Proceedings of the Third International Joint Conference on Autonomous Agents and Multi-Agent Systems, Vol. 1, pp. 228-235, June 2004.

A Trading Agent and Simulator for Keyword Auctions

Brendan Kitts
iProspect
5 Water Street, Arlington, MA. USA
bkitts@excite.com

Benjamin J. LeBlanc
iProspect
5 Water Street, Arlington, MA. USA
bleblanc@iprospect.com

Abstract

We describe the lessons learned in deploying an intelligent trading agent for electronic Pay-Per-Click keyword auctions. Our discussion focuses upon the technical challenges in developing the bid management framework – an extensive internet communications infrastructure and database used to query market quotes, store data, and submit bids. In addition to the production system, we have also built an auction simulator for demonstration and testing purposes. The simulator allows the specification of a population of competitor bidding agents as tiny C# programs, which are dynamically compiled at run-time and run in the simulation. Through this simulator we present scenarios which demonstrate both the richness of our trading agent, and also some interesting scenarios in electronic auctions.

1. Introduction

A Pay per click (PPC) search auction is an auction for sponsored positions in search engines. For instance, if a user types in a search for "masters degree" at Google, they will get back a set of listings. These include sponsored sites which have paid on a PPC auction to have their companies shown.

PPC auctions run every minute of the day for every possible character sequence. In each auction, a competitor p enters a bid $b_k^{(p)}$ which is the amount they are willing to pay should a customer click on their advertisement in the search results for keyword k. The auctioneer (eg. Google) sorts the bids for keyword/auction k and awards position 1 to the highest bid, position 2 to the second highest bid, and so on. The participant will then pay an amount equal to the number of customers who visit their web-site multiplied by their bid price.

This paper will discuss the lessons learned in developing a successful trading agent for PPC auctions. The agent is fully deployed and managing funds for nearly four thousand auctions. We relate experiences that may be relevant to other trading agent/auction projects. We also illustrate our system using a simulator which we have developed for demonstration and testing. Using the simulator we construct scenarios which show the performance of different trading strategies.

2. Agent

The iProspect bidding agent is responsible for choosing prices on a portfolio of PPC auctions, at every hour during the day, such that an objective is maximized, its budget is not exceeded, and other user-defined price and cost constraints are achieved.

The agent can optimize profit, revenue, traffic or acquisitions. The agent maximizes its objective by predicting market demand and price over T future hours, and for every conceivable position on each auction using parameterized functions. After doing this, the agent finds a vector of bids that optimize its objective. Bids for the next T hours are computed, but only the bids for the current hour are submitted to the auction. The extra work of budgeting for the next T hours forces the agent to plan ahead. The optimization problem is solved using steepest ascent.

The optimization algorithm is supplemented by a variety of user definable rules. Many of the rules function as mathematical constraints for the optimizer. For instance, a bid-max rule forms an upper bound on the price of a keyword. The budget forms another constraint. A more complete description of the bidding algorithm can be found in [5].

The system presently manages 3,771 auctions and a considerable amount of money. Bids may be changed as fast as once per hour. The system automatically adjusts its bidding for budget under-shoots or over-shoots, time-of-day, competitor activity, and so on. The system also automatically explores the auction using an interval estimation-like procedure [4]. The system reports on its performance via a webpage with various monitoring graphs.

3. Bid Management Framework

In order to bid effectively, the agent needs to (a) read the auction market state, (b) record conversion events, (c) record data in a common database schema,

(d) execute the agent and (e) submit new bids to the auction. All tasks except for (d) fall under the rubric of the Bid Management Framework (BMF). The technical challenges for the BMF were just as numerous as those of the agent, and many of the lessons learned should translate readily to other trading agent systems.

The BMF operates on a Microsoft Windows platform. SQLServer was used as the database. All framework components were written in C# and rely on the functionality supplied by Microsoft's .NET Framework. The .NET framework provided extensive libraries for XML Serialization, Threading, and Remoting in addition to a level of standardization and comprehensiveness not found in Open Source solutions. The MatLab programming language (Mathworks) is used for the agent optimization routines. These implementation decisions greatly eased the development of each component.

The BMF can be divided into four parts: (1) Database, (2) Communications layer, (3) Task scheduler and (4) Conversion tracking system. We describe each part in the following section.

3.1 Database

We maintain separate database schemas for each auction in order to collect auction-specific data. For instance, Google provides a bid-position estimation utility that other auctions don't provide. We also maintain a single standard schema that provides a standardized view of all of the auctions. This standard schema allows components in the BMF and agent to simplify their logic.

Database access is achieved through a series of data abstraction layers. The first layer defines a series of stored procedures found within the database model. The second layer of abstraction comprises a series of Data Access Classes (DAC) modeled on the Data Transfer Object standard of [6]. DAC classes are built on top of Microsoft's ADO.NET data access library.

3.2 Auction Communications

The BMF communicates with each auction through an auction-specific Auction Service Gateway (ASG). Each ASG is designed to encapsulate all communication logic required by the auction and to provide a simple mechanism for submitting requests to the auction for all dependant components. Three auctions are presently supported:

Overture: The ASG communicates with Overture's DirecTraffic Center® (DTC) via an XML communication protocol [2]. This protocol defines a set of command and response messages capable of handling multiple requests at a time. Each set of XML messages are passed back and forth over a HTTPS connection. Overture's DTC interface also defines a set of restrictions and access fees based on command count and frequency. The scheduler keeps track of its access count and frequency to comply with these restrictions.

Google: The ASG communicates with Google Adwords™ using the HTTP protocol [3]. To accommodate this type of interface, a set of classes were developed which encapsulated the navigation and parsing logic needed to submit requests for reports from Google. Google has an access restriction of 3 to 6 seconds per request, and the ASG complies with these restrictions.

Simulated: The simulator is an external program which runs independently and inserts market data directly into the database allowing the agent to function as if it were participating in a live auction.

3.3 Task Scheduler

The task scheduler is responsible for scheduling the execution of both agent launch and database synchronization tasks. Scheduling information is stored in the database and monitored every few minutes. Once it is determined that a task is ready for execution, the task is marked as executing and a work request is placed in the appropriate work queue.

Agent Launch Service: The agent launch service is responsible for executing MatLab script by means of a command line shell. Communication with this service is accomplished through a closely monitored work queue.

Database Synchronization Service: The database synchronization service (DSS) is responsible for synchronizing a BMF database and its remote auction counterpart. The DSS implements two tasks: capture of market-state and bid updates. Market state capture involves gathering all data available about a keyword including clicks, impressions, and time of quote. A bid update involves submitting new bids for keywords. Communication with this service is accomplished through a work queue.

3.4 Conversion Tracking

The Conversion Tracking System provides a means for capturing conversion events in real-time. The Conversion Tracking System records two types of events. When the customer first clicks on the PPC listing, the customer is re-directed to a Conversion Tracking Server which records the click, time, and PPC listing, creates a cookie to identify the customer session, and then sends the customer on to their destination. Secondly, on conversion pages a special Image tag sends a request to our Conversion tracking server along with a query parameter that specifies the revenue value of the conversion that has just transpired. When our server receives this request it knows that a conversion has occurred. Every fifteen minutes, Conversion Tracking System data is downloaded into our database, and from there merged with other information that we have recorded about the auction. This allows the agent to witness changing market conditions and respond to consumer behavior in near real-time.

4. Deployment Lessons

We learned a number of lessons that may be generalizable to other trading agent implementations.

4.1 Rules

One of our earliest discoveries was the value of rules as a way of constraining the optimization. The potential for mischief in PPC auctions is staggering. A single keyword put into the wrong position can generate thousands of dollars in cost in just hours. We have therefore developed safeguards to support our automated bidding solution.

Constraints: Pricing rules were developed hold prices to within a definable percentage of historical prices. This was important for reproducing past performance on an account that we were asked to take over, before gradually improving performance. As we monitored the account and verified that optimizer's actions were having the desired effect, the constraints were able to be relaxed, allowing the prices to move closer to their optimum settings (figure 1).

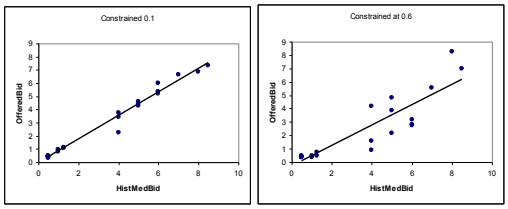

Figure 1. Account with prices constrained to within 10% of historical median, 60% of historical median.

Strategic auctions: Strategic auctions are keywords that carry much of the profit or traffic to a client. Strategic auctions exist in all client accounts because revenue share per auction generally follows a Zipf-like distribution (figure 2) – most of the revenue is generated by a select few auctions. These auctions are problematic because if conditions change this can seriously affect a customer's viability. Thus, like securities trading, the secret to success is having a diverse portfolio; an account with many moderate performing auctions is better than an account with one or two star performers.

In order to protect strategic auctions, we first found and then put special rules in place for these auctions. After this we monitored them as "bell-weathers" to verify that these auctions were being managed properly.

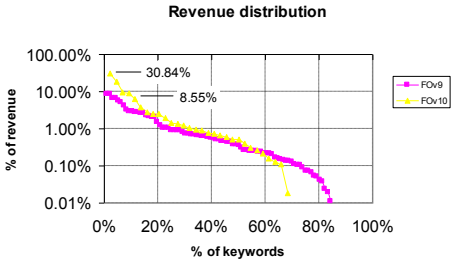

Figure 2. FOv10 is more vulnerable than FOv9, because the former contains a strategic auction that is responsible for 30% of the account's revenue. The arrival of a competitor on this auction could seriously affect FOv10's profitability.

4.2 Monitoring Systems

Failure logging, email notification, and auction monitoring graphs were also important. Logs were invaluable for identifying conditions that were causing problems during our early testing. An auction monitor system was created which collected several dozen graphs on various aspects of the bidding, including forecasts (eg. figure 3). These were

published to a webpage, so that each client account could be monitored in real-time.

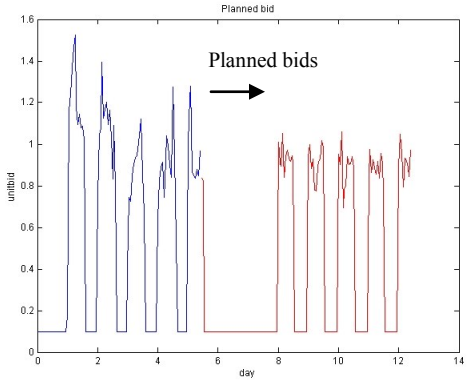

Figure 3. One of our monitoring graphs showing average price of bids submitted in the past and planned in the future. The arrow shows the start of the bid plan. This client suspends bidding during night-time and on weekends, and these shut-downs can be seen in the forecast.

4.3 Constraints aren't really fixed

When formulated as a numerical optimization problem, the budget is traditionally thought of as a quantity that is magically provided by the customer. However, in reality, we discovered that the budget was really an unknown for even our customers, and they were interested in suggestions for what amount might give them the best returns. Because the agent had extensive models for every keyword auction, we found that we could provide this kind of insight. The what-if analysis can be performed by running under hypothetical budgets, keyword-rules, or positions, and observing the predicted impact on acquisitions, clicks, position, and revenue under each condition, based on the optimizer's best bid allocation under these constraints. An example what-if analysis is shown in figure 4. This kind of analysis provided clients with a deep understanding into the efficiency of their programs. For instance, PPC auctions often show diminishing returns as spending increases. Using the what-if analysis, we found that we could cut the spending of many clients in half and generate almost as many acquisitions.

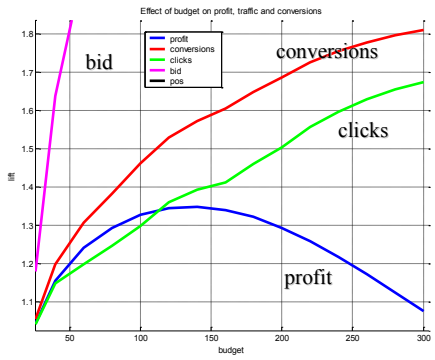

Figure 4. What-if analysis for ten potential budgets. Above a budget of $130 profit begins to decline as positions are captured which are too expensive to justify their cost. All metrics are the same scale, with the value at budget of $20 equal to 1.0.

5. Multi-agent Simulator

The simulator module is a Visual C# system that allows the production system to be taken off-line and tested in a safe environment. The simulator module reads and writes to the same database tables that are used in production. This allows the switch from production to test environment to be seamless from the perspective of the agent.

A simulator file defines the parameters of the simulation. The simulator file is written using the XML standard format [2] and defines a number of functions. These include:

Duration: The length of time for which the simulator will be running.

ClickTotal(k,t): This is the total number of clicks that will be generated by keyword *k* at time *t* in the simulation. In real web traffic data, we find that most clicks occur around midday excepting the lunch break. Least traffic occurs around 3am. Figure 5 shows this function modeled as a sine wave. Obviously international websites may have a different profile.

ClickProb (k,t,p): This is the proportion of the traffic that will be absorbed by a particular position. For instance, if we are in position 1, we may collect 50% of the clicks. If we are in position 3, we may collect 10%. Empirically this distribution tends to follow an exponential distribution, although figure 5 depicts a linear function.

ConversionProb(k,t,p): This is the probability of a customer who has visited the simulated site, of converting to a paid customer. There is a lore in the search engine community that customers who click

lower down in search results will be more likely to convert, since someone in position 100 of a list is assumed to be studiously looking for a service. If this were true, ConversionProb would be properly modeled as a function of position. Although our experimental evidence to-date has not supported this hypothesis, we can test the impact of such an eventuality in the simulator.

ConversionValue(k,t,p): The revenue a customer generates once they click through onto the site.

DatabaseAgent(c,k,t,p): We can define any number of robot Database Agents. These are each tiny Visual C# programs which define behavior for various agents. For example, figure 6 depicts an agent which repeatedly bids one cent above your bid.

```
<ClickTotal>
return (int)(5 +
40*(1+Math.Sin((CurrentTime.TimeOfDay.Hours/23
.0)*2*Math.PI)) );
</ClickTotal>
<ClickProbability>
        if(Position == 0) return 1.0;
        else return ( (7.0-Position) / 6.0);
</ClickProbability>
<ConversionProbability>
        return 0.5;
</ConversionProbability>
<ConversionValue>
        return 4.0;
</ConversionValue>
<BidAgent listingId="1">
<BidFunction>
  if (MarketBids == null) { return A; }
  Random random = new Random();
  if (PreviousBid LessThan F) { return F; }
 if (random.NextDouble() LessThan C) {
return (PreviousBid + ( random.NextDouble() * D
– D/2.0)); } else { return PreviousBid; }
</BidFunction>
</BidAgent>
```

Figure 5. Random Bidding Simulation

5.1 Random bidding

The first simulator experiment we describe involves agents that shift their bids randomly. Our agent's initial price is *A*. Every 10 minutes, an agent with probability *C*, increments its price according to a uniform random distribution between –*D*/2 and *D*/2. If an agent's bid drops below a floor *F*, the agent's price resets to *F*. This same bounded random walk was used to model airline prices in the 2001 Trading Agent Competition [7]. Although simplistic, this provides a useful benchmark with which to understand the capabilities and deficiencies of the agent. The program for this bidding behavior is shown in figure 5.

```
<BidAgent listingId="1">
        <BidFunction>
        if(PreviousBids == null) return 0.01;
        return PreviousBids[1] + 0.01;
        </BidFunction>
</BidAgent>
```

Figure 6. Anti-social competitor

We now describe several experiments using our Random Bidding simulation. In figures 7 to 11, our agent is agent 1000001. In figure 12 and 13 our agent is agent 1. Finally, each figure uses three sub-windows. The top window is profit by time. The middle is bid by time, and the bottom is clicks by time. We denote the letters A, B, C on these graphs to draw attention to events of interest.

5.1.1 Optimize profit

In this scenario the agent has an objective of maximizing profit and is not subject to any constraints. In Figure 7 top window (point A), agent 100001's profit tends to be higher than that of any other agent. Agent 100001 tends to maintain a bid between $0.25 and $1.00, which is the optimum.

Figure 8 shows the same simulation on a different simulation run. Again note that agent 100001 has a profit higher than the other agents. About two-thirds of the way through the simulation (Figure 8, point A), agents 2, 5 and 6 all dramatically increase their bids. Profit for these agents plummets, with agent 6 clearly showing negative profit (point B). However agent 100001 does not follow suit and continues to generate good profit.

5.1.2 Maintain position under a maximum bid

In this experiment we wanted to see how the agent would fulfill competing constraints. The agent is instructed to maintain position of 2 with maximum bid < $1.50.

Refer to figure 9. At point B, the agent is in position 2 and following the bid as it goes higher. At $1.50 the agent can no longer afford position 2. At that point the agent drops down to position 3.

Position 3 becomes too high at point C, and the agent drops further into position 4. The agent remains in position 4 for the remainder of the simulation.

5.1.3 Surf the gap

A "gap" in a PPC auction exists when there is a group of competitors who are *d* cents away from each other, followed by a steep rise *D>>d* to another group of competitors. Say the auction has prices for positions 4 to 1 are $1.41, $1.42, $1.43, $3.09. A person who bids $1.44 would place themselves into position 2 for almost the same price as the person in position 4 ($1.41). The region $1.44 to $3.09 is known as a "gap". Heuristically it can be a fairly effective strategy to have your agent look for these gaps and situate itself in them.

We tested the performance of "gap surfing" by instructing the agent to situate itself in the largest gap greater than $0.20. In figure 10 the largest gap is initially in position 2 (point A), but later the auction changes and it is in position 4 (point B). Surprisingly, using this simple rule, the agent generates more profit than any of the other agents.

At the time of writing, at least five commercial products offer "gap surfing" rules, so this result is certainly interesting. One would expect that picking the right parameters for the gap-size, however, would be difficult without formal analysis.

5.1.4 Optimize traffic

In this scenario the agent has the task of maximizing traffic generated in each period. The agent is allowed an unlimited budget. Figure 11 middle (point A) shows that the agent picks the highest bid. The bottom figure shows that traffic is higher than any other agent. Interestingly, if we look at the top frame showing profit, the agent can be seen to be running at a considerable loss (eg. see point B) – a natural consequence of this kind of objective!

5.2 Save for a Rainy Day

In this next simulation we test the value of being able to predict and plan for the future. The XML for this simulation is provided in figure 14.

We construct a simulation in which competitors vary their bids in a periodic fashion throughout the day. This is not completely unrealistic – many PPC bid management programs allow users to specify timed bids, and we know of companies who bid low at night and high during the day. We model this with competitors who bid according to a sine wave.

We also model traffic as a sine wave. However, here's the catch. The price and traffic sine waves are out of phase. Thus, the market is cheap when traffic is high. This means that it is most profitable to bid high when the market is cheap.

We begin the simulation with an infinite budget, and around 24 hours into the simulation, we lower the budget available to the agent. At this point, the agent must decide how to allocate a very small budget. The optimal solution would be to allocate it during the cheap part of the day – which is 12 hours in the future; meaning that the agent should hold back on spending and wait for the cheaper period, before spending judiciously so as to get the maximum profit with its meager funds.

```
<ClickTotal>
        return (int)(30 + 30*(1+Math.Sin((
((CurrentTime.TimeOfDay.Hours+12) % 23)
/23.0)*2*Math.PI)) );
</ClickTotal>
<ClickProbability>
        if(Position == 0) return 1.0;
        else return ( (7.0-Position) / 6.0);
</ClickProbability>
<ConversionProbability>
        return 0.5;
</ConversionProbability>
<ConversionValue>
        return 4.0;
</ConversionValue>
<BidAgent listingId="-1"><BidFunction>
return (0.01 +
0.5*(1+Math.Sin((CurrentTime.TimeOfDay.Hours/2
3.0)*2*Math.PI)) );
</BidFunction></BidAgent>
```

Figure 14. Save for a Rainy day simulation. Five competitor agents similar to -1 shown above are defined.

Figure 12 and 13 show the simulation. Our agent is agent 1. Over the first day, as market sinks, the agent's own bid price decreases *and* it takes a higher position (figure 12D). On day 2 we drop its budget to $10,000 per day (figure 12B and figure 13A). In response, the agent immediately shuts down its bidding (figure 13B). The agent still has a budget but is refusing to spend – in fact, the agent is predicting the periodic cheapening of the market in the next 12 hours. When the market becomes cheap, the agent

resumes spending (figure 13C), and shuts down once more after the prices begin to increase. This is an excellent example of the ability of the agent to plan ahead via its future allocation plan.

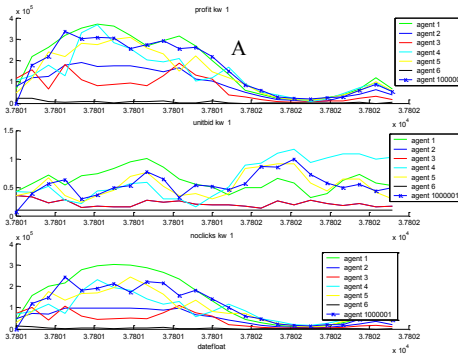

Figure 7. Profit optimization

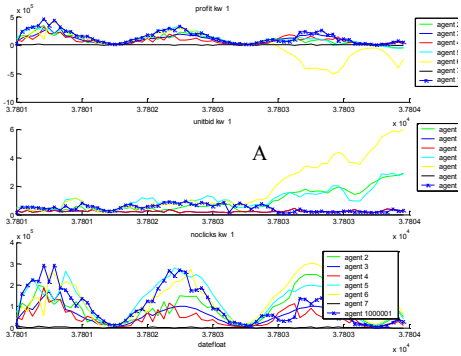

Figure 8. Profit optimization x2

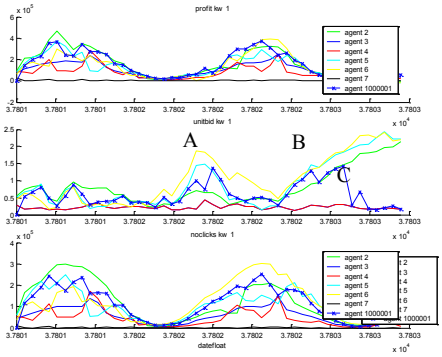

Figure 9. Hold position 2 and Bid<$1.50

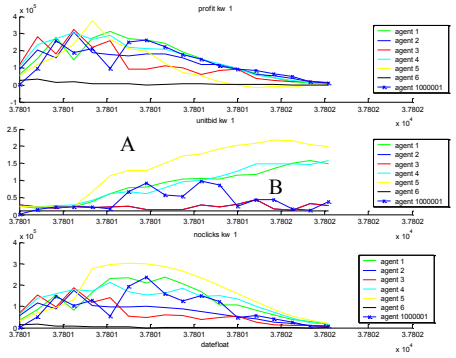

Figure 10. "Surf the gap"

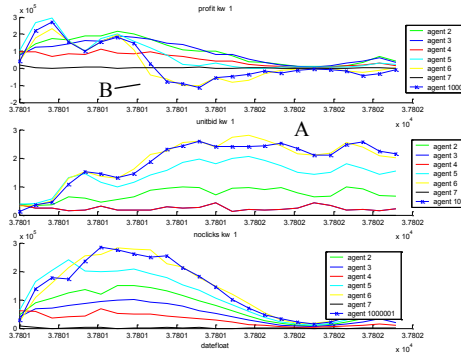

Figure 11. Maximize traffic

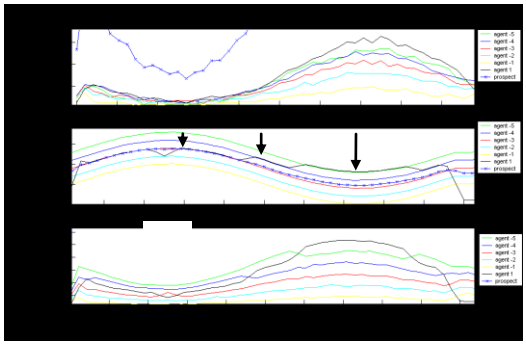

Figure 12. Adjust for periodic markets

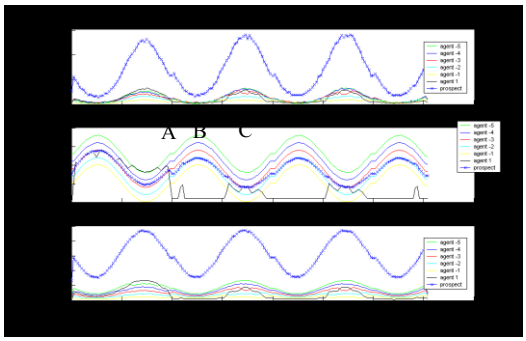

Figure 13. Plan for spending later in the day

Construction of the auction simulator has provided an invaluable tool with which to test our agent. We were also able to better understand the limitations of our system. For example, it is theoretically possible to design a competitor that always bids over our price – thereby forcing us into a less profitable position. Such an "anti-social" agent [1] can assume its position in the simulator faster than we can change our position, and consequently it can be persistent problem. In reality, we have found that our agent changes much faster than other participants (who are presumably mostly managing auctions manually), but the possibility is thought-provoking. We have also considered the possibility of building internal models of competitor behaviors as nearly all competitors in PPC auctions employ fixed rules that could be inferred after some probes; and then planning around these responses.

6. References

[1] Brandt, F. and Weiss, G., "Antisocial Agents and Vickrey Auctions", *Proceedings of the Eighth International Workshop on Agent Theories, Architectures*, Seattle. 2001.

[2] Bray, T., Paoli, J., Sperberg-McQueen, C., Maler, E., "Extensible Markup Language (XML) 1.0", *W3C Recommendation 6*, October 2000. http://www.w3.org/TR/REC-xml.

[3] Fielding, R., Gettys, J., et. al., "Hypertext Transfer Protocol -- HTTP/1.1", *Network Working Group RFC 2616*, 1999. http://www.w3.org/Protocols/

[4] Kaebling, L., *Learning in Embedded Systems, MIT Press*, Cambridge, MA. 1993.

[5] Kitts, B. and LeBlanc, B., "Optimal Bidding on Keyword Auctions", *Electronic Markets*, Vol. 14, No. 3. in press.

[6] Trowbridge, D., Hohpe, G., et al., *Enterprise Solution Patterns Using Microsoft .NET*, Microsoft Press, 2003.

[7] Wellman, M., Greenwald, A., Stone, P. and Wurman, P., "The 2001 Trading Agent Competition", *Electronic Markets*, Vol. 13, No. 1. 2003.

[8] NET Framework Developer's Guide, Microsoft Developer Network Web Page, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconoverviewofadonet.asp

[9] Trading Agent Competition, Detailed Rules, http://www.sics.se/tac/page.php?id=3&PHPSESSID=600dc3960f72e1b8a7b632dfc6b4d933

[10] Technology Information for the .NET Framework 1.1, Microsoft Corporation, http://msdn.microsoft.com/netframework/technologyinfo/