TEL-AVIV UNIVERSITY
RAYMOND AND BEVERLY SACKLER
FACULTY OF EXACT SCIENCES
BLAVATNIK SCHOOL OF COMPUTER SCIENCE

An Empirical Study of the Ad Auction Game in the Trading Agent Competition

Thesis submitted in partial fulfillment of the requirements for the M.Sc. degree in the School of Computer Science, Tel-Aviv University

by

Shai Hertz

The research for this thesis has been carried out at Tel-Aviv University under the supervision of Prof. Yishay Mansour

May 2012

Abstract

The contribution of this thesis is developing a top performing Ad Auction agent for the Trading Agent Competition (TAC). This agent models the users populations using particle filters, based on a Nearest Neighbor estimator (instead of game specific parameters). It also uses a simple and effective bid optimization algorithm, that applies the equimarginal principal combined with perplexity-based regularization. In addition, We study the empirical behavior of TAC-AA agents. We investigate the robustness of the agents to changes in the environment, and show that most agents, especially the top-scoring ones, are surprisingly robust. Furthermore, using the actual logs, we derive for each agent a *strategic fingerprint* and show that it almost uniquely identifies it, and that it is strongly related to its profit and robustness.

Contents

Chapter 1

Introduction and Related Work

Online advertising through sponsored search results is form of advertising, where query specific advertisements are placed alongside organic search-engine results. Sponsored search has become a multibillion dollar business in the past years. It also has been the object of a considerable amount of research [EOS07, Var07, LPSV07].

An important result of this research is a variety of ad auction mechanisms [LPSV07]. For example, the cost per click (CPC) can be set to be the price bid of the auction winner (*first price* auction), or the price of the second-best position (*second price* auction). A publisher can also take into account the quality of the ad (or of the advertiser) when determining the positioning of the ads. Many research approaches use restrictions and simplifying setting assumptions, e.g., focus on one-shot auction or focus on a single query auction [CDE$^+$07, Var07].

The Ad-Auction (AA) game in the Trading Agent Competition (TAC) presents a sponsored search scenario that employs an ad auction mechanism and a structured model of users [JW10]. While still simplified in some aspects, it presents a complex ad auction environment. Competitors in this game implement retailers that try to maximize their profit through the use of sponsored search advertising. This setting facilitates research of agent strategies in a multiagent competitive environment. Furthermore, it can be used to draw more general conclusions about ad-auction mechanisms and sponsored search [JWB10]. Ad Auctions games have been held since 2009, and in the course of time the performance of agents was improved by employing complex techniques and strategies [PCS10, BGNS10, PS11].

Previous approaches to the TAC-AA bidding optimization problem (e.g., [PCS10]) rely on the game description to accurately estimate the game parameters, as well as to model the users distribution and competitors actions. Both [PCS10, BGNS10] formulate the bidding optimization problem as a combinatorial (intractable) optimization problem and heuristically search for the best solution. Although such methods do achieve top scores in TAC-AA, they might be sensitive to modeling errors, both parametric (estimation errors) and structural (wrong model used).

One purpose of this work is to present our top performing TAC-AA agent, *tau11*. While based on observations from previous TAC-AA agents, this agent also employs machine learning techniques and approaches. For example, we implemented particle filters to model the users distribution, but our implementation uses K-Nearest Neighbor (K-NN) estimator to compute the input of the particle filter. Our optimizer was implemented to search for the equimarginal utility bid (See [BGNS10] for motivation). Using simple linear models for estimating cost-bid relation allowed for an efficient implementation of the optimization as a simple one-dimensional

search. Furthermore, to mitigate the risk of under-utilized capacity resulting from over-estimation of the sales potential in a users population, we introduced a tunable regularization factor that favors allocations across a high number of queries (i.e., allocations of high perplexity). All the above resulted in a top performing agent, achieving the third place in the final round of the TAC-AA 2011 competition, scoring within 3% of the winner.

We also used this agent in order to estimate the benefit that agents can obtain from improving their Machine Learning (ML) components. To that end, we modified the game server so it will send to our agent some of the unobservable parameters the agents try to estimate, so it would have a perfect knowledge of them. Our results indicate that even though Machine Learning models are inherently inaccurate, eliminating their error rates completely has only a minor effect on the performance of the agent. Therefore, we can speculate that improving these models is not likely to increase agents' score. We can deduce that the global performance of a learning system might not improve significantly even if all errors are eliminated. This is an excellent example where even significant improvements in the accuracy of the ML components would have a diminishing effect on the overall performance of the system.

Another purpose of this work is to draw general conclusions by analyzing TAC-AA agents using a black-box approach, i.e., we do not analyze the internal mechanisms of agents, but rather test their performance in various settings.

We try to understand the applicability of TAC-AA to real world scenario. The game specifies a synthetic environment, and the agents are developed to take advantages of various features of this environment. Our goal was to test whether the agents can adapt to a different environment and still perform well, as expected from agents in a real world. To that end, we modified the game parameters and tested the effect of this modification on the performance of some recent TAC-AA agents. We show that the top performing agents perform well even when the parameters are changed and exhibit robustness, although as expected, most of the agents are overfit to the specific game parameters. This result suggests that TAC-AA may indeed serve as a test-bed for solving real-life settings, and that techniques used by top performing agents may potentially be applied in the real world.

The final objective of our research is to define a *strategic fingerprint* of a TAC-AA agent and characterize its behavior. To achieve this objective, we define several observable attributes that are calculated from the logs for each agent in each game, and we incorporate them into an attribute vector we call a strategic fingerprint. We show that this strategic fingerprint identifies agents, and is also well correlated with their profit. Therefore, this fingerprint can be used to design better TAC-AA agents. In addition, it reflects the vulnerability of simple log anonymization, and demonstrates that it can be overcome using simple ML tools.

Related Work

Sponsored search research is dominated by a game theoretic point of view. Several auction mechanisms and allocation algorithms have been proposed and analyzed. These works have inspired the creation of the TAC-AA game in 2009. TAC-AA was held for three years, in which different methods and techniques were used to create competing agents. This game was also analyzed using game theoretic approach.

The works of [EOS07, Var07, LPSV07] present game theoretic models of the sponsored search setting, and analyse their equilibria. Edelman et al. [EOS07] analyse generalized second

price auction (GSP), the mechanism used in many real world ad auctions. They focus on one shot, simultaneous-move game, with restrictions on the advertisers' behavior, and call the equilibria satisfying these restriction "locally envy-free". They show that the set of locally envy-free equilibria contains an equilibrium in which the payoffs of the players are the same as in the dominant-strategy equilibrium of the Vickrey-Clarke-Groves (VCG) auction, even though the bids and the payment rules are different than these of the VCG mechanism. They also analyse the English auction which presumably resembles the underlying dynamics of GSP. They show that the English auction has a unique equilibrium in which all players receive VCG payoffs. In addition, this equilibrium is robust, i.e., it holds for any set of distributions of advertisers' private values.

Varian [Var07] proves the existence of pure Nash equilibrium in positional auctions (which are equivalent to GSP). He also provides empirical data showing that these Nash equilibria auction describe the basic properties of the prices observed in Google's ad auction reasonably accurately.

Lahaie et al. [LPSV07] describe auction mechanisms used for sponsored search and analyze their properties. They also present an alternative on-line ad allocation model, based on the budget limitation of the advertisers. They give two allocation algorithms for this problem and present their approximation ratio.

Aggarwal et al. [AFMP08] present a Markov model for users behavior in the sponsored search setting. They use this model to develop a position assignment mechanism which maximizes the total advertiser value derived from user clicks, and they show that it is slightly different than the GSP auction. In their model, the user views the ads sequentially and the probability that he will click on a certain ad decreases with its position. Their allocation algorithm uses VCG pricing, which is truthful in this setting.

The aforementioned research led to TAC-AA [JW10]. This game employs a Markov model of the users which is slightly more complex than the one described in [AFMP08]. In this model, users' click probability is determined not only by the ad position, but also by the user's internal state as well as the ad characteristics. Similar to [Var07], in this game the result of the auction is not determined solely by the advertisers' bids, but also by their quality. The purpose of this game is to encourage researchers and practitioners to produce both new ideas about bidding strategies for advertising, and insights about sponsored-search mechanisms.

Pardoe et al. [PCS10] report on the winning agent from the first (2009) competition, TacTex. Among other contributions, they present an algorithm for estimating the number of an agent's impressions based on the aggregated query report. This algorithm is specifically tailored for the TAC-AA setting. They use this model as an input to a particle filter model of the users population. They also describe a greedy optimization algorithm, and an advertiser model that aims at predicting the behavior of competing advertisers. Pardoe and Stone [PS11] describe another use of the particle filter technique in the AA setting - estimating the bids of other advertisers given a periodic ranking of their bids.

Berg et al. [BGNS10] present an implementation of the "equimarginal utility" principal, stated in another TAC competition [GNO$^+$07]. According to this principal, the revenue is maximized among possible uses of a resource when the return on the last unit of the resource is the same across all areas of use. They present two optimization algorithms which are based on this principal: a model-heavy greedy multiple choice knapsack algorithm and a model-light rule-based algorithm.

Finally, our analysis of the TAC-AA agents is inspired by [JWB10]. They analyze the first

TAC-AA, employing game theoretic approach. They construct an empirical game using the agents and mechanism from the competition, and derive equilibria of this game. In addition, they analyze agents' strategies according to their manufacturer specialty (MS). They show that the profit of an agent is highly correlated with its percentage of MS clicks.

Organization of the Thesis

The rest of this thesis is organised as follows. In Chapter 2 we present a brief description of the TAC-AA game specifications. We then describe our agent for TAC-AA 2011 - tau11 in Chapter 3. We use this agent in Chapter 3.5 to assess the possible contribution of better Machine Learning models to the performance of TAC-AA agents.

In Chapter 4 we analyze TAC-AA agents. Chapter 4.1 describes the experiments we held in order to assess the robustness of TAC-AA agents and their results. Finally, in Chapter 4.2 we define a *strategic fingerprint* of a TAC-AA agent, and show its relation to agents' performance and robustness.

Chapter 2

TAC-AA Game Specification

The Ad-Auctions game, one of several games run yearly by the Trading Agents Competition (TAC) forum, is a simulated sponsored search environment. The competing agents interact with a game server that simulates the users and the search engine's actions.

As illustrated in Figure 2.1, the ad auctions game dynamics are such that each advertising agent provides a daily bid bundle to the search engine system. The bid bundle consists of bids, limits and ad specifications, and this system uses the advertisers' bundles during the following day to run an auction for every search query performed by a user. The auction result determines the advertisers' ad positions and the price to be paid to the search engine system upon a user click on their ad (CPC - cost per click). This price is determined by a generalized second price auction with a reserve price.

Each simulated user, making at most one query per day, has a predetermined preference to one of three manufactures and one of three product types, thus there 9 different users populations. The game simulates 10000 users of each population. To keep the game simple, user's queries are limited to a choice of manufacturer and product type (not necessarily the preferred ones). The user may also choose not to mention in his query any manufacturer or product type and therefore there is a total of 16 possible queries.

The specific query made by a user depends on his *state*, reflecting his tendency to search, click, and make a purchase. All users start at a Non-Searching (NS) state, from which they transition with a given probability to an Informational Search (IS) state. In this state, users submit queries and click ads, but do not convert. From this state, users may transition to any of the Focused Searching states ($F0$, $F1$, $F2$). These states reflect the user's search sophistication, or its degree of knowledge about its internal preference, ranging from null query in focus level $F0$ to detailed search in focus level $F2$. Users may transition from a low level Focused Search to a higher one, and may also transact. After a user transacts, he may return to the Non Searching state and start the process all over again. The user's state transition probabilities are governed by a Markovian model that appears in the game specification.

To model bursts of search behavior, the transition probability from the Non Searching state to the Information Searching state may be increased significantly, with probability P_{burst}. A burst can last up to three days, after which the transition probability returns to its former state.

When a user submits a query, the auctioneer creates a ranked list of ads, called an *impression*. The user views the ads sequentially and she can click on one ad or more, with probability determined by a Click-Through Rate (CTR). The CTR for every query is determined by three factors: the advertiser effect (a baseline value randomly chosen by the server for each compet-

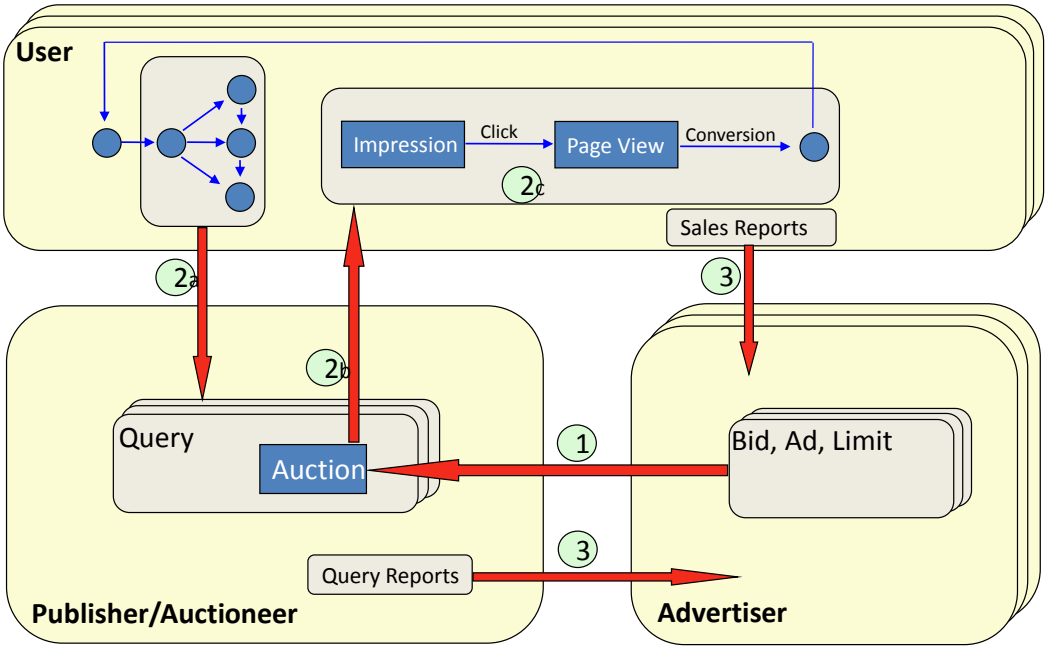

Figure 2.1: The Ad-Auctions game dynamics: The competing agents (the advertisers) submit bid bundles which are used by the system to run auctions for queries performed by simulated searching users. The users may click on ads and purchase products from the winning advertisers. Reports regarding past performance are provided daily to the advertisers so they can adjust their future actions

ing advertiser at game start); whether the ad is placed in a promoted slot or not; and whether the ad targeting matches the user's product preference.

The probability that a user that viewed a certain ad will continue to view the next ad is determined by a *continuation parameter*. Clicking the ad takes the user to the advertiser's web page, where she can decide whether to purchase a product (*convert*). The probability that a user will convert is called Conversion Rate (CVR) and is determined by three factors: the user's internal state, where higher focus levels convert at higher rates; whether the product matches the user's product preference; and whether the advertiser exceeds his sales capacity. The advertiser collects a predetermined revenue for each conversion.

Inter-query dependencies are introduced through a predefined 'soft' capacity constraint. Each competing agent is assigned one of three discrete capacity levels (Low, $Medium$ and $High$) at the beginning of the game session. An advertiser that sells over his capacity (during a moving window of five days) suffers a decrease of users conversion probability, which in turn reduces the Return on Investment (ROI), since the CPC remains the same.

The advertiser problem is therefore to maximize his accumulated net revenue (during 60

simulated days) by providing optimal bid bundles and considering for each query the potential costs and revenues (affected mainly by the user populations size and state, and by the competing agents bids). The agent may use daily reports that reveal information regarding the agent's performance during the previous day.

Some of the parameters of the game are given in the game specification, while others are randomly chosen at the beginning of each game from a known distribution. Hence, agents need to estimate the unobservable parameters, to model the users population, and then to optimize their bid bundle accordingly.

Notations

Throughout this work, we use certain notations for the game parameters and variables:

- The capacity of agent a is denoted by C^a. The advertiser name may be omitted when it's clear from the context.
- The number of impressions, clicks and conversions for query q on day d are denoted by $imps_q^d$, $clicks_q^d$ and $sales_q^d$.
- The reserve price (set by the server at the beginning of the game) is denoted by ρ.
- The continuation probability is denoted by γ.
- The advertiser effect, which is the baseline for the Click-Through Rate, is denoted by e_q^a, where q is the query type and a is the advertiser.

Chapter 3

Tau11 - a Winning TAC-AA Agent

3.1 Agent Structure

Our main target in this chapter is to describe our top-performing TAC-AA agent, that was built using Machine Learning approach and techniques. Observing the methods and results reported in [BGNS10] and [PCS10] we concluded that a good model of the users population is essential for top performance. We therefore implemented particle filters to model the users populations states. Our particle filter input however does not rely on spec-tailored computations of the total impressions as in [PCS10], rather we estimate this quantity using the *nearest neighbor* model-free method. Furthermore, our agent is model-free in that it does not assume a model for the competing agents bidding behavior.

We schematically partition our agent to three main components: A *Modeler* responsible to assess and predict the state of the changing environment, i.e., the users population and competitors state, in past, current, and future days; an *Estimator* responsible to discover the game parameters (this includes reserve prices, continuation probability etc.), and an *Optimizer* that uses the services provided by the estimator to come up with optimal bid bundles.

The service provided by our modeler is an assessment of the number of users submitting each of the queries for past current and future days. Our modeler also provides an assessment of the number of users that may submit a query but never convert (again, for each of the queries, and past and future days). As will become evident, these two quantities (specifically, their ratio) are a key metric in evaluating the potential profitability of bidding on a query, since users that click but never convert may result in significant loss to the advertiser.

Our estimator maintains estimates of the game parameters for each query (reserve prices, click through and continuation probabilities), and also the two-way mappings of bids to resulting *CPC*s and ad positions. As it turns out, a simple linear relation between bids and *CPC*s suffices (as an alternative to modeling the competitors' bids behavior) to achieve top scores.

Following notes in [BGNS10] suggesting that overall profits are maximized (subject to the constrained capacity) when the marginal utility (as a function of the bid) is equal across queries, our optimizer was implemented to search for the equimarginal utility bid. Using simple linear models for estimating cost-bid relation allowed for an efficient implementation of the optimization as a simple one-dimensional search. Furthermore, to mitigate the risk of under-utilized capacity resulting from over-estimation of the sales potential in a users population we introduced a tunable regularization factor that favors allocations across a high number of queries (i.e., allocations of high perplexity). The regularized optimization algorithm significantly increased the

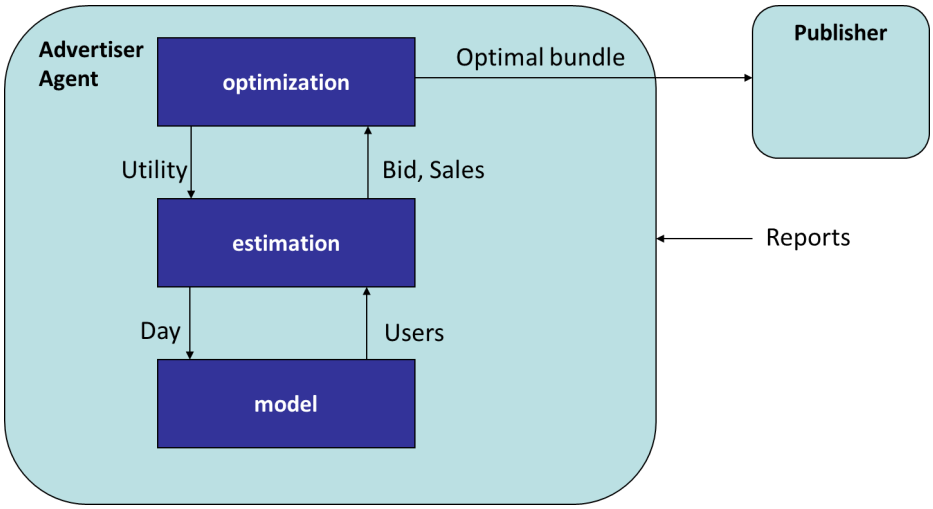

Figure 3.1: Schematic partition of the tau agent functionality. The agent (left) interacts with the publisher, which is implemented by the server

performance of our agent. All in all, combining the regularized equimarginal optimizer with the enhanced modeling resulted in a significant performance boost that (together with a series of limited-scope improvements, as described later) got our agent to the third place in the final round of the TAC-AA 2011 competition, scoring within 3% of the winner.

Our agent's architecture is illustrated in Figure 3.1. We now turn to describe in detail each of our agent's components.

3.2 Optimizer

According to the game specification, each agent has a limited capacity C, which should divide across a five-day window. At the beginning of each day, we estimate the number of units we sold in the last 4 days to calculate an allocation for the following day. We denote the capacity allocated for day d by Q_d. The optimizer's task is to find at every day d an optimal (highest profits) bid bundle for day $d+1$ that achieves this daily allocation.

The target daily capacity allocated for the following day Q_{d+1} (total to be further allocated across queries) is set such that it complements the units sold during the previous days $Sales_{d-3}, Sales_{d-2}, Sales_{d-1}$ and the estimation of the number of units to be sold today $\hat{Sales}_d$ to $(1+\delta)$ of the 5-day capacity C (the slight addition over the capacity is to allow for operation over the capacity at a range that will not hurt the conversion rates significantly). In all cases, the allocation is never less than a minimal amount which is half the average daily sales required

to achieve the capacity quota:

$$Q_{d+1} = \max\{\frac{C}{10}, (1+\delta)C - \hat{Sales}_d - \sum_{i=1}^{3} Sales_{d-i}\} \ . \tag{3.1}$$

The actual allocation, denoted $\tilde{Q}_{d+1}$, may be somewhat higher than Q_{d+1} since some of the queries get a minimal allocation even if the allocation scheme resulted in no allocation. This is due to the fact that we must take part in the auctions in order to get the respective reports, which we use to model the users and estimate game parameters. Finally, the estimate for 'today' sales $\hat{Sales}_d$ is the average of the quota allocated for today Q_d and the actual allocation $\tilde{Q}_d$.

Following the equimarginal utility principal stated in [BGNS10], we try to find an allocation for which the return on the last unit sold is the same across all queries. First, we express the utility U as a function of the units sold m.

Denote (for some fixed day d and query q, therefore omitted from the notation) by m_b the number of users submitting query q and converting (with some positive probability), and by m_n the number of users submitting query q and never converting (i.e., users in Information Searching state). We now show that the following relation holds:

$$U(m) = m\left(R - \frac{\textit{CPC}(b(m))}{\textit{CVR}}(1 + \frac{m_n}{m_b})\right) \tag{3.2}$$

where m is the number of units sold to the users, $U(m)$ is the total profit from selling m units, R is the revenue associated to a unit sold, $b(m)$ is the bid that results in m units sold, $\textit{CPC}(\cdot)$ is the mapping of the bid to the cost per click (*CPC*), and *CVR* is the conversion rate.

For a conversion to take place, a 'buying' user (whose state is one of the Focused Search states) should click the ad and then decide to convert. Since the users are simulated in a random order, the sales for a query are achieved by $m_b \cdot \textit{CTR}$ clicks from the 'buying' population and $m_n \cdot \textit{CTR}$ clicks from the 'non-buying' population. Therefore, $m_b \cdot \textit{CTR} \cdot \textit{CVR}$ sales requires $(m_b + m_n)\textit{CTR}$ clicks. The number of clicks required to achieve one unit sale is $\frac{(m_b+m_n)\textit{CTR}}{m_b \cdot \textit{CTR} \cdot \textit{CVR}} = \frac{1}{\textit{CVR}}(1 + \frac{m_n}{m_b})$.

Hence, the cost of each sale is $\frac{\textit{CPC}}{\textit{CVR}}(1 + \frac{m_n}{m_b})$. Since the *CPC* is determined by the bid $b(m)$, we get the relation (3.2). Note that since $b(m)$ (the bid resulting in at least m sales) is a step function, $U(m)$ is pieacewise linear in m. Moreover, the slope of $U(m)$ negatively depends on $\textit{CPC}(b(m))$ and is therefore decreasing in m.

We can now formalize the optimizer problem of finding the optimal bid bundle subject to the capacity constraint as the following program:

$$\begin{aligned} &\max_{\{b_q\}} \sum_q U_q(m_q(b_q)) \\ &\text{subject to} \sum_q m_q(b_q) \leq Q \end{aligned} \tag{3.3}$$

where $m_q(b_q)$ is the assumed number of units to be sold when bidding b_q on query q, and Q is the daily sales quota as set using (3.1). Now, making the simplifying assumption that $U_q(\cdot)$ are concave for every q, it is easy to see that for an optimal solution $\{m_q^*\}$ of (3.3) all the marginal utilities $\frac{dU_q}{dm}(m_q^*)$ are equal (otherwise, as long as the marginal utility is positive, we could move

a unit allocated to a query with lower marginal utility to a query with higher marginal utility and have a higher profit while still complying with the capacity constraint).

Since $U_q(m)$ is piecewise linear, equating marginal utility is equivalent to equating utility. Therefore, the optimization problem (3.3) is solved by a simple linear search for the maximal utility u for which the resulting total sales achieve the quota: $(m_1(u), m_2(u), \cdots, m_{16}(u))$ such that $\sum m_q(u) \geq Q$, where $m(u)$ is the inverse mapping of $u(m) = R - \frac{CPC(b(m))}{CVR}(1 + \frac{m_n}{m_b})$ and is provided to the optimizer by the estimator. Note that a decrease in the utility per unit sold occurs when the related cost per unit sold increases (reflecting a better ad position for the query), which leads to a higher number of conversions. Therefore $m(u)$ decreases with u.

As a result, our simple search for the optimal utility starts at a predefined constant high utility level u_h and decreases it repeatedly (in Δ sized steps) until our total estimated sales reaches the target daily allocated quota (or until a predefined constant low utility u_l is reached - this is to avoid the risk of negative utility, and to ensure that our algorithm stops even in cases where the total potential sales are lower than the allocation).

Algorithm 1 Basic optimizer algorithm.
Input: Q, $\{m_q(\cdot)\}$, $\{b_q(\cdot)\}$
Output: optimal bid bundle $\{b_q\}$

$u \leftarrow u_h$
while $(\sum_q m_q(u) \leq Q)$ AND $(u \geq u_l)$ **do**
  $u \leftarrow u - \Delta$
end while
$\{b_q\} = \{b_q(u)\}$

After noticing that in practice our optimizer tends to allocate most of the quota to very few queries (implying that overestimation of the sales - a common phenomenon of our estimator - led to an under-utilization of our quota) we added regularization to ensure that the allocation is spread over more queries and reduce the risk of under-utilization. The regularization is based on the perplexity of the allocation: denote by $\overline{m}(u) \in \Delta_{16}$ the vector of the relative parts of units allocated to each of the queries. The perplexity of a probability distribution $\overline{d} \in \Delta_n$ is defined as $p(\overline{d}) = 2^{H(\overline{d})}$ where $H(\cdot)$ is the entropy. Therefore $p(\overline{m})$ is larger as $\overline{m}$ represents a quota that is evenly distributes across the queries (and vice versa). In the regularized optimization algorithm we discount the estimated sales for a given utility level using a logistic function of the allocation perplexity. Specifically, by a factor $\frac{1}{1+\beta \cdot e^{-p(\overline{m})}}$. We used a parameter β to tune the regularization degree, where a high β represents a high preference to spreading the allocation across queries. Now, as u decreases, $m_q(u)$, the perplexity $p(\overline{m}(u))$, and the regularization factor increases. Therefore, our regularized algorithm (which is identical to Algorithm 1, but with $(1 + \beta \cdot e^{-p(\overline{m}(u))})$ multiplying Q in the while condition) will result in the highest uniform utility level that achieves a regularized quota (higher - since the regularization factor is smaller than 1).

3.3 Estimator

The main purpose of the estimator is to estimate the sales (and related bid) for a given target utility level. To do so, the estimator uses the modeler assessments m_n and m_b for every query

and estimates the other quantities appearing in the following relation, which is based on (3.2)

$$u(m) = R - \frac{\textit{CPC}(b(m))}{\textit{CVR}}(1 + \frac{m_n}{m_b}) \ . \tag{3.4}$$

Specifically, the estimator first solves (3.4) to find the required *CPC*, then it finds the target position that results in the required *CPC*, and finally it estimates the resulting sales figure (using the estimates of the continuation parameter and conversion rate). A similar computation is done (upon finding the optimal allocation) to set the bids. The bid for each query is set such that it results in the target number of units $m(u)$.

As we shall see, the estimator needs to provide estimations of the click-through rate (*CTR*), conversion rate (*CVR*), continuation probability and reserve prices. It also provides a mapping between bids and *CPC*s and between bids and ad-positions. We now describe in detail the methods used by the estimator for estimating each of the required quantities.

3.3.1 Bid-CPC Mapping

We make the simplifying assumption that the *CPC* and the bid are linearly related. Therefore, the estimator maintains the ratio and the upper and lower thresholds (i.e., the bid threshold beneath which our ad is not shown at all, and the bid threshold above which our ad is the first ad shown).

The ratio estimation is initialized based on previous games and is updated after each query report by averaging with the previous estimate. The lower and upper thresholds are also initialized based on previous games, and updated whenever they are inconsistent with the actual game data reported. Namely, when a bid lower than our minimal bid results in showing our ad we lower our minimal bid towards it. When a bid higher than our maximal bid doesn't result in our bid shown first we raise our maximal bid. Similarly, we maintain an estimate of the maximal *CPC*. Figure 3.2 illustrates the *CPC* and position relations with the bid.

Figure 3.2: The estimation of *CPC* (left) and position (right) as a function of the bid

3.3.2 Bid-Position Mapping

The total number of ad positions for a query is partially determined by the number of the bidders for that query. Therefore, we need to estimate the future number of bidders for a specific query, and use this number to estimate the position of an ad given a certain bid. This estimation is

Algorithm 2 bid2position

if $bid \geq maxBid$ **then**
  $position \leftarrow 1$
else if $bid < minBid$ **then**
  $position \leftarrow 0$
else
  $scale \leftarrow \frac{maxBid-bid}{maxBid-minBid}$
  $position \leftarrow num\_bidders \cdot scale + 1$
end if
return $round(position)$

Algorithm 3 position2bid

if $position = 0$ or $position > num\_bidders$ **then**
  $bid \leftarrow 0$
else if $position = num\_bidders$ **then**
  $bid \leftarrow minBid$
else
  $scale \leftarrow \frac{maxBid-minBid}{num\_bidders}$
  $bid \leftarrow maxBid - (position - 1) \cdot scale$
end if
return bid

quite naive. First, we initialize this number to its maximal value (5, according to the game specification). Then, after each query report we average the number of positions reported with the last estimate.

As before, we assumed an underlying linear relation and used the minimal bid and the maximal bid estimates. Therefore, we maintain the position-bid ratio and the thresholds. The resulting functions are bid2position and position2bid, shown in Algorithms 2 and 3 respectively.

3.3.3 Click through rate and reserve prices

The baseline *CTR* is the advertiser effect e_q^a, which is randomly chosen at game start. For games that have at least one promoted slot, we estimate e_q^a together with the two reserve prices ρ_{reg} and ρ_{pro} by solving the following three relations:

$$\rho_{pro} = \rho_{reg} + 0.5$$
$$cpc_r = \frac{\rho_{reg}}{(e_q^a)^\chi}$$
$$cpc_p = \frac{\rho_{pro}}{(e_q^a)^\chi}$$

where cpc_r and cpc_p are the minimal cost per click observed on regular slots and promoted slots, respectively.

The first relation is a simplifying assumption since by the game specification we always have $\rho_{\text{reg}} \leq \rho_{\text{pro}} \leq \rho_{\text{reg}} + 0.5$. The second and third are due to the generalized second price auction, the minimal price that an advertiser would have to bid in order to to get his ad shown is the squashed reserve price for the minimal regular or promoted position, respectively. The squashing parameter χ is given at game start so we can easily solve the three unknown variables and get estimates for both reserve prices and for $e_q^a = (2(cpc_p - cpc_r))^{-\frac{1}{\chi}}$. As this value is an approximation, it is averaged with the previous approximation whenever recomputed. Now, given the bid level we can assess whether our ad will be placed in a promoted slot or not. This allows us to use a good approximation of the true f_{pro}. Finally, knowing our ad type and the relevant population sizes we set f_{target} and get the desired approximation of the click through rate.

For games with no promoted slots we calculate the *CTR* iteratively. Given our reported position, the estimation of the continuation probability γ (as detailed in the next section) and the modeler assessment of the number of impressions on clicking users ($imps$), we first calculate the effective number of impressions - the number of users that actually considered clicking our ad (after passing higher positioned higher ads and continue without clicking), and then simply estimate the click through rate:

$$\begin{aligned} imps_{\text{eff}} &= imps \cdot [\gamma(1 - \textit{CTR}_{\text{old}} \cdot \textit{CVR})]^{position-1} \\ \textit{CTR}_{\text{new}} &= \frac{clicks}{imps_{\text{eff}}} \end{aligned} \tag{3.5}$$

3.3.4 Continuation Probability

For a game with promoted slots, once we have an estimate of the *CTR* we use the reported position and clicks and the modeler assessment of $imps$ to solve (3.5) where γ and $imps_{\text{eff}}$ are the only unknowns. Otherwise (to overcome the circular dependency between estimating *CTR* and continuation probability), we first calculate the *CTR* based on the previously calculated continuation probability and then calculate the new continuation probability.

3.3.5 Sales and bid

Sales estimation is the main service of the estimator provided to the optimizer. Given a target utility (profit) level for a query, the estimator returns the number of conversions (sales) that will result by bidding on the query such that the requested profit is attained. First, the target *CPC* is recovered using (3.4) and the modeler assessments of m_n and m_b. Next, the bid relations with *CPC* and position are used to find the position related to the target *CPC* (this is our target position). Then, the modeler assessment of total impressions and the estimation of the continuation probability are used to calculate using (3.5) the effective number of impressions for the target position. Finally, the estimates of click and conversion probabilities are used to provide the estimate of the number of conversions when bidding for the target position.

A related service provided by the estimator is the required bid level to achieve a target profit in a query (this is used by the optimizer once the optimal profit level is decided). To find the required bid level, the estimator does the first two steps of the sales estimation, returning the bid level that pertains to the target cost per click (using the bid and cost relation).

Both algorithms use a sales estimation function (Algorithm 4), which uses the estimations given by the modeler and the estimator to estimate to total number of sales that can be reached for a certain query and given utility. After finding the allocation that yields optimal utility for each query we create the bid bundle. We set the bid and the limit according to the allocation, but in addition we add 'exploration bids' for the queries not in the allocation. For these queries we choose at random a position between 1 and 5, and estimate the bid needed to achieve this position. We add this bid to the bundle, with a spend limit that suffices for a single sale.

3.4 Modeler

The modeler provides upon request $m_b^q(d)$ and $m_n^q(d)$, the maximal (potential) number of "buying" impressions and the maximal number of "non-buying" impressions (respectively) for

Algorithm 4 Sales(query q, utility)

$cpc \leftarrow (UNIT\_REVENUE(q) - utility) \cdot estimated\_CVR(q)$
$ctr \leftarrow estimateCTR(q, cpc)$
$position \leftarrow estimatePosition(q, cpc)$
if $position \leq 0$ **then**
  $sales \leftarrow 0$
  return
end if
$imps \leftarrow modelImpressions(q)$
while position > 1 **do**
  $imps \leftarrow imps \cdot est\_continuation\_prob.(q) * [1 - ctr \cdot est\_CVR(q)]$
  $position \leftarrow position - 1$
end while
$sales = impressions \cdot ctr \cdot estimated\_CVR(q)$
return $sales$

query class q on day d. As detailed in subsequent sections, $m_b^q(d)$ and $m_n^q(d)$ (specifically, their ratio) are a key metric in evaluating the utility of selling a unit by bidding on query q and as a consequence a key parameter to the bid optimization algorithm.

Denote by $n_s^p(d)$, the number of users in state s for population p on day d (recall that there are 9 user populations, one for each combination of manufacturer and product type). For clarity, in what follows we omit from the notation the dependence on the day d unless required to avoid ambiguity. Now, by the users behavior as described in the game specification, for each level-2 query $q = (M, C)$ we have

$$m_b^{(M,C)} = n_{F2}^{(M,C)}, \text{and } m_n^{(M,C)} = \frac{1}{3} n_{IS}^{(M,C)},$$

for each level-1 query $q = (M, \emptyset)$ or $q = (\emptyset, C)$ we have

$$m_b^{(M,\emptyset)} = \frac{1}{2}(n_{F1}^{(M,C1)} + n_{F1}^{(M,C2)} + n_{F1}^{(M,C3)}), \text{and } m_n^{(M,\emptyset)} = \frac{1}{6}(n_{IS}^{(M,C1)} + n_{IS}^{(M,C2)} + n_{IS}^{(M,C3)})$$

$$m_b^{(\emptyset,C)} = \frac{1}{2}(n_{F1}^{(M1,C)} + n_{F1}^{(M2,C)} + n_{F1}^{(M3,C)}), \text{and } m_n^{(\emptyset,C)} = \frac{1}{6}(n_{IS}^{(M1,C)} + n_{IS}^{(M2,C)} + n_{IS}^{(M3,C)})$$

and finally, for the (only) level-0 query $(\emptyset, \emptyset)$ we have (summing over the 9 populations):

$$m_b^{(\emptyset,\emptyset)} = \sum_p n_{F0}^p, \text{and } m_n^{(\emptyset,\emptyset)} = \frac{1}{3} \sum_p n_{IS}^p$$

Also, each user makes a query every simulated day and each query results in an impression for each of the winning advertisers. Therefore, by modeling $n_s^p(d)$, the number of users in state s for population p on day d, the modeler is able to provide upon request $m_b^q(d)$ and $m_n^q(d)$ for any query q.

We therefore maintain a particle filter for each of the 9 populations (each providing an estimate of $n_s^p(d)$). Each particle filter is a collection of particles, where each particle represents a distribution of the population over the possible states. The distribution represented by the

particle filter is the weighted average of the distributions of its particles, and the weight assigned to each particle reflects the plausibility of an observed quantity (in our case, as suggested by [PCS10], the estimated total number of impressions in the level-2 query that corresponds to the population) given its represented distribution.

The particle filter algorithm is therefore the following: we maintain a separate set of particles for each day *yesterday*, *today*, and *tomorrow*. When a new day starts, the particle set for *yesterday* is discarded, and the previous particle set for *today* is used for creating the new particle set for *yesterday* by reweighing and resampling it (upon receiving the query report and having an estimate of the total number of impressions). The new particle sets for *today* and *tomorrow* are created by advancing the new particle set for *yesterday* once and twice respectively. All this creates an updated estimate of $n_s^p(d-1)$, $n_s^p(d)$ and $n_s^p(d+1)$ at every day d of the game, allowing the modeler to service requests for m_b^q and m_n^q for days $d-1$, d, and $d+1$.

In the rest of the section describing our modeler we review in more detail the particle filter update steps and related implementation concerns. Of special interest (w.r.t. our model free approach) is the usage of nearest neighbor learning to estimate the particle filter input (in contrast to the direct calculation described by [PCS10]).

3.4.1 Reweighting

Given an observation of T estimated total impression, the weight $w(P|T)$ assigned to a particle representing a users distribution $P = (N_{NS}, N_{IS}, N_{F0}, N_{F1}, N_{F2}, N_{TR})$ is computed as the probability of a total of $T - N_{F2}$ successes in N_{IS} binomial experiments, each with success probability $\frac{1}{3}$ (this is because each of the N_{F2} users results in an impression for the related L2 query, and with probability $\frac{1}{3}$ each of the N_{IS} users results in an impression for the related L2 query). In practice, we use the normal probability approximation for the distribution with average $\frac{N_{IS}}{3}$ and variance $\frac{2N_{IS}}{9}$ and set

$$w(P|T) = \phi(\frac{3(T - N_{F2}) - N_{IS}}{\sqrt{2N_{IS}}}) \,, \tag{3.6}$$

where $\phi(\cdot)$ is the normal Cumulative Distribution Function. Upon re-weighting, the weights are normalized such that they sum to 1.

3.4.2 Resampling

Given a re-weighted particle set (the *baseline* set), resampling involves creating a new particle set in which the number of times each particle appears in the new set (the *resampled* set) is relative to its weight in the baseline set. Once resampled, the weights are discarded and weighted averages over the baseline set are equivalent to uniformly averaging over the resampled set.

Resampling a set of n particles with weights $w_1 \ldots w_n$ is implemented as follows: we consider a set of n evenly spread positions (distanced $\frac{1}{n}$) in the $[0, 1)$ interval, where the first position is randomly chosen in $[0, \frac{1}{n})$. We define the points $l_0 = 0$ and $l_i \triangleq \sum_{j \le i} w_j$ for $i = 1 \ldots n$ (note that $l_n = 1$). Now, the number of times we include particle i in the resampled set is the number of positions in the interval $[l_{i-1}, l_i)$, which is proportional to w_i by the construction.

A key challenge in modeling the users population is to identify bursts as soon as possible. A burst is characterized by a significant increase in the probability of transitioning from the non-searching state to informational-searching state and, as a consequence, a significant increase

in the observed total number of impressions. Therefore, overestimation of the total number of impressions (due to estimation errors) result in 'ghost bursts' - a situation in which the modeler gives high weights to particles that represent the occurrence of a burst while discarding particles that do not. Even when the estimations of the total number of impressions improve in subsequent days, it may take many iterations for the particle filter to readjust (since the particles that represent the 'true' situation do not survive the first overestimation).

To address such situations we only reweigh and resample a randomly chosen portion of the particles, leaving the rest of the particles unchanged regardless of the observation. Such particles might somewhat damage the accuracy of the modeled user population, but they allow for quick readjustment in case of estimation errors. The portion of the particles that is kept unchanged depends on our level of confidence in the observation: if we have complete confidence (i.e., our agent did not reach the spending limit and our position is positive) then the portion is 0%, if we have no confidence at all (i.e., our position is 0, indicating that our bid was too low and therefore no observation is available to us) then the portion is 100% (the whole particle population is kept unchanged), and in all other cases we set the portion to 5% (we also tried 10% and 15%, resulting in lower accuracy without noticeable robustness gain).

3.4.3 Advancing particles

Advancing a set of particles consists of advancing each particle of the set, simulating the daily state transitions of the users. A particle representing a users distribution is advanced by applying a transition model (which defines the state transition probabilities) to the represented user population. The transition model is given as part of the game definition, and is constant except for the transitions from focused searching states to the transacted state. Furthermore, the transition probabilities depend on the presence of a burst (effecting the transition probability from *NS* to *IS*, an effect that lasts for a sequence of days) and therefore each particle also maintains a burst-status which is used to select the appropriate transition model to use.

The particle advance algorithm is the following: First the appropriate transition model (*burst* or *regular*) is selected - this is a random choice (the probability of a burst depends on the current burst status). Second, the users N_S of each state S are transitioned according to the transition model. Finally, N_S^a (the advanced population of state S) is set as the aggregation of all users transitioned to state S. In the following two sections we address implementation issues regarding efficiency and other aspects of the advance algorithms.

Dynamic update of the Transition Model.

As mentioned in section 3.4.3, the conversion probabilities of the transition model depend on competitors actions and have to be estimated. Our first approach was to sample game logs (the population size in each state is reported for every simulated day) and use a linear regression to estimate transition probabilities by modeling the required transition (conversion) probabilities α, β, and γ (from states $F0$, $F1$, and $F2$, respectively) as follows:

$$\alpha N_{F0}^d + \beta N_{F1}^d + \gamma N_{F2}^d + 0.2 N_{TR}^d = N_{TR}^{d+1} \ ,$$

where N_S^d is the number of users in state S on day d. The relation is based on the fact that with probability 0.8 a user in the transacted state returns to the non-searching state, and the only transitions to the transacted state are from the three focused states $F0$, $F1$, and $F2$. This

regression is inherently inaccurate (for example, significantly different results are obtained on different game logs and different populations in the same game) and therefore we turned to a dynamic on-line estimation and update of the conversion probabilities, which resulted in much better results.

Our method is based on estimating the number of bidders b_q for each query q and the continuation parameter γ, both are daily estimated by the estimator component as described bellow and made available to the modeler. Moreover, we need the number of promoted slots s_p, a parameter provided to the agent at the beginning of each game. Finally, we need for each query the click probability and conversion probability of the *other* advertisers (CTR_q and CVR_q, respectively). Since we don't have access to CVR_q and CTR_q (which are randomly assigned to each advertiser for each query from a range that only depends on the focus level of the query), we take the average value of the possible range. (e.g. for $L1$ queries we use 0.23 and 0.35).

We compute the conversion probability as a function of the number of bidders $P_q(b)$ for a query q using the following recursion for $P_q(b_q)$:

$$P_q(1) = CTR_q \cdot CVR_q$$

$$P_q(b) = CTR_q \cdot CVR_q + \gamma(1 - CTR_q \cdot CVR_q)P_q(b-1) \ .$$

If the number of promoted slots s_p is positive, we assume that the highest ranked bidder is assigned a promoted slot. This only affects the top call to $P_q(b_q)$, in which we use a modified (somewhat higher) click probability.

3.4.4 Using NN for Total impressions estimation

The input (observation) of the particle filter is the total number of impressions during a simulated day. This quantity is not directly provided to the agent. [PCS10] describe a dedicated algorithm that may be used to find a value that is consistent with the reported data (the average position for each of the competitors, the position and number of impressions of the agent). However, consistent with our model-free approach, we used nearest-neighbor (NN) estimator to achieve a similar result.

We parsed several game logs to create a file with training samples. We augmented the data of each game with the true total number of impressions (retrieved from the game logs in retrospect). We then used this samples to create a a weka-based [HFH$^+$09] K-Nearest-Neighbor model that will be used by the agent to classify on-line (providing an estimation of the total number of impressions to the particle filter, given the reported data).

We used cross validation to assess the error rate of our model. Our experiments showed that using a training set that consists of 200000 samples, the average relative error rate of a 3-NN model was about 30%. Increasing the number of samples 10-fold reduced the average relative error to about 25%, but overall performance improvement was offset by the significantly higher memory requirements and initialization delay (to load the model into memory).

Comparing performance to the 'exact' computation of the total number of impressions

To compare our performance (using K-NN estimation of total number of impressions) to the performance of the algorithm that exactly computes the total number of impressions given the reported data we created a log-parser that uses two particle filters and the weka-model created

by previous training. The first particle filter is given (as the observation) the nearest-neighbor estimation of the total number of impressions and the other is given the true (as recomputed while parsing the log file) total number of impressions. It turns out that the estimation error has little influence on the overall ability of the particle filter to predict the user distribution over time.

3.5 Better Machine Learning Models

Many TAC-AA agents employ Machine Learning techniques and perform well in the game, despite the fact that the ML models used are inaccurate. Therefore, we wanted to estimate the potential score improvement that can be achieved by using more accurate models.

For that purpose, we modified our tau11 agent. We then modified the game server to send two types of information to our agent:

1. Parameters - The exact values of several unobservable parameters: advertiser's effect (e_a^q), continuation parameter (γ) and reserve prices (ρ).

2. Users distribution - The number of users in each state, for each query type.

We tested the effect this information had on our agent's performance in a competition against agents from the TAC-AA repository, who competed in the Ad Auction finals in 2010 and 2011.

The modified server sends the agent information that obviates the need for certain ML models, and in fact simulates the use of perfect models. Hence, these tests enable us to assess the additional profit one could hope to gain by improving these ML models.

In the first experiment we modified our agent to receive only the parameters from the server. These parameters enable it to make better assessment of Cost Per Click (CPC), click-through rates (CTR) and effective number of impressions.

In our original tau11, these parameters were estimated using iterative maximization methods (see Section 3.3 for details), and the error rate of the estimation was about 15%. Therefore, this information is a major estimation improvement. However, the average score of this agent was improved only by 1%, an improvement which is not statistically significant.

In the second experiment we modified our agent to receive only the users distribution from the server. Thus, the modified agent had perfect knowledge of the users' state in each time. In our original agent, this task was carried out using K-Nearest Neighbor approach, which was the input to a Particle Filtering model (see section 3.4 for details). The error rate of the K-NN model was 25%.

The score improved by 2%, which is not statistically significant ($p = 0.2$). This result implies that despite the high error rate of the K-NN model, the particle filter model creates good enough estimations of the users' state, which cannot be improved drastically.

Finally we modified our agent to receive both types of information. This agent had all of the information it needs in the game, except for the bid bundles of its opponents.

In this setting, the agent's average score was improved by 5%, an improvement which is statistically significant ($p < 0.05$). This result is somewhat surprising when considering the

minor effect that each set of information had by itself. It seems that improving only one model has little contribution, but having perfect models in both domains is more significant.

The results of these three experiments suggest that improving the accuracy of the ML models employed in tau11 has a limited contribution to the final result. Presumably, the high error rate of the ML models does not lead to impaired performance, and this error rate is somehow overcome during the optimization process of the agent.

3.6 Conclusion

The main novelty in our agent is the use of machine learning techniques in the modeler and the estimator. The ML models enabled us to implement a simple and effective optimizer, whose performance was further improved by the use of perplexity based regularization.

Even though the ML models we used are rather inaccurate, we showed that improving them would only lead to a minor performance improvement. We can conclude that the optimization process of our agent can overcome estimation errors.

Therefore, new ways should be sought in order to improve the performance of our agent. One possible direction is to change the optimizer so it would look several days ahead when creating the optimal bid bundle (instead of only one day ahead). This change requires creating a new modeler, which is able to estimate the future distribution of users. Such multi-day optimization could improve significantly the performance of our agent.

Another possible direction is to model the competitors of our agent in order to predict their future bids. Such prediction would enable us to predict more accurately the CPC and the ad position for a given bid, and would contribute to a better optimization process.

Chapter 4

Agents Robustness in TAC-AA

4.1 Robustness of TAC-AA Agents

In order to assess the robustness of TAC-AA agents, we ran several experiments in which we varied some of the game parameters and ran a standard 48-game competition in each new setting. The agents tested are agents from the TAC repository, who competed in the Ad Auction finals in 2010 and 2011. To complete the set to the required eight agents, we used two copies of one agent, TacTex (this also enabled us to estimate the effect of the game randomness on an agent's profit).

Since this combination of agents never participated in a public TAC-AA competition, a benchmark competition was first held. The results of this competition as well as the results of our first four experiments are detailed in Table 4.1.

For each experiment, we compared the score of each agent to its score in the benchmark competition, and noted the difference in the agent's position. We ran t-tests with 0.95 confidence level to find the statistical significance of this difference. We also compared the median score in each experiment to the median score of the benchmark competition, in order to understand the general effect of the changes we made.

Our results show that most agents, especially the top performing ones, are robust to changes in the game parameters, although they overfit to TAC-AA parameters to some extent.

AGENT	BENCHMARK	EX. 1.1	EX. 1.2	EX. 2.1	EX. 2.2	EX. 3.1	EX. 3.2	EX. 4
TACTEX10	58,146	67,627	61,294	50,903	62,544	53,578	61,866	36,737
TAU11	58,124	64,187	61,107	52,175	61,983	54,406	61,013	49,339
TACTEX(2)10	57,929	67,078	61,369	49,639	63,164	54,063	62,656	37,880
MERTACOR11	55,716	40,710	53,576	44,349	51,653	51,930	51,546	54,033
SCHLEMAZL10	55,413	62,766	60,952	51,323	59,553	53,145	59,246	47,139
CROCODILE11	50,735	51,456	50,521	44,682	54,700	45,593	53,369	40,386
TAU10	49,699	49,381	49,145	43,330	52,735	44,271	50,617	39,292
EPFLAGENT10	45,886	34,648	47,564	38,933	51,042	41,565	49,330	40,836
MEDIAN	55,565	57,111	56,032	47,161	57,127	52,538	56,308	40,611

Table 4.1: The results of the benchmark competition and experiments 1 - 4 from section 4.1. The numbers next to the agent name indicate the year in which this agent participated in the TAC-AA finals.

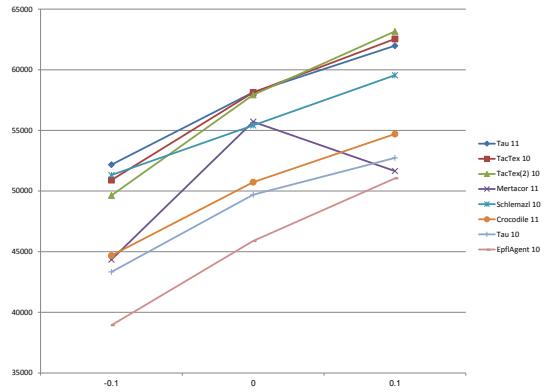

Figure 4.1: The results of experiment 2, where the advertiser effect is modified by ± 0.1 .

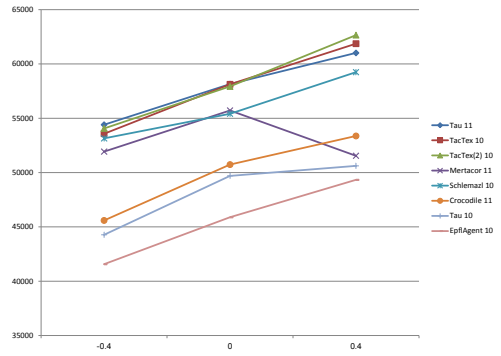

Figure 4.2: The results of experiment 3, where the conversion rate is modified by ± 0.04 .

4.1.1 Experiment 1 - Users Model

In the following experiments we modified the users model, i.e., we changed the transition probabilities in the users state machine, in order to differentiate between agents who rely heavily on the exact game parameters and agents that do not.

Ex. 1.1: We increased the transition probability from Non Searching mode (NS) to Informational Searching mode (IS) by a factor of 5 (from 0.01 to 0.05).

We hypothesized that this change will have a strong general effect, i.e., that it will increase the median score of the competition, since it increases the number of users which see ads, click on them and convert. We also expected that it will affect all agents similarly, and expected only a mild change in the relative positioning of the agents.

The median score of this competition was 57,111, which is significantly higher than the median of the benchmark competition. In addition, this change had a very different effect on different agents: while it increased the score of the top-performing agents, TacTex, tau11 and Schlemazl by 10-15%, it decreased the score of EpflAgent and Mertacor by about 25%. These differences were found statistically significant ($p < 0.01$). The other two agents maintained their old score - tau10 and EpflAgent.

Ex. 1.2: We slightly modified the users' transition matrix, but this time we changed the transition probability between the different focus level searching modes. Hence, the overall number of searching users did not drastically change, but the distribution of the users between the different focus levels changed.

We hypothesized that this change will differentiate between agents who heavily rely on the TAC-AA parameters and agents whose users models are more adaptive.

The median score of this competition was slightly higher than the median of the benchmark competition. This experiment had a milder effect on most agents. It significantly increased only the score of Schlemazl (by 10%, $p = 0.04$), while the scores of TacTex, tau11 and EpflAgent increased by about 5%, which was not found statistically significant. It also slightly decreased the score of Mertacor (by 4%, $p > 0.1$), as well as the scores of tau10 and Crocodile.

Combining the results of the two experiments, we can conclude that most agents are quite immune to changes in the users model. Specifically, TacTex, tau11 and Schlemazl seem less

dependent on the exact values of the user transition probabilities, while Mertacor seems dependent on these values.

4.1.2 Experiment 2 - Click-Through Rate

In order to change the users' click-through rate, we modified the range from which the advertisers' effect (e_q^a) is drawn. This parameter is the baseline probability that a user will click a generic ad of advertiser a, shown in query q. This probability can be modified by a targeting factor, for a targeted ad, and a promotion factor, for a promoted advertisement slot.

We ran two experiments - one in which we increased the expectation of e_q^a by 0.1 for all focus levels[1] (Ex. 2.1), and another in which we decreased it by the same amount (Ex. 2.2). We expected that increasing this parameter will increase the overall revenues and vice-versa, since increased advertiser effect will result in more clicks and therefore more conversions.

The results of these experiments are shown in Table 4.1, as well as in Figure 4.1. As expected, decreasing the advertisers' effect reduced the median score (by 15%), while increasing this effect raised the median score (by 3%). This effect was similar for most agents, except for Mertacor. The effect on all agents was found statistically significant ($p < 0.001$).

When examining the graph in Figure 4.1, we can clearly see that most agents exhibit a similar performance in some sense - their score in the benchmark competition exceeds the average of the low and high CTR experiments. This excess is due to overfitting the agents to the game parameters. As expected, we can see that almost all agents are optimized to the game parameters. However, the degree of this optimization varies drastically between agents. The most overfit agent is Mertacor - its score in the benchmark competition is higher by 14% than the its average score in the two CTR experiments. TacTex and tau11 show some overfitting (an increase of about 2%), and the other agents show very little overfitting.

Thus, we can conclude that most agents are robust against changes in the click-through rate, despite a slight overfitting to the game parameters. This result is not so surprising when keeping in mind that this rate is not known to the agents and that they estimate it during the game.

4.1.3 Experiment 3 - Conversion Rate

In these experiments we modified directly the Conversion Rate by 0.04,[2] in both directions (Ex. 3.1 & Ex. 3.2). The original parameter is known in advance to the agents, so we can assume that they all rely on it in their optimization.

We expected that changing this parameter will have a direct and similar effect on all agents, i.e., that an increased conversion rate will lead to higher scores.

The results of these experiments are shown in Table 4.1 and in Figure 4.2. As expected, decreasing the conversion rate reduced the median score (by 5.5%), while increasing it raised the median score (by 1.5%). This effect was similar for most agents, except for Mertacor whose score dropped in both scenarios by about 7%, an effect which was found statistically significant ($p < 0.01$).

[1] The values of these parameters are originally drawn uniformly at random from the ranges $[0.2, 0.3]$, $[0.3, 0.4]$ and $[0.4, 0.5]$ for focus levels $F0$, $F1$ and $F2$, respectively.

[2] The original conversion rates are 0.11, 0.23 and 0.36 for focus levels $F0$, $F1$ and $F2$, respectively.

As in the previous experiment, this setting also allows us to measure the overfitting of the agents to the exact value of CVR, by comparing the agent's benchmark score to the average of its scores in the two experiments.

In this experiment we see that most agents do not exhibit overfitting to the exact CVR. The only agents whose score exceeds the average significantly are Mertacor (by 7%) and tau10 (by 4.5%). This result is surprising since in TAC-AA the exact value of the CVR is a constant known to the agents, and so we expected that agents will be optimized to it.

4.1.4 Experiment 4 - Single Ad

In this experiment we reduced the number of advertising slots from 5 to 1, to simulate a banner-based advertising. This change is rather dramatic, and we expected that it will reduce drastically the median score. We also hypothesized that all agents will be affected in a similar way, since they all are optimized to multi-ad setting, where an agent can manipulate its position in order to optimize its sales and CPC. In the banner setting this flexibility is considerably diminished.

As we expected, the scores of all the agents dropped, and the median score was lower by 27%. However, the agents were not similarly affected - while the score of Mertacor was reduced only by 3% ($p > 0.2$), the score of other agents (tau10, tau11, Schlemazl and crocodile) dropped by about 15% and the score of TacTex dropped by 35%. The latter changes were found statistically significant ($p < 0.01$).

This experiment differentiates between agents who aim at lower positions and agents who aim at higher positions. The former agents are more affected by the elimination of these spots. Unlike the previous experiments, Mertacor was the most robust to this change, presumably since it aims at higher positions at all settings.

4.1.5 Experiment 5 - Population Size

In this set of experiments we varied the number of users in each product population, from 2000 to 20000. The population size in the original competition is 10,000 and is known to be fixed. Many agents employ Particle Filtering in order to estimate the users distribution, and knowing the exact population size is a precondition for this technique.

The results of these experiments are shown in Figure 4.3. Due to the capacity limit in TAC-AA, increasing the number of users does not increase the score significantly. However, reducing the number of users damages the performance of all agents drastically. The median score for increasing the number of users was approximately unchanged, while for decreasing the median score deteriorated quickly from -15% for 8000 users until a decrease of more than 80% for 2000 users. These decreases were found statistically significant ($p < 0.04$).

It is no surprise that the TAC-AA agents are not robust against changes in the population size. This size is known in advance and it's an important factor in the optimization process. Furthermore, it is reasonable to assume that most agents try to estimate the number of users in each state, and this estimation is based on a constant population size.

4.1.6 Conclusion

Our experiments show that most of the TAC-AA agents adapt well to different settings, despite being optimized to the exact game parameters. The top performing agents of TAC-AA - TacTex,

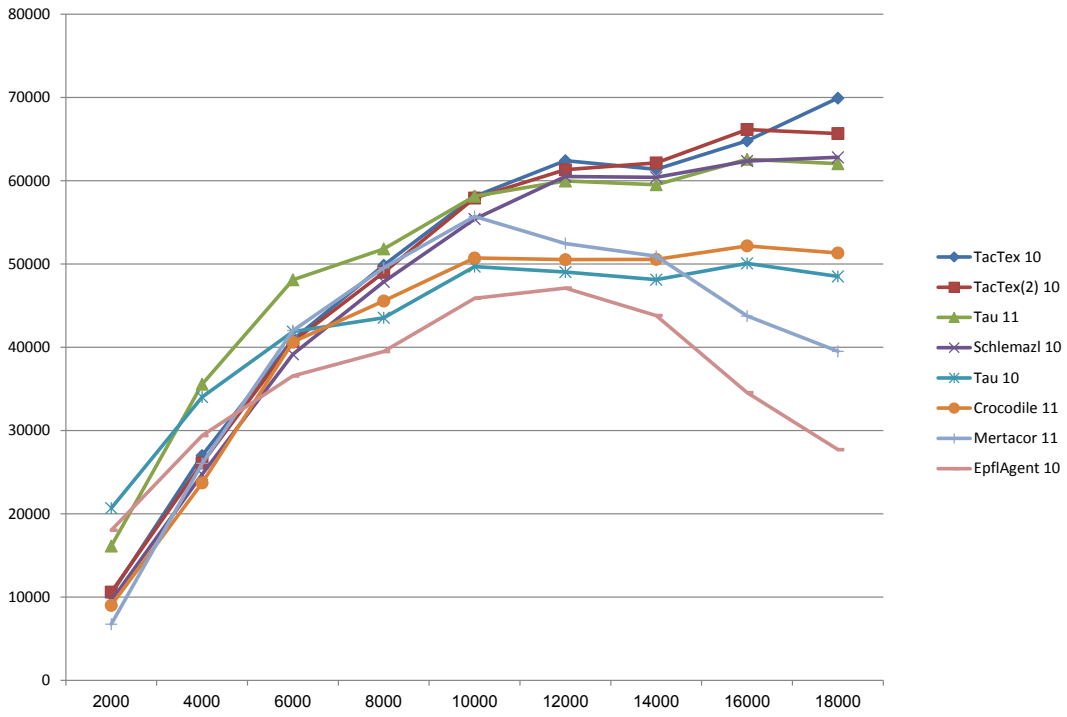

Figure 4.3: The results of experiment 5, where the users population size varies from 2000 to 20000.

Schlemazl and tau11 - are rather robust to most changes, but when the setting is changed drastically, as in experiments 4 and 5, their performance deteriorates.

This robustness result is somewhat surprising, since one could expect that the top performing agents in the TAC-AA would be more optimized to the exact game parameters and thus will be more affected by changes in these parameters (as is the case of Mertacor). However, the experiments show that most agents are less overfit to the game parameters than expected.

4.2 Agents Behavioral Identification

4.2.1 Strategic Fingerprint

In order to characterize an agent's strategic behavior, we use several attributes extracted from the games' logs, to form *strategic fingerprint* vectors. These vectors identify the agents, as each agent's strategic fingerprint vectors are in a different region in space. In addition, these strategic fingerprints are a good predictor of an agent's profit in a game.

Queries are naturally grouped into the three focus levels, and we further split them into specialty and non-specialty groups. Therefore, we use 5 distinct groups: $F2$ & $F1$ specialty

queries, and $F2$, $F1$ & $F0$ non-specialty queries. The attributes we use are:

1. **Query distribution of impressions, clicks and conversions:** We average across all days the relative part of impressions, clicks and conversions that the agent got from each of the above-mentioned focus-specialty groups. Namely, for each focus-specialty group g, we compute:

$$imps\_percent_g = \frac{1}{60} \cdot \sum_{d=1}^{60} \frac{\sum_{q \in g} imps_q^d}{|g| \sum_q imps_q^d}$$

$$clicks\_percent_g = \frac{1}{60} \cdot \sum_{d=1}^{60} \frac{\sum_{q \in g} clicks_q^d}{|g| \sum_q clicks_q^d}$$

$$sales\_percent_g = \frac{1}{60} \cdot \sum_{d=1}^{60} \frac{\sum_{q \in g} sales_q^d}{|g| \sum_q sales_q^d}$$

Since we measure only the percentage of impressions, clicks and conversions for each focus-specialty group, this attribute reflects only the way an agent distributes its budget across queries, and not the actual number of impressions, clicks and conversions. For example, using this attribute we can see whether an agent places ads only on his specialty products, or also on other, less profitable ads.

2. **Average ad position:** The average position of the agent's ads within each focus-specialty group, only for the days in which the agent's ad was shown.
For each query q in a focus-specialty group g the set D_q holds the days in which the agent's ad was shown in response to the query. We compute:

$$pos_g = \sum_{q \in g} \frac{\sum_{d \in D_q} pos_q^d}{5 \cdot |g| \cdot |D_q|}$$

This attribute tells us if the agent aims at higher or lower positions, and if it aims at different positions for different focus-specialty groups. For example, we can observe agents who try to get their ad shown first for their specialty products, but try to get lower positions for other queries.

3. **Proportion of active days:** The number of days in which the agent's ad was shown for each query, and then average within each focus-specialty group. For each focus-specialty group g (and D_q as defined above) we compute:

$$days_g = \frac{\sum_{q \in g} |D_q|}{|g| \cdot 60}$$

This attribute can be combined with the above-mentioned attributes to deduce the emphasis an agent puts on a certain focus-specialty group.

4. **The standard deviation of the agent's daily profit:** This attribute is oblivious to the relative part of the profit that comes from each focus-specialty group, but rather looks at the daily total revenues and total costs to compute a single attribute. We scale this attribute by the empirical maximal standard deviation observed across all games in order to normalize. Hence, this attribute is not directly related to the agent's profit, but rather to its stability.

Apparently, the strategic fingerprint vectors of each agent in various games lie in a typical range. To find these ranges we analyzed the logs of the TAC-AA 2011 finals and created a strategic fingerprint vector for each agent in each game.

To visually illustrate the strategic ranges of different agents, we used Principle Components Analysis (PCA) and projected the agents' vectors on the first 2 components. The result is shown in Figure 4.4, and we can see that each agent maintains a slightly different zone. However, there is no clear distinction between different capacities for each agent. We can conclude that the agents maintain similar behavior for all capacity values, and therefore the strategic fingerprint does not hold information about the agent's capacity.

To demonstrate agent identification using strategic fingerprints, we used a simple 3-Nearest Neighbor model that classifies agents based on their strategic fingerprint vectors. The error rate of this model was 5.9%. Most of the errors of the model are due to a specific agent, Mertacor, whose strategic range is rather wide, while the other agents are more accurately classified.

4.2.2 Relation to Profit

We also used the strategic fingerprints of the TAC-AA 2011 Finals in order to find the relation between strategic fingerprints and agents' performance.

The benchmarks to which we compare our results are two simple predictors - one that always outputs the agent's average score (has 19% relative error rate), and one that given an agent name and its capacity predicts the corresponding average (has 10.8% relative error rate). It should be noted that our model is oblivious to both the agent name and its capacity. In addition, it has no knowledge of the actual number of impressions, clicks and conversions that the agent was subjected to, nor its actual profit. It only has information about the emphasis it puts on different query groups and about its stability.

A simple 3-Nearest Neighbor model to predict an agent's profit from its strategic fingerprint had relative error rate of 14.7%, while a linear model had 12.5% relative error rate. Using Boosting with regression trees the relative error rate was reduced to 9.7%.

4.2.3 Relation to Robustness

The typical strategic ranges of agents, mentioned in Section 4.2.1, suggest an interesting relation to agents' robustness, as analyzed in Section 4.1. We hypothesized that the more robust agents are characterized by a medium strategic range. Presumably, an agent with a narrow strategic range is not flexible enough to overcome changes in the game settings. On the other end, an agent with a wide strategic range is not able to maintain its strategy across games and is too susceptible to changes.

To test this hypothesis, we created four measures for the width of agents' strategic range. We used the data from the benchmark competition (see Section 4.1) to calculate these measures. First, we computed the distance between the strategic fingerprint of each agent in each game to its mean strategic fingerprint across all games. We then computed for each agent the average, median and 0.8 percentile distance, as well as the variance of its distances. Since we assumed that having a middle value is optimal, we computed the median of each of the above-mentioned measures. Finally, the score of each agent was the absolute value of the difference between its measure and the corresponding median. Thus, agents with wide or narrow strategic ranges got high scores, and agents with medium ranges got low scores. The scores of each agent are shown in Table 4.2.

As a measure of robustness we used agents' ability to maintain their market share in the experiments detailed in Section 4.1. We computed the market share of each agent in the bench-

Agent Name	Abs. Diff. from Median of Average Distance	Abs. Diff. from Median of Median Distance	Abs. Diff. from Median of 0.8-percentile	Abs. Diff. from Median of Dist. Variances
SCHLEMAZL10	0	0	0	0.006
CROCODILE11	0.004	0.02	0.054	0.011
TACTEX10	0.045	0.035	0.014	0
TAU11	0.059	0.033	0.169	0.036
EPFLAGENT10	0.278	0.301	0.202	0.007
TAU10	0.447	0.405	0.58	0.058
MERTACOR11	0.621	0.693	0.635	0.087

Table 4.2: The scores of agents in the width measures mentioned in Section 4.2.3. The agent name column is an example of a permutation induced by the absolute deviation from the median of average distances measure.

mark competition and in each of our experiments. Then we computed the relative change in the agent's market share in each experiment, in comparison to its share in the benchmark competition. Thus, we got a robustness score for each agent in each experiment.

The strategic range measures and the robustness scores induce agent name permutations. We sorted agents' range-width scores in increasing order, to get four agent permutations. According to our hypothesis, the first agents in each such permutation are the more robust agents. Similarly, we sorted agents' robustness scores in decreasing order, and got 15 permutations, one for each of the competition in our experiments. The first agents in these permutations are the more robust agents.

We measured the Kendall's tau distance [Ken38] between each of the four range-width permutations and each of the fifteen robustness permutations. We then measured the Kendall's tau distance between four random permutations and each of the fifteen robustness permutations. The average distance between the strategic range-width permutations and the robustness permutations was 0.69. The average distance between the random permutations and the robustness permutations was 0.77. This difference was found significant in a t-test comparing the two groups of distances ($p < 0.007$).

However, when taking out Mertacor from the permutations, the result is no longer significant ($p = 0.12$), suggesting that Mertacor is the only agent for which there exists a strong relation between its strategic range and its robustness. For the rest of the agents this relation is weaker, and our hypothesis is rejected.

4.2.4 Conclusion

We can conclude that our strategic fingerprints model well the behavior of TAC-AA agents. Each agent has a typical range of strategic fingerprint vectors, and these vectors are well related to its performance in the game.

The strategic fingerprint vectors can be used to identify this agent with high precision. This identification technique may even overcome log anonymization, for example.

Since the strategic fingerprint reflects the agent's performance in the game, it could possibly be used for agent optimization, e.g., in a gradient descent method. However, further research is needed in order to assess the contribution of such optimization to the agent's performance.

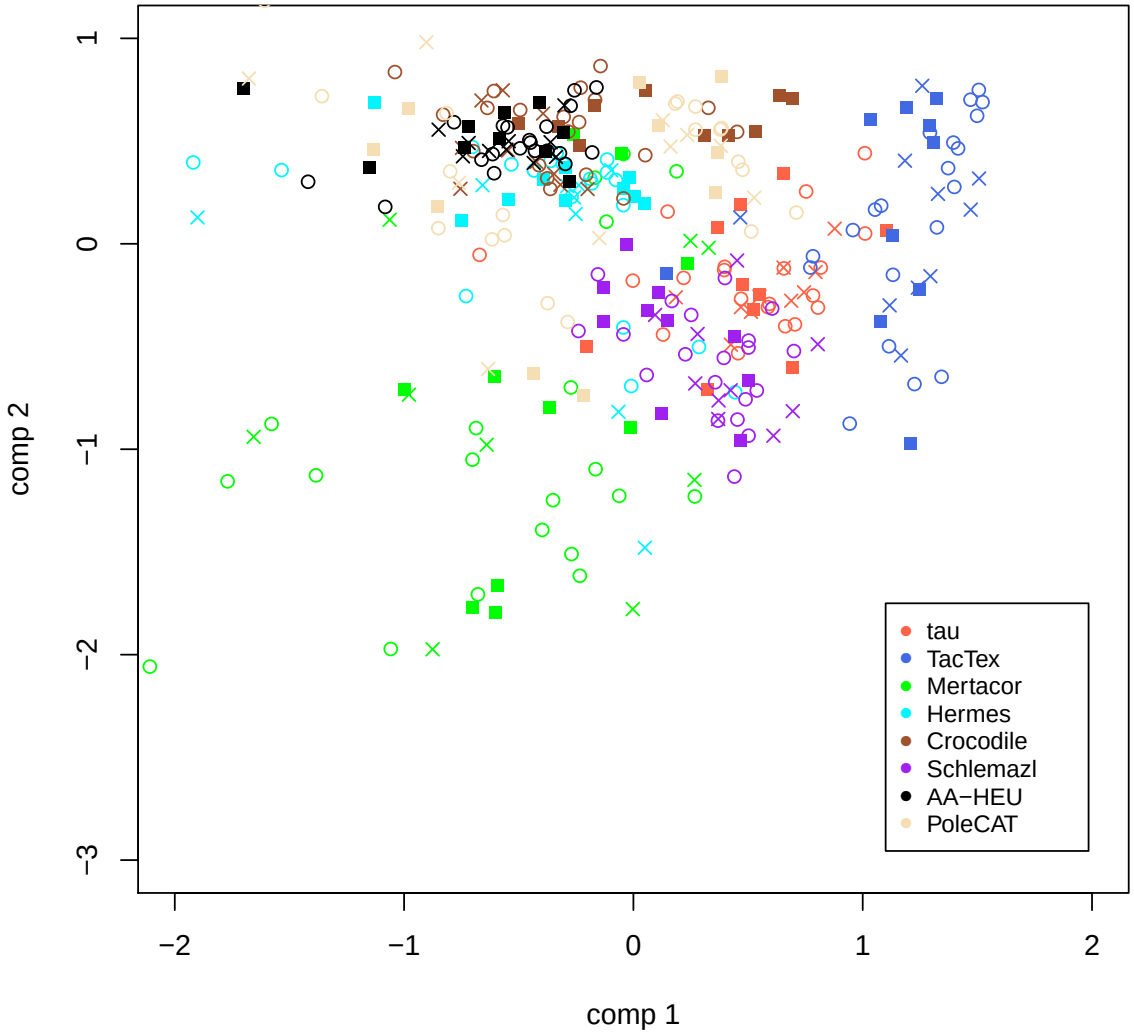

Figure 4.4: The Strategic Fingerprints of TAC-AA 11 finalists, projected on the 2 most principle components. The different point styles correspond to different capacities: high capacity is marked with full squares, medium capacity is marked with empty circles and low capacity is marked with crosses.

Chapter 5

Concluding Remarks

In this work, we presented our top performing TAC-AA agent. This agent uses ML models and techniques, which enable a simple and efficient optimization process. We showed that further improving these ML models would have only a minor impact on our agent performance. This result demonstrates that the the performance of a learning system might not improve significantly even if all errors are eliminated.

A more significant improvement could be possibly gained from changing the agent's optimizer to look several days ahead when optimizing the bid bundle. Another possible improvement could be to model the competitors behavior in order to predict their future bids.

In addition, we showed that most of the TAC-AA agents are robust to environmental changes, despite their optimization to the exact TAC-AA setting. This robustness is "good news", and it is a very important ingredient if one wishes to relate the agents back to real world scenario.

We also presented a behavioral model of TAC-AA agents that can be used to identify an agent with high probability and to predict its profit in a game. In addition, this model could possibly be used in an optimization process of a TAC-AA agent.

Bibliography

[AFMP08] G. Aggarwal, J. Feldman, S. Muthukrishnan, and M. Pal. Sponsored search auctions with markovian users. *CoRR*, abs/0805.0766, 2008.

[BGNS10] J. Berg, A. Greenwald, V. Naroditskiy, and E. Sodomka. A knapsack-based approach to bidding in ad auctions. In H. Coelho, R. Studer, and M. Wooldridge, editors, *ECAI*, volume 215 of *Frontiers in Artificial Intelligence and Applications*, pages 1013–1014. IOS Press, 2010.

[CDE+07] M. Cary, A. Das, B. Edelman, I. Giotis, K. Heimerl, A. R. Karlin, C. Mathieu, and M. Schwarz. Greedy bidding strategies for keyword auctions. In J. K. MacKie-Mason, D. C. Parkes, and P. Resnick, editors, *ACM Conference on Electronic Commerce*, pages 262–271. ACM, 2007.

[EOS07] B. Edelman, M. Ostrovsky, and M. Schwarz. Internet advertising and the generalized second-price auction: Selling billions of dollars worth of keywords. *American Economic Review*, 97(1):242–259, 2007.

[GNO+07] A. Greenwald, V. Naroditskiy, T. Odean, M. Ramirez, E. Sodomka, J. Zimmerman, and C. Cutler. Marginal bidding: An application of the equimarginal principle to bidding in tac scm. In J. Collins, P. Faratin, S. Parsons, J. A. Rodríguez-Aguilar, N. M. Sadeh, O. Shehory, and E. Sklar, editors, *AMEC/TADA*, volume 13 of *Lecture Notes in Business Information Processing*, pages 217–239. Springer, 2007.

[HFH+09] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The weka data mining software: an update. *SIGKDD Explor. Newsl.*, 11(1):10–18, November 2009.

[JW10] P. R. Jordan and M. P. Wellman. Designing an ad auctions game for the trading agent competition. In E. David, E. Gerding, D. Sarne, and O. Shehory, editors, *Agent-Mediated Electronic Commerce: Designing Trading Strategies and Mechanisms for Electronic Markets*, chapter Lecture Notes in Business Information Processing. Springer-Verlag, 2010.

[JWB10] P. R. Jordan, M. P. Wellman, and G. Balakrishnan. Strategy and mechanism lessons from the first ad auctions trading agent competition. In D. C. Parkes, C. Dellarocas, and M. Tennenholtz, editors, *ACM Conference on Electronic Commerce*, pages 287–296. ACM, 2010.

[Ken38] M. G. Kendall. A new measure of rank correlation. *Biometrika*, 30:81–89, 1938.

[LPSV07] S. Lahaie, D. M. Pennock, A. Saberi, and R. V. Vohra. Sponsored search auctions. In N. Nisan, T. roughgarden, E. Tardos, and V. V. Vazirani, editors, *Algorithmic Game Theory*, chapter 28, pages 699–716. Cambridge University Press, New York, NY, 2007.

[PCS10] D. Pardoe, D. Chakraborty, and P. Stone. Tactex09: a champion bidding agent for ad auctions. In W. van der Hoek, G. A. Kaminka, Y. Lespérance, M. Luck, and S. Sen, editors, *AAMAS*, pages 1273–1280. IFAAMAS, 2010.

[PS11] D. Pardoe and P. Stone. A particle filter for bid estimation in ad auctions with periodic ranking observations. In L. Sonenberg, P. Stone, K. Tumer, and P. Yolum, editors, *AAMAS*, pages 887–894. IFAAMAS, 2011.

[Var07] H. R. Varian. Position auctions. *International Journal of Industrial Organization*, 25(6):1163–1178, 2007.