The First International Trading Agent Competition: Autonomous Bidding Agents

Edited by *

Peter Stone

AT&T Research, 180 Park Ave., Room A273, Florham Park, NJ 07932
E-mail: pstone@research.att.com

Amy Greenwald

Department of Computer Science, Brown University, Box 1910, Providence, RI 02912
E-mail: amygreen@cs.brown.edu

This article summarizes the bidding algorithms developed for the on-line Trading Agent Competition held in July, 2000 in Boston. At its heart, the article describes 12 of the 22 agent strategies in terms of (i) bidding strategy, (ii) allocation strategy, (iii) special approaches, and (iv) team motivations. The common and distinctive features of these agent strategies are highlighted. In addition, experimental results are presented that give some insights as to why the top-scoring agents' strategies were most effective.

Keywords: automated trading agent, auctionbot, e-commerce, multiagent systems

1. Introduction

The first international Trading Agent Competition (TAC-2000) challenged its entrants to design an automated trading agent that was capable of bidding in simultaneous on-line auctions for complementary and substitutable goods [13]. A TAC agent is a simulated travel agent whose task is to organize itineraries for a group of clients who wish to travel from TACTown to Boston and back again during a five-day period in July.[1] Travel goods, such as airline tickets and hotel reservations, are complementary, and tickets to entertainment events, such as the Boston Red Sox and the Boston Symphony Orchestra, are substitutable. The trading agent's objective is to win items that best satisfy its clients' preferences as inexpensively as possible. The goals of the tournament included providing a benchmark problem in the complex domain of e-marketplaces, and motivating researchers to apply unique approaches to a common task.

* This article is the result of the efforts of many people. The agent descriptions were originally written by team members as listed in Table 1 and the appendix.

[1] The TAC workshop was held at ICMAS '00 in Boston in July, 2000.

This article reports on the competition from the participants' perspective. It describes both the task-specific details of, and the general motivations behind, 12 of the 22 competing agents (see Table 1).[2] The participants' motivations illuminate the general applicability of the TAC setup and its relevance to today's research agendas. We present the task-specific details of the particular agent strategies as instantiations of the participants' varied research agendas. Agent designs are reported in terms of (i) bidding strategy, (ii) allocation strategy, (iii) special approaches, and (iv) team motivations.

Agent	Country	Designers
`ATTac`**	USA	Peter Stone, Michael Littman, Satinder Singh, Michael Kearns
`RoxyBot`**	USA	Justin Boyan, Amy Greenwald
`Aster`**	USA	Andrew Goldberg, Umesh Maheshwari
`UmbcTAC`**	USA	Youyong Zou
`ALTA`**	Russia	Andrey Tarkhov, Dmitry Uspensky, Eugene Vostroknautov
`DAIHard`**	USA	Rajatish Mukherjee, Partha Dutta, Sandip Sen
`RiskPro`**	Sweden	Magnus Boman, Sven-Erik Ceedigh
`T1`**	Sweden	Lars Olsson, Erik Aurell, Lars Rasmusson, Martin Aronsson, Per Larsson, Glenn Lawer
`Nidsia`*	Switzerland	Nicoletta Fornara, Luca Maria Gambardella, Marco Colombetti
`EZAgent`*	USA	Betsy Strother
`UATrader`	USA	Daniel Zeng, Jiang Zhu, Bart Wilson
`EPFLAgent`	Switzerland	Omar Belakhdar, Patrice Jaton, Boi Faltings

Table 1
The TAC agents represented in this article, their countries, and their designers. Semi-finalists are indicated by an asterisk (*), finalists by two asterisks (**).

The results of the TAC competition are shown in Figure 1. The first graph depicts the scores in qualifying rounds (90 games, with the lowest 10 scores dropped), and the second graph depicts the scores on competition day (13 games). Throughout the qualifying rounds and up until the finals, the agents were continually changing as development proceeded. The agent descriptions that appear in this article reflect their characteristics during the final round. As the graphs illustrate, the four top-scoring teams in the final round, `ATTac`, `RoxyBot`, `Aster`, and `UMBCTac`, finished in a statistical tie.

[2] All participants were invited to contribute; those who chose to do so are represented herein.

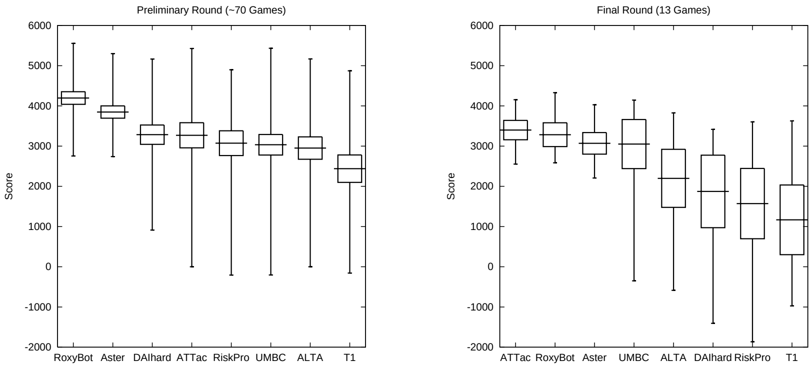

Figure 1. (a) Preliminary Round (90 games, lowest ten scores dropped): horizontal lines indicate mean, minimum, and maximum scores; box delimits 95% Confidence Interval. (b) Final Round (13 games, no scores dropped): horizontal lines indicate mean, minimum, and maximum scores; box delimits 95% Confidence Interval.

This article is organized as follows. Section 2 summarizes the intricacies of the TAC domain. In Section 3, we highlight the common and contrasting approaches taken in the general strategic design of TAC agents. The heart of this article is found in Section 4, which contains the details of the agent strategies. Following the individual agents' descriptions, Section 5 includes experimental results that are intended to give insights as to why the winning agents' strategies were most effective. We conclude in Section 6 with participants' suggestions as to how the design of future trading agent competitions might be improved.

2. Market Game

A TAC agent is a simulated travel agent whose task is to organize itineraries for a group of clients who wish to travel from TACTown to Boston and back again during a five day period. Travel and entertainment goods are traded at simultaneous auctions that run for fifteen minutes. An agent's objective is to secure the goods necessary to satisfy the particular desires of its clients, but to do so as inexpensively as possible. An agent's score is the difference between the utilities it earns for its clients and the agent's expenditures. In this section, we summarize the design of a TAC game instance.

2.1. Supply

The market supply consists of three types of travel goods: (i) flights to and from Boston, (ii) hotel room reservations at two competing hotels, namely, the Grand Hotel and Le Fleabag Inn, and (iii) entertainment tickets for the Boston Red Sox, the Boston Symphony, and Phantom of the Opera. There is a separate auction corresponding to every combination of travel good and day, yielding twenty-eight auctions in total: eight flight auctions (there are no inbound flights on the fifth day, and there are no outbound flights on the first day), eight hotel auctions (two hotel types and four nights), and twelve entertainment ticket auctions (three entertainment event types and four nights). All twenty-eight auctions are *simultaneous*. The rules of the various auctions are as follows:

- An infinite supply of flights[3] is sold by the "TAC seller", a specially designated supplier, at continuously clearing auctions in which prices follow a random walk. Prices are initialized between \$250 and \$400, and perturbed every 30–40 seconds by a random value uniformly selected in the range [\$-10, \$10], but confined within the bounds of \$150 and \$600. No resale of flights is permitted.
- The TAC seller also makes available sixteen hotel rooms per hotel per night, which are sold at open-cry, ascending, multi-unit, sixteenth-price auctions. In other words, the winning bidders are those who bid among the top sixteen, and these bidders uniformly pay the sixteenth-highest price. Transactions clear when the auctions close, which typically occurs at the end of a game instance, although these auctions are subject to early closing after random periods of inactivity. No bid withdrawal or resale in hotel auctions is permitted.
- Entertainment tickets are traded among TAC agents in continuous double auctions, where agents can act as either buyers or sellers, and transactions clear continuously. Each agent receives an initial endowment of tickets for each event on each night—zero with probability 1/4, one with probability 1/2, and two with probability 1/4. Ticket resale is permitted.

2.2. Demand

A TAC game instance pits eight trading agents against one another, with each agent representing eight clients. The market demand is determined by the sixty-four clients' preferences. Each client is characterized by a random set of preferences for ideal arrival and departure dates (IAD and IDD, which range over days 1 through 4 and 2 through 5, respectively), a grand hotel room reservation value (HV, which takes integer values between 50 and 150), and reservation values for each of the three types of entertainment events (RV, SV, and TV—integers between 0 and 200—for Red Sox, symphony, and theater, respectively). A sample set of preferences appears in Table 2; these preferences were those of the clients assigned to `ATTac` during game 3070 of the competition.

The job of each TAC agent is to assemble a feasible package of goods for each of its clients. A *package* is characterized by arrival and departure dates (AD and DD, respectively, ranging over days 1 through 5), a hotel type (H, which takes on value G for Grand Hotel or F for Le Fleabag Inn), and entertainment tickets ($\mathrm{I}(j,k)$ is an indicator variable that represents whether or not the package

[3] Technically speaking, the flight "auctions" are not auctions, but they are handled by the auction server in a way that is similar to how it handles the other auctions.

Client	IAD	IDD	HV	RV	SV	TV
1	2	5	73	175	34	24
2	1	3	125	113	124	57
3	4	5	73	157	12	177
4	1	2	102	50	67	49
5	1	3	75	12	135	1110
6	2	4	86	197	8	59
7	1	5	90	56	197	162
8	1	3	50	79	92	136

Table 2
`ATTac`'s client preferences in game 3070.

includes a ticket on night j to event $k \in \{r, s, t\}$; we also write R1, for example, to indicate that the package includes a Boston Red Sox ticket on night 1). In order to obtain positive utility for a client, an agent must construct a *feasible* package for that client; otherwise, the client's utility is zero. A feasible package is one in which (i) the arrival date is strictly less than the departure date, (ii) the same hotel is reserved during all intermediate nights, (iii) at most one entertainment event per night is included, and (iv) at most one of each type of entertainment ticket is included. Given a feasible package, a client's utility for that package is calculated as follows:

$$\text{utility} = 1000 - \text{travelPenalty} + \text{hotelBonus} + \text{funBonus} \tag{1}$$

where

$$\begin{aligned}
\text{travelPenalty} &= 100(|\text{IAD} - \text{AD}| + |\text{IDD} - \text{DD}|) \\
\text{hotelBonus} &= \begin{cases} \text{HV} & \text{if H} = \text{G} \\ 0 & \text{otherwise} \end{cases} \\
\text{funBonus} &= \textstyle\sum_j \left[\text{I}(j,r)\text{RV} + \text{I}(j,s)\text{SV} + \text{I}(j,t)\text{TV}\right]
\end{aligned}$$

At the end of a TAC game instance, the agents had 4 minutes to report the final allocation of their goods to their clients; otherwise, the TAC server produced a default allocation in a greedy fashion. The final set of goods acquired by `ATTac` in game 3070 is listed in Table 3. Given the client preferences in Table 2 and the goods in Table 3, `ATTac` allocated goods to clients as shown in Table 4.

3. General Strategies

In this section, we summarize the common features of the agents described in this article, outline a range of differences among them, and give a brief high-level overview of each agent's particular focus.

Good	Day1	Day2	Day3	Day4	Day5
R	1	1	1	2	–
S	1	1	0	0	–
T	1	0	1	1	–
G	4	1	0	0	–
F	1	2	3	3	–
I	5	2	1	0	–
O	–	4	1	0	3

Table 3
ATTac's final set of goods in game 3070. R, S, and T denote tickets to the Red Sox, symphony, and theater, respectively; G and F denote the Grand Hotel and Le FleaBag Inn, respectively; I and O denote inbound and outbound flights, respectively.

Client	AD	DD	H	Tickets	Utility
1	2	5	F	R4	1175
2	1	2	G	R1	1138
3	3	5	F	T3, R4	1234
4	1	2	G	—	1102
5	1	2	G	S1	1110
6	2	3	G	R2	1183
7	1	5	F	S2, R3, T4	1415
8	1	2	G	T1	1086

Table 4
ATTac's final allocation in game 3070. In this case, no goods were left unallocated, and the total utility is 9443. During the game, the agent spent \$5364 acquiring goods and earned \$75 selling entertainment tickets, leading to a final score of $9443 - 5364 + 75 = 4154$.

3.1. Common Approaches

The basic decisions that comprise TAC agents' inner bidding loops are listed in Table 5. The details of making decisions 1, 2, and 3—on what goods to bid, for how many of each good to bid, and at what price to bid—are postponed until the specific agent strategies are described. The strategic timing of the placement of bids, however, exhibited common features across agents, and is described presently.

The timing aspects of TAC agents' bids in hotel auctions is particularly intriguing. By virtue of the design of the TAC auctions, the supply of flights is *unlimited* and their prices are *predictable* (the expected future price equals the current price, unless the current price is the lower bound on flight prices), while the supply of hotel rooms is *limited*, and their prices are *unpredictable*. Given the risks associated with the hotel auctions, together with their importance in securing feasible travel packages, hotels were the most hotly contested items during the TAC competition.

REPEAT
1. Decide on what goods to bid
2. Decide at what price to bid
3. Decide for how many to bid
4. Decide at what time to bid
UNTIL game over

Table 5
High-level overview of TAC agents' bidding decisions.

Recall that TAC hotel auctions are ascending (English) mth-price auctions subject to random closing times given sufficient levels of inactivity. Most TAC agents refrained from bidding for hotels early on, unless (i) the ask price had not changed recently, implying that the auction might close early, or (ii) the ask price was very low, in the hopes of being one of the winning bidders should the auction indeed close early. Ultimately, the most aggressive hotel bidding took place at the "witching hour"—in the final few moments of the game—although precisely when was determined by each agent individually. More often than not *TAC hotel auctions reduced to mth price sealed-bid auctions.*

Not only were final hotel prices unpredictable, they often skyrocketed (see Figure 2). 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.[4] During 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 $\sim$ \$1000, always winning the hotels on which they bid, but paying far less than their bids. Having observed a dominant strategy during the preliminary rounds, most agents adopted this high-bidding strategy during the actual competition. The result: many negative scores, as there were often more than m high bids.

For example, a one-night package in which the hotel room is purchased at the value of its marginal utility yields a negative score equal to the price of flights and entertainment tickets; but an agent cannot do better than to bid its marginal utility, since bidding any lower and therefore not purchasing the hotel room yields precisely the same negative utility,[5] whereas bidding any higher could potentially yield an even more negative score. In the final competition, the top-scoring TAC agents were those who not only bid aggressively on hotels, but who

[4] As stated, this observation holds only when the length of stay is exactly one night; for longer stays it relies on the further assumption that all other hotel rooms in the package are secured.

[5] Technically, this claim is not true of an mth price auction of m goods, although it is true of an $m+1$st price auction of m goods. Thus, in the case of TAC hotel auctions, the claim only holds true so long as the bid in question is *not* the mth highest bid, or the difference between the mth highest and the $m+1$st highest bids is sufficiently small.

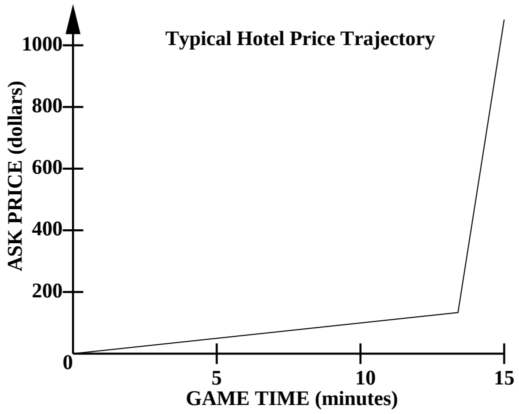

Figure 2. A typical hotel price trajectory. The price increases gradually until near the end of the game, at which point it skyrockets.

also incorporated risk and portfolio management into their strategy in order to reduce the likelihood of buying highly-demanded and highly-priced hotel rooms.

In the flight auctions, if we ignore the possibility of prices reaching the upper or lower bounds of their range, 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. Thus, given the simultaneous auction design, there is no incentive to bid on airline tickets before the witching hour, since by waiting there is some chance of obtaining information about hotel room and entertainment ticket acquisitions. There are, however, substantial risks associated with delaying the submission of bids. These risks arise from unpredictable network and server delays, which sometimes have the undesirable effect of causing bids placed before the end of a game instance to be received after the game's end.

In order 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, say l, and then place its flight bids as soon as game time exceeds $900 - 2l$ seconds. A more risk-seeking agent might place its flight bids after $900 - m$ seconds, where m is the minimum length of its five most recent bidding cycles. In practice, flight bids were placed anywhere from 5 minutes to 30 seconds before the end of the game. Recall that flight auctions are such that agents who place a winning bid pay not their bid, but rather the current ask price. Thus, at the time of their decision, most agents bid above the current price—often, agents bid the maximum possible price, namely \$600—to ensure that these bids, which were placed at critical moments, would not be rejected because of information delays resulting from network asynchrony.

3.2. Contrasting Approaches

TAC agents' bidding strategies differ most substantially in the realm of entertainment ticket auctions. While some agents focus on obtaining complete packages, others make bidding decisions on travel packages alone (*i.e.*, flights and hotel rooms) without regard for entertainment packages, essentially breaking the TAC problem down into two sub-problems, and then solving greedily. Although simpler to implement, the greedy strategy 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 this event exceeds the cost of the ticket and an additional hotel room plus any travel penalties incurred. Similarly, it is sometimes preferable to sell entertainment tickets and shorten a client's stay accordingly.

A further strategic dichotomy in TAC-agent design principles is evident in the methods by which agents *allocate* purchased goods to their clients.[6] Some agents focus on satisfying each of their clients in turn, whereas others make global decisions regarding all their clients' interests simultaneously. This aspect of an agent's design is relevant to both agent's on-going decisions as to what to bid on, and to the final allocation of purchased goods to clients at the end of the game. The top two teams' bidding strategies considered the TAC problem from a global perspective, but most of the other agents used the greedy method of satisfying each of their clients in turn. (One notable exception is the third-highest-scoring agent, which used a local search algorithm that considers clients in pairs.) Once again, the greedy approach is suboptimal, as the following example demonstrates.

Example 3.1. Consider two clients, A and B, with identical travel preferences, and the following entertainment preferences: A values the symphony at \$90 and the theater at \$80; B values the symphony at \$175 and the theater at \$150. Suppose each client is to be in town on one and the same night, and that the agent has one entertainment ticket of each type. An agent using a greedy approach who considers client A before client B will assign A the ticket to the symphony and B the ticket to the theater, obtaining an overall utility of \$140 rather than the optimal utility of \$155, which corresponds to the reverse assignment. □

The percentage of optimal allocations reported by each agent during the competition, as computed by the TAC organizing team, is listed in Table 6.

[6] The general allocation problem (*i.e.*, allocation without TAC's feasible package constraints) is NP-hard, as it is equivalent to winner determination [3], which in turn is equivalent to the weighted set-packing problem [8].

Agent	Aggregate	Minimum	Number
ATTac	100.0%	100.0%	13/13
RoxyBot	100.0%	100.0%	13/13
Aster	99.6%	98.0%	9/13
UmbcTAC	99.4%	94.5%	7/13
T1	98.8%	88.8%	7/13
DAIHard	98.3%	95.1%	1/13
ALTA	97.1%	90.4%	2/13
RiskPro	96.7%	88.1%	1/13
Betsy	98.2%	96.2%	0/6
nidsia	95.0%	80.7%	0/6
gekko	92.2%	54.8%	1/6
kuis	85.7%	73.4%	0/6

Table 6
The semi-finalists listed in order of their effectiveness in optimizing final allocations during the competition. The first measure ("Aggregate") is the percentage of the optimal utility (ignoring expenditures) achieved with the reported allocation, aggregated over the 13 games each of the top 8 agents played, and the 6 games each of the bottom 4 played. The second measure ("Minimum") is the minimum among these aggregated values. The third measure ("Number") is the number of times the agent reported an optimal allocation.

4. Individual Strategies

This section describes in detail the agents' particular bidding and allocation strategies, and any special approaches and motivations. To guide the reader, we first provide one sentence descriptions of each agent's approach. The numbers beside each sentence correspond to the subsection that describes that agent.

1. The key to `ATTac`'s success is its built-in adaptability, giving it the flexibility to cope with a wide variety of scenarios during the competition.

2. `RoxyBot` optimally solves the problems of *allocation*—assigning resources to clients—and *completion*—determining what quantity of each resource to buy and sell—using an innovative data structure called a *priceline*.

3. `Aster` is an agent that is neither strictly greedy, nor strictly optimal; instead its designers' goal was to create an agent whose performance would be scalable, since they expect many situations of practical interest to be more complex and less structured than TAC.

4. `UmbcTAC` conserves network bandwidth by being sensitive to network load and adapting the number of bids it places accordingly; on average, this agent updates its bidding data every 4–6 seconds, providing a significant advantage over the reported 8–20 second delays experienced by competing agents.

5. `ALTA`, the *Artificial Life Trading Agent*, uses a search/allocation strategy based on genetic algorithms.

6. `DAIHard` was developed by a research group that is interested in empirically studying, from the buyer's perspective, the utility of participating in one or more auctions of varying types in which similar goods are sold.
7. `RiskPro` is designed as a risk-averse, rather than a risk-seeking, agent, equipped with security levels and threshold values that lead to decisions consistent with the risk attitude of the decision maker.
8. `T1` was the unique entrant that came out of a development effort that created a set of collusive agents designed to investigate the power of collusion in TAC.
9. `Nidsia`'s team is researching the design of algorithms that construct optimal bidding policies in combinatorial auctions for complementary and substitutable goods.
10. `EZAgent` was designed to perform effectively using simple heuristics.
11. The general bidding strategy employed by `UATrader` can be characterized as a "myopic" trading strategy with iterative adjustments based on neighborhood search.
12. `EPFLAgent` represents a distributed solution to the TAC game; it uses the CSP formalism and caching to anticipate its decision-making needs which it outsources to its Solver.

4.1. ATTac

`ATTac` placed first in TAC using a principled bidding strategy, which includes several elements of *adaptivity*. This adaptivity gave `ATTac` the flexibility to cope with a wide variety of possible scenarios during the competition. The design of `ATTac` was motivated by the multiagent learning research interests of its developers.

Bidding

At every bidding opportunity, `ATTac` begins by computing the most profitable allocation of goods to clients (which we shall denote G^*), given the goods that are currently owned and the current prices of hotels and flights. (As specified under *On-Line Adaptation*, `ATTac` actually uses *predicted* closing prices of the hotels based on the results of previous game instances.) For the purposes of this computation, `ATTac` allocates, but does not consider buying or selling, entertainment tickets. In most cases, G^* is computed optimally using mixed integer linear programming, as described under *Allocation*.

`ATTac` bids in two different modes: *passive* and *active*. The *passive* mode, which lasts until the witching hour, is designed to keep as many options open as possible. During the passive mode, `ATTac` computes the average time it takes for it to compute and place its bids, T_b. Call the time left in the game T_l. When $T_l \leq 2 \times T_b$, `ATTac` switches to its *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 2 bidding iterations in active mode.

Based on the current G^*, its current mode, and T_l, ATTac bids for flights, hotel rooms, and entertainment tickets. Full details of ATTac's strategy are available in [12]. Here we focus on strategies for bidding on entertainment tickets, allocation of goods to clients, and adaptivity.

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 that it currently owns. In all cases, the prices depend on the amount of time left in the game (T_l), becoming 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^*. ATTac offers to sell E for $\min(200, V(E) + \delta)$ where δ decreases linearly from 100 to 20 based on T_l.[7]

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 already scheduled for that type of entertainment in G^*).

Finally, ATTac bids to buy each type of entertainment ticket E (including those that it is also offering to sell) based on the increased value of G^* that would be derived by owning E (*i.e.* 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.

Allocation

As is evident above, ATTac, relies heavily on computing the current most profitable allocation of goods to clients, G^*. Since G^* changes as prices change, ATTac needs to recompute it at every bidding opportunity. By using a mixed-integer linear programming approach, ATTac was able to compute optimal final allocations in every game instance during the tournament finals — one of only 2 entrants to do so (see Table 6).

The mixed-integer LP approach used by ATTac works by specifying 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 is guaranteed to find the optimal allocation, and usually does so in under one second on a 600 MHz Pentium computer.

On-Line Adaptation

ATTac was entered in TAC in large part due to the developers' research interests in multiagent learning, and past success in agent tournaments [11]. Based on the problem description, the domain appeared to be a good candidate for

[7] \$200 is the maximum possible value of E to any client under the TAC parameters.

applying machine learning techniques.

However, TAC was conducted in such a way that it was impossible to determine how much each competitor was bidding in the auctions; only the current ask prices are accessible. This precluded learning detailed models of opponent strategies. `ATTac` instead adapts its behavior on-line in three different ways:

1. `ATTac` decides when to switch from the passive to the active bidding mode based on the observed server latency T_b during the current game instance (see *Bidding* section).

2. `ATTac` adapts its allocation strategy based on the amount of time it takes for the linear program to determine optimal allocations in the current game instance (see *Allocation* section)

3. Perhaps most significantly, `ATTac` is adaptive in its risk-management strategy to account for potentially skyrocketing hotel prices. It was stated above that `ATTac` computes G^* based on the current prices of the hotel rooms. In fact, it uses the predicted closing prices of hotel auctions based on their closing prices in previous games.
   `ATTac` divided the 8 hotel rooms into 4 equivalence classes, exploiting symmetries in the game (due to the uniform distribution of client preferences, hotel rooms on days 1 and 4 should be equally in demand as should rooms on days 2 and 3), assigned priors to the expected closing prices of these rooms, and then adjusted these priors based on the observed closing prices 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 all of its allocation values.
   Empirical testing indicates that this strategy is extremely beneficial in situations in which hotel prices do indeed escalate, while it does not lead to significantly degraded performance when they do not [12]. Indeed, `ATTac` performed as well as the other top-finishing teams in the early TAC games when hotel prices (surprisingly) stayed low, and then out-performed the competitors in the final games of the tournament when hotel prices rose to high levels.

4.2. RoxyBot

`RoxyBot`'s algorithmic core is based on AI heuristic search techniques and approximates optimal behavior. In particular, `RoxyBot` incorporates an optimal solver for the problem of *allocation*—assigning purchased resources to clients at the end of the game so as to maximize total utility—and an approximately optimal solver for the more general problem of *completion*—finding the optimal quantity of each resource to buy and sell given current holdings and forecast prices. This section describes the formulations of and solutions to these two problems in the context of `RoxyBot`'s overall strategy. The formulation of the completion problem involves a novel data structure called a *priceline*, which is designed to handle (estimated) closing prices, (estimated) supply and demand, sunk costs, hedging, and arbitrage in a unified way. RoxyBot's high-level strategy is outlined in Table 7; full details are available in Boyan and Greenwald [6].

(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*

Table 7
RoxyBot's high-level strategy.

Allocation

We first describe the allocator, although it runs at the end of a game instance, since it helps to motivate the completer algorithm that is 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, an optimal solution based on A^* search is tractable for the dimensions of TAC.[8] Indeed, using an intricate series of admissible heuristics, RoxyBot managed to prune down the search tree of possible optimal allocations from roughly 10^{20} to 10^3 or 10^4 possibilities, resulting in provably optimal allocations typically being discovered in just half of a second. RoxyBot produced optimal allocations in 100% 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 decisions of which legal travel package—*i.e.*, 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 decisions of which entertainment package—*i.e.*, sets of entertainment tickets of different types on different days—to assign to clients 1 through 8. There are 73 entertain-

[8] For descriptions of standard AI search techniques, including A^* search and beam search, see [9].

ment packages, though many of these are infeasible due to earlier assignments of travel packages. The heuristics compute an upper bound on a quantity—*e.g.*, the maximum possible number of legal packages using good hotels, or arriving on day 3—and 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 instance enable these heuristics to be computed very quickly.

Completion

The *completer* that runs during each bidding cycle is the heart of `RoxyBot`'s strategy. Its aim is to determine the optimal quantity of each resource to buy and sell, given current holdings and forecast closing prices. Like the allocator, it considers all travel resources from a global perspective, and makes integrated decisions about which hotels and flights to bid for, which entertainment tickets to buy and sell, and how many of each. Unlike the allocator, the completer faces the added complexity that the resources being assigned may not yet be in hand; they may still need to be purchased at auction. Furthermore, in the case of entertainment tickets, resources which are in hand might be more profitably sold on the market than allocated to `RoxyBot`'s own clients.

To reason about the resource tradeoffs involved, `RoxyBot`'s completer relies on a data structure called a *priceline* for each resource, which transparently handles either one-sided or double-sided auctions, short-selling of resources, hedging, and both limited and unlimited supply and demand. Using this construction, the completer's task is much simplified: a package's cost is computed by popping off the leading prices from the corresponding pricelines. The value of a package to a client equals the client's utility for that package less 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: *i.e.*, how to "complete" the current set of holdings by transforming it into an optimal set of holdings by game end. 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, non-admissible heuristic, and a variable-width beam search over the same search space, in practice `RoxyBot` usually found an optimal completion within about 3 seconds of search. Therefore, during the competition, `RoxyBot` used beam search rather than provably optimal A^* search.

Estimation

`RoxyBot`'s pricelines are data structures in which to describe the costs of market resources. In auctions such as those fundamental to the TAC setup, however, 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). Inspired by its creators' primary research interest, RoxyBot was designed to use machine learning techniques to produce these estimates. However, the final round of the TAC competition was both too short and too different (due to changing agent strategies) from the preliminary rounds in order 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 [4]. 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.

4.3. Aster

Designed by members of the Strategic Technologies and Architectural Research (STAR) Laboratory at InterTrust Technologies Corp., Aster finished third on competition day. Aster's framework for cost estimation is flexible and can be tuned to 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 of the desired resources. After all auctions close, Aster runs a sophisticated algorithm to compute the final allocation.

Estimating Costs

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 observes the principle that sunk costs are no costs. For flights, the estimated cost is zero for tickets that are currently held, and the current ask price for additional tickets.

For hotels, estimating costs is tricky because both price and holdings are uncertain until an auction closes. Aster predicts the closing price for a hotel room by linearly extrapolating previous ask prices based on current time. This extrapolated price is then adjusted as follows: For rooms for which Aster holds hypothetical winnings, 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 only provides one bid and ask quote per entertainment ticket, Aster assumes the cost of buying an additional ticket is the current ask price, and that 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.

Tentative Allocation

On each iteration, `Aster` computes a tentative allocation of resources using a local search algorithm that considers pairs of clients in turn, given estimated costs and current holdings. It starts with a null allocation of resources to all clients. Then it iterates over all pairs of clients, deallocating their current resources and allocating new resources so as 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.

Bidding

`Aster` bids using one of two strategies, depending on whether the stage of the game is pre-commit (before the witching hour) or committed. Like `ATTac`, `Aster` initiates its committed 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 committed stage. During the pre-commit stage, `Aster` does not bid on flights; during the committed stage, `Aster` places all necessary flight bids to achieve the current allocation.

In the pre-commit stage, `Aster` places limited bids on hotel rooms in the hopes of capturing early closings. At the same time, `Aster` tries to avoid engaging in price hikes by placing bids at the minimum allowable increment, namely 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 having purchased unnecessarily. If it were to bid for more than two nights, and the price on a middle night were to shoot up, it could get stuck with an extra 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 is equal to the utility due to that client.

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

Final Allocation

`Aster` uses heuristic search to compute its final allocation. It searches a tree consisting of all possible travel packages (*i.e.*, arrival dates, departure dates, and hotel types) for all clients to compute the globally optimal allocation of its travel goods to its 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 need not be optimal over all clients viewed from the global perspective. Thus, after this first search terminates, `Aster` starts another search 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 6).

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

4.4. UmbcTAC

`UmbcTAC`, created at University of Maryland Baltimore County, placed fourth in the competition. By being sensitive to network load and adapting to network performance, it received the most frequent updates regarding market prices.

Bidding

`UmbcTAC` maintains the most profitable itinerary for each client individually based on the latest price quotes (as opposed to solving the full 8-client optimization problem). `UmbcTAC` balances strategies that avoid switching travel plans too frequently against strategies that encourage switching travel plans early on:

- When a client's itinerary is changed, the value of the goods that will no longer be needed are subtracted from the value of the new itinerary as a penalty for changing plans. Thus, the client's travel plans will not change unless the gain in profit overrides the value of wasted goods. Subsequently, wasted goods are marked as "free" goods: *i.e.*, they are treated as sunk costs. As such, the price of free goods is set to 0, which encourages their use in the itineraries of other clients.
- `UmbcTAC` only changes a client's travel plans if the profit difference between the new and the old plans exceeds a threshold value (typically between $10 and $100).
- When it is necessary to change a client's travel plan, it is important to do so as early in the game as possible: the earlier the plan changes, the more likely it is that the obsolete bids will either not win, or will win at a low price. `UmbcTAC` risks wasting one good in each bidding iteration in order to ensure that at least one client changes to a better plan. It does so 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 to be 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 (*i.e.*, 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 only bids for airline tickets and raises hotel bids to their limit 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. Other clients can then try to make use of free goods to change travel plans if a greater utility would result.

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. The agent never bids for any entertainment tickets during the last three minutes of the game in order to save network bandwidth. The agent also does not change travel plans during its last two or three bidding opportunities in order to save time. On average, `UmbcTAC` updates its bidding data every 4–6 seconds, providing a significant advantage over the reported 8–20 second delays experienced by competing agents.

4.5. ALTA

`ALTA` is an *Artificial Life Trading Agent*, named after the company of its designers, namely Artificial Life, Inc. `ALTA` is unique in that its search/allocation strategy is based on genetic algorithms.

Bidding strategy

ALTA limits its concern to travel packages, participating only in the ascending auctions for hotel rooms, but ignoring the continuous double auctions (*i.e.*, buying and selling of entertainment tickets) altogether. In this way, the search space is limited to only 20 feasible travel packages, although ALTA did attempt to allocate its endowed entertainment tickets. ALTA uses combinatorial search (plus heuristics) to maximize the expected utility of entertainment ticket allocations.

Like most other agents, ALTA divides the entire duration of the game into two parts. During the first period, current ask prices are dynamically monitored and decisions regarding what to bid for are made (using optimization techniques). ALTA also estimates the final prices of auctions during this phase, and increases its bids based on these predictions, the ask current quote, and cycle time. The point in time at which ALTA converts to its second phase is defined dynamically based on the performance of the AuctionBot, but is intended to guarantee at least three additional iterations.

After the first phase is complete, ALTA ceases any further optimization, freezing the current allocation: days of arrival/departure and hotel type. At this point (*i.e.*, during the second phase), ALTA focuses on ensuring the purchase of the desired goods. In doing so, ALTA bids aggressively, sending off high bids and increasing those bids whenever the desired holdings are not realized. Also during the second phase, ALTA bids the maximum possible amount for flights; this guarantees that flight purchases will be successful regardless of possible last minute price fluctuations. Lastly, it should be noted that in addition to concerns raised earlier about the feasibility of search, ALTA also decides to ignore continuous double auctions simply because it does not commit to an allocation until the second phase.

Allocation Strategy

As mentioned previously, ALTA's allocation strategy is based on the principles of genetic algorithms. As soon as the game ends, ALTA compiles the list of acquired resources, and begins searching for the optimal allocation of its goods among its clients given their preferences. According to the TAC setup, there are 392 feasible packages, plus the null allocation. Each chromosome, therefore, consists of 8 genes (one corresponding to each client), with every gene numbered between 0 to 392. ALTA simulates a population of 500 such chromosomes, with each one representing one possible allocation. Using genetic algorithms, ALTA searches for the optimal allocation by selection, mutation, crossover, and replacement within the population of chromosomes, seeking that which maximizes overall utility. This optimization process converges within the appropriate time frame. Although ALTA rarely reports an optimal allocation, it did find allocations within 97% of the optimal on competition day (see Table 6).

Special Approaches/Motivations

In attempt to optimize the performance of `ALTA`'s communication/transport level, `ALTA` was built in Java, based on the pure auction server API, rather than the C++ bidding agent classes. Another unique feature in the design of `ALTA` was its ability to control its expenditures. Although hotel prices often escalated at the end of the competition games, as other agents engaged in price wars, `ALTA` raised its final bids to an extent, but tried to ensure that its total expenses were below its total expected utility.

The designers of `ALTA` participated in TAC not just for the fun of it, rather because their company, Artificial Life, Inc., implements market and auction-based approaches for tasks such as scheduling and resource allocation for distributed search engines and network management. The designers are also interested studying the market-driven behavior of multiagent systems, in order to gain an understanding of the problems and scenarios of future automated e-commerce systems.

4.6. DAIHard

`DAIHard` was created by the Distributed Artificial Intelligence Group at the University of Tulsa.

Bidding Strategy

`DAIHard` bid for three items for each client, as follows:

Airline tickets: `DAIHard` bids for airline tickets only after a pre-specified time period (a parameter of the agent). This time window is large enough to account for the possibility of having to submit multiple bids because of changing auction prices. Delaying the bidding for airline tickets enables the agent to be more flexible in setting the length of each client's stay.

Hotel rooms: Initially, `DAIHard` bids for both bad and good hotel rooms for the duration of each client's stay. However, if an ask price exceeds a certain value, the agent stops bidding for one of the hotel rooms. In particular, if the difference between the price of the Grand Hotel and the price of Le Fleabag Inn exceeds the good hotel bonus, then DAIHard bids for Le Fleabag Inn; otherwise, DAIhard bids for the Grand Hotel.

There is also a reserve price which the agent does not exceed when it bids for hotel rooms. The reserve price in an English ascending auction is the agent's valuation of the good, which in the case of TAC hotel rooms is a function of airline ticket prices, entertainment ticket prices, and the prices of alternative hotel rooms. For a given hotel room, `DAIHard` computes this value as follows: for all pairs of arrival and departure days,

1. The pre-hotel utility of the package is computed using the formula

```
1000 + Fun Bonus + Hotel Bonus - (Current Flight Cost +
       Entertainment Ticket Cost + Travel Penalty)
```

2. The hotel cost is computed by calculating the sum of the current ask prices over the duration of stay.
3. The difference between 1 and 2 is stored as the package utility.

Now for each pair of arrival and departure days, there is an associated utility. The pair for which this utility is maximum is chosen, and the estimate of the corresponding hotel reserve prices are set at the value obtained in step 1. If all utilities are negative, `DAIHard` stops bidding for the client altogether.

Entertainment tickets: `DAIHard` begins bidding on entertainment tickets at the start of each game instance. The agent never bids above the entertainment reserve value for each client. In a competitive environment, where the objective is to win, rather than to maximize utility and minimize costs, agents may sacrifice a competitive advantage by selling unneeded tickets. Hence, in selling any entertainment tickets, `DAIHard`'s minimum ask price is fixed at \$100.

Allocation Strategy

The problem of finding optimal allocations, in general, is computationally intensive. Nonetheless, `DAIHard` does successfully compute a near-optimal allocation, which it accomplishes as follows:

1. It first allocates its airline tickets depending on each client's set arrival and departure days.
2. It then allocates hotel rooms to each client depending on a stored flag in the client data, indicating good or bad hotel. If the agent is unable to fully accommodate a client for the entire duration of its stay, it puts that client on a waiting list. Similarly, unassigned hotel rooms are stored in a common hotel-room pool – `DAIHard`'s hotel bidding strategy may result in the purchase of extra hotel rooms. After iterating over all its clients, the agent tries to allocate the hotel rooms in the common pool to the clients on the waiting list. Priority is given to clients with shortest stays.
3. `DAIHard` optimally, rather than greedily, allocates entertainment tickets.

Special Approaches/Motivations

TAC is closely related to the other projects of interest to the DAI-HARD research group. In general, the group is interested in developing pro-active, market-aware agents that can educate the on-line consumer about his/her domain of interest (*e.g.*, travel, shopping). This objective involves (i) analyzing market conditions, knowing user preferences, so as to facilitate users taking advantage of fleeting opportunities, and (ii) organizing and presenting the information structure of a domain in such a way as to enable users to formulate effective queries. In addition, the DAI-HARD group is interested in empirically studying, from the buyer's perspective, the utility of participating in one or more auctions of varying types in which similar goods of interest to the user are sold. Participation in

TAC enabled the group to begin to understand the dynamics of on-line auctions among multiple intelligent agents.

4.7. RiskPro

Bidding Strategy

`RiskPro` periodically updates bids for all types of auctions, until the end of the game instance. The expected utility of each client is monitored and compared with the total client debits, assuming that all auctions will clear at the current asking price. `RiskPro` bids in all auctions as long as the expected utility exceeds the expected client debits. Otherwise, the agent withdraws all the bids for the client in question in all auctions that have not already cleared. `RiskPro` sleeps for a predefined number of seconds after completing each bidding round, depending on the amount of time left in the game. In the first half of the game, the sleep time is ten seconds, in the second half five seconds, until three minutes remain when sleep is disabled.

`RiskPro` tries to minimize the risk of buying flight tickets early in the game that will not be of use at the end of the game, due to a lack of hotel rooms. Another reason for postponing the purchase of flights is the possibility of hotel prices inflating at the end of the game. `RiskPro` therefore waits until the last three minutes of the game before submitting flight bids. Since the ask price is updated independently of the submitted bids, `RiskPro` bids well above the ask price in order to ensure winning the goods even if the ask price increases before the next auction clears. As the supply of tickets is unlimited, `RiskPro` needs only submit flight bids once in order to win the flights at the next clear.

By contrast, the strictly limited supply of hotel rooms leads `RiskPro` to bid for both types of hotel simultaneously. `RiskPro` initially submits bids of $1 in auctions for both hotel types, modulo client preferences. The chance of winning goods early and at a low price is hence increased, if it so happens that the demand for one of these days is low, resulting in the auction closing prematurely. At the very last minute of the game, `RiskPro` uses a bid increment of half the asking price, in order to avoid the risk of losing goods to other agents.

The creators of `RiskPro` observed that the outcome of efficiently allocating the entertainment ticket endowment to clients can be more successful than attempting to trade tickets with other agents via the continuous double auctions. At the outset, `RiskPro` allocates tickets to all clients whose preferences exceeds $75, checking for each available ticket that it is allocated to the client with the highest utility for that ticket, and that this client does not already hold such a ticket. The remaining tickets in the endowment are saved for trading, which is initiated halfway into the game. The reason for the delay is to avoid aiding other agents at the beginning of the game, and to hopefully create some disorder by initiating a sudden change in prices in the middle of the game.

Allocation Strategy

`RiskPro` uses a simple allocation strategy which identifies clients with the highest preferences for the goods acquired. The strategy does not involve finding optimal allocations by evaluating the utilities of all possible combinations of goods. Instead it generates an allocation of all available goods by regarding subsets of goods as a single bundle, and searches for bundles according to a satisficing scheme [10]. Using this scheme, the final allocation of goods is independent of `RiskPro`'s information as to which goods were intended for which client.

Special Approaches/Motivations

Two relevant past experiences influenced the design of `RiskPro`. The first is prior work within the research lab (`www.dsv.su.se/DECIDE`) related to decision analysis and risk modeling, and the second was participation in several RoboCup competitions (`www.dsv.su.se/~robocup/`).

Boman, one of `RiskPro`'s designers, is a researcher interested in developing theories, methods, and tools for human decision analysis, some of which have recently been applied to artificial decision makers [1]. His main agenda has been to extend rational choice theory with realistic models and tools for risk management. In brief, his approach introduces general risk constraints [5], which allows for the setting of security levels and threshold values, leading in turn to recommendations of future actions consistent with the risk attitude of the decision maker. The main concern has been low-probability outcomes with catastrophic utility (close to, or equal to $-\infty$).

In the case of TAC, this perspective led to the following observation: a night without a hotel room might be considered a small disaster. In order to mitigate the effects of such disasters, `RiskPro` is a risk-averse agent. Realizing that this approach might adversely affect their standings, `RiskPro`'s developers accepted small performance degradations in order to connect their agent to their research agenda, and perhaps more importantly, to make it suitable to more realistic scenarios. Indeed, a possible future version of TAC might enhance the negative effects of infeasible allocations with catastrophic outcomes, as a game manifestation of, for example, a client suing the trading agent company. `RiskPro`'s code for optimizing with respect to server speed and network load, based on experiences from testing and participating in qualifying rounds, compensated for the non-optimality (at least in part) caused by a devotion to risk mitigation.

4.8. T1

The design of `T1` was a joint project between the Swedish Institute of Computer Science (SICS: `http://www.sics.se/`) and Industrilogik (`http://www.L4i.se/`). `T1` finished eighth in the competition, as the last-place finalist.

Bidding Strategy

T1 is essentially a set of parameterized heuristics optimized for the TAC setup. T1 has three stages of execution: initial bidding, an intermediate loop that updates bids every fifteen seconds and estimates final prices, and a final stage when it determines a tentative allocation and buys flights.

In the initial stage, T1 bids \$5 for 8 rooms in all the hotel auctions, ensuring that it will have already obtained any necessary rooms in the event that an auction closes prematurely. Similarly, it bids \$3 in all entertainment ticket auctions. (Tickets that it buys but turn out not to be useful are later auctioned off for \$110 or more. The \$110 limit was set based on the intuition that low prices could provide the buyer — a competing agent — with more utility than the transaction yields for T1, resulting in a relative net loss.)

In intermediate looping stage, T1 updates its bids every fifteen seconds, by placing a bit at \$10 above the current ask price if the hypothetical quantity won is lower than the desired quantity. Also during this stage, T1 uses current ask prices to estimate final prices using a thresholding heuristic. The price trajectories for the hotel auctions were roughly characterized by T1's designers as falling within one of the following of three categories: steady low prices, linearly increasing prices, or "unlimited" growth. This behavior was captured in a heuristic that estimates final prices using a pair of breakpoints for every hotel auction. If the current ask price is below the first breakpoint, T1 predicts that the price will close at that breakpoint. If the price is above the first breakpoint but below the second, T1 predicts that the price will end at the second breakpoint, unless the calculated linear trend is lower than this point, in which case T1 predicts the price will close at the calculated trend price. If the current price is higher than both breakpoints, T1 predicts according to the calculated trend price. The break points are parameters supplied to the agent at startup; the actual values were tuned based on the price data observed during the qualifying rounds.

With 2 minutes left in the game, T1 changes its focus to trying to avoid hotel room auctions whose prices it predicts will skyrocket. Specifically, T1 does not consider any hotel room in subsequent allocations whose price increase between the two last quotes exceeds a pre-specified amount. At the same time, T1 also estimates the cost of not winning each hotel room, *i.e.*, its marginal utility. This utility is approximated as the average of the inflight prices on the two days before the night in question plus the average of the outflight prices on the two days after that night plus a tuning parameter. The agent adds this calculated utility to the current ask price and places the resulting amount as the final bid in the hotel auctions. With 45 seconds left of the game, T1 buys the necessary flights, assuming it will obtain all the hotel rooms for which it has bid.

Allocation Strategy

Once final prices are estimated, T1 performs a greedy search to decide whether it should buy more of any resources. The agent computes tentative allocations of resources to its clients so to maximize utility, considering each in turn. The allocation process is repeated for all $8! = 40,320$ permutations of client orderings, and the optimal allocation among those considered is then used to decide whether to buy more resources. This search is suboptimal, but the post-competition analysis shows that it performs reasonably well (within 98% of the optimal allocation), with a few catastrophic failures (around 75% of the optimal). During the last 2 minutes of a game instance, the tentative allocation is only performed on a randomly selected subset of permutations of the client orderings.

Motivations and Observations

T1's designers were primarily interested in TAC as a route to understanding the nature of the difficulties that a real world combinatorial trading agent faces, such as the effect of response time, server failures, price wars, and malicious bidding. Initially, the team registered several agents, on the one hand because they were considering several different approaches, but in addition, because they wanted to determine whether the rules of the game favored collusion among agents. They discovered, for example, that one could exploit the fact that agents were given unlimited credit; in particular, one agent could earn billions by selling expensive event tickets to a colluding agent that was deliberately losing millions. Eventually, this type of collusion was disallowed by the organizers.

The design of T1 involved a number of simple heuristics together with a large number of tunable parameters. This approach appears not to have been sufficient for the TAC setup, since it requires the ability to observe several representative games in order to manually tune the parameters. Since the characteristics of the game changed quite drastically between the time of the qualifying rounds and the final competition, T1's parameters were unfortunately not tuned for the appropriate situation.

4.9. Nidsia

The Nidsia team focused their efforts on solving an open research problem, of which TAC provides a particular instantiation. The nature of the TAC setup is such that clients' utility functions dictate the value of complete packages, but the value of any particular good within a package, taken independently, is not always well-defined. Given an auction mechanism and an independent resource valuation, auction theory provides an optimal bidding strategy. Such valuations do not exist in TAC, however, nor do they exist in other combinatorial auctions for complementary and substitutable goods. Thus, the relevant bidding problem in such settings is how to best construct bids for individual goods, which are sold in separate auctions, but are of no value when considered in isolation, and only take on value in conjunction with other relevant goods. Nidsia's approach to this problem, inspired by the paper of Boutilier *et al.* on sequential auctions [2], is to construct an agent bidding policy conditioned on the possible outcomes of its bids.

Bidding Strategy

Nidsia's bidding strategy considers clients in turn; this discussion therefore applies to an arbitrary client. State s_t is a bit vector where each bit $s_{t,i}$ describes Nidsia's current holdings for said client at time t. A bidding policy is a function from states to actions, where an action a is a vector of bids, one per auction, and in general, each bid is a price-quantity pair. Under certain simplifying assumptions, Nidsia computes an optimal bidding policy.

The number of possible bids (and therefore actions) is infinite, if one considers all possible values of price and quantity. To reduce the space of actions to a manageable size, Nidsia only considers bids in which the quantity is 1 and the price for auction i is the ask quote $q_{t,i}$ at time t plus a fixed increment δ. With this simplification, an action a is a bit vector, where $a_i = 1$ if a bid is submitted at price $q_{t,i} + \delta$ and $a_i = 0$ if no bid is submitted.

To further reduce the search space, Nidsia focuses only on auctions for travel goods (*i.e.*, flights and hotels), and therefore, by the nature of the TAC auction mechanisms, is primarily concerned with hotel auctions. For each client, Nidsia computes the expected utility of each of 256 possible actions—each corresponding to whether or not each of the 8 possible hotel rooms is included—given current holdings. Nidsia bids according to the action that maximizes expected utility.

The expected utility $\mathbb{E}[U(s_t, a)]$ of taking action a in state s_t is the sum over all possible states s_{t+1} of the probability $P(s_{t+1}|s_t, a)$ of reaching state s_{t+1} times the utility $V(s_{t+1})$ of state s_{t+1}. The quantity $P(s_{t+1}|s_t, a)$ is computed as the product of the probability of the outcomes of the bids described by the action a, taken in state s_t, that lead to state s_{t+1}. This formulation assumes that the probability distributions among the various auctions are independent.

The probability of obtaining item i is assumed to be near 0 at the beginning of the game and near 1 at the end of the game. For the purposes of TAC, these probabilities were given by the equation for the straight line $F(t) = mt + b$ that satisfies the conditions $F(1) = 0.1$ and $F(15) = 1$; that is, $m = 0.9/14$ and $b = 0.1 - m$ (recall that 15 is the number of minutes in a TAC game). The probability of failing to obtain item i at time t is simply $1 - F(t, i)$.

The utility $V(s_t)$ at state s_t is taken to be the reward $r(s_t)$ for being in state s_t less the cost $c(s_t)$ of obtaining the items held in this state: *i.e.*, $V(s_t) = r(s_t) - c(s_t)$. The cost $c(s_t) = \sum_i s_{t,i} c_{t,i}(h)$, where $c_{t,i}(h) = 0$ if Nidsia owns hotel i at time t, and $c_{t,i}(h) = q_{t,i}$ otherwise. The reward $r(s_t)$ is taken to be the maximum possible value attainable among all feasible packages that include the hotels indicated by bit vector s_t. Formally,

$$\mathbb{E}[U(s_t, a)] = \sum_{s_{t+1}} P(s_{t+1}|s_t, a)\, V(s_{t+1})$$

$$P\left(s_{t+1}|a,s_t\right) = \prod_i P\left(s_{t+1,i}\right)$$
$$P\left(s_{t+1,i}\right) = s_{t+1,i}F\left(t+1,i\right) + \left(1-s_{t+1,i}\right)\left(1-F\left(t+1,i\right)\right)$$

Allocation Strategy

Due to time constraints during development, `Nidsia` allocates its goods to clients according to a fixed heuristic, rather than computing optimal allocations (using *e.g.*, integer programming). One minute before the end of the game, `Nidsia` bids on flights that coincide with the hotel room auctions that it expects to win for each client. Also at this time, the initial endowment of entertainment tickets is greedily allocated to clients. Unused tickets are auctioned off at the current bid-quote price, and useful tickets that are currently on sale are purchased at the current ask-quote. At the end of the game, `Nidsia` confirms that its clients have all the necessary goods to complete their travel, and it heuristically tries to allocate any unused goods so as to satisfy as many clients as possible.

Special Approaches/Motivations

The approach taken by Nidsia's developers to the TAC competition incorporated aspects of their more general research agenda, including the creation of techniques for computing optimal bidding policies. The Nidsia algorithm is not tailored to the particular auction mechanisms of TAC; rather it is more general in its applicability to any combinatorial auctions of substitutable and complementary goods. As a result, the implementation of Nidsia's TAC agent required some strong simplifying assumptions. Nonetheless, Nidsia's overall performance illustrates the promise of this general method. Had Nidsia's general algorithms been tailored to the specific TAC setup, Nidsia's performance would have improved.

4.10. EZAgent

`EZAgent` was one of the 12 TAC semi-finalists. The objective of its designer was to obtain positive utility with minimum coding.

Bidding Strategy

`EZAgent` enforces that all clients travel on the days that they prefer. No analysis was performed to see if changing the travel days could increase the utility. (This decision turned out to be a costly mistake, as often it is more beneficial to travel on different days to maximize utility.)

Given each client's travel days, `EZAgent` calculates how much utility it could obtain by purchasing the Grand Hotel for a client and compares this value with the current cost of rooms at the Grand Hotel. If it obtains positive utility from the Grand Hotel, the agent bids on the Grand Hotel. If the utility is less than the current cost, the agent bids on Le FleaBag Inn. If Le FleaBag Inn costs more than \$300, the agent opts not to acquire travel arrangements for this client.

During the last minute of the game, EZAgent does not change its hotel preference since doing so could lead to acquiring multiple accommodations.

EZAgent updates bids continuously until a game is complete. If it wants a hotel and does not have a winning bid, EZAgent bids $10 above the current ask price. An analysis of the executions during the final round showed that it bid approximately every 10 seconds. Since an unknown period of inactivity could result in an early closing of a hotel auction, no delay was inserted in between bidding cycles.

During the last five minutes of the game, the agent acquires airline tickets for any travelers for whom it has obtained or is still trying to obtain hotel arrangements. Also during this time, entertainment tickets are auctioned off. The ask price is set at a small increment over the value of the ticket to the client to whom it is currently allocated. A ticket is bought if it is offered at less than the utility that would be gained by assigning it to a client.

Allocation Strategy

After the last auction closes, EZAgent first distributes its hotel rooms among its clients. Clients with higher good hotel values are given preference for the Grand Hotel. If any rooms at Le FleaBag Inn were obtained, they are allocated to clients as yet without rooms according to their potential entertainment bonus. After allocating hotel rooms, the entertainment preferences of those clients with accommodations are ordered. Tickets are distributed in a descending manner until all possible tickets are allocated. This does not obtain an optimal allocation, but generally performs well. In a review of the games played, EZAgent averaged 98% optimal resource allocation (see Table 6). Since the allocation mechanism only considers one possible allocation of resources, it is able to quickly distribute its resources.

Special Approaches

EZAgent attempted to obtain positive utility with minimum coding. EZAgent did not use an optimal strategy, but rather it used a strategy that its designer hoped would result in a positive score. EZAgent originally attempted to gain more hotel rooms than necessary so that other agents would not be able to secure complete travel packages. Most other agents were not willing to pay high prices for hotel rooms, and dropped out of the market when the price exceeded some threshold. EZAgent's policy of greedy acquisition resulted its paying high prices, hoarding rooms, and achieving low utility. This feature (bug) was disabled during the semi-finals.

4.11. UATrader

The travel agent UATrader developed at the University of Arizona participated in the TAC qualifying rounds.

Bidding Strategy

The general bidding strategy employed by `UATrader` can be characterized as a "myopic" trading strategy with iterative adjustments based on neighborhood search. After analyzing several game instances and observing the behavior of other trading agents, the Arizona team realized that the decisions as to what days each client should stay in Boston play a decisive role in agent performance. Thus, `UATrader` actively participates in the flight and hotel auctions and seeks to coordinate its bidding activities in these auctions, while its involvement in the entertainment ticket auctions is secondary and largely opportunistic.

`UATrader` bases most of its bidding decisions on the *anchor solution*—a hypothetical assignment of travel dates for each client. The anchor solution is initialized to reflect each client's preferred arrival and departure dates. Whenever a price change is observed, `UATrader` evaluates small variations (neighbors) of the current anchor solution on a client-by-client basis. This evaluation is based on the sum of the potential changes across all three types of auctions given the current price quotes. If the overall impact of switching from the current anchor solution to some variation is positive, this variation is made the new anchor solution for the corresponding client.

`UATrader` bids in two phases. The duration of the first phase is 13 minutes; then with 2 minutes remaining, it switches to the second phase.

Flight Auctions. In the first bidding phase, `UATrader` submits fixed low bids (e.g., $165) for all available flights to take advantage of possible low airfares. In the second phase, `UATrader` assures the booking of the flights that match the anchor solution for each client by submitting high bids.

Hotel Auctions. `UATrader`'s behavior in hotel room auctions is coordinated with its bidding strategy for flights. In the first phase, `UATrader` submits dummy bids at fixed time intervals to keep the auctions from closing. In the second phase, `UATrader` relies on anchor solutions to examine whether the Boston Grand Hotel (BGH) or Le Fleabag Inn (LFI) rooms should be targeted. At first, `UATrader` prefers that all clients stay in BGH. Then, as the auctions proceed, for each client, `UATrader` estimates the utility change of switching from BGH to LFI (or switching from LFI to BGH, if the current target is LFI) based on the current prices. If, for certain nights, prices of both hotels exceed pre-specified thresholds, `UATrader` automatically modifies the corresponding anchor solutions and adjusts its bids in the flight auctions to avoid those nights.

Entertainment Ticket Auctions. In the first bidding phase, `UATrader` is not active in any of the entertainment ticket auctions. In the second phase, after the anchor solution is booked, `UATrader` makes a one-time decision based on the current price quotes pertaining to selling and buying tickets with the objective of maximizing total client utility.

Allocation Strategy

UATrader relies on the default allocator strategy provided by the TAC server.

Motivations

The University of Arizona team consists of researchers from Management Information Systems, Computer Science, and Experimental Economics. The team is currently conducting research on comparing different electronic exchanges including auctions and negotiations; developing intelligent trading strategies in software agents for various online trading institutions; and investigating human-agent interaction for strategic decision-making tasks. TAC was a valuable research experience, closely related to their research goals.

4.12. EPFLAgent

EPFLAgent is the middle tier of a multiagent system composed of the TAC AuctionBot and a slave agent responsible for solving constraint satisfaction problems (CSPs) submitted by the EPFLAgent (see Figure 3).

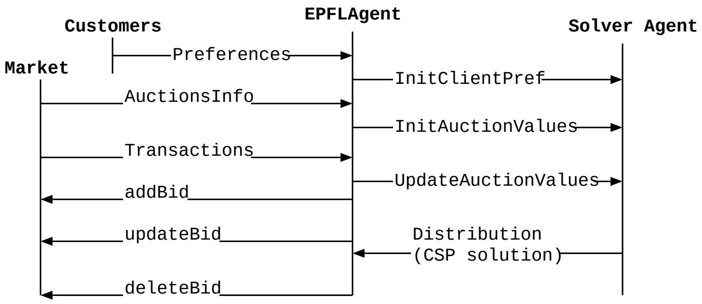

Figure 3. General overview of EPFLAgent communications with the other agents

The Solver uses constraint satisfaction algorithms provided by the JCL (Java Constraint Library) for solving the CSPs it receives. The CSPs are described in CCL (Constraint Choice Language) using an XML content description.

Bidding Strategy

EPFLAgent initializes the Solver using a CSP representation of the problem's variables (e.g., the client preferences, market state, etc.). Then, the agent classifies its clients into different classes, from the most constrained client to the least constrained, according to their preferences. Specifically, the clients are classified into three categories — very favorable (VF), favorable (FV), and unfavorable (UF) — based on their utilities (difference between the client's valuation of a package and its cost) and experimental thresholds. As the market evolves, EPFLAgent continually reclassifies the clients.

EPFLAgent starts by focusing on the clients who are FV, trying to satisfy them first since it is likely to be easy to satisfy VF clients when the end of the game is near. The agent continues spending the client's money (bidding slightly above the market price) until the client becomes UF or it is satisfied with an agent's utility. The bidding is done in parallel for all the FV clients.

Allocation Strategy

At the end of a game instance, the solver agent is given a limited amount of time to find a better solution than the current allocation. Since the solver is time-constrained, it uses simple heuristics to limit the search space. It may fail to find an optimal solution.

CSP-based Multiagent Approach to TAC

EPFLAgent represents a distributed solution to the TAC game. It focuses on the execution of a high-level strategy, delegating all specific tasks to other agents to achieve its goals. Specifically, it uses the CSP formalism and caching to anticipate its decision-making needs which it outsources to its Solver. The motivation for such an approach is to investigate the potential of multiagent solutions with the hope of discovering ways to tackle complex problems efficiently.

5. Results

We now describe some controlled experiments that explore specific aspects of two of the agents, namely ATTac and RoxyBot.

5.1. Adaptivity

In the final rounds of the TAC competition, adaptive hotel bidding played an important role. ATTac, who performed as well as the other teams in the early games when hotel prices (surprisingly) stayed low, out-performed its competitors in the final games of the tournament when hotel prices suddenly rose to high levels. Indeed, in the last 2 games, some of the popular hotels closed at over \$400, but ATTac successfully steered clear of these hotel rooms. In order to study the impact of an adaptive hotel bidding strategy in a controlled manner, we ran several game instances with ATTac playing against two variants of itself:

1. *High-bidder* computes its optimal tentative allocation based on the current hotel prices, as opposed to using priors and averages of past closing prices.
2. *Low-bidder* computes its optimal tentative allocation as do high-bidders, but then bids for hotel rooms at only \$50 above the current ask price, as opposed to bidding its marginal utility, which tended to be more than \$1000.

At the extremes, playing ATTac vs. 7 high-bidders, at least one hotel price skyrockets in every game since all agents bid very high for the hotel rooms. On

#high	agent 2	agent 3	agent 4	agent 5	agent 6	agent 7	agent 8
7 (14)	8671	8248	9369	11673	9253	9059	10406
6 (87)	11696	10552	10557	10328	10644	10295	1389
5 (84)	10579	10551	10474	10118	9830	2707	2650
4 (48)	10172	9982	9851	10016	3873	3360	4812
3 (21)	5085	5521	4595	3227	3533	4513	3284
2 (282)	*214*	*203*	2809	2918	2877	2295	2652

Table 8

The difference between ATTac's score and the score of each of the other seven agents averaged over all games in a controlled experiment. All scores are statistically significant at the 0.001 level, except the two marked in italics. Each row corresponds to a different number of high-bidders (excluding ATTac itself). The first column presents the number of high-bidders as well as the number of experiments we ran for that scenario in parentheses. The column labeled "agent i" shows how much better ATTac did on average than agent i. Scores above the stair-step line are for high-bidders and scores below the line are for low-bidders. In all cases, ATTac beats the other agents, and in all but the first two columns of the last row it does so with a very high level of statistical significance.

the other hand, playing ATTac vs. 7 low-bidders , hotel prices never skyrocket since all agents but ATTac bid close to the ask price. Our goal was to measure whether ATTac could perform well in both extreme scenarios as well as the various intermediate ones. Table 8 summarizes our results.

Each row of Table 8 corresponds to a different number of high-bidders in the game; for example, the row labeled with 4 high-bidders corresponds to ATTac playing against 4 high-bidders and 3 low-bidders. In the first column, we also show in parentheses the number of games played for the results in each row—each row reflects a different number of runs. In all cases, we ran enough game instances to achieve statistically significant results, but in some cases we ran more instances than turned out to be required. The column labeled agent i shows the difference between ATTac's score and the score of agent i averaged over all games. In all scenarios, these differences are positive, showing that ATTac outscored all other agents on average. Note, however, that ATTac's average score (and the scores of all other agents) tended to decrease with increasing numbers of high-bidders, since games became more volatile.

Statistical significance was computed from paired T-tests; all results are significant at the 0.001 level except for the two marked in italics. As mentioned before, as the number of high-bidders increases (in fact, any number greater than or equal to 3 suffices), the price for contentious hotels also rises. In all such scenarios ATTac outperforms all the other agents with a very high level of statistical significance. In the last row, when the number of high-bidders is only 2, very little bidding up of hotel prices ensues. As a result, we did not obtain statistically significant results relative to the two high-bidders (agent 2 and agent 3), since ATTac's strategy essentially reduces to high-bidding in this case. We do achieve high statistical significance relative to all the low-bidding agents, however. Overall, ATTac's adaptivity was successful when hotel prices did skyrocket, without impeding success when they did not.

The results of Table 8 provide strong evidence for ATTac's ability to adapt robustly to varying numbers of competing agents that bid up hotel prices near the end of the game. Combined with the fact that the top three finishers in the competition all had some form of price prediction strategy for avoiding expensive hotels, we can conclude that price prediction and adaptivity to hotel prices is a necessary component of a successful trading agent.

5.2. Allocation

We now present empirical results on the allocation problem. We compared the integer linear programming (ILP) approach of ATTac and the heuristic search approach of RoxyBot.[9] As our testbed, we used the 16 games of the TAC finals. There are eight agents per game; thus, these games comprise client preferences and final holdings for 128 agents.

In our first series of tests we verified that the optimal utility values output by RoxyBot's A^* algorithm were consistent with those of the ATTac's ILP on our 128 datapoints. A^* achieved a median run time of 0.59 seconds on a 600 MHz linux PC. Using CPLEX 6.5.3 on a 400 MHz SPARCstation with 2Gb of RAM, the ILP's median run time was only 0.02 seconds. This series of tests confirm what some TAC participants had already learned: optimal solutions are tractable for the dimensions of TAC.

Next, we experimented with the scalability of the algorithms. We produced instances of the allocation problem with 16, 32, and 64 clients by concatenating, for each game, first two, then four, and finally all eight agents' datapoints. Our complete dataset included 240 allocation instances: 64 16-client cases, 32 32-client cases, and 16 64-client cases in addition to the original 128 8-client cases. While A^* did not scale even to the 16-client cases, ILP fared much better, solving all but one of the 64-client cases. On the problematic case, however, the machine exhausted its 2Gb of RAM after six hours and aborted. The ILP also struggled with several other cases, spending five hours on one and over an hour on two others. Figure 4 summarizes ILP performance as a function of problem size. Each datapoint is graphed as a box plot reflecting the minimum, 25th percentile, median, 75th percentile, and maximum value of the running time. The ILP's running time can be characterized as fast on average but with very high-variance.

By contrast, beam search scales predictably: its running time is quadratic in the number of clients and linear in the beam width. We therefore designed experiments to test its degree of suboptimality. We ran a series of tests with beam widths varying from 1 to 1280 on all our instances except the one whose optimal solution was unknown (because the ILP was unable to solve it). We found that a

[9] Summaries of these approaches appear in Sections 4.1 and 4.2, respectively. Full details can be found in Stone *et al.*, [12] and Greenwald and Boyan [6].

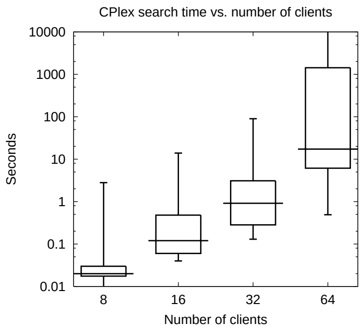

Figure 4. ILP running time at different problem sizes. (Note the log scale of the y-axis.)

beam width of only 1 (*i.e.*, best-first search) yielded a median accuracy of 99.4% in the 8 client case, with a median running time of less than 0.01 seconds. In the case of 64 clients, a beam width of 1 achieved a median accuracy of 97.9% in roughly 1 second. At the other extreme, a beam width of 1280 produced a median accuracy of 99.4% in the 64-client cases, but the median run-time was nearly 22 minutes. It seems a practical choice might be a beam width of 40, which achieved a median accuracy of 99.2% in the 64 client cases in 41.71 seconds.

Figure 5 illustrates how search quality and running time increase with beam width. Each datapoint is graphed as a box plot reflecting the minimum, 25th percentile, median, 75th percentile, and maximum value. Note that the running times have extremely low variance, while the performance is reliably above 96% of optimal for all but the smallest beam widths.

As allocation is simpler than completion, we do not expect completion to prove any less complex for the ILP. We also do expect completion to prove any *more* complex for beam search, since it solves allocation and completion in an identical way. We conclude that approximate heuristic search is a practical choice for use in the inner loop of a real-time bidding agent.

6. Conclusion

The first international trading agent competition was a very successful event, drawing twenty-two entrants from around the world. This article has compared and contrasted the strategies of twelve agents, including all of the finalists and most of the semi-finalists.

The large number of entrants and the range of agent strategies indicates that the organizers successfully created a problem that was relatively simple to attack, yet complex enough to prevent a trivial optimal strategy. From examining the different agent strategies, we can draw the following conclusions.

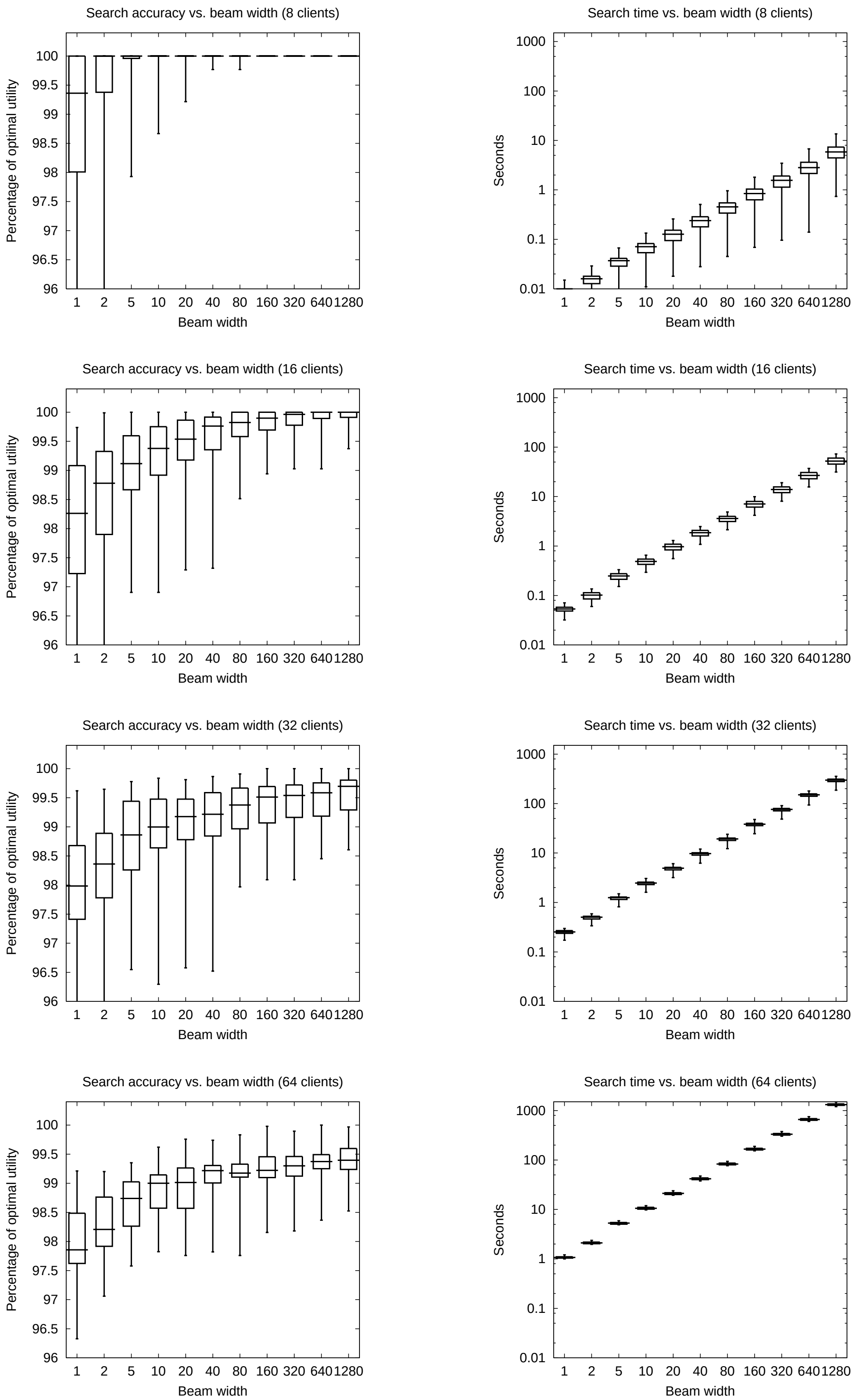

Figure 5. LEFT: Optimality of beam search, as a function of beam width and number of clients. (Note the offset on the y-axis.) RIGHT: Beam search runtimes, as a function of beam width and number of clients. (Note the log scale on the y-axis.)

- In a dynamic marketplace such as that modeled by TAC, the strategice timing of bid placement is crucial. Most of the agents strategies' incorporated some mechanism for placing bids as close to the end of the game as possible.
- When multiple goods with interacting values are auctioned simultaneously, agents must devise an efficient method for determining the values of sets of goods. Much of the effort of the more successful TAC agents was devoted to the allocation problem. The experiments reported in Section 5.2 explore the trade-off between optimal and scalable algorithms.
- When auction closing prices can vary dramatically, agents must incorporate some mechanism for predicting closing prices in advance, or at least taking into account the possibility that prices could shift unexpectedly. Indeed, the 3 top-scoring TAC agents had some form of price prediction or risk management strategy for the hotel auctions. The experiments reported in Section 5.1 serve to demonstrate the importantance of such adaptivity.

Notwithstanding the enthusiasm of the participants, a few suggestions have been proposed for the structure of future tournaments:

- There is no incentive to buy airline tickets until the end of the game. Were the price of flights to tend to increase, or were availability 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. There was usually no incentive for agents to reveal their clients' 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 on-line auction houses such as eBay. Roth and Ockenfels [7] argue that in such auctions, it is an equilibrium strategy to place two bids (with increasing valuations) and to place the second bid at the last possible moment. This strategy 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. In contrast, Amazon runs on-line 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, in equilibrium, *all bidders bid their true valuations before time* T.

  Were the TAC hotel auctions to be implemented in the style of Amazon, rather than eBay, agents would likely bid earlier. In this way, it would be possible for TAC agents to obtain information pertaining to the specific market supply and demand induced by the random client preferences realized in each game instance, and to use this information to estimate hotel prices. Unfortunately, Amazon-style auctions are not as well-suited for an agent competition, since there is no prespecified end time. Nevertheless, parallel auctions Amazon-style for substitutable and complementary goods would almost surely induce fascinating market dynamics.

- Activity in the entertainment auctions was limited during the first trading agent competition. This outcome, however, is not obviously correlated with the design of the entertainment auction mechanism. On the contrary, if more structure were added to the flight auctions, and if the hotel auctions were modified, perhaps in the way suggested above, interest in entertainment ticket auctions might be augmented.
- The information structure of the TAC setup was such that it was 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 (i) the dimensions of the TAC game should be extended such that the impact of any individual agent's bidding patterns is truly negligible; or (ii) to avoid issues of scalability and at the same time facilitate strategic reasoning, it should be possible to directly model the behavior of each individual agent, perhaps by associating names with bids. Were information provided regarding the bidding behavior of the agents (such that agents could infer other agents' clients' preferences, and therefore market supply, demand, and prices), TAC agents would potentially be able to learn to predict market behavior as the game proceeds.

The second trading agent competition took place in October, 2001 in Tampa, Florida, and incorporated some of these suggestions. TAC continues to be a worthwhile and exciting event in the emerging domain of designing autonomous agents for e-commerce.

Appendix

`ATTac`: Peter Stone, Michael Littman, Satinder Singh, Michael Kearns
AT&T Labs – Research
180 Park Ave.
Florham Park, NJ 07932
{pstone,mlittman,baveja,mkearns}@research.att.com

`RoxyBot`: Justin Boyan
NASA Ames Research Center and
MIT Artificial Intelligence Lab
545 Technology Square NE43-753
Cambridge, MA 02139
jboyan@mail.arc.nasa.gov

Amy Greenwald
Department of Computer Science
Brown University, Box 1910
Providence, RI 02912
amygreen@cs.brown.edu

`Aster`: Andrew Goldberg, Umesh Maheshwari
Strategic Technologies and Architectural Research (STAR) Laboratory
InterTrust Technologies Corp.
4750 Patrick Henry Drive
Santa Clara, CA 95054-1851
{`umesh,goldberg`}`@intertrust.com`

`UmbcTAC`: Youyong Zou
ECS 201
Department of Computer Science and Electrical Engineering
University of Maryland at Baltimore County
1000 Hilltop circle
Baltimore, MD,21250
`yzou1@cs.umbc.edu`

`ALTA`: Andrey Tarkhov, Dmitry Uspensky, Eugene Vostroknoutov
Artificial Life, Inc.
Four Copley Place
Skylobby, Suite 102
Boston, MA 02116
{`Dmitry.Uspensky,Andrey.Tarkhov,Eugene.Vostroknoutov`}`@artificial-life.com`

`DAIHard`: Rajatish Mukherjee, Partha Dutta, Sandip Sen
600 South College Avenue
Mathematical And Computer Sciences Department
University Of Tulsa
Oklahoma 74104
{`rajatish,partha`}`@euler.mcs.utulsa.edu, sandip-sen@utulsa.edu`

`RiskPro`:

Magnus Boman	Sven-Erik Ceedigh
SICS	Department of Computer & Systems Sciences
Box 1263	Stockholm University & The Royal Institute of Technology
SE-164 29 Kista	Electrum 230
Sweden	SE-164 40 Kista, Sweden
`mab@sics.se`	`s-e-ceed@dsv.su.se`

`T1`:

Erik Aurell, Martin Aronsson, Glenn Lawyer	Lars Rasmusson, Lars Olsson
Industrilogik L4i AB	SICS
Gvlegatan 22, P.O. Box 21024	P.O. Box 1263
SE-100 31 Stockholm, Sweden	S-164 29 Kista, Sweden
{`eaurell,mar,glenn`}`@L4i.se`	{`Lars.Rasmusson,larre`}`@sics.se`

`Nidsia`: Nicoletta Fornara, Luca Maria Gambardella
IDSIA
Switzerland
{`nicoletta,luca`}`@idsia.ch`

Marco Colombetti
Università della Svizzera Italiana
Lugano, Switzerland
`marco.colombetti@lu.unisi.ch`

`EZAgent`: Betsy Strother
North Carolina State University
Raleigh, NC 27695
`epriggin@eos.ncsu.edu`

`UATrader`: Daniel Zeng, Jiang Zhu, Bart Wilson
Department of Management Information Systems
Department of Computer Science
Economic Science Laboratory
University of Arizona
Tucson, AZ 85721
`zeng@bpa.arizona.edu`, `jiangzhu@cs.arizona.edu`,
`bwilson@econlab.arizona.edu`

`EPFLAgent`: Omar Belakhdar, Patrice Jaton, Boi Faltings
Artificial Intelligence Laboratory
Swiss Federal Institute of Technology
Lausanne, Switzerland
`belakdar@lia.di.epfl.ch`

References

[1] M. Boman. Norms in artificial decision making. *Artificial Intelligence and Law*, 7:17–35, 1999.

[2] C. Boutilier, M. Goldszmidt, and B. Sabata. Sequential auctions for the allocation of resources with complementarities. In *Proceedings of 16th International Joint Conference on Artificial Intelligence*, volume 1, pages 478–485, August 1999.

[3] J. Boyan, A. Greenwald, R. M. Kirby, and J. Reiter. Bidding algorithms for simultaneous auctions. In *Proceedings of IJCAI Workshop on Economic Agents, Models, and Mechanisms*, pages 1–11, 2001.

[4] D. Cliff and J. Bruten. Zero is not enough: On the lower limit of agent intelligence for continuous double auction markets. HP Technical Report HPL-97-141, 1997.

[5] L. Ekenberg, M. Boman, and J. Linnerooth-Bayer. General risk constraints. *Journal of Risk Research*, Forthcoming.

[6] A. Greenwald and J. Boyan. Bidding algorithms for simultaneous auctions: A case study. In *Proceedings of Third ACM Conference on Electronic Commerce*, 115-124 2001.

[7] 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, 2000.

[8] M. H. Rothkopf, A. Pekeč, and R. M. Harstad. Computationally manageable combinatorial auctions. *Management Science*, 44(8), 1998.

[9] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, 1995.

[10] H. Simon. Rational choice and the structure of the environment. In *Models of Bounded Rationality*, volume 2. MIT Press, Cambridge, 1958.

[11] P. Stone. *Layered Learning in Multiagent Systems: A Winning Approach to Robotic Soccer*. MIT Press, 2000.

[12] P. Stone, M. L. Littman, S. Singh, and M. Kearns. `ATTac`-2000: An adaptive autonomous bidding agent. In *Fifth International Conference on Autonomous Agents*, 2001.

[13] M. P. Wellman, P. R. Wurman, K. O'Malley, R. Bangera, S.-d. Lin, D. Reeves, and W. E. Walsh. A trading agent competition. *IEEE Internet Computing*, April 2001.