The 2002 Trading Agent Competition
An Overview of Agent Strategies

Amy Greenwald

■ This article summarizes 16 agent strategies that were designed for the 2002 Trading Agent Competition. Agent architects use numerous general-purpose AI techniques, including machine learning, planning, partially observable Markov decision processes, Monte Carlo simulations, and multiagent systems. Ultimately, the most successful agents were primarily heuristic based and domain specific.

Suppose you want to buy a used Canon AE-1 SLR camera at an online auction. It would be quite a daunting task to manually monitor prices and make bidding decisions at all web sites currently offering the camera—especially if accessories such as a flash and a tripod are sometimes bundled with the camera and sometimes auctioned separately. However, for the next generation of trading agents, autonomous bidding in simultaneous auctions will be a routine task.

Simultaneous auctions, which characterize internet sites such as eBay.com, are a challenge to bidders, particularly when complementary and substitutable goods are on offer. *Complementary goods* are items such as a flash and a tripod that would complement a camera, but a bidder wants the flash and tripod only if he/she is certain to acquire the camera. *Substitutable goods* are goods such as the Canon AE-1 and the Canon A-1—a bidder desires one or the other but not both. In combinatorial auctions, bidders bid on combinations of items, such as "camera and flash for $295"; in these auctions, the (NP-complete) problem of determining how to allocate the goods to maximize revenue falls in the hands of the auctioneer. In simultaneous auctions, however, the complexity burden lies with the bidders.

The International Trading Agent Competition (TAC) annually challenges its entrants to best design online agents capable of bidding in simultaneous auctions for substitutable and complementary goods. The original TAC was designed and operated by a group at the University of Michigan AI Laboratory (Wellman et al. 2001). In 2002, for the third rendition of the competition, the TAC software platform was redesigned by the Intelligent Systems Laboratory at the Swedish Institute of Computer Science (SICS).

Since the first International Trading Agent Competition back in 2000, TAC agent design has come a long way. In TAC-00, agent designs were primarily centered around designing algorithms to solve an NP-complete optimization problem. However, by the second year, it became common knowledge that this problem was tractable for the TAC travel game parameters. During the second year, agent designs focused on estimating clearing prices, and some agents designed algorithms that made use of distributional price estimates. Agent design in TAC-02, however, cannot be described so succinctly. Agent architects used numerous general-purpose AI techniques, including machine learning, planning, partially observable Markov decision processes, Monte Carlo simulations, and multiagent systems. Ultimately, however, the most successful agents were primarily heuristically based and domain specific.

Rules

Eight agents participate in a TAC game instance. Each TAC agent simulates a travel agent with eight clients interested in traveling from

Copyright © 2003, American Association for Artificial Intelligence. All rights reserved. 0738-4602-2003 / $2.00

Agent	AI Technology	Institution
ATTAC	Machine Learning	ATT Labs, Research
CUHK	Fuzzy Logic	Chinese University, Hong Kong
KAVAYAH	Neural Networks	Oracle
LIVINGAGENTS	Multiagent Systems	Living Systems, AG
PACKATAC	A Decision-Theoretic Agent	North Carolina State University
ROXYBOT	Online Heuristic Search	Brown University
PAININNEC	Genetic Algorithms	NEC Research
SICS	Optimization	Swedish Institute of Computer Science
SOUTHAMPTONTAC	An Adaptive Agent	University of Southampton
THALIS	Risk Aversion	University of Essex
TOMAHACK	POMDPs	University of Toronto
TNITAC	Planning	Politehnica University Bucharest
UMBCTAC	Risk vs. Return	University of Maryland at Baltimore County
WALVERINE	Competitive Equilibrium Analysis	University of Michigan
WHITEBEAR	Offline Heuristic Search	Cornell University
ZEPP	Multiagent Systems	Politehnica University Bucharest

Table 1. AI Technologies in TAC Agents.

TACtown to Tampa and home again over a five-day period. Each client is characterized by a random set of preferences for the possible arrival and departure dates, hotels (The Tampa Towers, AKA the good hotel, and Shoreline Shanties, AKA the bad hotel), and entertainment tickets (alligator wrestling, amusement park, and museum).

A TAC agent's score in a game instance is the difference between the total utility it obtains for its clients (based on its clients' preferences) and the agent's expenditures. To obtain utility for a client, an agent constructs a complete travel package for this client by purchasing airline tickets to and from TACtown and securing hotel reservations. It is also possible to obtain additional utility by supplementing a travel package with tickets to entertainment events. Each item is sold separately at auction: In total, there are 28 goods (4 flights in, 4 out, 4 good hotels, 4 bad, and 4 of each type of entertainment, for a total of 12). Thus, there are 28 simultaneous auctions; flights and hotels are complementary, and entertainment events are substitutable.

Airline ticket prices follow a biased random walk—prices are more likely to increase than decrease. Hotel room reservations are sold in ascending *English auctions* (auctions such as those for antiques and art). These auctions clear once a minute in an unspecified random order. Entertainment tickets are traded in continuous *double auctions* (auctions like those on the New York Stock Exchange).[1]

TAC Software

SICS hosted TAC in 2002 using a brand new server (Fritschi and Doerr 2002), which has been under development since November 2001. The purpose of this effort was to provide an open-source system to TAC participants as well as other researchers and educators in the field. The SICS TAC server consists of three main subsystems: (1) the market server, (2) the game server, and (3) the information server. The market server and the game server are written in SICSTUS Prolog. These servers run the actual markets and games and communicate with trading agents. The information server is written in JAVA. This server manages participants and schedules games; it collects and distributes game information and statistics on the web and is responsible for game monitoring applets.

The SICS server has proven itself to be very robust. It crashed on one occasion when one

team (accidentally) created thousands of agents, all of which requested simultaneous connections, a classic denial-of-service attack. Otherwise, the SICStus Prolog system proved extremely reliable, allowing code to be upgraded during continuous operation. The (noncritical) JAVA systems halted occasionally because of software errors; by design, these components can be stopped and restarted at any time without interfering with ongoing games. Nonetheless, future versions of the SICS TAC server will be implemented entirely in JAVA to facilitate community development.

A new addition to the game-monitoring applet, the chat room, led to unprecedented communication between participants. It was also an efficient channel for bug reports and public communication with the server developers.

Finally, SICS also developed new TAC AGENTWARE in the form of a JAVA software tool kit for TAC agents. Most of the new TAC entrants based their agents on SICS' TAC AGENTWARE.

Agent Strategies

This section describes 16 TAC agent strategies.[2] Fifteen of these 16 agents participated in the semifinals. A list of the 16 agents appears in table 1, along with the primary AI technology that characterizes each one.

ATTAC: A Machine Learning Agent

The core of ATTAC's approach is a learning algorithm that builds a model of price dynamics based on empirical data and utilizes this model to compute bids. ATTAC-02 is essentially the same agent as ATTAC-01 (Stone et al. 2002), but because ATTAC uses a learning approach and because the training data in TAC-02 differed from that of TAC-01, the agent's behavior differed from one year to the next.

Given posted prices for goods, one can compute a set of sales, purchases, and an allocation of goods to clients that maximizes profits. Similarly, if auction clearing prices are known, optimal bids can be computed: Bid high to buy, bid low to sell. Thus, one could predict precise clearing prices, or point estimates, for each open auction. However, the utility of this approach is very sensitive to the accuracy of predictions.

An alternative approach to price prediction is to construct a model of the probability distribution over clearing prices, stochastically sample price, and compute profit predictions, as defined earlier, for each sampled set of prices. This form of profit prediction models uncertainty. Moreover, it can be used to reduce uncertainty when used in conjunction with a scheme that estimates the value of delaying decisions for the purpose of gaining additional information.

ATTAC uses a distribution-based approach, relying on a general boosting-based algorithm for conditional density-estimation problems of this kind, that is, supervised learning problems in which the goal is to estimate the conditional distribution of a real-valued label (Schapire et al. 2002). As well as using this price prediction method successfully in competitions (ATTAC was a top finisher in the TAC-01 finals and the top scorer in the TAC-02 seeding round), its creators have validated this approach in controlled empirical experiments (Schapire et al. 2002).

ROXYBOT: A Retrospective

ROXYBOT-02 generalizes ROXYBOT-00 and ROXYBOT-01. At the heart of ROXYBOT-00 was a two-phase bidding policy: (1) solve the *completion* problem, that is, determine the set of goods on which to bid (Boyan and Greenwald 2001), and (2) associate numeric valuations with the goods in that set. Step 1 was accomplished using an optimization routine based on heuristic search (Greenwald and Boyan 2001); step 2 was accomplished using a marginal utility calculator. (The marginal utility of good x is defined as the utility of all goods, including good x, less the utility of all goods, excluding good x.) Policy determination in ROXYBOT-00 relied on a set of price point estimates.

ROXYBOT-01 generalized ROXYBOT-00 by computing policies not simply from price point estimates but rather from estimated price distributions. In particular, ROXYBOT-01 accom- plished step 1 by determining a set of goods that was likely to be of value under many samples of the estimated price distributions. (This algorithm proceeded by determining an initial set of goods that is desired under many samples, adding that set to the set of current goods and repeating. A larger and larger set of goods was built up by conditioning on those goods that were desired in earlier iterations.) Step 2 was accomplished by averaging marginal utilities across many samples of the estimated price distributions.

ROXYBOT-02, an agent based on Monte Carlo simulations, generalizes both earlier versions of ROXYBOT. Specifically, this agent (1) generates a bidding policy either according to ROXYBOT-01, using estimated price distributions, or according to ROXYBOT-00, obtaining a set of price point estimates by sampling from estimated price distributions, and (2) evaluates this bidding policy by averaging its score across many samples of estimated price distributions.

This search through the space of bidding policies continues until time expires (in TAC-02, time expires every minute on the minute), at which point RoxyBot-02 bids according to the best bidding policy.

CUHK: The Minority Agent

Agent CUHK's architecture is reminiscent of early TAC designs (Stone and Greenwald 2003). It consists of three major components: (1) a cost estimator, (2) an allocation and acquisition solver (AAS), and (3) bidders. The AAS allocates items using a greedy, heuristic search: Packages are allocated client by client without replacement. This approach is motivated by the fact that beam search was shown to output near-optimal solutions in TAC-00 (Greenwald and Boyan 2001), and by design, it is scalable to larger problems.

The input to the AAS are cost estimates. Like RoxyBot-00 (Greenwald and Boyan 2001), CUHK assumes sunken costs for goods that are already owned, opportunity costs for goods that could otherwise be sold, and inflated prices for duplicate copies of goods. Otherwise, the estimated cost of items with higher average prices in past games is greater than that of items with lower average past prices. This approach is intended to steer CUHK away from high-cost items and toward low-cost items so that CUHK's demands will be like those agents in the minority of the population. Ultimately, CUHK bids on (1) few goods, (2) few expensive goods, and (3) not too many low-cost goods.

CUHK's bidders rely on different strategies. Flight bidders vie for tickets twice an instance—in the beginning and at the end of the game. Initial flight-bidding decisions depend only on historical hotel clearing prices; later flight-bidding decisions depend on game dynamics. The hotel bidders bid low prices on all hotel rooms in the first few minutes, hoping to stimulate competitors to actively participate. After the first hotel auction closes, CUHK bids aggressively. The entertainment bidders bid on tickets using a strategy based on fuzzy reasoning. They place bids even if they do not reach, but are at least close to, their target prices.

KavayaH: Learning Market Demands

Three key features make up KavayaH: (1) a hotel demand and clearing price predictor, (2) a value assessor, and (3) bid submitters. We focus on its predictor because KavayaH is one of the few agents that only relies on machine learning.

Hotel price prediction is arguably one of the most important aspects of TAC. However, general price trends cannot be captured completely because they depend on the identity of the participating agents. KavayaH's price predictions are based on the demand for hotel rooms, as measured by the number of bids submitted in an auction; it is assumed that this parameter is less dependent on the participating agents. KavayaH predicts prices using a back-propagation neural network.

More specifically, demand for a day is defined as the average number of serious bids placed in both hotel auctions on the day. A bid is considered serious only if it exceeds the previous price by a certain threshold value. There are 12 input to the neural network, namely, the differences in starting flight prices. This setup is based on the observation that if such a price difference is significant, then demand tends to shift toward lower-priced days and away from higher-priced days. The output is classified into five classes: (1) very low, (2) low, (3) normal, (4) high, and (5) very high.

LivingAgents: Top Scorer in TAC-01

The LivingAgents team is based on the LivingAgents run-time system (lars) of Living Systems AG. lars agents operate in various research domains, such as TAC and RoboCup, as well as business platforms. LivingAgents used the same strategy and agents in TAC-02 as TAC-01. This strategy involved the following six agent types (the number in brackets is the number of running copies of each agent type): (1) TACManager [1] is responsible for starting and stopping the other agents. (2) TACClient [8] is responsible for the calculation of the best combination of tickets for the client it represents. (3) TACDataGrabber [5] is responsible for providing current auction information. (4) TACAuctioneer [4] is responsible for bidding according to the suggestion of the TACClients. (5) TACEntertainmentAuctioneer [1] is responsible for bidding in entertainment ticket auctions and regularly observing the current auction prices for opportunities to buy or sell tickets. This agent was as an extension of the simple TACAuctioneer type. (6) TACResultGrabber [1] is responsible for grabbing information of previous auction rounds and generating statistics to be used in later auctions.

LivingAgents makes most of its decisions at the very beginning of a TAC game instance. Only decisions to buy or sell entertainment tickets can occur later in the game. This strategy ensures considerable savings in the cost of flights. By using the average hotel prices from earlier games as forecasts and calculating the potential benefits of all permutations of packages, each LivingAgents client finds an approximately optimal allocation of tickets.

PACKATAC: A Hybrid Agent

The PACKATAC agent is a hybrid open-loop, closed-loop agent that commits to some aspects of packages at the beginning of the game and later completes packages by optimizing over current and predicted states of the game.

At the beginning of the game, the agent buys both flights for easy trips (that is, those that rely on goods that are not expected to be in great demand) and one anchor flight for trips of greater difficulty. Once two hotels have closed, the agent again selects flights to purchase by optimizing over predicted hotel clearing prices. The optimization formulation includes constraints to "level the demand" by limiting the number of clients in town on any given night as well as the distribution of those clients over the good and bad hotels.

Hotel prices are predicted using historical information. Like agents in TAC-01, PACKATAC conditions price predictions on the set of hotels that have already closed and the current ask price of each auction. The agent computes and bids the marginal value of each hotel room, given the predicted prices of the other hotel rooms. PACKATAC also places low-valued bids on some rooms early in the game to hedge against the possibility of prices skyrocketing later on. PACKATAC offers to buy or sell entertainment tickets based on its needs at prices determined by a hard-coded linear schedule.

PAININNEC: Genetic Algorithms

PAININNEC's strategy incorporates a combination of heuristics, including a genetic algorithm–based optimization technique. This optimization routine takes as input the auctions' expected clearing prices, as well as the clients' preferences, and outputs a set of goods to buy and sell.

In the genetic algorithm, each client's package is represented by a six-digit string $the_1e_2e_3e_4$. The first string element t ranges from 0–9 and encodes the travel dates of the package (for example, $t = 0$ represents arrival on day 1 and departure on day 2). The second element h is either 0 or 1, depending on the hotel type. The remaining elements of the string represent the entertainment package, with one entertainment ticket type per day. For example, if $e_1 = 0$, then this package does not include any entertainment on day 1, whereas if $e_1 = 1$, then this package includes an entertainment ticket of type 1 on day 1. A string of length 48 represents the 8 clients' travel packages.

The genetic algorithm optimizes the sum of each client's value for its assigned package less the cost of the goods. Strings are selected for reproduction using the *tournament method*; that is, two strings are selected at random, and with high probability, the string with the greater objective value reproduces, although errors occur with low probability. The genetic algorithm parameters used in the optimization routine during the TAC tournament were as follows: The population size was 200, the number of generations was 40, the crossover probability was 0.9, and the mutation probability was 0.05. The average run time for the genetic algorithm was about 20 seconds, and the genetic algorithm steered clear of packages with very high hotel costs.

SICS: Branch-and-Bound Optimization

SICS's TAC-02 agent, SICS, explored a risk averse strategy and a new optimization algorithm.

The optimizer operates on price lines (Boyan and Greenwald 2001), which are vectors of predicted clearing prices for different quantities of each good, for example, 0 to 8 hotel rooms at the good hotel on day 3 at prices: price(good-hotel; 3; [0; 155; 387; 726; 1210; 1892; 2838; 4138; 5912]). One such vector is constructed for each auction, trading-i (selling when possible) to 8-i (buying) items, where i is the number of items owned. The predictions were based partly on seeding round data, using the public TAC-02 game data tool kit, and partly on auction quotes and heuristics. The main heuristic used was risk averseness; for example, fourth-minute predictions were taken to be the ninetieth percentile of historic fourth-minute prices.

Given input of this form, the optimizer performs branch-and-bound search in a separate process, delivering a stream of gradually better solutions. The process is interrupted after 20 seconds, and the best solution found thus far is returned. Often, the best answer is found in the allotted time, and occasionally, it is proven optimal by completing the search. All variables are assigned in a general way by the optimizer, but ordering heuristics—for example, best-first assignment of possible hotel-flight combinations to a client—are crucial for performance.

A simplified version of the optimizer was used for calculating TAC-02 scores, usually finding the optimal solutions in milliseconds, but on some exceptional instances, it took minutes. Both optimizers are part of the SICS open-source TAC software.

The least developed part of the SICS agent is the bidding strategy. There is no principled mechanism for risk-consequence analysis. This is an area for future research.

SouthamptonTAC: An Adaptive Agent

SouthamptonTAC is an adaptive agent that varies its bidding strategy according to the prevailing market conditions.

Building on SouthamptonTAC01 (He and Jennings 2002), the agent classifies TAC game instances into three types of environments: (1) *noncompetitive*, in which there is very little competition for hotels, and an agent can obtain the rooms it wants at low prices; (2) *semicompetitive*, where prices are in the middle; and (3) *competitive*, where the prices of the hotels are (possibly very) high. The agent makes this classification based on the outcomes of the most recent games and the current prices of the hotels in the various auctions.

In games it deems noncompetitive, SouthamptonTAC buys all its flight tickets at the beginning of the game and never changes the travel plan of its clients. In competitive games, it buys flights according to its assessment of the flight category (rapidly rising flight prices cause the agent to buy near the beginning, whereas stagnant flight prices cause it to wait until near the end). In these games, it can alter the customers' travel plans to avoid staying in expensive hotels for long periods. Moreover, the agent continuously monitors the ongoing market situation and changes its assessment of the game type—and accordingly its strategy—depending on observed prices.

SouthamptonTAC uses fuzzy reasoning techniques to predict hotel clearing prices. In an attempt to capture the many different factors that affect hotel clearing prices, three rule-bases are used: (1) when both the good and bad hotel auctions are open, (2) when the counterpart hotel has just closed, and (3) when the counterpart hotel has been closed for more than one minute. The factors considered in the prediction are the price of the hotel, the price of the counterpart hotel, the price change in the previous minute, and the previous price change of the counterpart hotel (when it closed).

Thalis: A Risk-Averse Agent

In its initial allocation, an optimizer decides on the best package for each client based on its preferences and statistics about average hotel clearing prices. If any travel package involves more than three nights, it only issues requests for hotel auctions and reserves the flight requests for a later time (at least the fourth minute, or at most the fifth minute, depending on flight prices). Once the initial allocation has been completed, the optimizer checks if (1) the number of hotel nights desired on any given day exceeds a threshold or (2) the total number of hotel nights exceeds a given threshold. If so, the optimizer increases the estimated costs of the relevant hotel auctions, forcing a suboptimal solution that does not violate either of these constraints.

In terms of bidding, Thalis buys flights as soon as possible. It delays buying flights only in cases where the number of nights requested by a client is above a threshold (say, three) or when the selected package yields little revenue. For hotels, Thalis bids after the start of the fourth game minute and issues its initial bids at relatively high prices (determined by the statistics) to buy some learning time for the optimizer. It then monitors any increase in prices to dynamically configure later bid prices based on its estimates of future clearing prices. Finally, Thalis only sells entertainment tickets during the first six minutes of the game; thereafter, it only buys entertainment tickets.

TomaHack: A Partially Observable Markov Decision Process Approach to TAC

The TomaHack team's goal was to model TAC as a sequential decision process, specifically a partially observable Markov decision process (POMDP). Although TAC is a multiagent domain that might more accurately be modeled as a partially observable Markov game, the TomaHack team made the simplifying assumption that the other agents are part of the environment.

In the TomaHack formulation, the state of the game encompassed elapsed time, client preferences, current quotes, outstanding bids, and the agent's holdings. A key difficulty in this choice of representation was how to best determine the underlying state-transition function. Each transition, from one state to the next, requires simulation of all aspects of the dynamics of a TAC game—including the actions of rival agents.

In an attempt to uncover the state-transition function, the TomaHack team built models of the bidding behavior of each of its opponents. The models were constructed offline, using game logs as the source of data because of insufficient information about individual agents' actions during the play of a game. Each opponent was modeled as a function from observed state to active bids. Using neural network regression to approximate this function, promising results were obtained.

Given the significant complexity and the generative nature of the POMDP model, the TomaHack team planned to use a model-free approach inspired by Meuleau et al.'s (1999) policy search algorithm to learn a reasonable finite-state controller using simulation. Unfor-

tunately, there was insufficient time to implement this POMDP-based agent. With only limited success using an agent based on one of their opponent models, the TOMAHAC team entered a simple high-bidding agent in the final rounds of TAC-02.

TNITAC: A Planning Agent

TNITAC's designers transformed TAC into a planning problem. The agent is assigned an initial state, it decides on a goal state, and it creates a plan for reaching this goal state. A state is defined by goods owned, active bids, client preferences, and probabilities of winning in open auctions. The agent associates a utility value with each state, and the state with the highest utility value is the goal state.

State values are determined by summing the values of the goods the agent owns and the values of goods for which its bids are active. The value of an owned good is computed in one of two ways, depending on its relation to a travel package: (1) When the good is associated with a complete package, its direct value represents the additional utility gained by using this good with the complete package; (2) when the good is associated with an incomplete package, its collateral value represents a fraction of the package's utility, depending on how many other goods are necessary to complete that package and what the probabilities of winning these other goods are. The value of a good in an open auction, in which the agent has an outstanding bid, is computed in the same manner as the value of an owned good, except that this value is multiplied by the probability of winning the good.

Because TAC market conditions are continuously changing, TNITAC's main algorithm is loop based: During each iteration, it adapts its strategy to the current game situation. More importantly, a new goal state is computed each iteration. To find its new goal, TNITAC starts at its old goal. It first eliminates all lost goods and then explores a small neighborhood to find a state of maximal value, which becomes the new goal state.

UMBCTAC: A Balanced Bidder

UMBCTAC handles flight and hotel auctions separately from entertainment ticket auctions. Early in the game, UMBCTAC commits to a travel plan and places all its flight and hotel auction bids. Thereafter, it focuses on entertainment bidding. UMBCTAC is a balanced bidder: It never bids on too many rooms in any one hotel auction, and it tries not to hold too many entertainment tickets of any one type.

Because UMBCTAC commits to its travel plan early in the game, it seeks a well-balanced, robust plan. To find one, UMBCTAC uses a model of risk versus return, which operates as follows: (1) preprune highly supoptimal candidate plans, (2) select plan combinations with minimal risk, and (3) select plan combinations with maximal return. Hotel price predictions are simple historical averages. However, average prices correspond to average demand; thus, UMBCTAC adjusts its own demands to create average demands across hotel auctions, increasing the likelihood of accurate predictions.

Regarding entertainment auctions, UMBCTAC uses a probability-based model that handles each auction separately. In particular, the agent computes a range of buying and selling prices and probabilities of buying and selling at these prices. For example, if the agent holds three tickets of a certain type on a given day, it will sell one ticket at its lowest selling price if it expects another agent to place a bid above the price with high probability; however, if it holds only one ticket, it will sell the ticket at its highest selling price, even it expects another agent to bid above that price with only low probability.

WALVERINE: Walras + Wolverine

The University of Michigan's TAC entry, WALVERINE, aims to bid optimally based on a competitive equilibrium analysis of the TAC travel economy.

Any package optimization routine must be based on a forecast of hotel prices, the major source of uncertainty in TAC. WALVERINE predicts hotel prices by calculating the Walrasian competitive equilibrium for the game. It derives the equilibrium using a tâtonnement approach, a measure of expected demand derived from the known distribution of client preferences. This approach is designed to exploit information provided by the initial flight prices, which influence agents' choices of preferred travel days. Although the competitive assumption is clearly false for individual agents, it might support an accurate model of aggregate behavior.

By sampling outlier values around the predicted prices, the prediction is used to compute an "outlier hedged" set of optimal travel dates (hence, flight purchases) and marginal values for the available hotel rooms. During each hotel-bidding round, WALVERINE constructs bids that maximize expected value, taking into account its marginal value of each room unit, the probability of winning for a given bid, the probability that this bid will actually be the sixteenth (and, thus, set the price), and the expected price if it does not bid. These probabilities are derived from an expected distribution

Order	Agent	Score	Average
1	WHITEBEAR	3556.48	3412.78
2	SOUTHAMPTONTAC	3492.03	3385.46
3	THALIS	3351.23	3246.27
4	UMBCTAC	3320.65	3235.56
5	WALVERINE	3315.62	3209.52
6	LIVINGAGENTS	3309.83	3180.89
7	KAVAYAH	3249.86	3099.44
8	CUHK	3247.83	3068.77

Table 2. TAC-02 Final Results.

Each agent's score is simply an average, but the lowest score is dropped.

of other agents' valuations as well as a competitive analysis of the trading domain.

As described earlier, WALVERINE's approach to flight and hotel bidding is completely model based: There are no empirically tuned parameters. In contrast, WALVERINE's approach to entertainment bidding is completely model free. Specifically, WALVERINE executes a policy derived through Q-learning over thousands of auction instances. Bid actions are described in terms of offsets from marginal values, as calculated from the optimizations based on predicted hotel prices described earlier.

ZEPP: A Multiagent System

ZEPP is a TAC agent whose core is a multiagent system.[3] Each of ZEPP's clients is assigned a personal internal agent that is responsible for meeting his/her demands. This design is intended to mirror the principles of a live travel agency, where the employees work on behalf of their own clients but not on behalf of the clients of other employees.

There is no competition inside ZEPP. On the contrary, if one internal agent owns a good that is not useful to its client, it can offer this good to the other internal agents using the central coordination system (CCS). The CCS is transparent to the internal agents; in particular, a personal agent interacts with the CCS just as it would with the TAC auction server (except that flights and hotels can only be transferred internally).

During a TAC game instance, each internal agent creates a list of possible solutions for its client and computes the utility of each possible solution less estimated costs. At prespecified times, each agent sends bids for its best possible solution. This design permits rapid changes in bidding strategy, both in terms of what goods are bid on and at what price.

WHITEBEAR: Top Scorer in TAC-02

The WHITEBEAR approach is to generate domain-specific heuristics (for example, strategies used in TAC-01) and experiment with these heuristics to determine which are the most effective. Sometimes, the most effective heuristic is, in fact, a combination of two or more heuristics.

In the hotel auctions, WHITEBEAR ultimately combines two extreme heuristics: (1) strategy A is "bid some small increment greater than current prices," and (2) strategy B is "bid marginal utility." WHITEBEAR combines these strategies to form its actual bidding strategy, which lies in the middle ground: "use strategy A, unless the marginal utility is high, in which case use strategy B." In the flight auctions, heuristics are distinguished by their bidding behavior at the beginning of a game instance. Two extremes are "buy everything" and "buy only what is absolutely necessary." WHITEBEAR ultimately bids on flights as follows: "Buy everything, except any tickets that are dangerous." Dangerous tickets are those that restrict diversification across hotel auctions.

WHITEBEAR does not necessarily bid on an optimal set of goods for its clients, nor does it use sophisticated machine learning techniques to predict hotel clearing prices (its predictions are simply historical averages). WHITEBEAR's performance suggests that domain-specific heuristics, coupled with extensive experimentation, are a recipe for success in the TAC travel game (Vetsikas and Selman 2002).

Results

The competition on 28 July 2002 consisted of a morning with two simultaneous rounds of semifinals, each involving eight agents, followed by an afternoon of finals, involving the four top-scoring agents in each semifinal round. The top-scoring agents in the semi-finals rounds were Southamptontac and Whit Bear.[4] These agents were also the top-scoring agents in the finals (table 2).

Conclusion

Participants in TAC-00 laid the foundations for many TAC-02 agent architectures. The completion problem was formulated (Boyan and Greenwald 2001), and solutions based on integer linear programming (Stone, Littman, Singh, and Kearns 2001) and heuristic search (Greenwald and Boyan 2001) were proposed, enabling agents to determine at least a near-optimal quantity of each good to buy and sell, given its inventory, its clients' preferences, and clearing prices. Some TAC-02 agents extend the

earlier architectures with sophisticated AI techniques, such as machine learning, Monte Carlo simulations, and POMDPs. However, both SOUTHAMPTONTAC and WHITEBEAR, the best-performing TAC-02 agents, extend this architecture with domain-specific heuristics.

What was most successful in the design of TAC-02 agents (and likely to be of greatest assistance in your camera purchase) is precisely what is successful in most other branches of AI. Championship chess programs, which rely on the provably optimal αβ-pruning algorithm, also incorporate elaborately hand-coded openings and end games. Speech recognition, which is built on the elegant machinery of hidden Markov models, became ubiquitous only after painstaking engineering of language models and acoustic features. As with other complex domains, the best-performing internet trading agents are likely to combine clever algorithms with hand-tuned heuristics.

Acknowledgments

The following TAC agent designers also contributed to the text of this article: Viswanath Avasarala, Chi-Lun Chau, Li Ding, Maria Fasli, Clemens Fritschi, Minghua He, Sverker Janson, Nicholas Jennings, Pascal Poupart, Ravi Prakash Putchala, George Stan, Peter Stone, Ovidiu Trascu, Ioannis Vetsikas, Michael Wellman, and Peter Wurman.

Notes

1. Complete descriptions of the rules are available at www.sics.se/tac and tac.eecs.umich.edu/.

2. All agent designers were invited to contribute to this article. Those included are those who chose to be included.

3. ZEPP is the only agent reported on in this article that did not participate in the semifinals.

4. The organizers prefer to refrain from declaring any winners because far too few games can be run in a one-day workshop to achieve statistical significance.

References

Boyan, J., and Greenwald, A. 2001. Bid Determination in Simultaneous Auctions: An Agent Architecture. In Proceedings of the Third ACM Conference on Electronic Commerce, 210–212. New York: Association of Computing Machinery.

Fritschi, C., and Dorer, K. 2002. Agent-Oriented Software Engineering for Successful TAC Participation. Paper presented at the First International Joint Conference on Autonomous Agents and Multiagent Systems, 15–19 July, Bologna, Italy.

Greenwald, A., and Boyan, J 2001. Bidding Algorithms for Simultaneous Auctions: A Case Study. In Proceedings of Third ACM Conference on Electronic Commerce, 115–124. New York: Association of Computing Machinery.

He, M., and Jennings, N. R. 2002. SOUTHAMPTONTAC: Designing a Successful Trading Agent. Paper presented at the Fifteenth European Conference on Artificial Intelligence, 21–26 July, Lyons, France.

Meuleau, N.; Peshkin, L.; Kim, K.-E.; and Kaelbling, L. P. 1999. Learning Finite-State Controllers for Partially Observable Environments. In Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence, 30 July–1 August, Stockholm.

Schapire, R. E.; Stone, P.; McAllester, M.; Littman, M. L.; and Csirik, J. A. 2002. Modeling Auction Price Uncertainty Using Boosting-Based Conditional Density Estimation. Paper presented at the Nineteenth International Conference on Machine Learning, 8–12 July, Sydney, Australia.

Stone, P., and Greenwald, A. 2003. The First International Trading Agent Competition: Autonomous Bidding Agents. *Journal of Electronic Commerce Research*. Forthcoming.

Stone, P.; Littman, M. L.; Singh, S.; and Kearns, M. 2001. ATTAC-2000: An Adaptive Autonomous Bidding Agent. *Journal of Artificial Intelligence Research* 15:189–206.

Stone, P.; Schapire, R. E.; Csirik, J. A.; Littman, M. L.; and McAllester, D. 2002. ATTAC-01: A Learning, Autonomous Bidding Agent. Paper presented at the Workshop on Agent Mediated Electronic Commerce IV: Designing Mechanisms and Systems, 16 July, Bologna, Italy.

Vetsikas, I., and Selman, B. 2002. WHITEBEAR: An Empirical Study of Design Trade-Offs for Autonomous Trading Agents. Paper presented at the Eighteenth National Conference on Artificial Intelligence Game-Theoretic Decision Theoretic Agents Workshop, 28 July–1 August, Edmonton, Alberta, Canada.

Wellman, M. P.; Wurman, P. R.; O'Malley, K.; Bangera, R.; Lin, S.-D.; Reeves, D.; and Walsh, W. E. 2001. A Trading Agent Competition. *IEEE Internet Computing* 5(2): 43–51.

Amy Greenwald is an assistant professor of computer science at Brown University in Providence, Rhode Island. Her primary research area is the study of economic interactions among computational agents. Her primary methodologies are game-theoretic analysis and simulation. Her research is applicable in areas ranging from dynamic pricing using pricebots to autonomous bidding agents. She was recently awarded a Computational Social Choice Theory career grant by the National Science Foundation and named one of the Computing Research Association's digital government fellows. Before joining the faculty at Brown, Greenwald was employed by IBM's T. J. Watson Research Center, where she researched information economies. Her paper entitled "Shopbots and Pricebots" (joint work with Jeff Kephart) was named Best Paper by IBM Research in 2000. Her e-mail address is amy@brown.edu.