Semester Project: Bidding Agent for Advertisement Auctions

Elias Mirza

Ludek Cigler

Abstract

One year after the Trading Agent Competition, I try to come with a new improved bidding strategy that would optimize the profits in an Ad Auction. I present and analyze some results of the past competition. I then describe the strategy I tried to implement. I will focus on the user population estimation I implemented, as I spent most of time designing and implementing it.

1. Contents :

2. Introduction

2.1. Internet Advertising:

Internet advertising has become a very important part of today's marketing processes. Advertisers compete among each other to get the best promotion of their products on the internet. Sponsored search is a popular facet of internet advertising where each advertiser buys, in an ad auction, their ad position on a page for a given query. Given the huge amounts of money spent for internet advertisement campaigns, a new problem has risen; the use of automated auctions address the combinatorial problem of quoting the appropriate bid for each display slot for each query, in order to sell the most while minimizing the costs.

2.2. Defining the Trading Agent Competition for Advertisement Auctions (TAC-AA) game:

The TAC-AA was held for the first time in 2009. The Trading Agent Competition simulates the scenario of the Ad Auction, the retail market and the Users behavior. The main goal of this competition is to discover new bidding techniques that may optimize the profits of advertisers. The auction follows the second price pricing model, which means that we have N bidders and K items to sell (K slots), where every bidder prefers higher slot to a lower one (for the same price), and I^{th} highest bid gets I^{th} slot and pays what the $(I-1)^{th}$ bid was.

2.3. Defining the TAC-AA parameters:

Each agent is assigned a product to sell. This product is represented a component and its manufacturer. So each advertiser has a Component and Manufacturer Specialty. The Manufacturer Specialty increases the revenue of a conversion. The Component specialty increases the conversion rate. There are 3 Components (TV, Audio, and DVD) and 3 Manufacturers (PG, Flat and Lioneer), so we get a total of 9 different agents. (i.e.: An agent may want to sell a Product (PG, TV), he would thus prefer to buy ads for queries that contain the keywords PG and TV, which represent his "Specialty")

The TAC-AA game also simulates users that will generate queries, click on Ads, and buy products. These users are distributed into 9 different populations that have a product preference as described above with the agents. Of course each population would be more likely to buy from the advertiser that sells the product that matches their preference. Each population is represented by 10 000 users. These users generate only queries containing keywords with their manufacturer and component preferences. The users in a population may be in 5 different states: {NS, IS, F0, F1, F2} (I describe these each states below). These states define the queries typed by a user from such a population. (Figure.1) shows the allowed transitions between the different states.

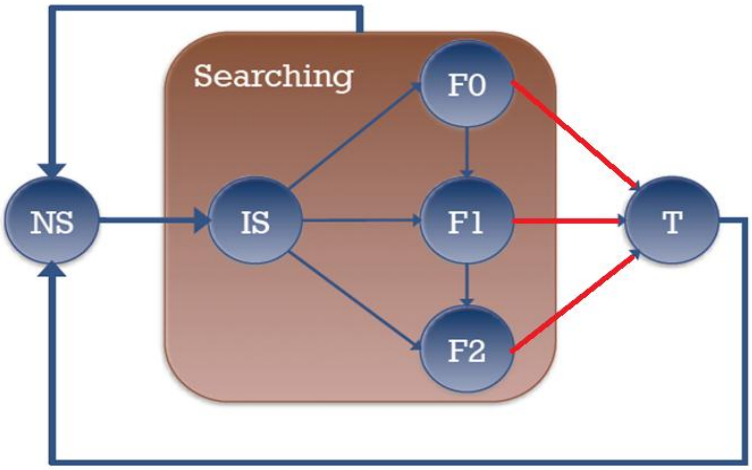

Figure 1

Let's define the states of the simulated users' population:

- NS: Non-searching users, they do not generate queries they only wait to get to the IS state to start generating queries.
- IS: Informational-searching users, they can generate F0, F1, or F2 queries, but do not transact.
- F0: Focus level 0-searching users, they generate the (Null, Null) query they focus on none of the specialties, these users may transact.
- F1: Focus level 1-searching users, they generate (Manufacturer specialty, Null) OR (Null, Component specialty) with equal probability, these users may transact.
- F2: Focus level 2-searching users, they generate (Manufacturer specialty, Component Specialty) queries, these users may transact.
- T: Transacted users, they have clicked on an ad, and then made a transaction.

So as a total number of possible generated queries for a certain population are: 1 F0 query, 2 F1 queries and 1 F2 query. So for all the populations we get 1 F0 query, 6 F1 queries and 9 F2 queries. The conversion probability is higher for higher focus level users.

Another important factor in the TAC specifications is the capacity distribution constraint. Each advertiser is assigned a capacity distribution at the beginning of the game this capacity varies between LOW (300), MEDIUM (450), and HIGH (600) on a window of 5 Days. Once an advertiser has sold more than his allowed capacity, the conversion rates decrease proportionally with the difference between the capacity distribution and the quantities sold, following the next function:

$I_d = \lambda^{\left(\left(\sum_{i=d}^{d-(W-1)} c_i\right) - C^{cap}\right)^+}$, with:

$C^{cap} = The\ critical\ distribution\ capacity = HIGH,\ MEDIUM\ or\ LOW$
$\lambda = Distribution\ capacity\ discounter = 0.996$
$W = Capacity\ distribution\ window = 5$

3. Our Strategy and its motivations:

3.1. Bid on the most profitable keyword:

In fact we know that once we sell more than critical capacity our revenue will start decreasing and then we will start losing money with each click, as we will get lots of clicks that will cost us money and almost no conversion so no revenue!

A first thought would be fixing a budget limit, that is: Once the value per click is lower than the cost per click we would stop bidding so that we get no bad clicks. This strategy makes us avoid the negative value clicks, but doesn't make a good profit of the positive value clicks, because the bids are high and the margin is small. Another way, is bidding lower, the problem in this case is that we may not get enough clicks, and thus enough conversions to make full use of our capacity. So here an idea arises, let's bid on the most profitable keyword in order to use our full capacity the best profitable way.

The figure 2 shows the effect of setting a budget limit, and the one of using smaller bids:

c^*: *sales with budget limit*
c_2: *sales with lower bids*
c_1: *sales without budget limit*

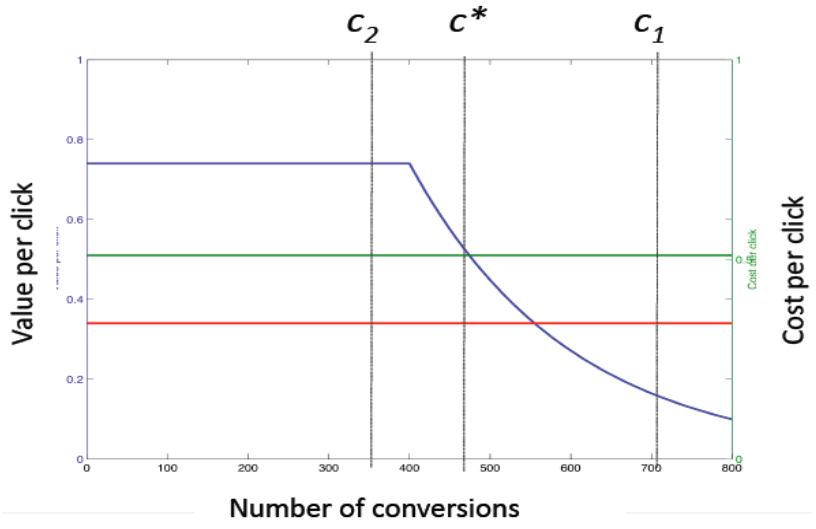

Figure 2

3.2. We define the needed parameters to decide what the most profitable keyword is:

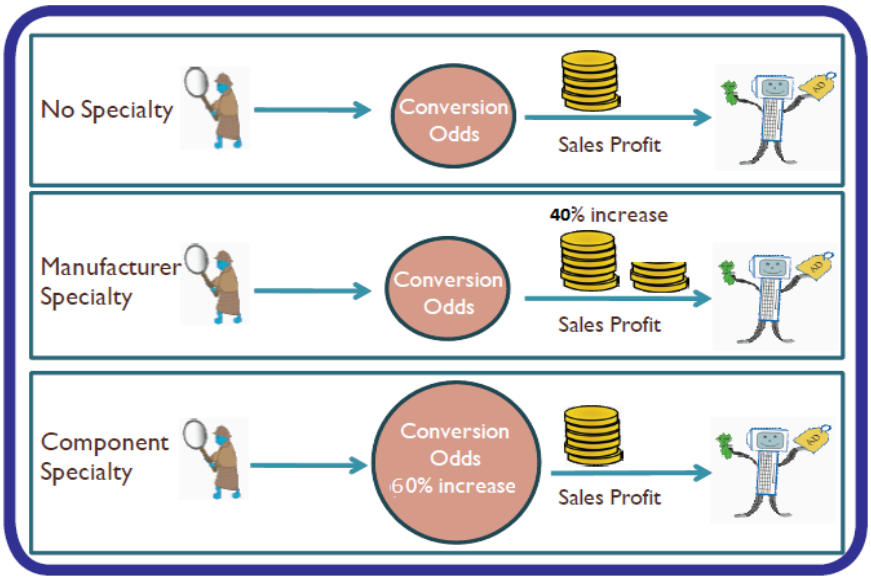

Figure 3

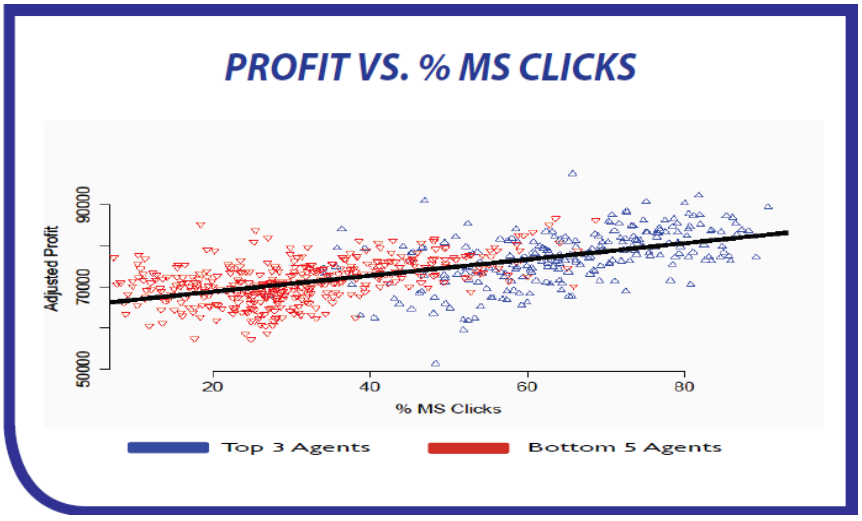

Figure 4

The specializations: Manufacturer specialty raises the profit per conversion. The component specialty raises the conversion rate. In both case, we obviously see a raise in the value of a click. Obviously keywords with specialization are more interesting than non specialty matching keywords.

A new question arises: on which type of specialty should we focus? Is it the component or the manufacturer specialty? An interesting analysis of last year's competition showed that the best agents focused their sales on Component specialty ads as we can see in the

figure 3. As we can see the best agents have more than 60% of their clicks on manufacturer specialty keywords.

So we thought that a good strategy would be to focus on keywords that will give us the *best profit per-conversion*, obviously these keywords are the ones that have manufacturer specialty.

3.3. Useful estimators for other interesting game parameters:

Now that our goal is fixed, having good estimates of the game will probably improve our choice of the most profitable keyword to bid on. The paper of last year's winner agent "TacTex" led us to the creation of a population estimator that will give us the number of users per state for each population, which will let us know better the number of impressions we should expect for each query. We then thought of estimating the click through rate for different bid values for a defined query.

Thus we would obtain the number of impressions, and click through rate, so we get an estimated number of clicks for each bid/ query. Calculating the conversion rate knowing that the ad is clicked is an easy task, as it does not depend on the bid, it depends only on the state of the users, the capacity constraint and the manufacturer specialty, which are all well known and defined parameters.

4. Presentation of our strategies:

We now present in more details the design of our strategies. In fact we designed 2 strategies, a simple one without the estimators, and an improved one that makes use of the estimators that will improve our evaluation of the *most profitable keyword.*

4.1. The simple Strategy:

We designed a simple strategy that works as follows:

a) We try to have the same optimal profit from all keywords, while not exceeding the capacity constraint by 20%.

b) We increase the bids until 1.2*Capacity is reached.

c) We decrease the bids if we are over the 1.2*Capacity.

Let's calculate the effect of going over the 1.2*Capacity:

$$I_d = I_d = \lambda^{\left(\left(\sum_{i=d}^{d-(W-1)} c_i\right) - C^{cap}\right)^+} = \lambda^{(1.2*C^{cap} - C^{cap})^+} = \lambda^{0.2*C^{cap}+} = \begin{cases} 0.996^{0.2*300} = 0.79 \\ 0.996^{0.2*450} = 0.70 \\ 0.996^{0.2*600} = 0.62 \end{cases}$$

After some thoughts we chose to fix a heuristic value of $C^{cap} + 90$, so that the effect would almost be the same for the 3 different possible capacities which is: $0.996^{90} = 0.70$.

This Strategy follows the basic principle described in the part 2 that is not setting a budget limit, but instead finding the right bid value to maximize the profit per-click, and make best use of the capacity allowed.

4.2. The improved Strategy:

We try to compute the optimal bid which maximizes the profit function described below:

$$Profit = \sum_{k \in Queries} Imp(k) * CTR(k) * \left(Rev(k) * ConvR(k) - Bid(k)\right)$$

Imp(k) = Impressions for query k: *which is calculated using the population estimator we implemented, as for each type of population we know the number of users per state, which generate impressions.*

CTR(k) = Click-Through-Rate of our Ad given that we have an impression on this query k: *which is calculated using the CTR estimator which bases its estimations which depends on the position of our ad, that we try to guess depending on our bid.*

Rev(k) = Revenue we get if we have a conversion on this query k: *which depends on our manufacturer specialty.*

ConvR(k) = Conversion-Rate given that we have a click on the query k: *which depends on our component specialty, and the states of the users.*

Bid(k) = The bid we made for the query k: *This bid will be the solution of our optimization problem, we defined before.*

Let's now describe the algorithms used to compute the needed estimations:

4.2.1. The population Estimator:

a) *Overall description:*

Let's first describe briefly in words the principle of particle filtering used by our estimator: For each population of the game we simulate different "population particles" that represent a possible states distribution of the real population. We then compute the probability that this particle exists given some observations of the game, we then assign a weight to each particle given its probability of occurrence.

b) *The parameters of the estimator:*

Now that we fixed our goals, let's determine the parameters that will let us create possible simulations. As we described earlier, the population users switch from a state to another following a Markov Chain (Fig.1). The parameters that we possess give us the transitions probabilities between all the states except the probability of getting a transaction, that is the transitions $F_0 -> T, F_1 -> T, and\ F_2 -> T$, but for the moment let's suppose that all our state transitions have well known fixed probabilities including the three latter undefined transitions. We get that all the outgoing edges from a given state follow a multinomial distribution, so we have to generate random distributions from a given state and a given number of users in this state (i.e.: If we have a 200 users in F_0, after 1 simulation we would get: 23 NS, 130 F_0, 47 F_1 OR 20 NS, 140 F_0, 40 F_1 OR ....). So we generate 1000 particles for each population following this principle. There is one more thing to indicate, there exists transitions bursts that occur with probability 10% between IS and NS, which means that the transition rate of the users of a certain population on a particular day from NS to IS increases by a factor of 20 whenever a burst occurs. This burst has to be simulated as well each day, as considering an average, between the transition rate with a burst and the normal transition rate, is non functional.

c) *Calculating the weights, normalizing and re-sampling:*

Once the daily particles generated, we start filtering these particles using some observations from the game. For the filtering process we will give each particle "p" on each day "d" a certain weight "w_d^p":

$w_d^p := w_{d-1}^p * prob^p_{n_{F_2}, 1/3}(n_{IS}|Impressions_{F2})$, with $prob^p_{n_{F_2}, 1/3}(n_{IS}|Impressions_{F2})$ described below. I also added another condition:

$w_d = \begin{cases} 0 & if\ n_{F_1} > Impressions_{F_1} \\ w_d & otherwise \end{cases}$, thus taking into account the fact that n_{F_1} cannot be greater than $Impressions_{F_1}$.

Let's define P := {the set of all particles}, for the normalization we do:

$\forall\ i \in P :\ w_d^i = \frac{w_d^i}{\sum_{j \in P}\ w_d^j}$;

Once the normalization complete, we resample our whole set of particles using the ***normalized*** weights previously calculated, following this simple algorithm:

For each particle p in P do:
if (w_d^p == 0): delete the particle
else: generate n particles p, with n := (w_p^i * total number of particles)

Let's now show what $prob_{n_{IS},1/3}(n_{F2}|Impressions_{F2})$ means, and how it is calculated. We know that for any population all the F_2 users generate F_2 queries, and each IS user may generate with the same probability an F_0, F_1, or F_2 query. So from the impressions we get for an F2 query we can calculate the probability of existence of each particle, in fact:

$$Imp_{F2}(q) \approx N_{F_2}(q) + \frac{1}{3}N_{IS}(q) \quad (1)$$

$Imp_{F_2}(q) = \#\ of\ Impressions\ for\ query\ q\ of\ type\ F_2$

$N_{F_2}(q) = \#\ of\ F_2\ users\ from\ the\ population\ that\ has\ preferences\ defined\ like\ q$

$N_{IS}(q) = \#\ of\ IS\ users\ from\ the\ population\ that\ has\ preferences\ defined\ like\ q$

Expression (1) becomes: $Imp_{F2}(q) - N_{F_2}(q) \approx \frac{1}{3}N_{IS}(q)$ we thus can get a binomial distribution $Bin \sim X(n,p)$ with parameters: $n = N_{IS}(q)$ and $p = \frac{1}{3}$ and we calculate the desired probability:

$$\boldsymbol{prob_{n_{IS},1/3}(n_{F_2}|Impressions_{F_2})} := P\left(X = Imp_{F_2}(q) - N_{F_2}(q)\right)$$

d) *Estimating the unknown transitions probabilities to T:*

For the moment we supposed that we knew all the transition probabilities, let's now see how we could estimate the 3 missing transition probabilities ($F_0 -> T, F_1 -> T\ and\ F_2 -> T$):

We know that in order to get a transaction, we need a click and then a conversion. We have the click probabilities: $p_{clickF_0} \in [0.20, 0.30], p_{clickF_1} \in [0.30, 0.40]\ and\ p_{clickF_2} \in [0.40, 0.50]$ (these probabilities may vary), and the conversion probabilities: $\pi_{F_0} = 0.11$, $\pi_{F_1} = 0.23\ and\ \pi_{F_2} = 0{,}36$. We also know that everyday 5 ads are shown and a user can transact from any of these 5 ads, starting from the top ad, and continuing to the next Ad with a continuation probabilities $\gamma_{F_0} \in [0.20, 0.50]$, $\gamma_{F_1} \in [0.30, 0.60], \gamma_{F_2} \in [0.40, 0.70]$ until he transacts or arrives to the end of the ads. We thus have the Markov chain shown in (Figure.6).

After resolving numerically this chain using approximate values for each unknown parameter of the chain, I got:

$p_{TF_0} = 0.063$, $p_{TF_1} = 0.19\ and\ p_{TF_2} = 0.36$. As these values were not precise, and because the transaction probabilities depend on the ads (generic or targeted) I decided to make use of the fact that I possess 1000 particles (so I could have different random transaction probabilities), so I created a normal distribution $N_{F_i} \sim Normal\ (p_{TF_i}, (p_{TF_i}/4)^2)$ around each transaction probability p_{TF_i}. I then pick randomly a new transaction probability from the corresponding Normal distribution.

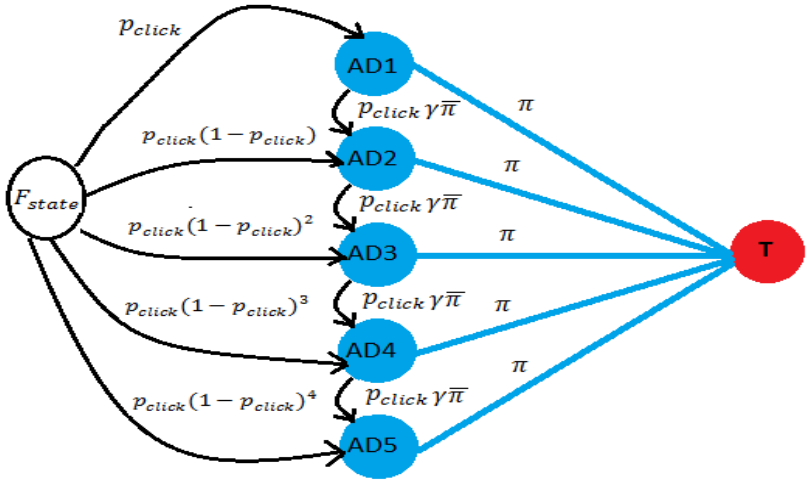

Figure 6

e) *Using the estimated particles to get an overall estimation:*

Once done with the particles' generation, filtering and re-sampling, we have to obtain an estimation of the population. I just do an average of the number of users per state in the 1000 particles, as these have all the same weight after the normalization and re-sampling.

4.2.2. The Click-Through-Rate (CTR) estimator:

As described earlier, the highest positioned Ad has the highest click probability, thus it has the highest CTR. We also know that the highest bid gets the highest Ad position, the 2nd highest gets the 2nd position and so on ... Thinking this way there may be a direct relation between the bids submitted and the CTR. When deriving the profit function we notice an exponential relation between the bids and the CTR (i.e.: $CTR = a_1 e^{a_2\, bid}$). The idea now would be finding the constants a_1 *and* a_2 such that we could get an estimation of the CTR for a given bid. I used the exponential regression approximation to calculate these parameters. Once the CTR estimator was ready, I tried it with the data. Unfortunately it didn't work. I then plotted the CTR as a function of the bids, and noticed that there was no relation at all between the bid and CTR, the only existing relation was between the CTR and the Ad position (which I plotted also, and it showed an exponential slope). After thinking, it was obvious that there was neither a direct relation between the bid and the CTR, nor between the bid and the Ad position. As the Ad position depends on our bid AND on the bids of the other advertisers.

So I couldn't get an estimation of the CTR from the bids I submit. This parameter is needed for my optimized strategy, thus I couldn't try it. The "future work" section will suggest a possible approach for estimating the relation between the CTR and the submitted bid.

4.2.3. Optimizing the profit using the estimators:

We have the profit following profit function:

$$Profit = \sum_{k \in Queries} Imp(k) * CTR(k) * \big(Rev(k) * ConvR(k) - Bid(k)\big)$$

With the Capacity constraint:

Capacity= $C^{cap} = \sum_{d=day-window+1}^{day} \sum_{k \in Queries} Imp_d(k) * CTR_d(k)$

We create and optimize the following Lagrangian function:

$Profit - \lambda * (C^{cap} - \sum_{d=day-window+1}^{day} \sum_{k \in Queries} Imp_d(k) * CTR_d(k))$

We will get λ as a function of many of the game's parameters. I implemented the Newton method to get the value of λ.

5. Analysis and Testing:

The data provided from the semi-finals game showed that there was not a really important relation between the CTR and the bids made. The data seemed pretty random, and this randomness seems explained by the fact that the position not only depended on our bid but also on the bids of the other advertisers.

The population estimator somewhat worked as I almost never got a total weight of 0 during the 60 days for all the populations simulated. However comparisons with the game data after the game showed that the estimator was still not very precise. This is in some way due to the fact that we have data with 1 day late so I always have to do 1 extra unchecked simulation and submit its result for the profit optimizer as it needs it right away. Another issue occurs with the bursts that we get with probability 10% on each day, these bursts are sometimes hard to detect, and it may happen that on some day a particle with burst is kept with another particle without burst which is totally wrong may corrupt the results. All these issues are still open for solutions...

(I couldn't complete this part on time ...) I'll add nice graphs to show the results..

6. Future work:

As stated before, there are still some unfixed issues with improved strategy, on theory it seems but it would be better to complete its implementation and see how it works.

So a future possibility would be finding a way to estimate the opponent advertisers' bids as a way to obtain an approximation of our ad position given our bid. This would complete the missing part of my agent. Another thing to be considered is correcting the small problems with the population estimator, finding a solution for the "1 day late problem" and for the "burst problem" should improve the results a lot. ...

(I couldn't complete this part on time ...)

7. References:

[1] D. Pardoe, D. Chakraborty, and P. Stone. TacTex09: *A champion bidding agent for ad auctions,* May 2010.

[2] L. Cigler. *Bidding Agent for Advertisement Auctions*, 2009.

[3] M. Sanjeev Arulampalam, Simon Maskell, Neil Gordon, and Tim Clapp. *A Tutorial on Particle Filters for Online Nonlinear/Non-Gaussian Bayesian Tracking.* February 2002.

[4] M. Mühlich. *Particle filters an overview*.

[5] G. Balakrishnan and Ye Wang. *The Bidding War Trading Agents and the 2009 TAC Ad Auctions Tournament Poster.* 2009.

[6] P. R. Jordan, B. Cassell, Lee F. Callender, A. Kaul and M. P. Wellman. *Trading Agent Competition, Ad Auctions.* March 2010.