TacTex09: Champion of the First Trading Agent Competition on Ad Auctions

Technical Report UT-AI-10-01

David Pardoe, Doran Chakraborty, and Peter Stone
Department of Computer Science
The University of Texas at Austin
{dpardoe, chakrado, pstone}@cs.utexas.edu

1. INTRODUCTION

Sponsored search [4] is one of the most important forms of Internet advertising available to businesses today. In sponsored search, an advertiser pays to have its advertisement displayed alongside search engine results whenever a user searches for a specific keyword or set of keywords. An advertiser can thereby target only those users who might be interested in the advertiser's products. Each of the major search engines (Google, Yahoo, and Microsoft) implements sponsored search in a slightly different way, but the overall idea is the same. For each keyword, a *keyword auction* [5] is run in which advertisers bid an amount that they are willing to pay each time their ad is clicked, and the order in which the ads are displayed is determined by the ranking of the bids (and possibly other factors).

The Trading Agent Competition Ad Auctions Game (TAC /AA) [3] was designed to providing a competitive sponsored search environment in which independently created agents can be tested against each other over the course of many simulations. In this report, we describe TacTex, the winning agent in the first (2009) TAC/AA competition. TacTex operates by estimating the full game state from limited information, using these estimates to make predictions, and then optimizing its actions with respect to these predictions. This report is intended to supplement our conference paper on TacTex [6] by providing a more thorough description of the agent. Sections 4 and 5 contain significant added material, while Sections 6, 7, and 8 contain some added details. Algorithms 1 through 11 have been added to summarize certain parts of the agent described in the text. For discussion of the competition and experimental results, see the conference paper.

2. GAME DESCRIPTION

We begin by providing a summary of those parts of the TAC/AA game that are most important for understanding the design of TacTex. For full details, see the game specification [2].

Overview: In each TAC/AA game, eight autonomous agents compete as advertisers to see who can make the most profit from selling a limited range of home entertainment products over 60 simulated game days, each lasting 10 seconds. Products are classified by manufacturer (*flat*, *pg*, and *lioneer*) and by component (*tv*, *dvd*, and *audio*) for a total of nine products. Search engine *users*, the potential customers, submit queries consisting of a manufacturer and a component, although either or both may be *null*, i.e., miss-

ing. There are thus 16 total query types, divided into three *focus levels*: F0 (the query with both manufacturer and component null), F1 (the six queries with one null and one specified), and F2 (the nine queries with both specified). Each day, for each of the 16 query types, a keyword auction is run. For each auction, an advertiser submits i) a (real, non-negative) bid indicating the amount it is willing to pay per click, ii) an ad, and iii) a spending limit (optional). Ads can be either *targeted* (specifying both a manufacturer and product) or *generic* (specifying neither). (The set of 16 (bid, ad, spending limit) tuples can be said to be an advertiser's action space, and Sections 4 through 8 essentially describe how TacTex maps its observations into this space each day.) The top five bidders have their ads shown in order, but if an advertiser hits its spending limit (as a result of having its ad clicked enough times), its ad is not shown for the rest of the day, and all advertisers with lower bids have their ads move up one position. Bids must exceed a small reserve price.

Users: There is a fixed pool of users, each of which remains interested in a specific product throughout the game and only submits queries corresponding to this product. However, users cycle through states corresponding to the focus levels according to a specified transition model. Users begin in a non-searching (NS) state, and can then transition through a searching (IS) state (which may submit a query of any focus level but will not make a purchase) to one of three buying states (F0, F1, or F2, each of which submits a query of the corresponding focus level and makes a purchase with a probability that increases with the focus), and eventually back to the non-searching state. For each product, the total number of users in any state can vary widely and rapidly.

Click model: Every searching or buying user submits one query per day and then proceeds through the resulting ads in order of advertiser ranking. When an advertiser's ad is shown, it is said to receive an *impression*, but not all impressions result in clicks. The default user behavior is as follows. If a user submitting query q reaches the ad of advertiser a, the probability of a click is e_q^a, a hidden parameter drawn randomly at the start of each game. If the user clicks, it will then *convert* (make a purchase) with a probability dependent on the user's focus level. For each conversion, the advertiser receives $10. (This amount is technically the sales profit before considering advertising costs, but we will simply refer to it as the agent's *revenue*). If the user does not convert, it proceeds to the next ad with probability γ_q, another randomly drawn, hidden game parameter. Thus, the higher the position of the ad, the more likely it

| | Agent actions | | | | Results | | | | | |
Advertiser	Bid	Sq. bid	Ad	Sp. limit	CPC	Imps	Clicks	Convs	Impression range	Avg pos
MetroClick	0.315	0.109	generic	50.93	0.310	426	164	16		1.000
QuakTAC	0.266	0.107	lioneer:dvd	-	0.194	718	156	6		1.593
TacTex	0.235	0.091	generic	0.236	0.201	77	1	0		3.000
UMTac09	0.216	0.078	generic	7.583	0.209	700	36	6		2.719
munsey	0.190	0.075	generic	-	0.174	718	16	2		3.675
epflagent	0.214	0.068	generic	-	0.184	641	3	0		4.510
AstonTAC	0.158	0.059	generic	500.0	0.133	292	1	0		4.938
Schlemazl	0.062	0.020	flat:dvd	5.617	-	0	0	0		-

Table 1: Results for the query *null:dvd* from one game day of the 2009 TAC/AA finals

is to be clicked. A number of factors can modify this default behavior. First, if an advertiser's ad is targeted, the click probability is raised or lowered depending on whether the ad matches the product desired by the user. Second, each advertiser has a component and manufacturer specialty. If the product desired by the user matches the component specialty, the conversion probability is increased, and if it matches the manufacturer specialty, the advertiser's revenue is increased. Finally, the conversion probability decreases if the advertiser has exceeded its capacity, as described below.

Auctions: Ads are ranked using a generalized second price auction. Rather than ranking ads solely by bids, the search engine also considers click probability. If for query type q an advertiser's bid is b_q and its default click probability is e_q^a, then its *squashed bid* is defined as $(e_q^a)^\chi b_q$, where χ is a random but known game parameter. Ads are ranked by squashed bid, and each time an advertiser's ad is clicked, it pays the minimum amount it could have bid while still beating the squashed bid of the advertiser ranked below it.

Capacity: Each advertiser is assigned a capacity c which serves as a soft constraint on how many products it can sell (of any type) over a five day period. Whenever an advertiser's ad is clicked, if the number of products n sold over five days (including those sold so far on the current day) exceeds c, then the conversion probability is multiplied by a *distribution constraint* equal to 0.995^{n-c}. Note that the distribution constraint changes during the day as the advertiser sells more products.

Information: Advertisers must operate in the face of limited information about customers and competitors. For each query type, the advertiser receives a daily report stating how many impressions, clicks, and conversions the advertiser received and the average cost per click (CPC). The only other information available is a report on the ad used by each advertiser and the average position of that ad. An advertiser that wishes to increase its number of clicks would therefore have little information about how much it would cost to increase the position of its ad or how many clicks it might expect in the new position. Advertisers are also unaware of the types (specialties and capacities) of other advertisers.

Example: Table 1 shows the results for the query *null:dvd* from a sample game day. The eight advertisers are shown in order of their squashed bids, which differs from the order of the true bids due to differing e_q^a values. The ads and spending limits (where used) of each agent are also shown. The results of these actions are shown on the right side of the table: the cost per click, impressions, clicks, and conversions. In addition, the *impression range* column shows a graphical representation of the period for which each adver-

tiser's ad was shown, with the day progressing from left to right. On this day, 718 users submitted the query *null:dvd*, but due to spending limits, only two agents, QuakTAC and munsey, received the full 718 impressions. TacTex was the first agent to hit its spending limit (after a single click - this was a probe, as described later). At that point, all lower advertisers increased by one position, and since epflagent reached the fifth position, its ad began to be shown. Hence, the impression range column shows epflagent starting where TacTex stopped. Although Schlemazl reaches the fifth position at the end of the day, its ad is not shown because its bid is below the reserve. Finally, the average position for each advertiser is shown. Note that the average positions are not in the same order as the squashed bids, and that the average is only for the period in which the ad was shown (thus never above 5). From this table, the only information available to TacTex was its own row (except for the squashed bid) and the ad and average position columns. Much of TacTex's computational effort is devoted to estimating the rest of this information so that its decisions can be based on as much information as possible.

3. TACTEX OVERVIEW

At a high level, TacTex operates by making predictions or estimates concerning various factors (such as unknown game parameters, user populations, and competitor bids) and then by finding the optimal actions given this information. These tasks are divided among a number of modules that we describe in detail in the following sections. Here, we give an overview of these modules. Figure 1 depicts the relationship between the modules, including the inputs and outputs of each.

At the start of each new day, the game server sends TacTex a report on the results of the previous day. The first module to be called is the Position Analyzer, a preprocessor that converts some of this information into a more useful format. The goal of the Position Analyzer is essentially to reconstruct the impression range column of Table 1 for each query type.

TacTex then performs all necessary prediction and estimation using three modules. The User Model uses the total number of queries for each query type to estimate the composition of each of the nine user populations. From these estimates, predictions about future user populations can be made. The Advertiser Model takes information relating to the actions of other advertisers and predicts the actions these advertisers will take in the future. The Parameter Model maintains estimates of unknown game parameters by finding those parameters that best fit the known auction outcomes.

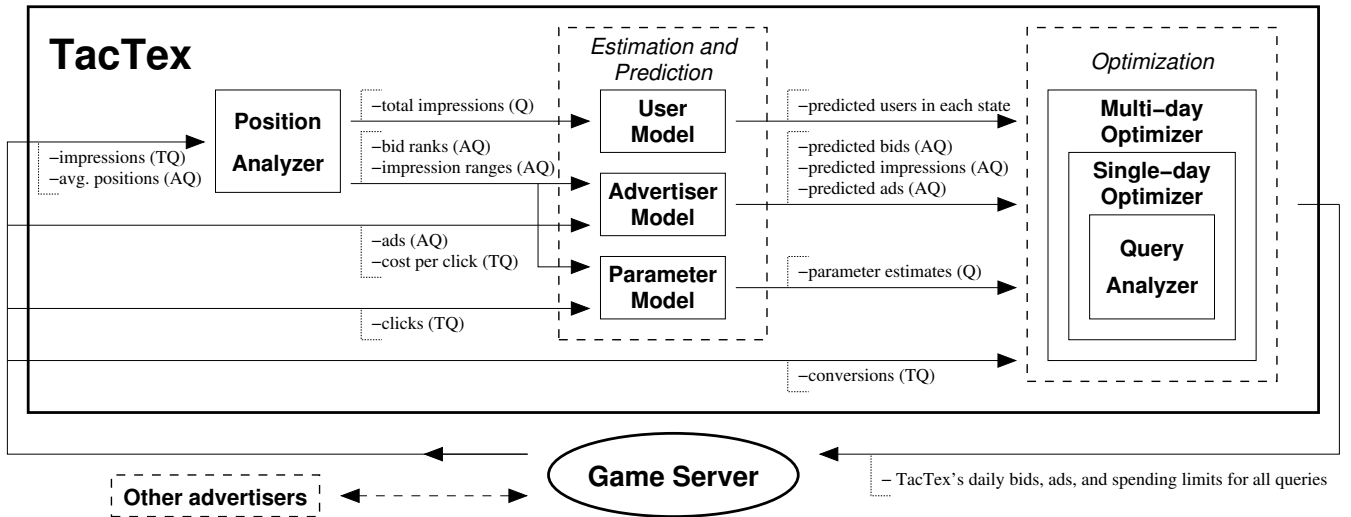Finally, TacTex must use these predictions and estimates

Figure 1: Flow of information in TacTex. T = TacTex only, A = all advertisers, Q = all queries.

to choose the optimal bids, ads, and spending limits to submit to the game server for the next day. The Query Analyzer uses the information received to compute the expected outcomes of actions, such as how many clicks and conversions would occur for a given query type. If there were no distribution constraint, then TacTex could optimize for each query type independently, but instead it must choose how to allocate its available capacity among query types and even among multiple days. The Multi-day Optimizer is responsible for dividing capacity among the remainder of the game days, and it calls the Single-day Optimizer to divide each day's capacity among query types using the information provided by the Query Analyzer.

4. POSITION ANALYZER

For each query type, the position analyzer takes as input i) the average position of each advertiser, ii) the number of impressions for TacTex, and iii) an upper bound on the total number of impressions. Using this information, the position analyzer attempts to determine i) the ranking of the squashed bids, and ii) the first and last impression for each advertiser – in other words, it attempts to reconstruct the impression range column from Table 1.

For advertiser a, the three values that we wish to determine are i) $rank_a$, the ranking of the advertiser's squashed bid (1-8); ii) $start_a$, the first impression for which the advertiser's ad was displayed; and iii) end_a, the impression after the last. As part of the process we will also need to determine $limitOrder_a$, the order in which the advertiser hit its spending limit with respect to other advertisers (again 1-8, with ties broken by rank). If $rank_a < 6$, $start_a$ will be 1; otherwise, it will be end_b, where b is the advertiser for which $limitOrder_b = rank_a - 5$ (the fact that b hits its spending limit is what causes a to rise into fifth position). If advertiser a never hits its spending limit, then end_a will be the total number of impressions plus one. Table 2 shows all of these values for the query shown in Table 1.

The game server computes the average position for advertiser a by taking the sum of positions over all of the advertiser's impressions (sum_a) and dividing by the number of

impressions:

$$averagePosition_a = \frac{sum_a}{end_a - start_a} \qquad (1)$$

Suppose that we already know each $rank$ value. Then we can express sum_a as follows:

$$sum_a = \sum_{b \,:\, rank_b \leq rank_a} max(0, min(end_b, end_a) - start_a) \qquad (2)$$

The contribution of each advertiser b to sum_a is the number of impressions for which its ad was displayed above the ad of advertiser a (advertiser a's ads are included because the ranking is 1-based). If we also know each $limitOrder$ value, then we can rewrite sum_a without the max and min, giving us a system of linear equations that we could try to solve. However, we have twice as many variables as equations ($start_a$ and end_a for each advertiser a), giving us a severely under-constrained system. In addition, we do not in fact know the $rank$ and $limitOrder$ values, and would need to try solving the system corresponding to each set of values.

An additional piece of information that we have not taken advantage of is the fact that despite being represented by a Java double, each average position is a fraction (as shown in Equation 1). We can therefore use the method of continued fractions to find the fractional representation for each average, $numerator_a/denominator_a$. These values are shown in Table 2. (Note that the average positions shown are truncated.) In some cases, the numerator and denominator of this fraction will equal the values shown in Equation 1, giving us exactly the information we need. For example, using average position for UMTac09, we obtain a fraction of 1903/700, and UMTac09 indeed had 700 impressions. However, if the fraction in Equation 1 is reducible, with the numerator and denominator having a GCD of gcd_a, then we need to multiply the resulting numerator and denominator by the unknown value gcd_a to obtain the needed information. For example, for QuakTAC we obtain the fraction 572/359 and must multiply by 2, and for TacTex we obtain the fraction 3/1 and must multiply by 77. So now instead of two variables for each advertiser a, we have only one, gcd_a.

Advertiser	Rank	Start	End	Impression range	LimitOrder	Avg pos	Fraction	GCD
MetroClick	1	1	427		2	1.000	1/1	426
QuakTAC	2	1	719		4	1.593	572/359	2
TacTex	3	1	78		1	3.000	3/1	77
UMTac09	4	1	701		3	2.719	1903/700	1
munsey	5	1	719		5	3.675	2639/718	1
epflagent	6	78	719		6	4.510	2891/641	1
AstonTAC	7	427	719		7	4.938	721/146	2
Schlemazl	8	-	-		-	-	-	-

Table 2: Values computed by the Position Analyzer for the data from Table 1

We could again try to solve for these variables by setting up a system of linear equations, but in this case the system is homogeneous, again resulting in a large space of possible solutions. As before, we also do not know the *rank* and *limitOrder* values.

Instead of solving a system of equations, we perform a depth-first search of the space of possible values of each *rank*, *limitOrder*, and *gcd* by cycling through nodes of corresponding types in the search tree. Although the search space is large, by utilizing a number of pruning rules we can usually search the tree quickly. Figure 2 shows a portion of the search tree for the data from Table 1. For clarity, we will say that the *level* of a node is the number of ancestor nodes of the same type plus 1 (rather than its actual depth), and the nodes of Figure 2 are labeled with these levels. Algorithms 1-4 summarize the search process. Note that in these algorithms, all values to be determined are treated as global variables, and any variable set at a search node must be unset when backtracking.

The root of the tree is the level 1 rank node. At a level i rank node, we choose which advertiser a_i has rank i. An advertiser with average position p cannot have a rank less than $\lceil p \rceil$. Thus in Figure 2, there is only one choice at both $rank(1)$ and $rank(2)$, while at $rank(3)$ there are two choices.

Search then proceeds from a level i rank node to a limitOrder node of the same level. At this node we choose the order in which the advertiser a_i chosen at the preceding *rank* node hits its spending limit with respect to all previously chosen advertisers, set its *limitOrder* accordingly, and increment any *limitOrder* values that have already been set and are as large or larger. Again, our choices are constrained by the advertiser's average position. In order to have average position p, an advertiser's *limitOrder* cannot be less than $i+1-\lfloor p \rfloor$. (The agent starts in ith place and must move up to at least position $\lfloor p \rfloor$.) In Figure 2, at $limitOrder(1)$ we assign MetroClick a *limitOrder* value of 1 (the only possible value at level 1), at $limitOrder(2)$ we assign QuakTAC a *limitOrder* of 2 (it cannot be 1 according to the constraint), and at $limitOrder(3)$ we can assign UMTac09 a *limitOrder* of 2 or 3.

A level i limitOrder node leads to a level i gcd node. Here we must choose the value of *gcd* for the advertiser a_i chosen at the preceding *rank* node. Since we know the *rank* and *limitOrder* values of all advertisers with higher rank, and we know that the numerator and denominator of the fraction in Equation 1 are equal to $gcd_{a_i} numerator_{a_i}$ and $gcd_{a_i} denominator_{a_i}$, respectively, we can derive the following from Equations 1 and 2:

$$gcd_{a_i}(numerator_{a_i} - denominator_{a_i}(i - limitOrder_{a_i} + 1)) =$$

$$\sum_{\substack{b\,:\,rank_b < rank_{a_i},\\ limitOrder_b < limitOrder_{a_i}}} max(0, end_b - start_{a_i}) \quad (3)$$

If we know the *end* values for all advertisers with higher rank and both sides of the above equation are nonzero, then it is straightforward to determine $start_{a_i}$, use the equation to find gcd_{a_i}, and then compute end_{a_i}. Unfortunately, both sides of the equation will be zero whenever $limitOrder_{a_i} = 1$.

As a result, we have three cases to consider. In the first case, $limitOrder_{a_i} = 1$, and we continue without determining a value for end_{a_i} or gcd_{a_i}. In the second case, the right hand side of Equation 3 is nonzero and contains no unknown end_b values. In this case, we can solve for gcd_{a_i}. In the third case, the right hand side of Equation 3 is nonzero but does contain unknown values. In this case, we attempt to guess the value of gcd_{a_i}, and then we determine the values of $start_{a_i}$ and end_{a_i} (possibly obtaining expressions including these unknown end_b values). We also plug these values into Equation 3 and store the resulting equation. As we proceed down the search tree, we will usually obtain enough equations to be able to solve for any unknowns. If at any point the system of equations is inconsistent, we backtrack. The possible values of gcd_{a_i} are constrained by the *end* values of the advertisers with the previous and next values of *limitOrder*, and by the upper bound on total impressions. Still, a wide range of values may be possible. Because we have a limited amount of time to search, and because lower values of gcd_{a_i} are most common, we implement a form of iterative broadening search. We repeat the search until we find a solution or hit a time limit of 300ms, and on our nth attempt, we consider only the first $5n$ possible values for gcd_{a_i} at each gcd node. In some cases, there will be no possible values for gcd_{a_i}, and we backtrack. Finally, we note that when advertiser a_i is our own agent, we know our own number of impressions and thus our *gcd*. Unless forced to backtrack, a level i gcd node leads to a level $i+1$ rank node.

Figure 2 shows examples of cases 1 ($gcd(1)$) and 3 ($gcd(2)$ and $gcd(3)$). At $gcd(1)$, we know $start_{a_1} = 1$ (because MetroClick is ranked in the top five), but we cannot determine gcd_{a_1} or end_{a_1}. At $gcd(2)$, we can also set $start_{a_2} = 1$. Because $limitOrder_{a_2} > limitOrder_{a_1}$, we know that $end_{a_2} \geq end_{a_1}$; however, we do not know end_{a_1}. As there are not yet any advertisers with a higher *limitOrder*, our only upper bound on end_{a_2} is the upper bound on the total number of impressions. Assuming this value is sufficiently high, we will have at least five values to consider for gcd_{a_2}. This is our first attempt at finding a solution, so we try only the first five (1 through 5). First, we try $gcd_{a_2} = 1$. Plugging known values into Equation 3 gives us $1 \cdot 213 = end_{a_1} - 1$, and so we can set $end_{a_1} = 214$. Note that this value is incor-
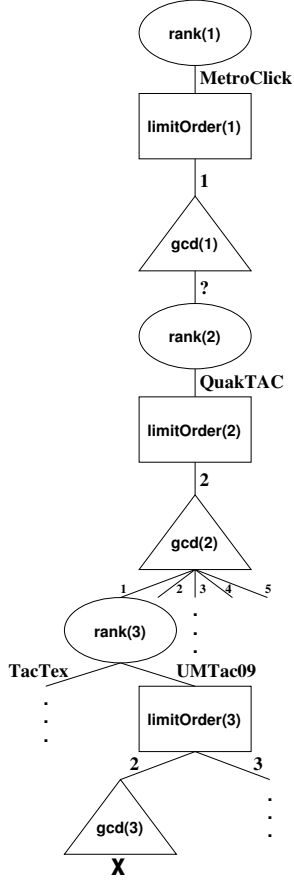
Figure 2: Part of the search tree for Table 1

Algorithm 1: POSITION-ANALYZER

begin
    input : all average positions, TexTex's impressions,
              upper bound on impressions
    output: ranking, impressions per advertiser, total
              impressions

1 $n \leftarrow 0$
2 $Equations \leftarrow \{\}$
3 $Solutions \leftarrow \{\}$
4   **while** $Solutions = \{\}$, $time < 300ms$ **do**
5 $n \leftarrow n + 1$
6   RANK-NODE(1)
7   Score solutions using heuristics and select the best

Algorithm 2: RANK-NODE

begin
    input: $level$
1   **if** $level > number\ of\ advertisers\ with\ a\ position$ **then**
2   valid solution found; add it to $Solutions$
3   **else**
4   determine set A of remaining advertisers that could
         have rank $level$
5   **for** $\forall adv \in A$ **do**
6 $a_{level} \leftarrow adv$
7 $rank_{a_{level}} \leftarrow level$
8   LIMIT-ORDER-NODE($level$)

Algorithm 3: LIMIT-ORDER-NODE

begin
    input: $level$
1   determine possible values of $limitOrder_{a_{level}}$
2   **for** $each\ value\ v$ **do**
3   **for** $1 \leq i \leq level - 1$ **do**
4   **if** $limitOrder_{a_i} \geq v$ **then**
5 $limitOrder_{a_i} \leftarrow limitOrder_{a_i} + 1$
6 $limitOrder_{a_{level}} \leftarrow v$
7   GCD-NODE($level$)

---

rect; it would be correct had we chosen the correct gcd_{a_2} of 2. We can also set $end_{a_2} = start_{a_2} + denominator_{a_2} gcd_{a_2} = 360$. At $gcd(3)$, we are forced to backtrack. Having chosen $limitOrder_{a_3}$ to be 2 (meaning $limitOrder_{a_2}$, for QuakTAC, has been incremented to 3) UMTac09 must hit its spending limit before QuakTAC, and so we must have $end_{a_3} < end_{a_2}$. But this implies $1 + 700gcd_{a_3} < 360$, an impossibility.

If we reach a rank node with all advertisers having been assigned a rank, then we have found a valid solution. If any variables remain unsolved, we set them to the median of their possible range. Because there may be multiple valid solutions, we record this solution and then backtrack to search for more. At the end of our search, if there is only one solution, we return it. In rare cases (around 1% of the time), we hit the 300ms time limit without finding any solutions. Often, multiple solutions are found, and we choose the best one by scoring each solution according to a number of heuristics. Solutions are favored if: i) the total number of impressions is low, ii) there are no unsolved variables, iii) multiple advertisers share the maximum *end* value, iv) no advertisers share any other *end* value, and v) only the advertiser with greatest *rank* has an average position of 5.0. Typically the solution chosen is correct or very nearly so.

5. USER MODEL

The User Model maintains estimates of the user popula-tion states by using a particle filter (specifically a Sampling Importance Resampling filter [1]) for each of the nine populations. A particle filter is a sequential Monte Carlo method that tracks the changing state of a system by using a set of weighted samples (called particles) to estimate a posterior density function over the possible states. The weight of each particle represents its relative probability, and particles and weights are revised each time an observation (conditioned on the current state) is received. In this case, each of the 1000 particles used per filter represents a distribution of the 10,000 users of that type among the six individual user states (NS, IS, F0, F1, F2, and T). At the beginning of the game, the particles are chosen to reflect the possible populations resulting from the initialization process performed by the game server. Each succeeding day, a new set of particles is generated from the old. For each new particle to be generated, an old particle is selected at random based on weight, and the new particle's user distribution is randomly generated from the old particle based on the user transition dynamics. These dynamics are known with the exception of the probability of a user transitioning to the Transacted state as a result of a conversion. This probability depends on the ads seen by the user, and thus on the behavior of the advertisers, but we can estimate it fairly accurately from

Algorithm 4: GCD-NODE

```
begin
    input: level
1   if rank_{a_level} < 6 then
2   │   start_{a_level} ← 1
3   else
4   │   find b such that limitOrder_b = rank_{a_level} − 5
5   │   start_{a_level} ← end_b        /* may include unknown */
6   if limitOrder_{a_level} = 1 then
7   │   leave gcd_{a_level} and end_{a_level} unset for now
8   │   RANK-NODE(level + 1)
9   else if RHS of Equation 3 has no unknowns then
10  │   solve for gcd_{a_level}
11  │   end_{a_level} ← start_{a_level} + denominator_{a_level} gcd_{a_level}
12  │   if end_{a_level} is feasible then
13  │   │   RANK-NODE(level + 1)
14  else
15  │   determine feasible values of gcd_{a_level}
16  │   for up to 5n values v do
17  │   │   gcd_{a_level} ← v
18  │   │   end_{a_level} ← start_{a_level} +
        │       denominator_{a_level} gcd_{a_level}
19  │   │   add Equation 3 to Equations
20  │   │   try to solve for unknowns in Equations
21  │   │   if end_{a_level} is feasible and Equations is
        │       consistent then
22  │   │   │   RANK-NODE(level + 1)
```

past games.

The new particles are then re-weighted based on Tac-Tex's observations. The daily observations that depend on the user populations are the total impressions, clicks, and conversions for each of the 16 queries. Some of these observations are more informative and straightforward to use than others. Observations for queries in which the manufacturer or component are not specified (F1 and F2 queries) are less informative because they represent the behavior of users from multiple populations. Conversion rates depend on the position and distribution constraint at the time of the click, and suffer from small sample sizes. The most informative observations are the total impressions for queries that specify both manufacturer and component (F2 queries), and we found that we were able to estimate user populations accurately using only these observations. Note that unless TacTex had its ad displayed for every impression for a query type (an uncommon case), the total impressions for that query type must be determined using the Position Analyzer.

The number of impressions for an F2 query is the number of users submitting that query, which is the number of F2 users plus those IS users that chose the F2 query. An IS user has a 1/3 probability of choosing the F2 query, so the probability of observing N total impressions when there are x_i IS users and x_{f2} F2 users is the probability that $N - x_{f2}$ of the x_i IS users choose the F2 query, which can be determined from the binomial distribution $B(x_i, 1/3)$ or (in our case) estimated using the normal approximation to this binomial distribution, $N(x_i/3, 2x_i/9)$. Each particle has its weight set to this probability, and weights are then normalized to sum to one.

The resulting set of particles represents our estimated probability distribution over the user population state on the previous day. To obtain the expected user population n days in the future, we update each particle $n + 1$ times according to the user transition dynamics and take the weighted average of the particles.

Figures 3 and 4 show the estimated and actual number of users in the IS, F0, F1, and F2 states for two different products in one game from the 2009 finals. The game server models users as exhibiting occasional bursts of interest in a product (i.e., moving from the NS to IS state in large numbers), and these bursts can be seen in the IS user plots. If these bursts can be detected, then the estimated number of IS users will be very accurate, and the estimates of users in other states will follow. In Figure 3, these bursts are detected perfectly, and accuracy for all four user states is extremely good. For the product depicted in Figure 4, TacTex's estimates of total daily impressions were poor on some days, and on these days the User Model could not be sure whether a burst occurred. As a result, the IS estimate shows a number of small "blips", where some particles reflected a burst and others did not, and one true burst was missed. Nevertheless, the User Model was able to recover on succeeding days, and the estimates for all four states remain good overall.

6. ADVERTISER MODEL

The Advertiser Model makes three types of predictions about the actions of the competing advertisers.

6.1 Impression predictions

In addition to predicting the bids of other advertisers, it is also important to predict whether and when they will hit their spending limits. If other advertisers set low spending limits, a relatively low bid could still result in a high average position and number of clicks. For each query type, the Advertiser Model predicts the maximum number of impressions that each advertiser could receive before hitting its spending limit. In general, the Advertiser Model predicts that this maximum will be the same number of impressions as the advertiser received on the previous day; however, in cases in which an advertiser did not hit its spending limit by the day's final impression, we assume that the advertiser effectively had no spending limit and will also not hit its spending limit on the coming day. Information about the impressions received by other advertisers is provided by the Position Analyzer.

6.2 Ad predictions

The Advertiser Model also predicts the ads (targeted or generic) that other advertisers will choose. For each query type, the Advertiser Model maintains a count for all the ads it has seen so far from each advertiser. The predicted ad for that query is then the majority ad, i.e., the ad having the highest count amongst all posted ads for that query type.

6.3 Advertiser bid estimation

The third task performed by the Advertiser Model is to maintain estimates of the bids submitted by each advertiser for each query and then to predict what future bids will be. Advertiser bid estimation is a hard problem because the bidding dynamics of other advertisers are unknown – while bids often change only gradually, it is not uncommon for large jumps to occur. In addition, the Advertiser Model receives only partial information about the bids of other advertisers (the bid ranks and TacTex's CPC). During the development
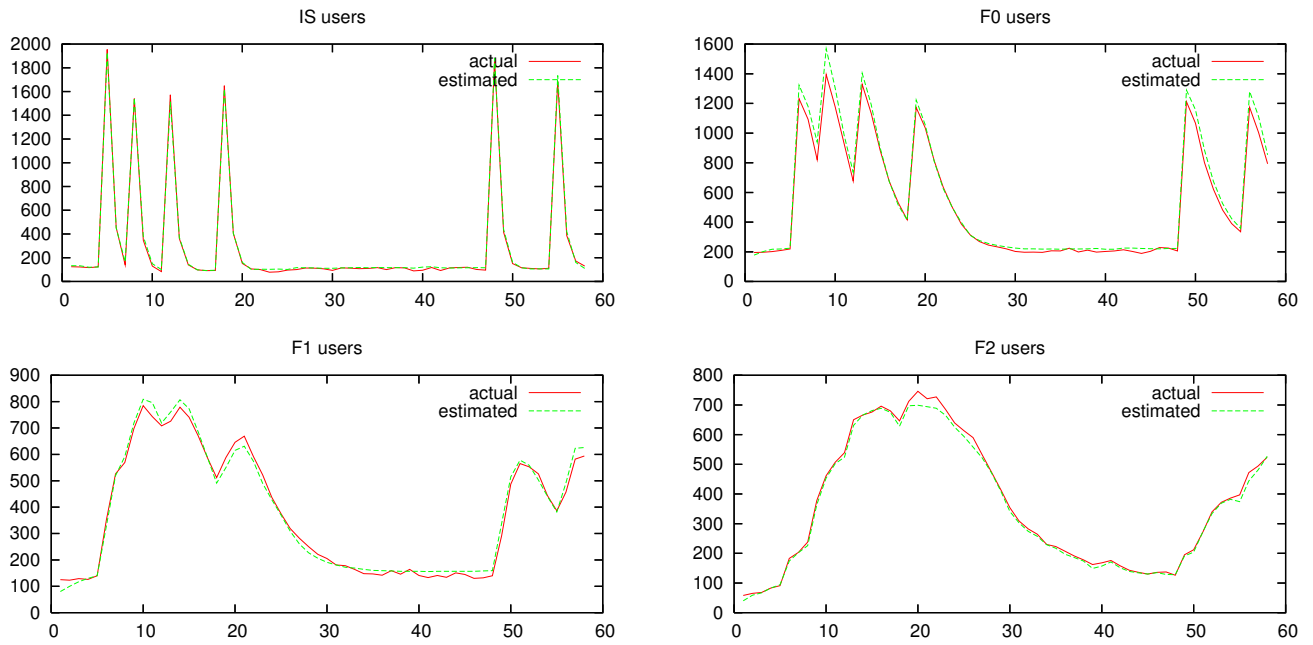
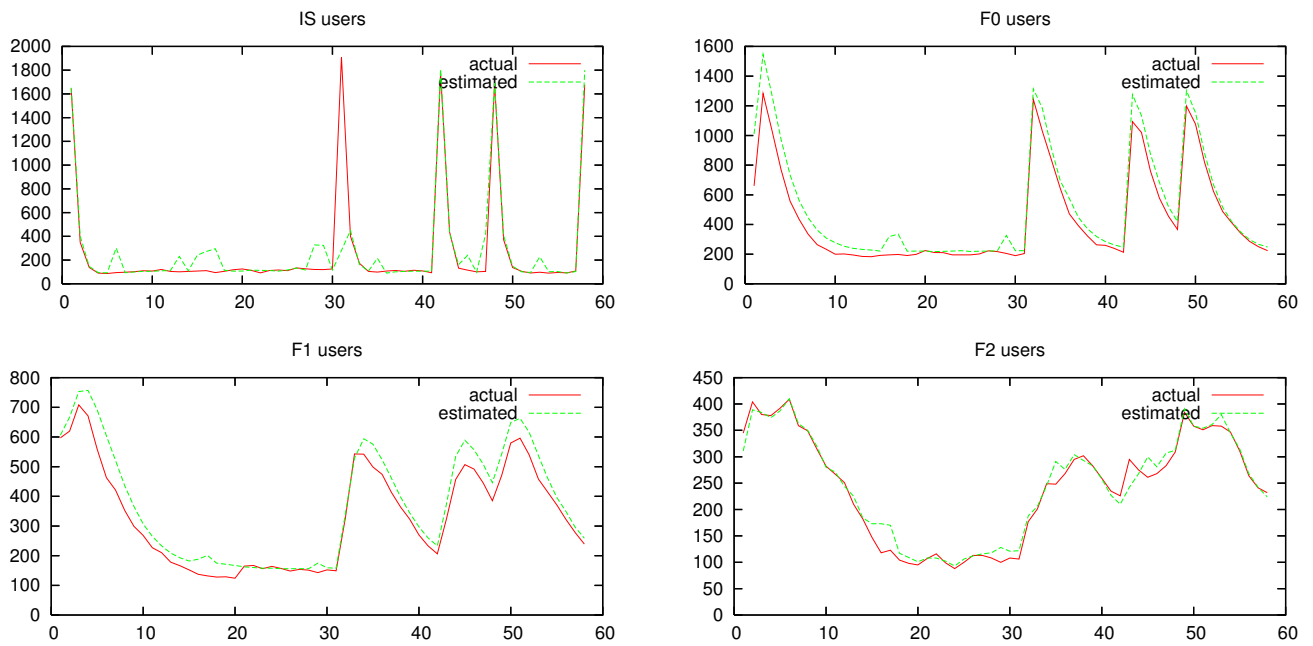**Figure 3: User population estimates for one product**

Figure 4: User population estimates for a second product

of TacTex, we created two very different and independently designed bid estimators. Our preliminary testing did not show either approach to be superior; however, we found that an ensemble approach that averaged the output of the two estimators outperformed either one alone, and so the Advertiser Model uses both estimators in this fashion. The primary difference between the two is that the first estimator models all advertisers' bids jointly, while the second models bids independently.

The estimates obtained are for the bids active on the previous game day; however, our goal is to predict future bids for use in planning. Currently, the Advertiser Model simply assumes that the most recently estimated bids will persist for the rest of the game, but improving these predictions is an important focus of future work.

First bid estimator : The first approach (Algorithm 5) uses particle filtering to estimate the bids of other advertisers. We use one particle filter for each of the 16 query types. Each of the 1000 particles per filter represents one set of bids for all other advertisers for that query type. Associated with the particles is a probability distribution that gives the likelihood of each particle representing the current state. On each new day, each particle filter samples from the underlying distribution to obtain the next set of particles. Then it updates each particle based on the observations received that day, i.e., the cost per click and bid rankings. Once the particles have been updated, the filter recomputes the probability distribution for the new set of particles.

The sampling step is straightforward. The next step in particle filtering is to update each particle using the known dynamics. In the absence of such known dynamics, we propose a departure from the traditional vanilla particle filter and use a part of the observation to do the update. Let cpc_{t+1} and r_{t+1} be the cost per click and ranking for that query seen on day $t+1$. We use these values to reset some of the bids made by other advertisers in some particles (where necessary) in an attempt to improve the respective particle. On each iteration of the update, the bid, b^x_{t+1}, of an advertiser x is adjusted while holding the bids of the other advertisers fixed. The bids are adjusted for only those cases where the order of a bid is incorrect with respect to the known bid ranking. Algorithm 6 gives an outline of how the bid adjustment is done. The two cases where the order is correct and there is no need for bid adjustment are:

$$r^x_{t+1} > (r^{TacTex}_{t+1} + 1) \quad \wedge \quad b^x_{t+1} < cpc_{t+1}$$
$$r^x_{t+1} < r^{TacTex}_{t+1} \quad \wedge \quad b^x_{t+1} > b^{TacTex}_{t+1} \qquad (4)$$

The conditions when b^x_{t+1} needs to be updated, and how these updates are made, are mentioned below. $rand(a, b)$ denotes a random draw from the range (a, b). z denotes the particle and $z(r)$ denotes the bid of the advertiser ranked r in z.

$$b^x_{t+1} = \begin{cases} cpc_{t+1} & \text{if } r^x_{t+1} = r^{TacTex}_{t+1}+1 \\ rand(0, \text{least bid value in } z) & \text{if } r^x_{t+1} = \text{undefined} \\ rand(z(r^x_{t+1}+1), z(r^x_{t+1}\text{-}1)) & \text{otherwise} \end{cases}$$

Note that the "otherwise" case excludes the conditions mentioned in 4. The whole process is repeated a fixed number of times, holding one advertiser fixed each time, with each iteration improving upon the former (20 iterations is sufficient). At the end of this update step, we have a better particle having closer predictions of other advertiser bids.

Next comes the step of recomputing the probability distri-

Algorithm 5: ADVERTISER-BID-ESTIMATOR

begin
  input : $\vec{z}, Pr(\cdot), cpc, ranks$
1 **for** $\forall z \in \vec{z}$ **do**
2    $z \leftarrow$ ADJUST-BIDS$(z, cpc, ranks)$
3 Sample n particles from the filter
4 **for** $\forall z \in \vec{z}$ **do**
5    $Pr(z) \leftarrow$ RECOMPUTE-DISTRIBUTION$(z, ranks)$
6 normalize $Pr(\cdot)$

Algorithm 6: ADJUST-BIDS

begin
  input : $z, cpc, ranks$
  output: z
1 LO-BID $\leftarrow$ set a low bid limit
2 $size \leftarrow$ no of advertisers in $ranks$
3 $r^{TacTex} \leftarrow$ own rank from $ranks$
4 $b^{TacTex} \leftarrow$ own bid for q
5 $count \leftarrow$ no of iterations
6 **while** $count$ $times$ **do**
7   **for** $(\forall x \in advertiser\ set)$ **do**
8      $r^x \leftarrow$ rank of x from $ranks$
9   **switch do**
10   **case** $r^x = r^{TacTex} + 1$
11          $b^x \leftarrow cpc$
12   **case** $r^x = -1$ *(undefined)*
13          $b^x \leftarrow rand(\text{LO-BID}, z(size))$
14   **case** $r^x > (r^{TacTex} + 1) \wedge b^x \geq cpc$
15          $b^x \leftarrow rand(z(r^x + 1), z(r^x - 1))$
16   **case** $r^x < r^{TacTex} \wedge b^x \leq b^{TacTex}$
17          $b^x \leftarrow rand(z(r^x + 1), z(r^x - 1))$

---

bution of the sampled particles (Algorithm 7). Although the true likelihood of a particle whose ranking does not match the true ranking r_t is zero, there may be few particles with the correct ranking, and so we instead use a likelihood function designed to give some weight to all particles. We compute the difference of ranking for each advertiser from the two available sources, i.e., r_t and the rank from z. For a distance δ, we define $\kappa(\delta) = exp(-\frac{\delta^2}{4.9})$. The likelihood of each particle is set to the product of these $\kappa(\delta)$ values over all advertisers, and thus the particles whose predicted rankings are closer to r_t get assigned higher values. These values are normalized over all 1000 particles to give the true probability distribution captured by the particles.

Second bid estimator: The second bid estimator differs from the first in two main ways. First, for each query it maintains separate bid distributions for each advertiser, rather than a single joint distribution over all bids. Second, while both estimators approximate continuous distributions using discrete distributions, the second filter does so using a fixed set of bids rather than a changing set of particles. The bid space $[0, 3.75]$ is discretized into values v_1 through v_{100} by setting $v_i = 2^{i/25-2} - 0.25$ (thus $v_{50} = 0.75$). Discretizing the bid space in this way allows better coverage of low bids, which are most common, while still maintaining the ability to represent very high bids, and it also simplifies our modeling of changes in bids, described below.

On day $t + 1$, for each advertiser x, we wish to estimate the distribution of the new bid, b^x_{t+1}, over these discrete v

Algorithm 7: RECOMPUTE-DISTRIBUTION

```
begin
    input  : z, ranks
    output: Pr(·)
1       Pr(z) ← 1
2       for ∀a ∈ advertiser set do
3           δ ← difference of a's rank between z and ranks
4           δ_f ← diff/7
5           Pr(z) ← Pr(z) × exp(−δ_f²/C)
6       normalize Pr(·)
```

Algorithm 8: UPDATE-DISTRIBUTIONS

```
begin
    input  : rankings r_{t+1}, TacTex's bid and cpc
1   for each advertiser x do
                    /* let T_i^x be Pr(b_{t+1}^x = v_i|b_1^x...b_t^x) */
2       for i = 1 ... 100 do
3           set T_i^x according to Equation 7
4   for n = 1 ... 10 do
5       for each advertiser x do
6           for i = 1 ... 100 do
                /* let O_i^x be Pr(r_{t+1}|B_{t+1}^{-x}, b_{t+1} = v_i) */
7               if n = 1 then
8                   O_i^x = 1
9               else
10                  set O_i^x according to Equation 8
11      for each advertiser x do
12          for i = 1 ... 100 do
13              Pr(b_{t+1}^x = v_i) ← O_i^x T_i^x
14      normalize Pr(·)
```

values, conditional on the observed ranking r_{t+1}, previous bids b_1^x ... b_t^x, and the bids of other advertisers, B_{t+1}^{-x}. We make the simplifying assumptions that r_{t+1} and b_1^x ... b_t^x are conditionally independent given b_{t+1}^x, and that B_{t+1}^{-x} and b_1^x ... b_{t+1}^x are independent. Applying Bayes' rule twice and rearranging, we derive:

$$Pr(b_{t+1}^x = v_i|r_{t+1}, B_{t+1}^{-x}, b_1^x...b_t^x) \propto$$
$$Pr(r_{t+1}|B_{t+1}^{-x}, b_{t+1} = v_i)Pr(b_{t+1}^x = v_i|b_1^x...b_t^x) \quad (5)$$

The first term in the R.H.S. of Equation 5 is the probability of the observation while the second term is the transition model of bids for x, both unknown.

We model bid transitions by assuming that bids change in one of three ways. First, with 0.1 probability, b_{t+1}^x jumps uniformly randomly to one of the v_i values. This case covers sudden jumps that are difficult to model. Next, with 0.5 probability, b_{t+1}^x changes only slightly from b_t^x. We assume that the probability of changing from v_i to v_j is proportional to $\phi_{0,6}(|i - j|)$, where $\phi_{0,6}$ is the density function of the zero-mean normal distribution with variance 6. Because we discretized the bid space in such a way that bids increase exponentially, the use of this distribution reflects the assumption that the logarithms of the ratios of successive bids are distributed normally with zero mean. Finally, we assume that with 0.4 probability, the bid changes according to a similar distribution, but the change is with respect to the bid 5 days ago, b_{t-4}^x. This case captures the fact that bids often follow 5 day cycles due to the 5 day capacity window. The probabilities for these three cases were chosen to provide robustness to a variety of agent behaviors in pre-competition experiments. Let $tr(j,i)$ denote the resulting probability of the bid transitioning to v_i from v_j using the above normal distribution and normalizing.

Summing the three cases gives us the following:

$$P(b_{t+1}^x = v_i|b_t^x = v_j, b_{t-4}^x = v_k) = \quad (6)$$

$$0.1 \cdot 0.01 + 0.5 \, tr(j,i) + 0.4 \, tr(k,i)$$

and so we can find the probability of each bid by summing over our previous distribution estimates:

$$P(b_{t+1}^x = v_i) = 0.1 \cdot 0.01 + 0.5 \sum_{j=1}^{100} P(b_t^x = v_j)tr(j,i) + \quad (7)$$

$$0.4 \sum_{k=1}^{100} P(b_{t-4}^x = v_k)tr(k,i)$$

Our estimate for b_1^x is initialized to a distribution consistent with observed game data, and when $t < 5$, we substitute b_1^x for b_{t-4}^x.

The observation probabilities are now conditioned on a single advertiser's bid, rather than a set of bids as in the first bid filter. Let y be another advertiser in the game apart from x. If advertiser y is TacTex, then we know the bid; otherwise we have a distribution representing our estimate of the bid for y. Thus the conditional probability of the set of rankings r_{t+1} given a fixed $b_{t+1}^x = v_i$ and a fixed value of the distribution B_{t+1}^{-x} is :

$$P = \Pi_{\forall y \neq x} \begin{cases} Pr(b_{t+1}^y > v_i) & \text{if } r_{t+1}^x > r_{t+1}^y, \\ Pr(b_{t+1}^y < v_i) & \text{if } r_{t+1}^x < r_{t+1}^y, \\ 1 & \text{otherwise} \end{cases} \quad (8)$$

where P denotes $Pr(r_{t+1}|b_{t+1}^x = v_i, B_{t+1}^{-x})$. Note that ranks will only be equal if neither advertiser had any impressions; in this case we have no information about the relative bids. Also, whenever y is TacTex, the R.H.S. will be 1 or 0 since we know our own bid. In addition, in some cases it is possible to use TacTex's cost per click to exactly determine the bid of the advertiser below it. Finally, we have been treating B_{t+1}^{-x} as if it were known, but in fact these are the other advertisers' bids that we are trying to estimate simultaneously. We address this problem by applying Equation 5 for 10 iterations, using the latest estimates for each bid distribution, as this resulted in sufficient convergence in testing. Algorithm 8 summarizes the update procedure used each day by the second bid estimator.

7. PARAMETER MODEL

Recall that for each query type q, the parameter γ_q represents the probability that a user will progress from one ad to the next, while each advertiser a has a parameter e_q^a that affects the probability of a user clicking its ad (and thus also its squashed bid). Given the bid rankings and impression ranges computed by the Position Analyzer and the User Model's population estimate, we can determine the distribution over the number of clicks that TacTex would receive for any set of these parameter values. The Parameter Model estimates the values of γ_q and e_q^{TacTex}, as these are the parameters that have the most impact on TacTex and about which our observations provide the most information.

There is insufficient information to effectively estimate e_q^a values of other advertisers, and so we assume they equal the mean possible value. To perform estimation, the Parameter Model maintains a joint distribution over (γ_q, e_q^{TacTex}) pairs by discretizing the possible space of values uniformly and setting the likelihood of each pair to be proportional to the probability of all observations given that pair, that is, the product of the probabilities of each day's number of clicks. When performing calculations involving these parameters, the Parameter Model takes the weighted average for each parameter as its estimate. While these estimates do not necessarily converge to the correct values by the end of a game (and might not ever be expected to given our decision to not estimate other advertisers' e_q^a values), they are much more accurate on average than simply assuming the expected parameter values.

8. OPTIMIZATION

To this point, we have described those modules that estimate the game state and make predictions about the future. We now turn to the challenge of using this information to select actions. In particular, each day TacTex must choose bids, ads, and spending limits for each query. The key factor in the optimization process is the distribution constraint. Recall that while there is no hard cap on capacity, exceeding a certain number of conversions results in a reduced conversion rate. Beyond some point, marginal returns per conversion can become negative. As a result, TacTex performs optimization by reasoning about conversions and then choosing actions expected to result in those conversions, rather than reasoning directly in the space of possible actions.

The optimization process consists of three levels: a Multi-day Optimizer (MDO), a Single-day Optimizer (SDO), and a Query Analyzer (QA). Because we want to maximize profit for the entire game, not a single game day, the top-level decision that must be made is how many conversions to target on each remaining game day, and this decision is made by the MDO. Computing the expected profit for a given day and conversion target requires deciding how to divide the conversions among the 16 query types, and the MDO calls the SDO to perform this task. Finally, the SDO calls the QA to i) determine the bid, ad, and spending limit that are expected to result in a given number of conversions, and ii) compute the expected cost and revenue from those conversions. We describe these three levels from the bottom up.

8.1 Query Analyzer

For any given bid, ad, and spending limit, it is fairly straightforward to determine the expected cost, revenue, and conversions from a specific query type. The QA does this by taking the expected user population, iterating through all impressions, computing our position and CPC, and then computing the probability that the user i) reaches our ad, ii) clicks on it, and iii) converts. (Note that at this level we are not considering the distribution constraint, which may lower the conversion rate.) However, the problem we face is essentially the reverse: the QA is given a conversion target and needs to determine the bid, ad, and spending limit that will produce those conversions in the most profitable way. Up to a certain point, raising either the bid or the spending limit will increase the number of conversions, while the effect of ad choice depends on the user population, so there may be a number of ways to reach a given number of conversions.

We simplify matters by using no spending limits. During the course of a day, the expected profit per impression can only increase as other agents hit their spending limits. If an agent above us hits its spending limit, our position improves along with our conversion rate. (Higher positions have higher conversion rates due to a higher ratio of F users to IS users – IS users never convert and thus are more likely to reach ads at lower positions.) If the agent below us hits its limit, then our CPC is reduced, as we are participating in a generalized second price auction. It is therefore usually preferable to control conversions using the bid rather than the spending limit.[1]

For any given bid, we can evaluate each of the relevant ads and pick the one that gives the highest profit per conversion. The spending limit can then be set based on the expected cost. As a result, the query-level predictor's primary task is to determine the bid that will result in the desired number of conversions. There is one difficulty remaining, however: because our prediction for each advertiser's bid is a point estimate, any bid between the nth and $n+1$st predicted bids will result in the same position, $n+1$, and the function mapping bids to conversions will be a step function. In reality, there is uncertainty about the bids of other advertisers, and we would expect this function to be continuous and monotonically increasing. We create such a function by linearly interpolating between the expected results for each position. In particular, we assume that the number of conversions expected for the nth position will result from bidding the average of the n-1st and nth bid. For $n = 1$, we use a bid 10% above the predicted highest bid, and for $n = 8$, we use a bid 10% below the predicted lowest bid. We generate functions for cost and revenue in the same way.

The complete procedure followed by the query-level predictor is therefore as follows. First, we find the eight bids (along with corresponding optimal ads) that correspond to the eight possible positions, and determine the expected conversions, cost, and revenue for each. Next, we use linear interpolation to create functions mapping bids to conversions, cost, and revenue. Finally, for a conversion target c, we can find the bid resulting in the target from the conversions function and determine the resulting cost ($cost_q(c)$) and revenue ($revenue_q(c)$) from the corresponding functions. The ad to use is the ad corresponding to the closest of the eight bids.

8.2 Single-day Optimizer

Using this information about each query type, the SDO can now determine the optimal number of conversions to target for each query type given a total daily conversion target c and the initial capacity used u. The initial capacity used (the sum from the past four days) is important because it, along with the total conversion target, determines the distribution constraint, which can in turn have a large impact on the profit from each conversion and the optimal solution. To illustrate, suppose we need to choose between targeting a single conversion from query type A with an expected revenue of 10 and expected cost of 8.5, and a single conversion from query type B with an expected revenue of 15 and expected cost of 13.3. With a distribution constraint of 1 for this conversion, the profits would be 1.5 and 1.7, respec-

---

[1]During the 2009 TAC/AA competition, TacTex used high spending limits as a precaution, but our experiments have shown that this was unnecessary, and so we omit them from the agent described here.

tively. With a distribution constraint of 0.9, however, the profits would be 0.5 and 0.2, changing the optimal choice from B to A. It is also important to note that although we are targeting one conversion, with a distribution constraint of 0.9 we would actually only expect 0.9 conversions; we will address this issue below.

Computing the precise impact of the distribution constraint is difficult because it decreases after each conversion, meaning that we would need to know when a conversion occurred to compute its profit. We solve this problem by making the simplifying assumption that the day's average distribution constraint applies to each conversion. We denote this value $\bar{d}(u,c)$ because it can be computed from the initial capacity used and the total conversion target; in fact, we precompute all possible $\bar{d}(u,c)$ values before the game begins. The goal of the SDO is thus to find values of c_q maximizing $\sum_q [\bar{d}(u,c)revenue_q(c_q) - cost_q(c_q)]$, where the c_q values correspond to the query types and sum to c. Again, although we are targeting c conversions, we would actually only expect $\bar{d}(u,c)c$ conversions. In general we reason in terms of conversions *before* adjusting for the distribution constraint, and so for clarity we will use the term *adjusted* conversions when referring to the actual number of resulting conversions. Note that u is expressed in terms of adjusted conversions, while c is not.

We are now left with a fairly straightforward optimization problem: allocating the total conversion target among the queries so as to maximize profit. This problem can be solved optimally using dynamic programming by casting it as a multiple choice knapsack problem, with each (*query type, conversion*) pair representing a single item and each query type representing a class from which only one item can be chosen. This solution is too slow for our needs, unfortunately, and so instead the SDO uses a nearly-optimal greedy solution in which we repeatedly add conversions from the most profitable query type. If it were always the case for each query type that as the number of conversions increased, the marginal profit per query type decreased, then we could add the most profitable conversion at each step and be guaranteed the optimal solution. In order to receive more conversions we must increase our bid, and thus our cost per click increases. However, as our position improves, is it possible for our conversion rate to improve enough to offset this cost, and so the marginal profit per conversion may actually increase. As a result, instead of adding a single conversion at each step of our greedy approach, we consider adding multiple conversions. For each query type, we determine the number of additional conversions (bounded above by the number of conversions remaining before we hit our target) that maximizes the average profit per additional conversion, and we then add the conversions from the query type with the highest average profit. This greedy approach is not guaranteed to be optimal, but tests show that the resulting expected profit differs from the results of the optimal dynamic programming approach by less than 0.1% on average. The greedy optimizer is summarized in Algorithm 9.

8.3 Multi-day Optimizer

The SDO determines bids for any given conversion target and amount of capacity already used. Determining the bids to submit for the next day therefore requires only that we choose the conversion target. Because the bids submitted today affect not only tomorrow's profit but also the capac-

---

Algorithm 9: SINGLE-DAY-OPTIMIZER

begin
    input : capacity used u, capacity target c
    output: profit
1    $cSum \leftarrow 0$
2   **for** *each query type q* **do**
3   obtain $cost_q()$ and $revenue_q()$ from the QA
4        $c_q \leftarrow 0$
5   **while** $cSum < c$ **do**
6   Find q and c_q' maximizing
$$\frac{\bar{d}(u,c)(revenue_q(c_q') - revenue_q(c_q)) - (cost_q(c_q') - cost_q(c_q))}{c_q' - c_q}$$
        such that $c_q' > c_q$ and $cSum + c_q' - c_q \leq c$
7        $c_q \leftarrow c_q'$
8        $cSum \leftarrow cSum + c_q' - c_q$
9    $profit \leftarrow \sum_q \bar{d}(u,c)revenue_q(c_q) - cost_q(c_q)$

---

ity remaining on future days, we cannot simply choose the conversion target myopically. In order to maximize expected profit over the remainder of the game, we must consider not only tomorrow's conversion target, but also the actions we will take on all succeeding days.

The MDO operates by finding the optimal set of conversion targets for the remainder of the game. The expected profit from any set of conversion targets can be determined by successively applying the SDO to each remaining game day. The goal of the MDO on day d is is therefore to find the conversion targets c_t maximizing $\sum_{t=d+1}^{59} SDO_t(c_t, u_t)$, where SDO_t returns the expected profit from applying the SDO on day t, and u_t represents the total adjusted conversions over four days preceding t (which can be computed from the c_t values). Note that planning for the entire game requires calling the QA (and thus predicting the bids of other agents) for all remaining game days, not only the next day.

Once again, an optimal solution can be found using dynamic programming, but we choose another approach due to time constraints. The dynamic programming approach requires working backward from the last day and finding the optimal conversion target given the number of (adjusted) conversions on each of the previous four days. The MDO instead uses a form of hill climbing search to solve this optimization problem. We begin by setting each c_t value to be one-fifth of TacTex's capacity. Then for all t, we consider increasing or decreasing c_t by one and compute the expected profit in each case. We then choose the most profitable deviation over all t. This process repeats until no deviation is profitable. In comparing these two optimization approaches, we found it necessary to use a somewhat coarse degree of granularity (increments of five conversions) when implementing the dynamic programming approach due to memory limitations, and as a result the hill-climbing approach was actually slightly better than the dynamic programming approach. The optimization procedure followed by the MDO is summarized in Algorithms 10 and 11.

Once the optimal set of conversion targets is found, the MDO takes tomorrow's conversion target and submits the bids determined by the SDO. Essentially, we plan for the rest of the game and take the first step of this plan. On the next day, we repeat this process using updated information.

There is one remaining special case. It will often be the case that we are not interested in bidding on a particular

Algorithm 10: FIND-PROFIT

begin

    input : day d, conversion targets $c_d \ldots c_{end}$,
                  capacities used $u_{d-1}, u_{d-2}, u_{d-3}, u_{d-4}$

    output: profit

1    $profit \leftarrow 0$

2   **for** $t = d \ldots end$ **do**

3        $u \leftarrow u_{t-1} + u_{t-2} + u_{t-3} + u_{t-4}$

4        $profit \leftarrow profit + \text{SINGLE-DAY-OPTIMIZER}(u, c_t)$

5        $u_t \leftarrow \bar{d}(u, c_t)$

Algorithm 11: MULTI-DAY-OPTIMIZER

begin

    input : day d, capacities used $u_{d-1}, u_{d-2}, u_{d-3}, u_{d-4}$

    output: conversion target c_d

1   **for** $t = d \ldots end$ **do**

2        $c_t \leftarrow \frac{total\ capacity}{5}$

3   **while** *maximum profit improves* **do**

4   **for** $t = d \ldots end, change \in \{1, -1\}$ **do**

5            $c_t \leftarrow c_t + change$

6            $profit_t^{change} \leftarrow$
                $\text{FIND-PROFIT}(d, c_d \ldots c_{end}, u_{d-1} \ldots u_{d-4})$

7            $c_t \leftarrow c_t - change$

8   find t and *change* maximizing $profit_t^{change}$

9        $c_t \leftarrow c_t + change$

---

query. When this happens, TacTex submits a probe bid designed to provide information about the bids of other advertisers. The bid chosen is one that we expect to be the nth ranked bid, where n is the rank between 2 and 6 that we have hit least recently. We set a spending limit equal to the bid so that we will likely only receive a single click.

8.4 First two days

On the first two game days, we have not yet received any information about auction results, and so we cannot use the bidding strategy described above. Many agents use hard-coded bids, but we choose our bids using information from past games. For each query type, we choose an average position to target using a simple heuristic: the target begins at 5, and if the query type matches one of our specializations, the target decreases by an amount depending on our capacity. We compute the bid that we expect to result in this position by performing linear regression on data observed in previous games to learn a function mapping bids to positions. A separate function is learned for each day and focus level.

9. ACKNOWLEDGEMENTS

10. REFERENCES

[1] S. Arulampalam, S. Maskell, N. Gordon, and T. Clapp. A tutorial on particle filters for on-line non-linear / non-gaussian bayesian tracking. *IEEE Transactions on Signal Processing*, 50(2):174–188, Feb. 2002.

[2] P. Jordan, B. Cassell, L. Callender, and M. Wellman. The Ad Auctions Game for the 2009 Trading Agent Competition. Technical report, 2009.

[3] P. Jordan and M. Wellman. Designing an ad auctions game for the Trading Agent Competition. In *IJCAI-09 Workshop on Trading Agent Design and Analysis*, July 2009.

[4] S. Lahaie, D. Pennock, A. Saberi, and R. Vohra. Sponsored search auctions. In N. Nisan, T. Roughgarden, E. Tardos, and V. Vazirani, editors, *Algorithmic Game Theory*. Cambridge University Press, 2007.

[5] D. Liu, J. Chen, and A. Whinston. Current issues in keyword auctions. In G. Adomavicius and A. Gupta, editors, *Handbooks in Information Systems: Business Computing*. Emerald, 2009.

[6] D. Pardoe, D. Chakraborty, and P. Stone. TacTex09: A champion bidding agent for ad auctions. In *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2010)*, May 2010.