulation is modeled as a Markov
chain. In the TAC/AA scenario, the users can be in one of six of possible states. The user state transition model is shown in Figure 2.2. All the users are initialised in the non-searching state (NS), in which they do not submit queries, and they can transition to the informational search state (IS) where they generate F0 or F1 or F2 queries with equal probability. The users can remain in this state or transition to one of the three focused searching states (F0, F1, F2) in which they generate corresponding queries. When in one of these states, they can go from lower focused states to higher, they can transact (T) or they can return to the non-searching state. The probability of a purchase increases with the focus. After transacting, the users can remain in their present state or go back to the non-searching state.

Figure 2.2: User state transition model. Each state also has an implicit self-loop (not shown).

The click model that is adopted in TAC/AA is a hybrid of the cascade model [22, 23] and the model proposed by Das et al. [24]: after generating a query, the users move from higher positions to lower. This means that ads that are located at higher positions are more likely to be clicked. When an advertisement is displayed, its owner receives an impression. The impression will or will not be followed by a click. If a user reaches an advertisement, the probability of clicking depends on the estimated click probability of the advertiser for the query $q$, $e_{q,a}$, whether the advertisement is placed in a promoted slot and whether, in case of a targeted advertisement, it matches the preference of the user. Specifically, the click probability is given by:

$$Pr(\text{click}) = \eta(e_{q,a}, f_{\text{target}} \cdot f_{\text{pro}})$$

(2.1)

where

$$\eta(p, x) = \frac{p \cdot x}{p \cdot x + (1 - p)}$$

(2.2)

The promotion factor $f_{\text{pro}}$ is greater if the position of the advertisement is a promoted slot as these slots enjoy an enhanced click rate. For a regular slot, $f_{\text{pro}} = 1$, while for a promoted slot, $f_{\text{pro}} = 1 + PSB$ (PSB = 0.5 in the game). The targeting factor $f_{\text{target}}$ increases or decreases depending on whether the targeted advertisement that has been
selected matches the preference of the user:

\[
 f_{\text{target}} = \begin{cases} 
 1 + TE & \text{if targeted advertisement matches} \\
 1 & \text{if generic advertisement} \\
 1/(1 + TE) & \text{if targeted advertisement does not match}
\end{cases}
\]

where \( TE \) is the targeting effect (\( TE = 0.5 \) in the game).

If a user did not click on an advertisement or clicked but did not make a purchase, it will proceed to the next advertisement with continuation probability \( \gamma_q \). The parameters \( e_q, a \) and \( \gamma_q \) are hidden and are drawn randomly from a prespecified interval at the start of each game.

The conversion behaviour of users depends on many factors [25, 26]. In the TAC/AA scenario, a user that clicks on an advertisement may convert or not. The probability depends on three parameters. The first parameter is the baseline conversion probability, \( \pi_l \), which depends on the user’s focus level. Higher focus level queries convert at higher rates, that is \( \pi_{F2} \geq \pi_{F1} \geq \pi_{F0} \). The second factor increases the probability of a conversion in cases where the user’s desired product is the same component as the advertiser’s component specialisation. The last parameter captures the case in which advertisers sell too many products in a short period. In this case, their inventories run short and they are not able to serve the users immediately. As a result, users are less eager to buy their preferred product from these advertisers and the advertisers are led to limited conversions. The distribution constraint effect is given by:

\[
 I_d = \begin{cases} 
 \lambda^{((\sum_{i=d}^{d+(W-1)} c_i) - C^{\text{cap}})} & \text{if } (\sum_{i=d}^{d+(W-1)} c_i) - C^{\text{cap}} > 0 \\
 1 & \text{otherwise}
\end{cases}
\]

where \( c_d \) is the total number of conversions on day \( d \), \( W \) is the aggregation window for distribution capacity (\( W = 5 \) in the game) and \( C^{\text{cap}} \) is the critical distribution capacity beyond which conversion rates start decreasing. As mentioned earlier, in the TAC/AA scenario, each advertiser is assigned one of three distinct capacity levels \{HIGH(600), MED(450), LOW(300)\} at the start of the game. The conversion probability is:

\[
 \Pr(\text{conversion}) = \begin{cases} 
 \eta (\pi_{l} \cdot I_d, 1 + \text{CSB}) & \text{if user matches component specialty} \\
 \frac{\eta}{\pi_{l} \cdot I_d} & \text{otherwise}
\end{cases}
\]

where \( \text{CSB} \) is the component specialisation bonus (\( \text{CSB} = 0.6 \) in the game).

### 2.5 Summary

In this Chapter, we gave an overview of the TAC/AA game which was used as the testbed in order to evaluate the agents that we implemented. We initially provided general information about the game and subsequently, we presented the three entities that are encountered in the TAC/AA scenario, the publisher, the advertisers and the users. In the next Chapter, we review related work in the field of sponsored search.
Chapter 3

Background and Related Work

This Chapter surveys previous work in the field of sponsored search. The purpose of this Chapter is to point out the many contributions of previous researchers and to place our work in the proper context. Initially, we present the most popular auction mechanisms that have been used by the major search engines. We then summarise a number of sponsored search bidding strategies both developed in the context of the TAC/AA game and as general bidding strategies for real sponsored search auctions. Finally, we present three techniques that are incorporated into our agent and constitute key components for its efficient functioning.

3.1 Auction Mechanisms

In the early beginning of sponsored search, advertisers did not pay on a per-click basis, but they bought a specific number of impressions after negotiating with the search engine. They usually paid large amounts to show their advertisements a fixed number of times and the whole process was very slow. This situation changed dramatically in 1997 when Overture, one of the major search engines of the time, introduced a completely new model of selling Internet advertising, the Generalised First Price (GFP) auction mechanism. Each advertiser selected a keyword in which he was interested and submitted a bid reporting the amount he was willing to pay each time a user clicked on his advertisement. The advertisers could now target their advertisements, as they submitted bids only on keywords that were relevant to their products or services. The position of an advertisement in the results page depended on the submitted bid. The advertisements were displayed in descending order of bids and as a result, higher bids led to higher positions. When a user clicked on an advertisement, the advertiser paid the bid he placed on the corresponding keyword. After the launch of the new auction mechanism, Overture’s sponsored search platform flourished. However, the GFP mechanism proved unstable due to the dynamic nature of the environment. Google managed to address the GFP’s problems when it introduced the Generalised Second Price (GSP) auction mechanism [13]. GSP is used by the majority of the search engines as the mechanism to allocate their slots. In the simplest form of the GSP auction, advertisers submit bids for their keywords of interest stating the maximum amount
they are willing to pay when a user clicks on their advertisement (called their valuation). In most cases, the advertisements are ranked in decreasing order of bids and the final rank determines the relative positions in which they are displayed in the search results page. In particular, the advertisement with the highest bid is displayed at the top, the advertisement with the second highest bid is displayed in the second position, and so on. If a user clicks on an advertisement placed in a specific position, the corresponding advertiser is charged an amount that equals the bid of the second advertiser plus a minimum increment. The rationale behind this is that a bidder in a specific position will only want to pay the minimum increment above the bid of the advertiser in the next position. Several search engines take into consideration the fact that different advertisers have different probabilities of being clicked when placed in the same position along with the submitted bids in order to rank the advertisements. These probabilities, which are called click-through rates (CTR) are multiplied by the advertisers’ bids and the advertisements are ranked according to these values. When a user clicks on an advertisement, the advertiser is charged the minimum amount sufficient to exceed the next advertiser’s bid times their estimated CTR ratio. The GSP auction mechanism makes the market more user friendly and stable. A lot of research has focused on the properties of the GSP auction, but a further analysis of these properties is beyond the scope of this dissertation (for a detailed survey see Maillé et al. [27]). An alternative to the GSP auction mechanism is the Vickrey-Clarke-Groves (VCG) mechanism [10–12]. VCG assigns the positions in a socially optimal manner, maximising the social welfare. For each click on his advertisement, an advertiser pays the externality imposed on the other bidders by its presence, that is, the harm he causes to them. The optimal strategy for a bidder participating in a VCG auction is to bid his true valuation of the object.

3.2 Bidding Strategies in Sponsored Search Auctions

Due to the ever increasing importance and rapid pace of development of sponsored search, the budget optimisation problem has been extensively studied over the last few years. The launch of the TAC/AA game gave extra impetus to the research in the field. The growing number of researchers is a guarantee for the further development of sponsored search. In the next sections, we present a number of bidding strategies adopted by advertisers participating in sponsored search auctions and by TAC/AA agents.

3.2.1 General Strategies

The budget optimisation problem has been troubling the minds of researchers for several years now. A lot of work has been done on developing bidding strategies that can efficiently deal with it. In this section, we provide details of some of these strategies.

Borgs et al. [28] propose a natural bidding heuristic in which advertisers attempt to optimise their utility by equalising their return-on-investment (ROI) across all the
keywords of their interest. ROI is defined as the ratio of the advertiser’s utility to his cost. Most of the methods that can be found in the literature aim at maximising the advertiser’s ROI. Borgs et al. state that the problem can be viewed as a discrete separable resource allocation problem. In these problems, the efficiency of an investment strategy can be calculated by measuring its marginal ROI. According to their bidding heuristic, a budget-constrained advertiser can optimise his utility if he adjusts his bids such that the marginal ROI across all keywords is the same. In general, this heuristic cannot be applied in practice, as the marginal ROI is difficult to estimate and is even undefined when the utility or the cost are discontinuous. However, the marginal ROI of a keyword can be approximated by its ROI. Thus, instead of equating the marginal ROIs of all the keywords, the bids are set such that the ROIs of the keywords equal some constant ROI.

Muthukrishnan et al. [29] assume that the advertiser’s daily budget and choice of keywords are known and give directions on how the advertiser should set his bids in order to maximise the number of clicks that he will receive. The authors state that the number of clicks an advertiser receives in a day is very hard to estimate as it depends on many factors. The budgets of other advertisers and their choices of keywords as well as the number of queries of relevance that users issue on that day, and the frequency with which the advertisements are clicked are only some of these factors. The values of these parameters are not known but most search engines use statistical methods to analyze past data and can provide probability distributions for them. The authors formulate three stochastic versions of the budget optimisation problem and for each one of them, they identify if a bid solution that maximises the expected number of clicks can be determined. The first model is the Proportional Model in which the number of users, their queries and clicks do not stay the same, but the proportions of clicks for different keywords are constant. In the second model, called the Independent Keywords Model, each keyword has its own probability distribution for the number of clicks, and the samples are drawn from these distributions independently. The last model is the Scenario Model in which there are a number of scenarios, each of which is sampled from a given probability distribution over scenarios and the number of clicks for each keyword is determined by the current scenario. The authors designed some algorithms that produce a special kind of solutions called prefix solutions. These algorithms sort the keywords in order of increasing cost-per-click and bid on the first $i$ of them. In the first model, these solutions are optimal. In the second model, they constitute good approximations, while in the third, the best prefix solution is arbitrarily far from the optimum.

Feldman et al. [30] present a two-bid uniform strategy, that is, a strategy that randomises between two bids on all keywords. This strategy aims at maximising the number of clicks that the advertiser is going to receive. The authors assume that the expected cost and clicks that result from placing a bid $b$ on a keyword are known or can be measured precisely using past history and they define the functions $\text{cost}(b)$ and $\text{clicks}(b)$ that return the cost and number of clicks for a bid $b$ that qualifies for a specific query auction. The data contained in these two functions is a collection of points and can be illustrated in a plot of cost vs clicks which the authors call a landscape. If the advertiser is interested in only one keyword, it is reasonable to move to the direction of increasing cost and submit the highest bid that ensures that he will remain within his budget. However, since there are only a finite number of points in the landscape, he is
not able to exploit his budget to its full potential. As a solution, the authors propose a randomised bidding strategy that combines two bids, the bids whose corresponding costs are immediately higher and lower than the available budget. For more than one keyword, the points from the individual queries are aggregated into a single landscape and the two uniform bid vectors that correspond to the two points are calculated. Although uniform strategies may seem naive, they are very efficient. There always exists a uniform bidding strategy that is \((1 - \frac{1}{e})\)-optimal.

Zhou et al. [31] address the problem of determining the bids that an advertiser should place on its keywords of interest at each time period so as to maximise their ROIs. In order to solve the problem, the authors cast it as a knapsack problem and present algorithms for the problem achieving efficient performance. The weight is equal to the total cost and must not be greater than the budget while the value is equal to the total profit, that is, the total revenue minus the total cost. In order to design their algorithms, the authors assume that the weight of each item is much smaller than the capacity of the knapsack and that the value-to-weight ratio of each item is both lower and upper bounded by \(L\) and \(U\) respectively. In the case of single slot auctions, the problem is modelled as an online knapsack problem and the authors design an algorithm with ratio \((\ln(U/L)) + 1\) compared to maximum profit attainable by the omniscient bidder who knows the bids of all the other advertisers. The efficiency of the method is the same regardless if the bids of the other advertisers are known or not. For the multiple-slot case, the authors model the problem as an online multiple choice knapsack problem. The difference between the knapsack and the multiple choice knapsack is that in the latter there are some classes of items and at most one item of each class can be selected. In the current problem, a class corresponds to an auction which consists of many items, that is, slots, but the advertiser can be placed to at most one of them. The authors provide an algorithm that achieves a \((\ln(U/L)) + 2\) ratio to that of the best offline algorithm for the multiple-slot case. The knowledge of the other advertisers’ bids and the click-through rates of all the available slots is necessary in order to achieve maximum performance.

Cary et al. [32] examine a class of greedy bidding strategies that can be used in a repeated auction that is run after a user has submitted a specific query. The authors assume that the other advertisers always place the same bids on the auctions and present some strategies an advertiser competing against them could adopt to maximise his utility. Specifically, they initially present the Balanced Bidding strategy. An advertiser who adopts this strategy targets the slot that maximises his utility and chooses a bid high enough above the bid needed for his target position in order to force the prices paid by his competitors to rise, but not so high that if one of his competitors bids just below him, he would mind getting a higher slot at a price just below his own bid. The second strategy that is presented is the Restricted-Balanced Bidding strategy. According to this strategy, the advertiser targets the slot that maximises his utility, but this slot is selected only among the slots with no higher click-through rate than his current slot. The bid is determined in the same way as in Balanced Bidding. The third strategy is called Competitor Busting. An advertiser who follows this strategy targets the slot that maximises his utility and bids just below the bid that is required to win the next higher slot. The aim of this strategy is to make competitors pay as much as possible and exhaust their advertising resources. The last strategy is the opposite of Competitor Busting and is called Altruistic Bidding. In Altruistic Bidding, the advertiser targets
the slot that maximises his utility, but instead of trying to decrease the profits of the other advertisers, he tries to help them by bidding the minimum amount to win its desired slot.

One of the first works on software agents that are designed to bid autonomously in sponsored search auctions and which are capable of solving the bidding problems faced by their owners is that of Kitts and Leblanc [33]. The authors formulate the problem as an integer programming problem trying to maximise the profit subject to some constraints. Initially, the authors present an algorithm that translates four rules (position minimum, position maximum, bid minimum and bid maximum) into a constraint that is used to bound the optimisation. Subsequently, they provide methods for estimating the values of some unknown parameters of the problem. The first parameter is the expected number of user clicks for each position. It is estimated using data from the past. If for a specific period there are no data available, the authors draw data from surrounding periods and predict the number of clicks for the period in question. Another parameter they try to predict is the position they will win given a specific bid. Their prediction is based on the previously submitted bids and the corresponding won positions. They use an exponential function which allows them to give more weight to recent observations and less weight to those taken further in the past. The last parameter they try to estimate is the expected revenue. Specifically, they assume that revenue per click is independent of position and time and model this variable as a simple average. After these parameters are estimated, the resulting model is tested for quality and if it does not manage to pass it, they try to repair the model by exploring the bidding landscape, so as to obtain sufficient samples.

3.2.2 TAC/AA Strategies

Many researchers that participated in TAC/AA competitions published reports describing the bidding strategies they followed as well as other interesting features they incorporated into their agents. In this section, we summarise most of these reports by describing the basic design principles of the corresponding agents.

Pardoe et al. [34] designed TacTex, one of the most successful agents of TAC/AA (ranked first in 2009 - 2011). TacTex exploits the information provided by the publisher in order to estimate the full game state. It uses the estimated game state to make predictions and optimises its actions based on these predictions. Specifically, after receiving a report on the results of the previous day, the agent tries to extract the ranking of the scores of the other advertisers for each query. Subsequently, TacTex attempts to estimate the composition of each one of the nine user populations. The agent maintains estimates of the user population states by using a particle filter for each subpopulation. Each day, for each filter the agent combines the old set of particles with the information provided by the publisher in order to generate a new set of particles and then each particle is updated using the user transition dynamics to obtain the next days expected user population. The next task of the agent is to predict the actions of the other advertisers. It predicts the maximum number of impressions of each advertiser, the type of the advertisement they will choose and the bids they will submit for each query. TacTex also tries to estimate unknown game parameters by using the available information and computing the best fitting values. The agent uses all these
estimates that were produced in the previous stages in order to choose the next day’s bids and advertisements. Initially, it calculates the optimal set of conversion targets for the rest of the game and it then splits the next day’s conversion target between the various query types. In order to assign an optimal number of target conversions to each query, it predicts the expected cost, revenue and conversions from each query using the expected user population and other previously estimated information. Finally, for each query, the agent determines the bid that will result in the desired number of conversions.

Chang et al. [35] designed AstonTAC. AstonTAC finished second in the first competition in 2009. The agent records and organises the static information provided by the server during game initialisation as well as the run-time information regarding previous day’s events provided by the publisher. Each day, it processes this information and turns it into knowledge which it uses to determine its bids and other tasks. The agent’s bids are based on the market-based value per click (MVPC). The MVPC of a query is the expected revenue minus advertising cost that a click on the corresponding advertisement can generate. The bid that it places on a query is equal to the product of three factors. The first factor is the MVPC of the specific query, while the other two factors depend on the agents distribution capacity and the previous days sales, and on the ranking mechanism adopted by the publisher respectively. The agent calculates its desired conversions for the next day and allocates them to queries that can generate high profit. Specifically, it sorts the queries in descending order of profit-per-conversion and selects a prefix of them, so as to fully exploit the expected available conversions. The spend limit of the selected queries is set to infinite, while the spend limit of the other queries is set to zero. The expected conversions of each query are calculated as a product of expected impressions, click probability and conversion rate. As regards the type of the advertisement, the agent chooses a generic advertisement when neither the component of the query matches the agents component specialty nor the manufacturer of the query matches its manufacturer specialty. In any other case, it chooses a targeted advertisement.

Quak-TAC, the agent developed by Vorobeychik [36], is a simple agent that follows a simulation-based game theoretic approach. The agent’s bids are determined by the linear formula \( b(u) = au \) where \( u \) is the advertiser’s value per click (VPC). The VPC of a query is the expected revenue from each click on the corresponding advertisement. For each query, the VPC is equal to the product of the probability of converting after a click on the advertisement and the expected revenue from such a conversion. The agent computes the conversion probability by estimating the next day’s proportion of focused shoppers and the current and next day’s number of conversions. As regards parameter \( a \), before the competition, the author used a simulation-based game theoretic analysis to estimate an equilibrium in this strategy space and to determine the value of \( a \) that optimally responds to the prediction of the other players’ bids. In order to operationalise an equilibrium, Vorobeychik restricted all agents to follow the same bidding strategy \( b(u) \). Moreover, in order to enable a more detailed analysis, he limited \( a \) to discrete values ranging from 0 to 1 with a step of 0.1. Vorobeychik used a best response dynamic to find an equilibrium. In this method, which is roughly analogous to self-play, a player computes a maximizing action in each iteration assuming stationary opponents. All the advertisers except from one follow the same strategy, while the strategy of this advertiser varies among games. The author chose to start
with \( a = 1 \) and after several iterations of best response dynamics, he obtained the equilibrium \( a = 0.2 \) which proved to be a best response to nearly every reasonable strategy and was the strategy used in the competition’s final rounds.

Chatzidimitriou et al. [37] designed agent Mertacor. Mertacor is based on the agent developed by Vorobeychik that was described above. The main tasks of Mertacor are the same as those of Quak-TAC. Specifically, the agent estimates the VPC for each query and chooses a proportion of VPC for bidding in each auction according to the linear formula: \( b(u) = au \). Mertacor calculates in a slightly different way compared to Quak-TAC the estimation of the conversion probability which is necessary for the estimation of the VPC. The difference lies in the calculation of the distribution constraint effect for the current and the next day. Besides this differentiation, the authors made two extensions. They used a k-Nearest-Neighbors (k-NN) algorithm to get a better estimation of the current day’s number of conversions and use it to further predict the capacity to be purchased on the next day. These improved estimations can contribute to a more accurate estimation of the VPC. The second extension they added was an associative to the state of the game n-armed bandit formulation of the problem for selecting the appropriate value of \( a \). The authors switched their bidding strategy from \( b(u) = au \) to \( b(a, u) = au \) in an attempt to make their agent adaptive and therefore, to be more competitive in a real environment.

Tau, the agent designed by Schain et al. [38], participated in the TAC/AA competitions of 2010 and 2011 and its major feature is its light modelling of the game. The agent that participated in the 2010 competition was very simple. It only bid for queries that matched its component or manufacturer specialty or both. Moreover, it placed only high bids on these queries reducing the action space. The next day’s target daily capacity was set such that it complemented the number of conversions during the previous three days plus the estimation of today’s conversions to a number a little greater than the 5-day distribution capacity. The agent calculated the fraction of the next day’s target capacity that is going to be allocated to each query using the Randomized Weighted Majority (RWM) algorithm which generated its results based on estimates of the cost-per-click and the conversion rate of each query. The agent’s performance was not satisfactory and the authors realised that they would have to model the user populations in order to achieve top performance. As a result, for the 2011 competition, inspired by agent TacTex, they used particle filters to model the user populations. However, in their implementation, the inputs of the filters were estimated using a k-Nearest Neighbours (k-NN) algorithm that was trained on data from previous games. In order to avoid the risk of under-utilized capacity resulting from incorrect predictions of the users model, the authors introduced a tuneable regularization factor. They also tried to estimate various game parameters as well as the two-way mappings of bids to resulting costs and corresponding positions. For the optimisation part, they used the equimarginal principle which states that the profit level must be equal across all queries. Therefore, instead of placing high bids on the queries of interest, the required bids were induced by the target profit level.

Siranovic et al. [39] implemented CrocodileAgent. For queries of type F0 and F1, the agent generates generic advertisements, while for queries of type F2, it chooses to advertise the product that matches the query’s manufacturer and component. CrocodileAgent maintains a constant bidding strategy during the game. The bids for every query are
defined with an intention to maximise the profit. Except from the first two days when there are no available reports from the publisher and the bids are predetermined, for the rest of the game, the bids are based on the previous day’s conversion rate, the query focus level and the agent’s specialties. If the previous day’s rate is satisfactory, the bid that is submitted for the next day is based on it. If the rate is not satisfactory, the new bid is based on the cost-per-click received on the latest report. Each day, the agent sets an aggregate spend limit from a set of fixed values that were experimentally proved optimal. These values depend on the agent’s capacity and the current day of the game. Finally, it also sets query specific spend limits. The values of these limits are defined according to the click ratio, to the conversion ratio and to previous profits.

Berg et al. [40] present some bidding strategies and evaluate their performance in the TAC/AA scenario. The authors concentrate only on the bidding strategies and do not provide any information about the way they predict the expected number of impressions, click probabilities, conversion probabilities and cost-per-click for each query. Specifically, for the optimisation part of the agent, two different classes of algorithms were tested: greedy multiple choice knapsack algorithms and rule-based algorithms. Initially, the authors modelled the problem as a penalised multiple choice knapsack (PMCKP) problem and used three different greedy algorithms to solve it. These algorithms return the bid that the agent should place on each auction in order to maximise its profit over all future days and not just the current day. The model takes also into consideration the distribution constraint effect and chooses the appropriate number of conversions to be made. Besides the greedy MCKP algorithms, the authors implemented a set of rule-based algorithms. These algorithms can make reasonable decisions, even when the accuracy of the used models is low. The first rule-based bidder that is presented searches for the bids at which some proxy of ROI is equated across queries. The authors also implemented agents that try to equate profit margins (PM), cost per order dollar (CPOD) and profit per cost dollar (PPCD) and they observed that the strategies that result from trying to equate any of these three metrics are equivalent.

3.3 Techniques Used

The final version of the TAC/AA agent that we implemented combines a wide variety of techniques. In this section, we present three techniques that constitute the core of our agent: the greedy algorithm that solves the Multiple Choice Knapsack Problem which is responsible for the optimisation part of the agent, the Particle Filter which is the key component of our estimator and the Hill Climbing algorithm which ensures that the number of conversions are maintained at the desired levels.

3.3.1 Multiple Choice Knapsack Problem

The Multiple Choice Knapsack Problem (MCKP) is a problem in combinatorial optimisation which generalises the classic Knapsack Problem (KP). In a KP, there is a set of items and each one of them has a specific weight and a specific value. In order to solve
the problem, we have to determine how many copies of each item to select so that the total weight is less than or equal to a specific limit and the total value is maximised. The problem is called a Knapsack Problem because it is similar to the problem faced by someone who owns a knapsack with limited size and wants to fill it with the most valuable items. The KP is a very common problem which arises in resource allocation and is studied in a variety of fields. The 0-1 KP is the most common form of KP problems. In the 0-1 KP, there is only one copy of each item and it can be added to the collection of selected items or not. The 0-1 KP can be formulated as:

$$\max \sum_{i=1}^{k} value_i \cdot x_i$$  \hspace{1cm} (3.1)$$

$$\sum_{i=1}^{k} weight_i \cdot x_i \leq W$$  \hspace{1cm} (3.2)$$

$$x_i \in \{0, 1\}, \text{ for all } 1 \leq i \leq k$$  \hspace{1cm} (3.3)$$

There are many algorithms which are able to solve knapsack problems. Most of them are based on dynamic programming approaches, branch and bound approaches or hybridizations of both approaches. Due to the limited amount of time that is available for the optimisation process in the TAC/AA scenario, it would be preferable to use a fast algorithm in order to find workable solutions even if they are not optimal. Because of the importance of the KP problems which derives from the fact that many real world problems can be modeled as such problems, there has been substantial research on creating and analyzing algorithms that approximate a solution. The most commonly used approximation algorithms are the ones that follow greedy approaches and generate nearly-optimal solutions.

The MCKP is an extension of the 0-1 KP. In the MCKP, there is a number of classes and each class consists of a set of items. The restriction in the MCKP is that at most one item of each class can be added to the collection of items that have already been selected. The MCKP can be modeled as:

$$\max \sum_{i=1}^{k} \sum_{j=1}^{m} value_{ij} \cdot x_{ij}$$  \hspace{1cm} (3.4)$$

$$\sum_{i=1}^{k} \sum_{j=1}^{m} weight_{ij} \cdot x_{ij} \leq W$$  \hspace{1cm} (3.5)$$

$$\sum_{j=1}^{m} x_{ij} \leq 1, \text{ for all } 1 \leq i \leq k$$  \hspace{1cm} (3.6)$$

$$x_{ij} \in \{0, 1\}, \text{ for all } 1 \leq i \leq k \text{ and all } 1 \leq j \leq m$$  \hspace{1cm} (3.7)$$

As with the majority of the KP problems, a variety of algorithms have been proposed for the solution of the MCKP. These types of problems are typically solved greedily. We will describe the greedy approach presented by Kellerer et al. [17] which was also adopted by Berg et al. [40]. The MCKP is converted into a standard KP by creating incremental items and the resulting KP is solved greedily.
For each class, we create a set of incremental items using the items that belong to this class. Hence, the number of the different sets of incremental items is equal to the number of classes. Subsequently, we describe the process of creating a set of incremental items from the items of a class: the first step is to create a set of items in which the items of the class are sorted in nondecreasing order of weight, that is, the lighter items precede the heavier. We then remove the dominated and LP-dominated items from the newly created set of items. Due to the fact that there are other items more profitable than these, it is not necessary to take them into account and they can be safely removed. An item \( i \) dominates another item \( j \) if \( \text{weight}_i \leq \text{weight}_j \) and \( \text{value}_i > \text{value}_j \). Items \( i \) and \( j \) LP-dominate item \( k \) if \( \text{weight}_i < \text{weight}_k < \text{weight}_j \), \( \text{value}_i < \text{value}_k < \text{value}_j \) and \( \frac{\text{value}_j - \text{value}_k}{\text{weight}_j - \text{weight}_k} \geq \frac{\text{value}_k - \text{value}_i}{\text{weight}_k - \text{weight}_i} \). The last step in creating the set of incremental items is to modify the weights and values of the items. Specifically, the first item maintains its original weight and value, but all the other items are modified according to the following equations:

\[
\text{w}'_i = \text{w}_i - \text{w}_i - 1 \\
\text{u}'_i = \text{u}_i - \text{u}_i - 1
\]

for \( 2 \leq i \leq n \) where \( n \) is the number of the items that remained after removing the dominated and LP-dominated items.

The algorithm that we adopted for the solution of the emerging problem computes the efficiencies of the first item of each class and based on these efficiencies selects the next item that will be inserted into the knapsack using a greedy approach. The efficiency of an item is equal to \( \text{efficiency} = \frac{\text{value}}{\text{weight}} \) where \( \text{value} \) and \( \text{weight} \) are the item’s value and weight that emerged after creating the incremental items. The algorithm finds the item with the highest efficiency and if its efficiency is positive and its weight is low enough so it can fit into the knapsack, the item is selected. The item is then removed from the set of incremental items of its class and the process is repeated until an incremental item with nonpositive efficiency is encountered or there are no more incremental items left. In this way, the MCKP is transformed into a standard 0-1 KP in which from the first incremental item of each class, we select the item with the highest efficiency. Finally, from the incremental items that were selected, we determine which of the original items will fill the knapsack. For each class that one or more of its incremental items were selected, the last incremental item that was added is the one that is selected in the solution of the original problem. Taking the first \( k \) incremental items of a class in the resulting 0-1 KP is equivalent to taking only the \( k \)th incremental item in the original problem.

### 3.3.2 Particle Filtering

Particle filters are sequential Monte Carlo methods that evolve a set of point mass representations of probability densities (i.e., particles) in order to perform inference in state-space models [16]. Particle filters extend the traditional Kalman filtering methods [41] as they can be applied to any state-space model. There are several variants of the particle filter and most of them, including the Sequential Importance Sampling (SIR) filter which is the filter that best applies to our case, are based on the Sequential Importance Sampling (SIS) algorithm.

A filter is used to estimate the state vector of a system, that is, a vector that contains all relevant information required to describe the system, at a specific time given all the
measurements up to that time. A discrete-time formulation of the problem is adopted and as a result, the evolution of the system is modelled using difference equations and measurements are assumed to be available at discrete times.

In a general discrete-time state-space model, the state of a system evolves according to:

$$x_k = f_k(x_{k-1}, v_{k-1})$$  \hspace{1cm} (3.8)

where $x_k$ is a vector representing the system at time $k$, $f_k$ is a non-linear function of the state of the system at time $k - 1$ and $v_{k-1}$ is the noise vector of the state of the system at time $k - 1$. Information about $x_k$ is obtained only through noisy measurements:

$$z_k = h_k(x_k, n_k)$$  \hspace{1cm} (3.9)

where $h_k$ is a possibly non-linear function describing the measurement process and $n_k$ is a vector representing the measurement noise. Our aim is to estimate $x_k$ based on the set of all available measurements $z_{1:k} = \{z_i, i = 1, ..., k\}$ up to time $k$.

In order to perform the required inference, we need a model that describes the evolution of the system with time and a model that applies the updates that are determined to be made after a measurement is received to the state of the system. These models are available in a probabilistic form and in conjunction with the fact that the system state has to be updated after a measurement is received make the Bayesian inference the most suitable method to estimate the required state. In a Bayesian approach to dynamic state estimation, our goal is to compute the posterior probability density function (PDF) of the state using the received measurements as well as other available information. Hence, our goal is to calculate the PDF $p(x_k|z_{1:k})$ which is a complete estimation of the systems state.

For a variety of problems, an estimate of the systems state is necessary to be computed after receiving a measurement. In this case, it is preferable to process the received measurements sequentially than to process the full set of received measurements when a new measurement is received. In this way, we can avoid storing the complete data set and reprocessing data that were previously processed. This type of filter is called a recursive filter and consists of two steps: the prediction step and the update step. The prediction step uses the system model to predict the PDF of the system’s state at the next time step. The prediction step computes the prior PDF of the system’s state at time $k$ using the equation:

$$P(x_k|z_{1:k-1}) = \int p(x_k|x_{k-1})p(x_{k-1}|z_{1:k-1})dx_{k-1}$$  \hspace{1cm} (3.10)

where $p(x_{k-1}|z_{1:k-1})$ is known due to recursion and $p(x_k|x_{k-1})$ is defined by (3.8) and the know statistics of $v_{k-1}$. The update stage uses the latest received measurement to modify the predicted PDF of the systems state so as to reflect its true state to a greater extent. The update is performed using the Bayes theorem:

$$p(x_k|z_{1:k}) = \frac{p(z_k|x_k)p(x_k|z_{1:k-1})}{p(z_k|z_{1:k-1})}$$  \hspace{1cm} (3.11)
There are some special cases in which the computations required for the prediction and update steps (equations (3.10) and (3.11)) can be carried out analytically and the state of the system can be estimated using the Kalman filter algorithm. However, in general, these equations cannot be solved analytically. In this case, the use of approximate methods such as Monte Carlo sampling is necessary. Sequential importance sampling (SIS) is the most basic Monte Carlo method used for this purpose. In order to describe the algorithm, it is useful to consider the full PDF \( p(x_{0:k-1}|z_{1:k-1}) \) at time \( k-1 \). In SIS the required PDF \( p(x_{0:k-1}|z_{1:k-1}) \) is represented by a set of random samples \( \{x_{0:k}^i, i = 0, ..., N_s\} \), which are called particles. Each particle has a weight \( \{w_k^i, i = 0, ..., N_s\} \) and these weights are updated such that they approximate the PDF \( p(x_{0:k}|z_{1:k}) \) at the next time step. Specifically, the PDF at \( k \) is approximated as:

\[
p(x_{0:k}|z_{1:k}) \approx \sum_{i=1}^{N_s} w_k^i \delta(x_{0:k} - x_{0:k}^i) \tag{3.12}
\]

The values of the weights are determined using the principle of importance sampling. Because it is difficult to sample directly from the target distribution \( p(x) \), we approximate it using samples drawn by a proposal distribution \( q(x) \) which is called an importance density. If the samples \( x_{0:k}^i \) were drawn by a proposal distribution \( q(x_{0:k}^i|z_{1:k}) \), the weights are determined by:

\[
w_k^i \propto \frac{p(x_{0:k}^i|z_{1:k})}{q(x_{0:k}^i|z_{1:k})} \tag{3.13}
\]

As mentioned previously, at each iteration, an approximation to the PDF \( p(x_{0:k-1}|z_{1:k-1}) \) is available from the previous recursion and we need to approximate the PDF \( p(x_{0:k}|z_{1:k}) \) with a new set of samples. If we choose a proposal distribution that can be factorised such that

\[
q(x_{0:k}|z_{1:k}) = q(x_k|x_{0:k-1}, z_{1:k})q(x_{0:k-1}|z_{1:k-1}) \tag{3.14}
\]

then the samples \( x_{0:k}^i \sim q(x_{0:k}|z_{1:k}) \) can be obtained by augmenting each of the existing samples \( x_{0:k-1}^i \sim q(x_{0:k-1}|z_{1:k-1}) \) with the new state \( x_k^i \sim q(x_k|x_{0:k-1}, z_{1:k}) \). The weights are updated using the equation:

\[
w_k^i \propto w_{k-1}^i \frac{p(z_k|x_k^i)p(x_k^i|x_k^i_{k-1})}{q(x_k^i|x_{0:k-1}^i, z_{1:k})} \tag{3.15}
\]

If \( q(x_k|x_{0:k-1}, z_{1:k}) = q(x_k|x_{0:k-1}, z_k) \), the proposal distribution becomes dependent only on \( x_{k-1} \) and \( z_k \). In this case, we only need to store \( x_k^i \) and discard the path \( x_{0:k-1}^i \) and the history of measurements \( z_{1:k-1} \), while the weight update equation becomes:

\[
w_k^i \propto w_{k-1}^i \frac{p(z_k|x_k^i)p(x_k^i|x_k^i_{k-1})}{q(x_k^i|x_{0:k-1}^i, z_k)} \tag{3.16}
\]

and the required PDF can be approximated:

\[
p(x_k|z_{1:k}) \approx \sum_{i=1}^{N_s} w_k^i \delta(x_k - x_k^i) \tag{3.17}
\]
As the number of particles increases, that is, \( N \to \infty \), the emerging estimate gets closer to the true PDF \( p(x_k|z_{1:k}) \), and the performance of the SIS filter increases.

As mentioned above, most particle filtering algorithms are variants of the basic SIS algorithm. One of these algorithms is the Sampling Importance Resampling (SIR) algorithm, introduced by Gordon et al. [42]. In the SIR algorithm, the importance density \( q(x_k|x_{k-1},z_k) \) is taken to be the prior density \( p(x_k|x_{k-1}) \). As a result, the update equations for the particles are given by:

\[
x_i^k \sim p(x_k|x_{k-1}^i) \quad (3.18)
\]

\[
w_i^k \propto w_{i-1}^k p(z_k|x_i^k) \quad (3.19)
\]

In the SIR algorithm, at each time index, a resampling step follows the update steps. After resampling, all weights \( w_{i-1}^k \) become equal to \( \frac{1}{N} \). Therefore, the \( w_{i-1}^k \) in equation (3.19) disappears and we have:

\[
w_i^k \propto p(z_k|x_i^k) \quad (3.20)
\]

### 3.3.3 Hill Climbing

In general, optimisation means to find the best solution from a set of possible solutions [43] to certain mathematically defined problems. The simplest form of optimisation problems consists of maximizing or minimizing a real function. Many of the problems that we encounter throughout our lives are optimisation problems. The most advanced and challenging applications of optimisation are in the areas of chemical reactor design, aero-engine design, structural design, resource allocation and scheduling.

Discrete optimization is a branch of optimization in computer science. In discrete optimisation, the problem’s variables are restricted to take only discrete values. In these problems, we can find the best solution by examining each possible solution individually. While this approach, which is called brute-force search, is guaranteed to find the best solution, in most cases and especially in real-world problems, it cannot be applied in practice due to its computational cost. Therefore, in such problems, it is necessary to use algorithms that are able to find the best or a high-quality solution without requiring to search the entire space. The Hill Climbing algorithm, Simulated Annealing and the Genetic Algorithm are three algorithms that are widely used for this purpose.

Hill Climbing is one of the simplest optimisation techniques [44]. In its purest form, the algorithm generates a new solution by performing the minimum possible modification to a single element of the solution and if the emerging solution is better than the previously found, the old solution is replaced. Otherwise, the current solution is retained and the new solution is discarded. The process is repeated for a certain number of iterations or until a good solution has been found. Hill Climbing is simply a loop that continually moves towards better solutions. An implementation of Hill Climbing is demonstrated in Algorithm 1 where next() is a function that modifies as little as possible a single element of the best_solution and quality() is a function that returns the quality of a solution. The algorithm gets stuck when it finds a solution whose immediate neighbours have lower values (Figure 3.1). Moreover, it stores no
Algorithm 1 Hill Climbing Algorithm

\[
\begin{align*}
\text{best_solution} & \leftarrow \text{ initialise_solution()} \\
\text{repeat} \\
& \quad \text{ new_solution} \leftarrow \text{ next()} \\
& \quad \text{ if quality(new_solution) } > \text{ quality(best_solution)} \text{ then} \\
& \quad & \quad \text{ best_solution } \leftarrow \text{ new_solution} \\
& \quad \text{ end if} \\
\text{until} \text{ termination condition met}
\end{align*}
\]

Figure 3.1: Hill Climbing algorithm stuck in local maximum.

information about the previous solutions that it encountered and it does not look ahead beyond the immediate neighbors of the current state.

For the purposes of our problem, we had to implement a variation of the general Hill Climber that was described above. The solution of the problem consists of a specific number of integers. The initial solution is not generated in a random manner, but all the integers are set equal to a specific number. In each iteration, a number of new solutions is generated by adding or subtracting 1 from each one of the integers of the current solution. The emerging solutions are evaluated and the best of them is compared with the current solution. If it is better than the current solution, it replaces it and the algorithm continues to the next iteration. The process is repeated for a certain number of iterations which depends on the available time or until a newly generated best solution is worse than the current solution.

3.4 Summary

In this Chapter, we reviewed work that has been done in the field of sponsored search before describing our study in detail. We initially offered an insight into the most popular auction mechanisms and we presented a number of bidding strategies both general purpose and employed by TAC/AA agents. In an attempt to pave the way for the next Chapter in which we will give a detailed description of the agents that were implemented as part of this project, we presented three techniques that constitute key components of our agents’ design.
Chapter 4

Bidding Agents for Sponsored Search Auctions

In this Chapter we describe the different versions of our agent. We first present the basic architecture of our agent which forms the foundation upon which we build our next agents. Subsequently, we present upgraded versions of the agent emphasising the new features that are added with each version. The Chapter starts with the most simple, low-performing agent and ends with the most complex and efficient agent that constitutes the flagship of our agents. The different versions of our agent are shown in Figure 4.1.

![Figure 4.1: Different versions of our agent.](image)

4.1 **stuBID_Version1**

Initially, we tried to implement a model-free agent, that is, an agent that completely ignored the user population and the parameters and details provided by the game specifications. Although it is known that model-free approaches cannot achieve the same levels of performance as the top-scoring TAC/AA agents [38], the design and implementation of such an agent is a first step towards the implementation of more complex systems. Specifically, the agent uses a rather naive approach to estimate the
next day’s cost and profit for each query. It models the problem as a multiple-choice knapsack (MCKP) problem and solves it greedily.

We considered important to spend some of the first days of the game to learn the number of agents competing for each query and the bids that we have to place to win specific positions in the 16 distinct auctions that run for the 16 queries. Moreover, in this way, we can acquire information about the conversions and revenue that each position of an auction can generate as well as their relative costs. As a result, in the first five days of each game instance, we place random bids on all the auctions. These bids vary within the range $1 – $1.5. In order to determine that range, we ran a series of experiments in which we recorded the bids of our competing agents. We then plotted these bids and we observed that in the majority of the games played, they lie between these limits. Based on these plots, we determined the appropriate bids that will allow us to explore efficiently the bidding space and to become aware of the other advertisers’ bids. Although, it is almost impossible to examine all the available positions in this way, we can get an idea of which auctions are most profitable and the potential conversions and costs that we expect by bidding on them. It would be desirable to repeat this learning process at regular intervals during the game. However, due to the fact that the duration of a game instance is quite short, re-exploring the bidding space negatively affects the agent’s performance (see section 5.6.3).

The type of the advertisements that we choose to display depends on the stage of the game. In the first five days, we do not set spend limits and we place bids on all the queries. As a result, it is possible to end up with more conversions than desired. In order to limit the number of conversions in the first five days, we choose generic advertisements for all the queries, as generic advertisements lead to fewer clicks compared to targeted and therefore, to fewer conversions. In the rest of the game, for the F0 queries, we choose generic advertisements, while for the F1 and the F2 queries, we choose targeted advertisements. More specifically, for the F2 queries, we choose targeted advertisements whose manufacturers and components match the corresponding manufacturers and components of the queries. For the F1 queries, if the manufacturer field is empty, we choose a targeted advertisement whose component matches the component of the query and whose manufacturer is our manufacturer specialty. If, on the other hand, the component field is empty, we choose a targeted advertisement whose manufacturer matches the manufacturer of the query and whose component is our component specialty.

After the initial learning stage, the strategy of our agent is to place bids that win the most profitable positions from all the auctions while at the same time taking into consideration the distribution constraint effect which according to Jordan et al [45] is one of the most important factors that determine the performance of an agent. Specifically, in order to determine the auctions and their corresponding positions that can lead to profit maximisation, we formulated the problem as a multiple-choice knapsack (MCKP) problem and solved it using a greedy algorithm. The MCKP approach has also been used by the agent designed by Berg et al. [40], however, we look at the problem from a different perspective. Our constraint in the MCKP model is not the number of conversions, but the maximum cost which is determined by the available budget. Budgets are the norm in real Internet advertising auctions as often advertisers have liquidity issues. However, these issues are not incorporated in the TAC/AA game.
Although advertisers are allowed to set budgets for the various queries, how these budgets will be determined is a big question. The correlation of the aggregate budget with the desired number of conversions was a reasonable solution. Specifically, we followed a very simple approach. We tried to correlate the agent’s total cost with its total conversions by calculating the fraction \( \frac{\text{total cost}}{\text{total conversions}} \) and updating it each time a new report from the publisher is received. In this way, provided that we have calculated our desired number of conversions (see next paragraph), we can set our budget equal to:

\[
\text{Budget} = \frac{\text{total cost}}{\text{total conversions}} \times \text{target conversions}
\]  

We should note that in this version of our agent, the budget is not a hard constraint. It is used as a constraint in the MCKP formulation of the problem, but the total cost is allowed to exceed it as no spend limits are set.

As regards the target conversions, that is, the number of products that we aim at selling, we have used two different approaches. In the first approach, we keep it constant and equal to \( \frac{\text{distribution capacity}}{\text{distribution window}} \). In this case, the conversions are relatively stable and the used capacity is close to 100%. However, if the conversions are lower than the critical distribution capacity, the agent will not make an attempt to exploit the full capacity potential, losing a potential extra profit. In the second approach, we add the conversions of the previous three days plus the conversions that we expect to carry out during the current day (previous day’s target conversions) and we set the next day’s target conversions equal to the distribution capacity minus this sum. This approach attempts to fully exploit the available capacity and addresses the inability of the first approach to better utilise the available capacity in cases where the number of conversions in the previous four days was lower than the desired. However, when this number is much lower that the aggregate target conversions of the previous four days, the target conversions oscillate between selling too many products in one day and selling almost no product in the next four days. The first approach proved more efficient and was incorporated into the agent.

As regards the optimisation part, as mentioned above, we modeled the problem as an MCKP. As mentioned in section 3.3.1, in an MCKP, there are a number of classes and each class contains one or more items. Each item has a corresponding weight and value. Our aim is to fill the knapsack using at most one item from each class so as to maximize the value that we can get from the items while having a total weight less than or equal to the knapsack size. In our problem, there are 16 classes, the 16 distinct auctions. Each class contains a number of items, the positions that the agent can win in the auction. Each item has a corresponding weight which is equal to the expected cost that will emerge if this specific position is won and a corresponding value which is equal to the average profit-per-click of the position multiplied by the expected clicks. Our aim is to select the positions (at most one for each query) that maximise the agent’s profit while keeping the total cost less or equal than the available budget. Hence, the problem can be written as below:

\[
\max \sum_{i=1}^{k} \sum_{j=1}^{m} \text{profitPerClick}_{ij} \cdot \text{clicks}_{ij} \cdot x_{ij}
\]  

(4.2)
\[
\sum_{i=1}^{k} \sum_{j=1}^{m} \text{cost}_{ij} \cdot x_{ij} \leq \text{Budget} \tag{4.3}
\]

\[
\sum_{j=1}^{m} x_{ij} \leq 1, \text{ for all } 1 \leq i \leq k \tag{4.4}
\]

\[x_{ij} \in \{0, 1\}, \text{ for all } 1 \leq i \leq k \text{ and } 1 \leq j \leq m \tag{4.5}\]

Figure 4.2 illustrates the emerging MCKP problem. As mentioned earlier, the MCKP can be solved greedily (see section 3.3.1). After solving the MCKP, we get a target position for each auction. For some auctions that do not have the potential to lead to high profits, there will be no target positions.

Our goal is to maximise the agent’s profit while staying under budget. However, due to the fact that in this version of the agent, no spend limits are set, it is not certain that the budget will not be violated. In order to calculate the expected profit of a position, we use information from the previous days about the position’s profit-per-click along with its expected number of clicks. The estimated profit-per-click of a position is set equal to the average profit-per-click of the position. After running a lot of game instances, we observed that in general, for each position, the values of profit-per-click remain roughly the same throughout each game instance and therefore, we can use their average value in order to estimate the expected profit of the position. The estimated cost of a position is set equal to the average cost that the agent incurred while placed in that position. The expected number of clicks depend on the estimated cost and we used a regression scheme to approximate it. Assuming that this data is representative of the events that will take place in the next day, the greedy algorithm generates a solution that contains the positions that we should target for in order to maximise our profit. Of course, to be able to generate a more reliable solution, it is
necessary that the agent has been placed in most of the positions of the auctions, if not in all of them.

The next problem the agent has to face is to determine the amount of money it is going to bid in order to win the desired positions. This is a very complex task as the agent’s bids must take into account the bids of its competitors which change from time to time and cannot be predicted. In order to estimate the required bid to win a specific position, we used the agent’s history of bids. All the bids that the agent submitted from the beginning of the game until the current day are taken into account. Specifically, after the initial placement of random bids, there is an indication on which bids win which positions. The bids are set using this information as well as more recent data. In particular, they are determined using an exponentially weighted moving average:

\[
\text{bid} = \frac{\sum_{i=1}^{k} w_i \cdot \text{bid}_i}{\sum_{i=1}^{k} w_i}
\]  

An exponential function is used to weight the past bids, so that observations taken further in the past receive less weight. The exponential weighting is given by: \( w = 0.95^{t_n - t_0} \) where \( t_n - t_0 \) is the difference in days between the next day and the day whose observation is currently being weighted. If after the last time the agent was placed in the desired position, it won a position immediately above or below this position, the bid that it submitted is compared to the exponentially weighted moving average and if required, the bid that it is going to submit is adjusted to correspond to the new market prices that are determined by the other advertisers’ bids. Specifically, if the agent won a higher position in the results page by placing a lower bid than the exponentially weighted moving average, the bid that the agent is going to place is set equal to this bid minus a random amount between $0 and $0.1. If, on the other hand, the agent won a lower position than the target by submitting a higher bid than the exponentially weighted moving average, the bid that it is going to place is re-adjusted and set equal to the bid that led to the lower position plus a random amount between $0.1 and $0.2. In the case that the agent has never been placed in the desired position and as a result, there are no past bids available indicating the amount it has to bid to win the position, we calculate the exponentially weighted moving average of the closest higher and lower won positions and we use linear interpolation to determine it.

### 4.2 stuBID Version2

The first step towards a more advanced model-free agent was the addition of query specific spend limits. We considered necessary to add spend limits to our agent’s design, as in real scenarios, budgets are practically always present and as it was evident that they could deter certain factors from decreasing the agent’s profit. By setting spend limits, the complexity of the agent would slightly increase, while the benefit in performance could be considerable.

Setting these spend limits was a very simple task as their values arise directly from the estimated costs that were used in the MCKP formulation. The greedy algorithm that is responsible for solving the MCKP fills the knapsack with at most one position
from each auction. Each position has an expected cost that we are going to incur if placed in it and the aggregate cost of all the positions that are selected by the greedy algorithm cannot exceed the total budget. For all the queries which have no positions selected by the greedy algorithm, their spend limits are set equal to zero. For all the other queries, the spend limits are equal to the estimated cost of their selected position. In the previous version of our agent, the estimated cost of a position was set equal to the average cost that the agent incurred while placed in that position. In this version, because of the spend limits, we know that the cost that the agent will pay, while placed in a position, will be at most equal to the budget that was set for the corresponding query. If the cost incurred the last time the agent won the position was equal to the budget that had been set, it is very likely that we could sell more product from the query and that the budget constituted an inhibiting factor. Hence, when this is the case for the most profitable queries, we increase the next day’s estimated cost by a specific amount ($20). In this way, if a profitable position restricted by a budget constraint has the potential to lead to more conversions, we increase its budget to take full advantage of it.

In the following paragraphs, we analyse how the budgets affect the agent’s performance. When we set a budget for a specific query and it is not exceeded, the corresponding advertisement is displayed for every impression (i.e., all the users that submit the query see the advertisement). In these cases, the spend limit does not play any role in the agent’s performance and it was not necessary to set it. The agent’s revenue and cost for the specific query would be exactly the same even if no send limit had been set. On the other hand, if the budget is exceeded, it will have either a positive or a negative impact on the agent’s performance.

Initially, we present the cases in which setting query specific spending limits can contribute to the increase of the agent’s profit. This is achieved by keeping the cost below specific levels. More specifically, in the cases when a burst in one of the nine subpopulations happens, a large number of users transition from the non-informational state (NS) to the informational state (IS). These users will submit queries relative to their preferred products and will click on the displayed advertisements, but they will not convert. These users can cause a great performance drop in the agent’s performance. The higher the position won, the greater the number of the informational users that will click on the advertisement and as a result the cost will be high. Using a spend limit for the query will remove the agent’s advertisement when a certain number of users have clicked on it and will not allow the cost to surpass a specific level. Moreover, as the number of conversions that the agent is going to carry out depends on the position it is going to win, the agent may sell much more products than desired if placed in a higher position than the one it aimed at. The query specific budgets contribute to keeping the number of conversions to the desired levels. In this way, the agent avoids selling too many products which will make users less eager to purchase their preferred product from it. The same applies for the case in which the estimated cost of displaying an advertisement for every impression of a query is much lower than the real cost. Setting a budget removes the advertisement when the estimated cost is exceeded. If the query is one of the most profitable, removing the advertisement can affect negatively the resulting profit. On the other hand, for the less profitable queries, the query specific spend limit prevents the agent from selling more low-profit products than desired. The advantages of setting budgets that were described above show that
the main purpose of setting them is as a form of safety in the cases where the cost from winning a position in a specific auction is much greater than the cost that was estimated.

Except from an increase in performance, setting a query specific spend limit can also cause a performance drop. This happens when the agent has the potential to sell more products through an advertisement that offers it high profits for each sold product, but the advertisement is removed because the cost has reached the budget that was set for the specific query.

4.3 stuBID_Version3

The model-free agent that was described in the previous section performs extremely well if we consider the way it estimates the expected profits-per-click and costs. As will be shown in the next Chapter, it managed to outperform most of the agents that participated in the previous TAC/AA competitions. This creates a lot of expectations about the performance of the agent if we attempt to generate more accurate estimations of these values. A good model of the user populations can offer better estimates and greatly increase the performance of the agent. In this version of our agent, we removed the budget constraints and we used particle filters to model the states of the user populations. We also added a hill-climbing algorithm that allows the agent to better manage the number of conversions it accomplishes.

In order to be able to make more accurate estimates about the next day’s events, we decided to use particle filters to model the states of the users. Following the approach of Pardoe et al. [34], we implemented 9 particle filters, one for each one of the 9 subpopulations. Each subpopulation consists of 10,000 users which are interested in only one of the 9 distinct products and is modelled as a Markov chain. These users are distributed among the six individual user states (NS, IS, F0, F1, F2, and T). Each filter consists of 1,000 particles and each particle represents a distribution of the users among the user states. Before the start of a game instance the particles are updated 10 times in order to reflect the possible populations resulting from the initialization process performed by the game server. Each succeeding day, the particles that had been generated to reflect the previous day’s distribution of the population are re-weighted based on the reports provided by the publisher. The information that plays the most crucial role in determining the distribution of the populations is the total impressions for the F2 queries, that is, the queries that specify both the manufacturer and the component. This information is enough to allow us to generate relatively accurate estimations of the user populations. Specifically, the total number of impressions of an F2 query is equal to the number of F2 users plus the IS users that choose to submit an F2 query. Each IS user can choose between submitting an F0, F1 or F2 query with equal probability. Hence, the total impressions are equal to \( F2_{\text{users}} + \frac{1}{3} IS_{\text{users}} \). The probability of observing \( N \) total impressions when there are \( x_{IS} \) IS users and \( x_{F2} \) F2 users is the probability that \( N - x_{F2} \) users of the IS users choose to submit an F2 query. This probability can be determined from the binomial distribution \( B(x_{IS}, 1/3) \) and can be estimated using the normal approximation \( N(x_{IS}/3, 2x_{IS}/9) \). The weights of the particles are set to this probability and are then normalized to sum to one.
The emerging particles are then resampled. A new set of particles is drawn from the previous set. Particles with large weights are likely to be drawn multiple times, while particles with very small weights are not likely to be drawn at all. The weights of the new particles are equal to \( \frac{1}{1000} \). After the resampling step, the set of particles that emerges represents the estimated probability distribution over the user population state on the previous day. We are interested in the expected user population on the next day. Hence, we update each particle 2 times and take the weighted average of the particles. When updating a particle, the new particle’s user distribution is randomly generated from the old particle based on the user transition dynamics. All these dynamics are given except from the probability of a user transitioning to the Transacted (T) state as this probability depends on the advertisements that the agents choose to display. Although it is not known, this probability can be estimated fairly accurately using data from past games.

Knowing the distribution of the users among the user states, we can calculate the estimated impressions, clicks, conversions, cost and revenue for all the positions of all the queries using the equations that were presented in Chapter 2. Some of the parameters in these equations are unknown. However, we are given the limits within which they lie \([19]\) and we replace them with their average values. Because of these approximations, the estimations cannot be precise, but the error is still small so that we can use these estimated values. We also attempted to track the other advertiser’s preferred queries and the advertisements they choose to display in order to become aware of the number of advertisers that submit bids on each auction which acts as an indicator of the less crowded auctions.

The problem that emerges at this point is that in order to get the total number of impressions for a specific query, the agent’s advertisement should be displayed for every impression. Displaying advertisements for the 9 F2 queries for every impression will lead to a large performance degradation as some of these queries will not be as profitable as the others and the number of sales will greatly exceed the critical distribution capacity. Moreover, this constraint does not allow us to set budgets as they will prevent the agent from getting aware of the total number of impressions of the F2 queries.

In order to address this problem, Pardoe et al. \([34]\) implemented a position analyser which allowed them to compute the total number of impressions for each query. We followed a completely different approach: we bid only on the queries that the agent expects to be the most profitable and as a result, we only win advertising positions in a subset of the F2 queries. For the rest of the F2 queries, no advertisements are displayed and we are not provided the total number of impressions and we cannot re-weight the previous day’s particles of the corresponding populations. In this case, we just update the corresponding filters. Only when the agent wins a position in an auction for an F2 query, we are able to re-weight the particles of the corresponding filter. Algorithm 2 demonstrates a pseudo-code of the particle filters’ updating process. Although we lose a lot of information about the distribution of the population, surprisingly, after winning a position in an F2 auction and receiving the total impressions, the predictions of the filter can get again close to the real distributions of the populations and the performance of the agent is very competitive.
Algorithm 2 Particle filters’ updating process

\[
\begin{align*}
&i \leftarrow 1 \\
&\textbf{while } i < 10 \textbf{ do} \\
&\quad \textbf{if } \text{impressions}(F_2, \text{query}_i) > 0 \textbf{ then} \\
&\quad \quad \text{re-weight()} \\
&\quad \quad \text{resample()} \\
&\quad \quad \text{update()} \\
&\quad \textbf{else} \\
&\quad \quad \text{update()} \\
&\quad \quad \text{update()} \\
&\quad \textbf{end if} \\
&\quad i \leftarrow i + 1 \\
&\textbf{end while}
\end{align*}
\]

Except from the particle filters that estimate the distribution of the users in the 9 sub-populations, we also added a hill-climber to calculate the next day’s target conversions. Specifically, each day, we approximate the optimal set of conversion targets for the next 10 days and use the next day’s value as the next day’s target conversions. We could calculate the set of conversion targets for the remainder of the game instead of the next 10 days, however, due to the time restrictions of the game (each day lasts for 10 seconds), we decided to use the aforementioned 10-days window and optimise for more iterations. As mentioned above, in order to solve this optimisation problem, we used a hill-climber. We begin by setting all the conversion targets equal to \(c_d\). Then, we generate 10 new solutions by increasing or decreasing the conversion target of each one of the next 10 days by 1 and compute their fitness. The most profitable deviation over all the generated solutions is chosen and the process is repeated for a number of iterations. An implementation of the hill climbing algorithm is demonstrated in Algorithm 3 where \(\text{generate_random}()\) is a function that generates a random number between 0 and 1 and \(\text{quality}()\) is a function that returns the quality of a solution.

The number of the next day’s target conversions is set equal to the corresponding number provided by the fittest solution. In order to compute the fitness of a solution, except from the numbers of conversions that are present in the solution, we also take into account the conversions of the previous 3 days and the conversions having occurred during the current day. For each conversion during the day, the fitness of the day is updated according to:

\[
\text{fitness}_{\text{day}} = \begin{cases} 
\text{fitness}_{\text{day}} + \lambda ((\sum_{d=d-4}^d c_i) - C_{\text{cap}}) & \text{if } (\sum_{d=d-4}^d c_i) - C_{\text{cap}} > 0 \\
\text{fitness}_{\text{day}} + 1 & \text{otherwise}
\end{cases}
\]

where \(c_d\) is the total number of conversions on day \(d\) and \(C_{\text{cap}}\) is the critical distribution capacity beyond which conversion rates start decreasing. Due to the fact that at the current day, we do not know the actual conversions of the previous day, we use the previous day’s target conversions instead. The total fitness of the solution is given by: \(\text{fitness} = \text{fitness}_{\text{current day}+1} + \text{fitness}_{\text{current day}+2} + \ldots + \text{fitness}_{\text{current day}+10}\). Figures 4.3, 4.4 illustrate how the fitness of a specific day and the total fitness of a solution are...
Algorithm 3 Modified Hill Climbing Algorithm

```
best_solution ← initialise_solution()

i ← 1

repeat
    while i ≤ 10 do
        if generate_random() < 0.5 then
            new_solution_i ← best_solution[i] + 1
        else
            new_solution_i ← best_solution[i] - 1
        end if
        i ← i + 1
    end while

i ← 1

while i ≤ 10 do
    if quality(new_solution_i) > quality(best_solution) then
        best_solution ← new_solution_i
    end if
end while

until termination condition met
```

calculated. This approach exploits to a greater extent the available capacity and can

Figure 4.3: Fitness of a specific day.

exploit the inability of the previous approaches to better utilise the available capacity.
4.4 stuBID_Verison4

As a next step to the design of our agent, we decided to set query specific budgets as it was our belief that these spend limits can increase the performance of the agent. The main problem in setting query specific budget is that the accuracy of the particle filters in estimating the distribution of the user populations will degrade to a large extent. A particle filter can be updated properly only if the advertisement for the corresponding F2 query is displayed for every impression which will allow the re-weighting of the particles such that they reflect the true distribution of the users among the various states.

Setting query specific budgets may not allow the agent to get aware of the total number of impressions, as when the budget is exceeded its advertisement is removed. In order to be able to re-weight the particles as often as possible, the budget for every query of interest is set equal to the estimated cost that the agent is going to pay if the desired position is won increased by a small amount ($20). The estimated costs are calculated as described in the previous section and the small amount is added in order to increase the probability that the cost will not exceed the budget. More specifically, if the estimated costs are accurate and the agent wins the desired or a lower position in the search page, it will probably have its advertisements displayed for every impression and will be able to successfully re-weight its particles. If, on the other hand, the estimated costs are lower than the real costs or the agent wins a position higher in the search page than its desired position, it will probably exceed its spend limit for the specific query before the end of the day and the number of impressions will not
be informative enough to allow the agent to re-weight the particles. In this case, the particles that represent the previous day’s distribution of the users are updated according to the user transition dynamics without being re-weighted and resampled at first. When a report describing the previous day’s events is received, the cost incurred from displaying an advertisement for an F2 query is compared with the query specific budget that was set. If the budget was not exceeded, we know that the advertisement was displayed for every impression. As a result, the particles of the corresponding filter are re-weighted using the number of impressions provided by the publisher which is equal to the total number of impressions. If the cost was equal to the budget, the total number of impressions is not available as the number provided by the publisher is lower than the total impressions and therefore, there is no sufficient information to re-weight the particles and keep the user model accurate. An implementation of the particle filter’s updating algorithm is demonstrated in Algorithm 4.

Algorithm 4 Particle filters’ updating process

```plaintext
i ← 1
while i ≤ 9 do
    if impressions(F2_query_i) > 0 then
        if cost(F2_query_i) < budget(F2_query_i) then
            re-weight()
            resample()
            update()
            update()
        else
            update()
            update()
        end if
    else
        update()
        update()
    end if
    i ← i + 1
end while
```

4.5 Summary

This Chapter provided a detailed description of the four agents that we implemented as part of this study. We presented the adopted bidding strategies and we highlighted the most important parts of the agents. In the next Chapter, we present and discuss the results that emerged from evaluating the agents as well as how changes in various components of the agents affect their performance.
Chapter 5

Experiments

In this Chapter we investigate the performance of the agents that were described in the previous Chapter. More specifically, a number of controlled experiments were designed to evaluate their performance and measure different aspects of the adopted bidding strategies. We test the accuracy of the particle filters and the estimators as well as how changes in some parameters of the different components of our agents affect the achieved scores. We present the results and try to analyse the reasons for the performance gain that we achieved after inserting new components into our agent’s design and to shed light on the relative importance of the various components.

5.1 Experimental Setting

In order to get reliable results, we had to test our agents against other agents that have proved to be efficient. The fact that many teams that competed in TAC/AA competitions have submitted binaries of their agents to the TAC agent repository\(^1\) contributed to this end. Specifically, we downloaded the binaries of 7 high-performing agents that participated in previous TAC/AA competitions and tested our agents against them. In order to test the performance of each version of our agent, we organised a tournament in which our agent along with the other 7 agents participated in 60 games. These tournaments are similar to the final round of the TAC/AA competition.

5.2 General Performance

The first group of experiments concerns the general performance of our agents. The scores from the four tournaments that were organised are shown in Table 5.1. The first agent that we implemented (stuBID_Version1) performed worse than all its competitors. The difference in performance especially in comparison to the top scoring agents was large. From the analysis of the games played, we observed that the distribution constraint effect was the main factor that kept the agent’s performance to low

\(^1\)http://tac.sics.se/showagents.php
Table 5.1: Average Performance (60 actual games with best repository agents)

<table>
<thead>
<tr>
<th>Agent</th>
<th>Set 1</th>
<th>Set 2</th>
<th>Set 3</th>
<th>Set 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>TacTex</td>
<td>55.797</td>
<td>55.720</td>
<td>56.121</td>
<td>55.461</td>
</tr>
<tr>
<td>stuBID_Version4</td>
<td></td>
<td></td>
<td></td>
<td>54.236</td>
</tr>
<tr>
<td>Mertacor</td>
<td>54.314</td>
<td>55.368</td>
<td>53.060</td>
<td>53.378</td>
</tr>
<tr>
<td>Schlemazl</td>
<td>53.927</td>
<td>54.949</td>
<td>53.744</td>
<td>54.033</td>
</tr>
<tr>
<td>stuBID_Version3</td>
<td></td>
<td></td>
<td></td>
<td>48.459</td>
</tr>
<tr>
<td>CrocodileAgent</td>
<td>44.864</td>
<td>48.109</td>
<td>46.333</td>
<td>47.511</td>
</tr>
<tr>
<td>stuBID_Version2</td>
<td></td>
<td></td>
<td></td>
<td>47.135</td>
</tr>
<tr>
<td>AstonTAC</td>
<td>44.709</td>
<td>42.281</td>
<td>43.800</td>
<td>46.388</td>
</tr>
<tr>
<td>DNAgent</td>
<td>40.744</td>
<td>42.174</td>
<td>39.867</td>
<td>42.372</td>
</tr>
<tr>
<td>epflagent</td>
<td>40.074</td>
<td>43.421</td>
<td>43.306</td>
<td>43.249</td>
</tr>
<tr>
<td>stuBID_Version1</td>
<td>33.776</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

levels. The agent sold more products than expected and the users were less inclined to buy decreasing the incurred revenue. The main reason that caused the increase in conversions was that the incurred cost was greater than the estimated. Due to the fact that the number of conversions depends on the incurred cost, the increased cost led to more conversions than expected and therefore, to the distribution problem.

In the next version of our agent (stuBID_Version2), we attempted to alleviate this problem by setting query specific spend limits. The agent remained simple, however, it also proved robust and it exceeded our expectations as it managed to outperform the majority of the agents that participated in previous TAC/AA competitions. From the results, we can conclude that the increase in performance that the query specific spend limits offer is quite high.

In the third version of our agent (stuBID_Version3), we added the particle filters, a new estimator and the hill climbing algorithm and we removed the query specific budget constraints. The performance of this agent was satisfactory as it remained at the same level as the performance of the second version (stuBID_Version2). The difference in performance between this version and the first version (stuBID_Version1) highlights the contribution of the new components that were added. From the analysis of the games, we uncovered that also in this version of our agent, the main limiting factor was the distribution constraint effect. The difference between the estimated and the experienced costs are mainly due to the inaccurate predictions of the particle filters and the error of the estimator that calculates the expected cost and the expected number of clicks.

Although the predictions were not as imprecise as in the first version (stuBID_Version1), the agent’s performance has the potential to increase if the number of conversions is kept to the desired levels. Hence, once again, our solution to this problem was to set query specific budget constraints. The introduction of query specific spend limits (stuBID_Version4) improves the agent’s performance to a large extent as the spend limits can guarantee that the next day’s cost will not be greater than the desired cost and therefore, that the number of conversions will be close to their desired number.

In general, as we can see from the results, setting spend limits for the various keywords, in which the agent is interested, offers a great boost in its performance. The increase
in the achieved profits after adding a spend limit show that the cases in which the budget affects negatively the agent’s performance are much more rare than the cases in which it avoids costs reaching high levels and that the extra profit that was not gained is much lower than the extra cost that the agent would have to pay if there were no spend limits. Hence, we can say that setting budgets is very beneficial and it offers a level of safety which guarantees that the agent’s performance will not vary significantly between the various game instances.

5.3 Particle Filters

As we saw previously, the particle filters that we used to maintain estimates of the distribution of the users among the six individual user states in the 9 subpopulations contributed significantly in the increase of the agent’s performance. Specifically, adding the particle filters to the agent’s design and combining them with query specific spend limits offered a boost to the agent’s performance and allowed it to reach the performance levels of the top-scoring agents. This increase in performance showed that it is very important to predict the users’ needs and desires when tackling the budget optimisation problem. In a real scenario, the bidding history and statistics provided by the search engines to the advertisers about the keywords of their interest can offer a valuable source of information for predicting searching trends of the future.

In order to verify the proper functioning of the particle filters, we tested the accuracy of the particle filters in estimating the distribution of the users among the 6 user states. Specifically, we run a series of experiments in which we compared the estimated distributions of the subpopulations with the actual distributions. In these experiments, the advertisements were displayed for every impression and the particles were updated at each time step using the total number of impressions provided by the publisher. Figure 5.1 shows the estimated and actual number of users in the F0, F1, and F2 states for one product in a randomly chosen game. We can see that the filter’s estimates are fairly accurate. Figure 5.2 illustrates the estimated and actual number of users in the F0, F1, and F2 states for one product in another randomly chosen game. In this game, the estimated distribution of the users which are interested in the product in the various states is close to the actual distribution, but the estimates are less accurate than these in the previous case.

In both versions of our agents that use particle filters to maintain estimates of the user subpopulations (stuBID_Version3 and stuBID_Version4), the total number of impressions is not known for all the F2 queries. Specifically, the agent that does not make use of query specific budget constraints (stuBID_Version3) displays its advertisements for every impression, but it does not display advertisements for all the 9 F2 queries. As a result, some of the filters that represent the distribution of a user population are just updated, thus increasing the uncertainty. In the final version (stuBID_Version4), except from the fact that not all the advertisements are displayed at each time step, there are also budget constraints. When the budget of a query is exceeded, the corresponding advertisement is removed and the total number of impressions cannot be calculated precisely. Only in the case when the budget is greater than the cost of the total clicks, we are able to observe the total number of impressions and use it to re-weight the
particles. It is very interesting to test the accuracy of the particle filters in these cases, as, although the majority of the filters are blindly updated at most time steps, the improvement in performance compared to the model-free version, shows that the estimates of the filters are not completely irrelevant. In general, we observed that in most days, the agent places bids on the same queries and as a result, the particles of the corresponding filters are frequently re-weighted. Of course, they are not re-weighted everytime the agent wins a position in the auction due to the budgets that are set, but they are updated several times during a game instance. Figure 5.3 shows the estimated and actual number of users in the F0, F1, and F2 states for such a product. We can see that the estimated number of users in the three states are close to the actual numbers, which confirms our conjecture that the agent places bids mostly on the same queries. For all the other F2 queries on which the agent rarely places bids, the filters’ estimates are imprecise. The filters are able to repair their estimations provided they are given the total number of impressions for the corresponding F2 query, however, due to the fact that these numbers are not received on a regular basis leads the filters back to inaccurate predictions. Figure 5.4 illustrates the estimated and actual number of users in the F0, F1, and F2 states for a product that the agent rarely chose to advertise. We can see that in the first five days in which the agent places bids on all the F2 queries without setting spend limits, the filter’s estimates are accurate. In the rest of the game, the agent is not interested in the specific F2 query, it does not display an advertisement.
for it and due to the fact that it does not receive information about the number of impressions, the filter’s predictions are way off. We can also see that at some point, the agent displayed an advertisement for the whole day and its estimated numbers returned close to the actual numbers. However, after this point, they once again diverged away from the true values. Although the estimated distributions of the users among the various states in populations whose total impressions are rarely been acquired are not accurate, as mentioned above, in most cases the agent concentrates on the most profitable queries and submit bids on them almost every day. As a result, it receives the total impressions for the corresponding populations on a regular basis and as a result, the estimated distributions of the users of these populations are close to the real distributions. This is the reason why the agent’s performance remains high, even if we set query specific spend limits.

5.4 Estimators

Due to the increase in performance that we observed after adding the particle filters that model the states of the user populations, we consider it very interesting to examine the accuracy of the estimators that calculate the expected costs, conversions and profits before and after adding the filters. Our initial estimators (stuBID_Version1 and

Figure 5.2: Example of user population estimates for a second product where the particles are always re-weighted.
Figure 5.3: Example of user population estimates for a product where the particles are often re-weighted.

stuBID_Version2) were very simple. Specifically, they set the expected cost, conversions and profit of a position equal to the average cost, conversions and profit of that position. The estimators that were implemented after the addition of the particle filters (stuBID_Version3 and stuBID_Version4) take into account the estimated impressions that are generated by the 9 filters as well as problem-specific information (the equations provided in the game specifications [19]) to predict these values. The expected cost, conversions and profit generated by these two approaches for random queries are illustrated in Figures 5.5, 5.6 and 5.7 respectively. We can see in the figures that the values that were generated by the first estimator were unrealistic for all the examined cases. The large difference between the actual and estimated values stems from the simplicity of the approach, as the estimates are calculated improperly. The average value can reveal where approximately these values lie, but it is not possible to provide accurate estimates. On the other hand, the quality of the second estimator is much higher. More specifically, the estimated costs are fairly accurate. As we can see in Figure 5.5, they are very close to the actual costs. The accuracy decreases in the case of the estimated conversions and profits. However, these predicted values are still much more accurate than the corresponding estimates of the first estimator. The equations that were mentioned above contain some hidden game parameters and some other parameters that depend on the actions of the other advertisers. As a result, the precise estimation of each position’s cost, conversions and profit is not possible. However,
Figure 5.4: Example of user population estimates for a product where the particles are rarely re-weighted.

Figure 5.5: Estimated costs of the initial (left) and the improved (right) estimator for two random queries.
the accuracy of the estimator can be improved if we apply machine learning to better approximate these unknown parameters and if we use data from previous games to predict the other agents’ behaviour.

5.5 Distribution Capacity

One of the primary factors that affect the agent’s performance is the distribution constraint effect. Each advertiser is assigned a critical distribution capacity and if the sum of the products it sold on the previous four days plus the products it sells on the current day exceeds this value, conversion rates decrease as users are less inclined to purchase. The two components of our agent that are responsible for maintaining the sales at the desired levels are the budget constraints along with the hill climber. The hill climber generates the next day’s desired number of conversions and the budget constraints keep the actual conversions close to this number. Figure 5.8 shows the percentage of the used capacity in two random games. In the first game, the agent...
made use of budget constraints (stuBID_Version4), while in the second it did not (stuBID_Version3). We can see that when budget constraints are used, the used capacity

![Percentage of the used capacity with and without budget constraints.](image)

Figure 5.8: Percentage of the used capacity with (left) and without (right) budget constraints.

is relatively stable and a little greater than 100%. This happens because even if the agent sells more products than the critical distribution capacity, the extra profit it makes is greater than the cost that is incurred due to the fact that some users click on its advertisement, but they don’t convert. In other words, there is a trade-off between the number of conversions and the conversion rates and the hill climber attempts to keep both of them high. In the case when we do not use budget constraints, as we can see, the number of conversions can show a particularly high increase which will affect negatively the agent’s performance.

### 5.6 Performance after Applying Modifications

In the following subsections, we modify some of the agent’s components and we also change the values of some parameters in order to investigate how the agent’s performance is affected by these modifications. As in the initial experiments, we measured the impact of these changes to our agent by running, for each new agent, a tournament of 60 games in which it competed against the 7 agents of the TAC/AA repository.

#### 5.6.1 Modified Estimation Window of Bids

We decided to investigate how the agent’s performance will be affected if we modify our bid estimator. The standard bid estimator that is used in all the versions of our agent takes into account all the bids that we have submitted for a specific query from the beginning of the game instance in order to calculate the required bid to win the desired position. We implemented three new agents that look at the bidding history of only the last 10, 5 and 3 days, while estimating the bids that are going to be submitted for the next day. The general method of calculating the bids remained the same. The
Table 5.2: Average Performance (60 actual games with best repository agents)

<table>
<thead>
<tr>
<th>Agent</th>
<th>Set 1</th>
<th>Set 2</th>
<th>Set 3</th>
<th>Set 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>TacTex</td>
<td>55.461</td>
<td>59.528</td>
<td>56.847</td>
<td>55.513</td>
</tr>
<tr>
<td>stuBID_Version4_Window3</td>
<td>54.236</td>
<td></td>
<td></td>
<td>54.571</td>
</tr>
<tr>
<td>Schlemazl</td>
<td>54.033</td>
<td>55.955</td>
<td>54.575</td>
<td>54.542</td>
</tr>
<tr>
<td>stuBID_Version4_Window10</td>
<td>53.190</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mertacor</td>
<td>53.378</td>
<td>53.182</td>
<td>53.859</td>
<td>53.594</td>
</tr>
<tr>
<td>stuBID_Version4_Window5</td>
<td>47.511</td>
<td>46.937</td>
<td>47.170</td>
<td>47.147</td>
</tr>
<tr>
<td>CrocodileAgent</td>
<td>46.388</td>
<td>46.040</td>
<td>46.087</td>
<td>43.415</td>
</tr>
<tr>
<td>AstonTAC</td>
<td>43.249</td>
<td>43.989</td>
<td>40.715</td>
<td>42.580</td>
</tr>
<tr>
<td>epflagent</td>
<td>42.372</td>
<td>41.312</td>
<td>43.638</td>
<td>41.203</td>
</tr>
</tbody>
</table>

results that emerged are shown in Table 5.2. The difference in performance between the agent that takes into account the whole bidding history of the current game instance and the three agents that look only at the bids submitted the last 10, 5 and 3 days is almost negligible. The average profit of the four agents ranged between $52.302 and $54.571. We believe that the reason that the performance is not affected by the modification of the estimation window is that in general, the agents keep their bids relatively constant for the majority of the days of each game instance. Moreover, the exponential function that is used to weight the past bids, assigns much greater weights to more recent bids than to bids that were submitted far in the past. As a result, even the initial bid estimator, which takes into account the whole bidding history of the current game instance, determines the value of the bids mainly based on the most recent observations.

5.6.2 Modified Budget

Due to the fact that the adopted method of determining the next day’s budget is very simple and we cannot guarantee that the assigned values are sufficiently efficient, we considered it important to test different values. As a result, we implemented three modified versions of our agent in which we multiplied the next day’s budget by 0.9, 1.1, 1.2 and 1.3 respectively. The average profits of the agents are illustrated in Table 5.3. As we can see in the table, the average profits of the four new agents remained at the same level as the last version of our agent. The difference in performance between the five agents is insignificant. Of course, if we multiply the budget by a much greater or smaller number, the agent’s performance will be affected to a larger extent. We speculate that the reason that small modifications in the value of budget do not alter the agent’s average score is that the next day’s conversions, which depend on the set budget, cannot be strictly limited to their desired number just by setting the budget constraints. Moreover, the deficit in conversions when the budget is multiplied by 0.9 and the extra conversions when it is multiplied by 1.1, 1.2 and 1.3 are not large enough to significantly affect the agent’s performance.
Table 5.3: Average Performance (60 actual games with best repository agents)

<table>
<thead>
<tr>
<th>Agent</th>
<th>Set 1</th>
<th>Set 2</th>
<th>Set 3</th>
<th>Set 4</th>
<th>Set 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>TacTex</td>
<td>55.461</td>
<td>53.711</td>
<td>56.189</td>
<td>57.667</td>
<td>57.242</td>
</tr>
<tr>
<td>stuBID_Version4</td>
<td>54.236</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Schlemazl</td>
<td>54.033</td>
<td>56.303</td>
<td>55.842</td>
<td>54.800</td>
<td>56.016</td>
</tr>
<tr>
<td>stuBID_Version4_Budget1.2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mertacor</td>
<td>53.378</td>
<td>54.391</td>
<td>55.791</td>
<td>54.019</td>
<td>54.662</td>
</tr>
<tr>
<td>stuBID_Version4_Budget1.1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>stuBID_Version4_Budget1.3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>stuBID_Version4_Budget0.9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CrocodileAgent</td>
<td>47.511</td>
<td>48.384</td>
<td>47.157</td>
<td>47.515</td>
<td>45.699</td>
</tr>
<tr>
<td>AstonTac</td>
<td>46.388</td>
<td>44.585</td>
<td>44.215</td>
<td>46.993</td>
<td>44.191</td>
</tr>
<tr>
<td>EpflAgent</td>
<td>43.249</td>
<td>40.972</td>
<td>44.415</td>
<td>43.557</td>
<td>42.297</td>
</tr>
<tr>
<td>DNAgent</td>
<td>42.372</td>
<td>43.301</td>
<td>41.759</td>
<td>41.648</td>
<td>41.948</td>
</tr>
</tbody>
</table>

Table 5.4: Average Performance (60 actual games with best repository agents)

<table>
<thead>
<tr>
<th>Agent</th>
<th>Set 1</th>
<th>Set 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>TacTex</td>
<td>55.461</td>
<td>53.216</td>
</tr>
<tr>
<td>stuBID_Version4</td>
<td>54.236</td>
<td></td>
</tr>
<tr>
<td>Schlemazl</td>
<td>54.033</td>
<td>50.991</td>
</tr>
<tr>
<td>Mertacor</td>
<td>53.378</td>
<td>54.821</td>
</tr>
<tr>
<td>CrocodileAgent</td>
<td>47.511</td>
<td>47.879</td>
</tr>
<tr>
<td>AstonTAC</td>
<td>46.388</td>
<td>45.279</td>
</tr>
<tr>
<td>epflagent</td>
<td>43.249</td>
<td>44.406</td>
</tr>
<tr>
<td>DNAgent</td>
<td>42.372</td>
<td>39.370</td>
</tr>
</tbody>
</table>

5.6.3 Re-exploration of the Bidding Space

Given the highly dynamic environment of sponsored search, it is reasonable to periodically explore the bidding space in an attempt to update the preferred queries and bids of the other agents and to take advantage of any promising queries. Thus, we decided to repeat the initial learning process twice during each game instance. The range of the randomly generated bids was kept the same, while the duration of the two new learning periods was three days. The average profit of the agent in 60 games is illustrated in Table 5.4. We can see that the average profit of the agent has decreased significantly compared to that of the fourth version of the agent. It is our belief that the agents participating in the TAC/AA game concentrate mainly on the same queries and their bids remain roughly the same throughout each game instance. As a result, a further exploration of the bidding space beyond the initial is not necessary. Moreover, due to the fact that the duration of a game instance is quite short, re-exploring the bidding space negatively affects the agent’s performance, as during this period, it places bids on all the auctions instead of the most profitable.
Table 5.5: Average Performance (60 actual games with best repository agents)

<table>
<thead>
<tr>
<th>Agent</th>
<th>Set 1</th>
<th>Set 2</th>
<th>Set 3</th>
<th>Set 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>TacTex</td>
<td>55.797</td>
<td>55.720</td>
<td>56.024</td>
<td>57.537</td>
</tr>
<tr>
<td>Mertacor</td>
<td>54.314</td>
<td>55.368</td>
<td>55.277</td>
<td>52.334</td>
</tr>
<tr>
<td>Schlemazl</td>
<td>53.927</td>
<td>54.949</td>
<td>53.086</td>
<td>53.813</td>
</tr>
<tr>
<td>stuBID_Version2_Hill-climber</td>
<td>51.810</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CrocodileAgent</td>
<td>44.864</td>
<td>48.109</td>
<td>44.073</td>
<td>47.439</td>
</tr>
<tr>
<td>stuBID_Version2</td>
<td></td>
<td></td>
<td></td>
<td>47.135</td>
</tr>
<tr>
<td>AstonTac</td>
<td>44.709</td>
<td>42.281</td>
<td>48.585</td>
<td>46.764</td>
</tr>
<tr>
<td>DnaAgent</td>
<td>40.744</td>
<td>42.174</td>
<td>34.762</td>
<td>44.037</td>
</tr>
<tr>
<td>EpflAgent</td>
<td>40.074</td>
<td>43.421</td>
<td>40.763</td>
<td>40.781</td>
</tr>
<tr>
<td>stuBID_Version1_Hill-climber</td>
<td></td>
<td></td>
<td></td>
<td>40.012</td>
</tr>
<tr>
<td>stuBID_Version1</td>
<td>33.776</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

5.6.4 Addition of Hill Climber in the First Two Versions

In the third version of our agent (stuBID_Version3), we used a hill climbing algorithm in order to optimise the desired number of conversions for the next 10 days and to use the next day’s value for the calculation of that day’s aggregate budget. The hill climber could have been incorporated into the first two versions of the agent. In order to investigate the size of its contribution to the rise in the agent’s performance as well as the particle filters’ contribution, we added it to our first two agents and their performance is shown in Table 5.5. As we can see in the Table, the increase in performance after the addition of the hill climber is large. More specifically, the average profit of the first agent increased by $6.236, while the corresponding gain for the second agent was $4.675. This large increase was not unexpected as the hill climber addresses the inability of the initial approach to better utilise the available capacity. Instead of keeping the next day’s desired number of conversions constant and equal to its value is determined based on the number of conversions that were carried out on the previous four days and those that are carried out on the current day, and it also takes into account the future conversions. The results also highlight how important is for the agent’s profit to keep the number of conversions close to the desired levels.

5.7 Summary

In this Chapter we experimentally evaluated the performance of the four agents that were described in the previous Chapter. Moreover, we investigated the accuracy of the particle filters that model the states of the 9 user subpopulations as well as the accuracy of the estimators that calculate the expected costs, conversions and profits for each position. Finally, we repaced some components of the agents with others and we measured the difference in the average profits that emerged. In the following Chapter we provide a conclusion and offer an outlook of future work.
Chapter 6

Conclusions and Future Work

In this study we focused on the budget optimisation problem faced by advertisers participating in sponsored search auctions. Advertisers seek to maximise their profit and for this reason, the development of bidding strategies that will allow them to achieve their goal is more essential than ever. The design of agents that can act on behalf of their owners and take the appropriate decisions has proved to be very efficient in the field of sponsored search. We provided an analytical description of the existing bidding strategies as well as the design principles and adopted strategies of several agents that participated in TAC/AA competitions. We implemented a software agent able to undertake the bidding tasks of its owner. Although in the game, advertisers do not suffer from liquidity issues and therefore, are not restricted by budget constraints, due to the fact that budgets are almost always present in real sponsored search auctions, we considered it important to introduce the concept of budget and to present a solution to the emerging budget optimisation problem. We initially equipped our agent with some basic bidding strategies and we gradually increased its complexity by adding new features. Our experiments showed that implementation of both modeling and optimization methods are required to achieve high performance in the TAC/AA game. The emerging agent proved particularly efficient and robust as it was placed second among some of the best TAC/AA agents. Its general performance highlights the efficacy of the adopted strategies. Finally, we attempted to identify key parameters that influence more or less the agent’s behaviour.

As a future work we would first like to improve the performance of our estimators. The expected clicks, costs, conversions and revenues depend on several hidden game parameters and the estimators can be improved only if the values of these parameters are estimated accurately. We would also like to build a model of the other advertisers and apply machine learning to predict their behaviour. By using data from previous games, it may be possible to analyse the other agents’ decisions and to get an insight into their adopted strategies. This model can increase significantly the performance of the agent and will also allow us to improve our bid estimators. It would also be very interesting to observe the contribution of budget to increasing the agent’s performance after increasing the accuracy of its estimators. It is our belief that the role of budget will become less significant as the agent’s predictions become more accurate. Finally, it is worthwhile to add some new bidding strategies to the agent’s design and to compare their performance against the existing strategies.
References


REFERENCES


