Autonomous Bidding Agents in the Trading Agent Competition

Designing agents that can bid in online simultaneous auctions is a complex task. The authors describe task-specific details and strategies of agents in a trading agent competition.

Amy Greenwald
Brown University

Peter Stone
AT&T Labs–Research

A natural offshoot of the growing prevalence of online auctions is the creation of autonomous bidding agents that monitor and participate in these auctions. It is straightforward to write a bidding agent to participate in an online auction for a single good, particularly when the value of that good is fixed ahead of time: the agent can bid slightly over the ask price until the auction closes or the price exceeds the value. In simultaneous auctions offering complementary and substitutable goods, however, agent deployment is a much more complex endeavor.

The first trading agent competition (TAC), held in Boston, Massachusetts, on 8 July 2000, challenged participants to design a trading agent capable of bidding in online simultaneous auctions for complimentary and substitutable goods. TAC was organized by a group of researchers and developers led by Michael Wellman of the University of Michigan and Peter Wurman of North Carolina State University. In a companion article, the tournament organizers present the design and operation of the competition.[1]

This article describes the task-specific details of, and the general motivations behind, the four top-scoring agents. First, we discuss general strategies used by most of the participating agents. We then report on the strategies of the four top-placing agents. We conclude with suggestions for improving the design of future trading agent competitions.

General Game Strategies

A TAC game instance lasts 15 minutes and pits eight autonomous bidding agents against one another. Each TAC agent is a simulated travel agent with eight clients, each of whom would like to travel from TACtown to Boston and home again during a five-day period. Each client is characterized by a random set of preferences for arrival and departure dates, hotel rooms, and entertainment tickets. A TAC agent's objective is to maximize the total

 1089-7801/01/$10.00

utility (a monetary measure of the value of goods to clients) minus total expenses.

Agents have two basic activities: *bidding* and *allocating*. An agent bids, or offers payment, for goods to gain utility; an agent allocates purchased resources to clients to maximize total utility, both during and at the end of the game. To obtain utility for a client, a TAC agent constructs a travel package for that client by placing winning bids in simultaneous auctions for hotel reservations and flights. An agent can obtain additional utility by buying and selling entertainment tickets. After the auctions close, agents have four minutes to report their final allocations of goods to clients. A TAC agent's score is the difference between its clients' utilities and the agent's expenditures; a higher score indicates better performance. For full details, see http://tac.eecs.umich.edu.

```
(A) REPEAT
    1. Get market prices from server
    2. Decide on what goods to bid
    3. Decide at what prices to bid
    4. Decide for how many to bid
    5. Decide at what time to bid
     UNTIL game over
(B) Allocate goods to clients
```

Figure 1. High-level overview of a TAC agent's bidding decisions. TAC agents run an inner bidding loop, getting price updates from the server and making bidding decisions. Note that the order in which these decisions are made varies according to the agent's strategy.

Bidding Strategies

Figure 1 lists the basic decisions in the agents' inner bidding loops. Most decisions—what to buy, how many to buy, and how much to bid—showed several noteworthy differences. Only the timing of bids showed common features across agents.

Hotel auctions. In TAC auctions, the supply of flights is infinite and airline ticket prices are predictable, but the supply of hotel rooms is finite and hotel prices are unpredictable. Given the risks associated with the hotel reservation auctions, together with their importance in securing feasible travel packages, hotels were the most hotly contested items during the TAC competition. The timing of bidding in hotel auctions is particularly intriguing.

Hotel room prices have no set maximum value. Instead, TAC hotel auctions are ascending (English), mth-price, multiunit auctions and are subject to random closing times given sufficient levels of inactivity. (In a multiunit auction, m units of a good are available; in an mth price auction, the bidders with the m highest prices win the m units, all at the mth highest price.) Most TAC agents refrain from bidding for hotels early in the game unless the ask price has not changed recently, implying that the auction might close early, or the ask price is very low. Ultimately, the most aggressive hotel bidding takes place at the "witching hour"—in the final moments of the game—although precisely when is determined individually by each agent. More often than not, TAC hotel auctions reduce to mth price sealed-bid auctions, resulting in unpredictable final hotel prices that were often out of bounds.

Treating all current holdings of flights and entertainment tickets as sunk costs, the marginal utility of an as-yet-unsecured hotel room reservation is precisely the utility of the package itself. (Note that this observation holds only when the length of stay is exactly one night; for longer stays, it requires the further assumption that all other hotel rooms in the package are secured.) Throughout the preliminary competition, few agents bid their marginal utilities on hotel rooms. Those that did, however, generally dominated their competitors. Such agents were high bidders, bidding approximately \$1,000, always winning the hotels on which they bid, but paying a price far below their bid. Most agents adopted this high-bidding strategy during the actual competition. The result: many negative scores, since there were often greater than m high bids on a hotel room. In the final competition, the top-scoring TAC agents were those that not only bid aggressively on hotels, but also incorporated risk and portfolio management into their strategy to reduce the likelihood of buying highly demanded, overpriced hotel rooms.

The top-scoring agents were those that not only bid aggressively, but also incorporated risk and portfolio management into their strategy.

Flight auctions. Unlike hotel prices, prices for flights are predictable, with a maximum value of \$600. In particular, expected future prices equal current prices. Since airline prices periodically increase or decrease by a random amount chosen from the set $\{-10, -9, \ldots, 9, 10\}$ with equal probability, the expected change in price for each airline auction is 0. (Indeed, it can be shown that if

Table 1. The agents' effectiveness in optimizing final allocations during the competition.

Agent	Strategy	Aggregate	Minimum	Number
ATTac	Optimal	100.0%	100.0%	13/13
RoxyBot	Optimal	100.0%	100.0%	13/13
Aster	Heuristic	99.6%	98.0%	9/13
UmbcTac	Greedy	99.4%	94.5%	7/13

Note: Aggregate is the percentage of the optimal utility (ignoring expenditures) achieved with the reported allocation, aggregated over the 13 games each of the top four agents played. *Minimum* is the minimum among these aggregated values. *Number* is the number of times the agent reported an optimal allocation. This information was provided by the TAC organizing team.

the airline auctions are considered in isolation, waiting until the end of the game to purchase tickets is an optimal strategy, except in the rare case where the price hits the lower bound on its value.) Thus, the auction design offers no incentive to bid on airline tickets before the witching hour, since by waiting there is some chance of obtaining information about hotel acquisitions.

There are, however, substantial risks associated with delaying bid submission. These risks arise from unpredictable network and server delays, and can cause bids placed during a game to be received after the game is over. To cope with these risks, most agents dynamically computed the length of their bidding cycles and then placed their flight bids some calculated amount of time before the end of a game. For example, a risk-averse agent might compute the average length of its three longest bidding cycles and then place its flight bids as soon as game time reaches 900 seconds minus twice that delay. A more risk-seeking agent might place its flight bids later, perhaps with game time of 900 seconds minus the minimum length of its five most recent bidding cycles. In practice, flight bids were placed anywhere from five minutes to 30 seconds before the end of the game.

Flight auctions are such that agents who place a winning bid pay not their bid but the current ask price. Thus, most agents bid above the current price—agents often bid the maximum—to ensure that these bids, which were placed at critical moments, were not rejected because of information delays resulting from network asynchrony.

Entertainment auctions. Agents' bidding strategies differ most substantially in auctions for entertainment tickets. While some agents focus on obtaining complete packages, others make bidding decisions on travel packages alone (that is, flights and hotel rooms) without regard for entertainment packages, essentially breaking the TAC problem down into two subproblems and then solving greedily. The greedy approach, however, is not optimal. For example, if a client does not already have a ticket to an event, then it is preferable to extend the client's stay whenever the utility obtained by assigning that client a ticket to the event exceeds the cost of the ticket and an additional night at the hotel plus any travel penalties incurred. Similarly, it is sometimes preferable to sell entertainment tickets and shorten a client's stay accordingly.

Allocation Strategies

When allocating goods to their clients, most of the agents greedily focus on satisfying each of their clients in turn. Only the top two teams' bidding strategies incorporated global decision-making in which the interests of all their clients were considered simultaneously. (The third-scoring agent used a heuristically based intermediate approach.) This aspect of an agent's design also affects its final allocation algorithm. The percentage of optimal allocations reported by each agent during the competition is listed in Table 1.

Although simpler, the greedy strategy is not always optimal. For example, consider two clients, *A* and *B*, with identical travel preferences and the following entertainment preferences: *A* values the symphony at \$90, the theater at \$80, and baseball at \$70; *B* values the symphony at \$175, the theater at \$150, and baseball at \$125. Suppose each client will be in town on the same night and that one ticket for each entertainment type is for sale for \$50 on that night. An agent using a greedy approach who considers client *A* before *B* will assign *A* the ticket to the symphony and *B* the ticket to the theater, obtaining an overall utility of \$140. It would be optimal, however, to assign *A* the ticket to the theater and *B* the ticket to the symphony, yielding an overall utility of \$155.

Top-Scoring Agents' Strategies

In the remaining sections of this article, we describe the bidding, allocation, and completion strategies; team motivations; and any unique approaches of the four top-scoring agents.

ATTac: Adaptability and Principled Bidding

ATTac placed first by using a principled bidding strategy. Several elements of strategic adaptivity gave ATTac the flexibility to cope with a wide vari-

ety of possible scenarios during the competition. ATTac's design was motivated by its developers' interest in multiagent learning.

Bidding. At every bidding opportunity, ATTac begins by computing the most profitable allocation of goods to clients (denoted G^*), given the currently owned goods and the current prices of hotels and flights. (For hotels, ATTac actually uses predicted closing prices based on the results of previous game instances.) For this computation, ATTac allocates, but does not consider buying or selling, entertainment tickets. In most cases, G^* is computed optimally through mixed-integer linear programming.

ATTac bids in either passive or active mode. In passive mode, ATTac computes the average time it takes to place a bid, keeping its bidding options open until the witching hour. When the time left in the game equals twice the time of an average transaction, ATTac switches to active mode, during which it buys the airline tickets required by G^* and places high bids for the required hotel rooms. ATTac expects to run at most two bidding iterations in active mode.

Based on the current G^*, its current mode, and the average time of its transactions, ATTac bids for flights, hotel rooms, and entertainment tickets. Stone et al. detail ATTac's strategy[2]; we focus on bidding strategies for entertainment tickets.

On every bidding iteration, ATTac places a buy bid for each type of entertainment ticket and a sell bid for each type of entertainment ticket it currently owns. In all cases, prices depend on the amount of time left in the game, and become less aggressive as time goes on.

For each owned entertainment ticket E, if E is assigned in G^*, let $V(E)$ be the value of E to the client to whom it is assigned in G^*. Since \$200 is E's maximum possible value to any client under the TAC parameters, ATTac offers to sell E for min (200; $V(E) + \delta$) where δ decreases linearly from 100 to 20 based on the time left in the game. ATTac uses a similar "sliding price" strategy for entertainment tickets that it owns but did not assign in G^* (because all clients are either unavailable that night or are already scheduled for that type of entertainment in G^*).

Finally, ATTac bids on each type of entertainment ticket (including those it is also offering to sell) based on the increased value of G^* that would be derived by owning it (that is, G^* is entirely recomputed with a hypothetical additional resource). Again, a sliding price strategy is used, this time with the buy price increasing as the game proceeds. The sliding price strategy allows the agent to take advantage of large value inconsistencies at the beginning of the game, while capitalizing on small potential utility gains at the end.

Allocation. ATTac relies heavily on computing the current G^*. Since G^* changes as prices change, ATTac needs to recompute it at every bidding opportunity. ATTac used a mixed-integer linear programming approach to compute optimal final allocations in every game of the tournament finals—one of only two entrants to do so (see Table 1).

Using a mixed-integer linear programming approach, ATTac specifies the desired output: a list of new goods to purchase and an allocation of new and owned goods to clients to maximize utility minus cost. ATTac searches for optimal solutions to the defined linear program using "branch and bound" search. This approach will find the optimal allocation, usually in under one second on a 600-MHz Pentium computer.

ATTac's "branch and bound" search usually found the optimal allocation in under one second.

Online adaptation. TAC appealed to ATTac's developers because it appeared to be a good application for machine-learning techniques, one of the developers' main research interests.[3] However, TAC was conducted in such a way that it was impossible to determine competitors' bids; only the current ask prices were accessible. This precluded learning detailed models of opponent strategies. ATTac instead adapts its behavior online in three ways:

- ATTac decides when to switch from passive to active bidding mode based on the observed server latency during the current game instance.
- ATTac adapts its allocation strategy based on the amount of time the linear program takes to determine optimal allocations in the current game instance.
- Perhaps most significantly, ATTac adapts its risk-management strategy to account for potentially skyrocketing hotel prices.

For flights, ATTac computed G^* based on current prices. For hotels, however, it predicted current game closing prices based on closing prices in previous games. ATTac divided the eight hotel rooms into four equivalence classes, exploiting symme-

```
(A) REPEAT
  1. Update current prices and holdings
  2. Estimate clearing prices and build
     pricelines
  3. Run completer to find optimal buy/sell
     quantities
  4. Set bid/ask prices strategically
    UNTIL game over
(B) Run optimal allocator
```

Figure 2. RoxyBot's high-level strategy. Roxybot's inner bidding loop refines the generic loop shown in Figure 1.

tries in the game (hotel rooms on days one and four should be equally in demand as rooms on days two and three), assigned priors to the expected closing prices of these rooms, and then adjusted these priors based on closing prices observed during the tournament.

Whenever the actual price for a hotel was less than the predicted closing price, ATTac used the predicted hotel closing price for computing its allocation values. This strategy works well both when hotel prices escalate and when they do not.[2] Indeed, ATTac performed as well as the other top-finishing teams in the early TAC games when hotel prices stayed low, and then outperformed its competitors in the tournament's final games when hotel prices rose to high levels.

RoxyBot: An Approximately Optimal Agent

RoxyBot's algorithmic core, based on AI heuristic search techniques, incorporates an approximately optimal solver for completion and an optimal solver for allocation. The formulation of the completion problem involves a novel data structure called a *priceline*, which is designed to handle future closing prices, future supply and demand, sunk costs, hedging, and arbitrage in a unified way. RoxyBot's high-level strategy is outlined in Figure 2; full details are available in Boyan and Greenwald.[4]

Allocation. RoxyBot's *allocator*, which runs at the end of a game, helps motivate the completer algorithm used during each bidding cycle. The allocator solves the following problem: Given a set of travel resources purchased at auction, and given the clients' utility functions defined over subsets of travel resources, how can the resources be allocated to the clients so as to maximize the sum of their respective utilities? Although this problem is NP-complete,[5] an optimal solution based on A^* search is tractable for the dimensions of TAC. Indeed, using an intricate series of admissible heuristics, RoxyBot pruned the search tree of possible optimal allocations from roughly 10^{20} to 10^3 possibilities. As a result, it typically discovered provably optimal allocations in one half of a second in all of the competition games.

The A^* search traverses a tree of depth 16. Search begins at the top of the tree with the given collection of resources. At each level of the tree, a subset of the remaining resources is allocated to a client, and those resources are subtracted from the pool. Levels 1 through 8 correspond to the choice of feasible travel package (that is, combination of flights and hotel rooms) to assign to clients 1 through 8, respectively. There are 21 such travel packages, including the null package. Levels 9 through 16 of the tree correspond to the choice of entertainment package (that is, sets of entertainment tickets of different types on different days) to assign to clients 1 through 8.

There are 73 entertainment packages, though many are infeasible due to earlier assignments of travel packages. The heuristics compute an upper bound on a quantity (for example, the maximum possible number of feasible packages using good hotels, or arriving on day three). Then, subject to these upper bounds, all as-yet-unassigned clients are assigned their preferred package among those remaining, ignoring conflicts. Caching tricks employed at the start of each game enable these heuristics to be computed very quickly.

Completion. The *completer* that runs during each bidding cycle is the heart of RoxyBot's strategy. It aims to determine the optimal quantity of each resource to buy and sell, given current holdings and forecasted closing prices. Like the allocator, it considers all travel resources from a global perspective and makes integrated decisions about hotel, flight, and entertainment bids. Unlike the allocator, the completer faces the added complexity that the resources being assigned may not yet be in hand, but may still need to be purchased at auction. Furthermore, it might be more profitable to sell in-hand entertainment tickets than allocate them to its own clients.

RoxyBot's completer relies on its pricelines to reason about the trade-offs involved with each resource. The priceline transparently handles either one-sided or double-sided auctions, short-selling of resources, hedging, and both limited and unlimited supply and demand. This construction greatly simplifies the completer's task because the cost of a package equals the totals of the corresponding pricelines, and the value of a package to a client

equals the client's utility for that package minus its cost. Given the pricelines and the corresponding client valuations of packages, A^* search can be used to find the optimal set of buying and selling decisions. Unfortunately, most of the A^* heuristics used in RoxyBot's optimal allocator were not applicable in the completer scenario, and running times for an optimal completer occasionally took as long as 10 seconds. Nonetheless, using a greedy, nonadmissible heuristic and a variable-width beam search over the same search space, in practice RoxyBot usually found an optimal completion within about three seconds. Therefore, during the competition, RoxyBot used beam search rather than provably optimal A^* search.

Estimation. RoxyBot's priceline data structures describe the costs of market resources. However, in auctions such as those fundamental to the TAC setup, costs are not known in advance. Therefore, the actual input to RoxyBot's pricelines are but estimates of auction closing prices and estimates of market supply and demand (current holdings are known), which RoxyBot produces using machine-learning techniques. However, the final round of the TAC competition was too short and the agent strategies too different from the preliminary rounds for Roxybot to effectively use most of the learning algorithms that were developed. Only entertainment ticket price estimates were adaptively set, using an adjustment process based on Widrow-Hoff updating.[6] In future competitions, RoxyBot's creators hope that TAC will be more suited to the use of learning algorithms for price-estimation based on bidding patterns observed during a game instance and an agent's own clients' preferences.

Aster: Flexible Cost Estimation

Aster, the third-placing agent, was designed by members of the Strategic Technologies and Architectural Research Laboratory at InterTrust Technologies. Aster's cost estimation framework is flexible and can respond to strategic behavior of competing agents. Aster's allocation heuristics are relatively simple and fast, and they produce high-quality solutions.

Like RoxyBot, Aster runs a loop. During each iteration, Aster gets the status of all auctions, estimates the costs of resources, computes a tentative allocation based on estimated costs, and bids for some desired resources. After all auctions close, Aster runs a sophisticated algorithm to compute the final allocation.

Bidding. Aster bids using one of two strategies that correspond to game stages before and after the witching hour. Like ATTac, Aster initiates its precommit bidding stage as late as feasible, based on previous delay in accessing the AuctionBot, with the hope that it will be able to complete at least one iteration during the active stage. During the precommit stage, Aster does not bid on flights. In committed stage, Aster places all necessary flight bids to achieve the current allocation.

In the precommit stage, Aster places limited bids on hotel rooms, trying to capture early closings. At the same time, it tries to avoid engaging in price hikes by placing bids at the minimum allowable increment (that is, the ask price plus $1). Aster limits its bids for each client to at most two consecutive nights, even if the allocator has scheduled the client for a longer stay. (With at most two nights, if the hotel price on one night shoots up, Aster can drop the expensive night without loss. If it bid for more than two nights and the price on a middle night were to shoot up, it could get stuck with a room or two for the outer nights.) During the committed stage, Aster bids for every night of each client's allocated stay. The amount of these bids equals the utility due to that client.

Aster's cost estimation framework is flexible and can respond to strategic behavior of competing agents.

Aster's strategy for buying and selling entertainment tickets is independent of game stage. It sets the bid and ask prices for tickets by using their utility in the current allocation as well as precomputed expected utilities for other trading agents. Notably, Aster's goal is to obtain greater utility than the other agents, not to maximize its own utility. In some games, Aster profited by buying and selling the same entertainment ticket.

Allocation. On each iteration, Aster computes a tentative allocation of resources by using a local search algorithm that considers pairs of clients in turn, given estimated costs and current holdings. It starts with all clients having no resources and then iterates over all pairs of clients, deallocating their current resources and allocating new resources to maximize utility. This procedure uses the cost vectors as stacks: Deallocating a resource frees up the cost of the last allocated copy, and allocating a resource incurs the cost of an additional copy. Repeated iterations are conducted until utility does not improve.

Aster uses heuristic search to complete its final allocation. To compute the globally optimal allocation of its travel goods to its clients, it searches a tree consisting of all possible travel packages (that is, arrival dates, departure dates, and hotel types) for all clients. Then, at each leaf of this tree, Aster computes an entertainment assignment by iterating over all pairs of clients, deallocating and reallocating entertainment tickets optimally, until the entertainment allocation cannot be improved.

The above search algorithm is not optimal because the entertainment ticket assignment process is only locally optimal, but is not optimal over all clients viewed from the global perspective. Thus, after this first search, Aster searches again, in an attempt to compute an optimal entertainment allocation over all clients while keeping their travel packages fixed. The allocation heuristic performs well, usually finding an optimal or a near-optimal solution (see Table 1).

UmbcTAC bases its bidding strategy on the computed network delay between the agent and the TAC server.

Aster uses pruning in both searches to cut execution time. Although not provably optimal, Aster's designers believe that such approximate approaches will scale better to larger games than exact approaches, since the size of the search tree can be explicitly controlled.

Estimation. Just as RoxyBot computes a priceline for each resource, Aster computes a cost vector, whose *i*th entry gives the cost of holding or acquiring the *i*th copy of that resource. Also like RoxyBot, when estimating costs, Aster treats sunk costs as no costs. For example, the estimated cost for flights is zero for currently held tickets and the current ask price for additional tickets.

For hotels, estimating costs is tricky because both price and holdings are unknown until an auction closes. Aster predicts the closing price for a hotel room by linearly extrapolating previous ask prices on the basis of current time. This extrapolated price is then adjusted as follows: For rooms Aster has hypothetically won, the cost is reduced; the amount of this reduction depends on the probability that these winnings would be ultimately realized (the higher the bid, the higher the probability, and the lower the estimated cost). For additional rooms, the cost is increased exponentially to model potential increases in closing prices due to Aster's own bids.

Since the AuctionBot provides only one bid and ask quote per entertainment ticket, Aster assumes that the cost of buying an additional ticket is the current ask price and that the cost of further tickets is infinite. For all tickets that Aster currently holds, the opportunity cost of one ticket is set to the current bid price, while the opportunity cost of all the remaining tickets is set to zero.

UmbcTAC: Network Sensitivity and Adaptability

UmbcTAC, created at University of Maryland Baltimore County, placed fourth. UmbcTac was particularly sensitive to network load: It adapted to network performance more frequently than competing agents and received more frequent updates.

Bidding. UmbcTAC maintains the most profitable itinerary for each client individually, based on the latest price quotes (as opposed to solving the full eight-client optimization problem). UmbcTAC balances several strategies to avoid switching travel plans too frequently against some strategies to encourage switching travel plans early on:

- When a client's itinerary is changed, the value of the goods that will no longer be needed is subtracted from the value of the new itinerary as a penalty for changing plans. Therefore, the client's travel plans will not change unless the new plan's profit overrides the value of wasted goods. Wasted goods that are already won, or would be won if an auction closed immediately, are marked as free goods, their prices are set to 0, and they are treated as sunk costs. Free goods may then be used in other clients' itineraries.
- UmbcTAC changes a client's travel plan only if the profit difference between the new and the old plans exceeds a threshold value (typically between $10 and $100).
- It is important to change a client's itinerary as early in the game as possible because the earlier the plan changes, the more likely it is that the obsolete bids either will not win or will win at a low price. To ensure that at least one client changes to a better plan, UmbcTAC risks wasting one good in each round by setting the penalty for the first wasted good to 0.

Once the desired goods have been determined, UmbcTAC sets its bid prices as follows:

- *Flights*. The agent bids a price significantly higher than the current price to ensure that the client gets the ticket.

- *Hotels*. The agent computes the price increment, defined as the difference between the current price quote and the previous price quote. It sets the bid price to be the current price plus the price increment. During the witching hour, UmbcTAC bids for hotels at a price such that if it wins a hotel at that price, the client's utility would be 0.
- *Entertainment*. The agent buys entertainment tickets for a client if the client is available (that is, in town and without an entertainment ticket for that night or of that type). It buys the ticket that the client most prefers at the market value. Any extra tickets are sold at auction at an ask price equal to the average of the preference values of all UmbcTAC's clients.

UmbcTAC continually bids for hotels to guard against the possibility of hotel auctions closing early. It bids for airline tickets and raises hotel bids to their limit only in the last few seconds of the game.

Allocation. At the end of the game, UmbcTAC allocates the purchased flights and hotel rooms greedily to the clients according to the most recent travel plans used during the game. If a client cannot be satisfied, its goods are taken back and marked as free goods, which other clients can then try to use to improve utility. Entertainment tickets are also allocated greedily. This strategy is simple and nearly optimal. UmbcTAC begins by allocating an entertainment ticket to the available client with the largest preference value for that ticket.

Bandwidth management. In TAC, prices change every second, and hotel auctions may close at any time. Therefore, keeping bidding data up to date is very important. UmbcTAC bases its bidding strategy on the computed network delay between the agent and the TAC server. When the network delays are longer than usual, UmbcTAC is more aggressive, offering higher prices and bidding for flights earlier. To save network bandwidth, the agent never bids for any entertainment tickets during the last three minutes of the game. To save time, the agent does not change travel plans during its last two or three bidding opportunities. On average, UmbcTAC updates its bidding data every four to six seconds, providing a significant advantage over the 8- to 20-second delays reported by others.

Suggestions for Future Competitions

The first trading agent competition drew 22 entrants from around the world. Although participants' experiences were overwhelmingly positive, we propose a few modifications to the structure of future tournaments.

- There is no incentive to buy airline tickets until the end of the game. If the price of flights tended to increase, or if availability was limited, agents would have to balance the advantage of keeping their options open against the savings of committing to itineraries earlier.
- The hotel auctions were effectively reduced to sealed-bid auctions. In particular, there was no incentive for agents to reveal their preferences before the very end of the game. As a result, it was impossible for agents to model market supply and demand and thereby estimate prices.

  The phenomenon of English auctions with set closing times reducing to sealed-bid auctions has been observed in other online auction houses such as eBay. Roth and Ockenfels argue that in such auctions, it is in fact an equilibrium strategy to place multiple bids (with increasing valuations) and to bid at the last possible moment.[7] This observation contradicts the usual intuition pertaining to second-price sealed-bid auctions, namely that a single bid at one's true valuation is a dominant strategy.

  Amazon runs online auctions in which the length of the auction is extended beyond its original closing time, say T, by 10 minutes each time a new (winning) bid is received. In this case, equilibrium behavior dictates that all bidders bid their true valuations before time T. If the TAC hotel auctions were implemented in the style of Amazon, rather than eBay, agents would likely bid earlier. In this way, TAC agents would obtain more information pertaining to the specific market supply and demand induced by the random client preferences realized in each game instance, and could use this information to estimate hotel prices. Unfortunately, Amazon-style auctions have the downside that they might never end!
- Activity in the entertainment auctions was limited. This outcome, however, is not obviously correlated with the design of the entertainment auction mechanism. On the contrary, if more

The information structure of the TAC setup made it impossible to observe the bidding patterns of individual agents.

structure were added to the flight auctions, and if the hotel auctions were modified, interest in entertainment ticket auctions might increase.

- The information structure of the TAC setup made it impossible to observe the bidding patterns of individual agents. Nonetheless, the strategic behavior of individual agents often profoundly affected market dynamics, particularly in the hotel auctions. It seems that either the dimensions of the game should be extended such that the impact of any individual agent's bidding patterns is truly negligible; or, to avoid issues of scalability, it should be possible to directly model the effect of the behavior of each individual agent. If information were available about the bidding behavior of the agents (such that other agents could induce clients' preferences, and therefore market supply, demand, and prices), TAC agents might learn to predict market behavior as a game proceeds.

The agents developed for TAC are a first step toward creating autonomous bidding agents for real, simultaneous interacting auctions. One such auction is the Federal Communications Commission's auction of radio spectrum.[8,9] For companies that are trying to achieve national radio coverage, the values of the different licenses interact in complex ways. Perhaps autonomous bidding agents will impact bidding strategies in future auctions of this type. Indeed, the ATTac developers created straightforward bidding agents in a realistic FCC Auction Simulator.[10] In a more obvious application, an extended version of TAC agents could become a useful tool to travel agents, or to end users who wish to create their own travel packages.

Acknowledgments

This article is the result of the efforts of many people, including all of the TAC finalists. The authors are particularly indebted to Justin Boyan (RoxyBot), Umesh Maheshwari (Aster), and Youyong Zou (UmbcTAC) for their contributions to their respective sections.

References

1. TAC Team, "A Trading Agent Competition," *IEEE Internet Computing*, vol. 5, no. 2, Mar./Apr. 2001, pp. 43-51.
2. P. Stone et al., "Attac-2000: An Adaptive Autonomous Bidding Agent," to be published in *Proc. Fifth Int'l Conf. Autonomous Agents*, 2001.
3. P. Stone, *Layered Learning in Multiagent Systems: A Winning Approach to Robotic Soccer*, MIT Press, Cambridge, Mass., 2000.
4. J. Boyan and A.. Greenwald, "RoxyBot: A Dynamic Bidding Agent for Simultaneous Auctions," available at http://www.cs.brown.edu/people/amygreen/ (current Mar. 2001).
5. A. Roth and A. Ockenfels, "Late Minute Bidding and the Rules for Ending Second-Price Auctions: Theory and Evidence from a Natural Experiment on the Internet," working paper, Harvard University, Cambridge, Mass., 2000.
6. D. Cliff and J. Bruten, *Zero is Not Enough: On the Lower Limit of Agent Intelligence for Continuous Double Auction Markets*, tech. report HPL-97-141, Hewlett-Packard, Bristol, UK, 1997.
7. M.H. Rothkopf, A. Pekefic, and R.M. Harstad, "Computationally Manageable Combinatorial Auctions," *Management Science*, vol. 44, no. 8, 1998, pp. 1131-1147.
8. R.J. Weber, "Making More from Less: Strategic Demand Reduction in the FCC Spectrum Auctions," 1996, available online at http://www.kellogg.nwu.edu/faculty/weber/PAPERS/pcs_auc.htm (current Mar. 2001).
9. P.C. Cramton, "The FCC Spectrum Auctions: An Early Assessment," *J. Economics and Management Strategy*, vol. 6, no. 3, 1997, pp. 431-495.
10. J.A. Csirik et al., "FAucS: An FCC Spectrum Auction Simulator for Autonomous Bidding Agents," submitted to *Proc. 17th Int'l Conf. Artificial Intelligence*, 2001; available at http://www.research.att.com/~pstone/papers.html.

Amy Greenwald is an assistant professor of computer science at Brown University in Providence, Rhode Island. Her primary area of interest is multiagent learning on the Internet, which she approaches using game-theoretic models of computational interactions. She completed her PhD at the Courant Institute of Mathematical Sciences of New York University, where she received the Janet Fabri Memorial Prize for a doctoral dissertation of exceptional quality.

Peter Stone is a senior technical staff member in the Artificial Intelligence Principles Research Department at AT&T Labs Research where he investigates multiagent learning. He received a PhD in 1998 and an MS in 1995 from Carnegie Mellon University, both in computer science. He received his BS in mathematics from the University of Chicago in 1993. Stone's research interests include planning and machine learning, particularly in multiagent systems. Stone is the author of *Layered Learning in Multiagent Systems: A Winning Approach to Robotic Soccer* (MIT Press, 2000) and was awarded the Allen Newell Medal for Excellence in Research in 1997.

Readers can contact Greenwald at the Department of Computer Science, Brown University, Box 1910, Providence, RI 02912, amygreen@cs.brown.edu; or Stone at AT&T Labs–Research, 180 Park Ave., Room A273, Florham Park, NJ 07932, pstone@research.att.com.