A Risk-Averse TAC Agent[*]

Peter Hultman

October 23, 2002

Abstract

The Trading Agent Competition sheds light on research issues in automated trade. Based on a challenging market scenario, agents compete through various strategies, seeking to complete travel package trips. Difficult issues include bidding strategies, market prediction, and resource allocation. A risk-averse agent "SICS" is presented, which uses a price prediction strategy based on heuristics and competition data. Its efficiency is evaluated using its performance in the TAC'02 competition. The results indicate that the use of basic game statistics and competition data gives a good ground for developing a successful agent.

[*]This thesis corresponds to 20 weeks of full-time work.

Contents

1 Introduction

1.1 Background

Trading in electronic markets is increasingly becoming a special topic of interest within the Artificial Intelligence (AI), Electronic Commerce and Multi Agent Systems (MAS) research communities. With the ambition to create a common domain for researchers to work on, where different solutions to a common problem could be discussed and compared, the first Trading Agent Competition (TAC) [2, 17, 15] was held in conjunction with the International Conference on Multi-Agent Systems (ICMAS-00). TAC-00 attracted 18 participants (research groups, students, and others) from all over the world, to develop agents with unique solutions to the problem. Based on the success of that event a sequel was held in 2001 [4, 20], and this year, 2002, the competition was held for the third time.

The software agents that enter the competition represent travel coordinators whose goal is to arrange travel packages for their clients. These packages consist of flight tickets, hotel rooms, and tickets to entertainment events, all of which the agents trade in simultaneous electronic auctions. An agent's objective is to secure goods serving the particular desires of its clients, but to do so as inexpensively as possible. The market game has been specially designed to present the agents with difficult decision problems and admit a wide variety of potential bidding strategies [22].

The first two years competitions were hosted on a version of the Michigan Internet AuctionBot server [17]. This year the competition was organized by the Swedish Institute of Computer Science (SICS) who re-implemented the server from scratch and made it downloadable open source [18]. The agents communicate with the server via a TCP-based agent programming interface (API), using an XML-based protocol, in order to get current market information and place its bids.

1.2 Aim

The purpose of this study is to test how a fairly simple, risk-aversive agent strategy, based on heuristics and statistics, performs in an online market game like TAC. The hypothesis is that statistics from previous games contains valuable information about the characteristics of the market and the expected closing prices for the auctions. This information can help the agent to perform better in the competition, in terms of average score.

1.3 Methodology

Beginning with literature studies, covering agents, conclusions drawn from previous years, and descriptions of the game rules, I tried to get a good understanding of the domain, and the problem itself. However, the time constraints introduced by the fact that the preliminary rounds of the competition were to be held only two weeks after I began my study, forced me in the beginning to mainly focus on having a working agent running in time for the competition. Most of the understanding of the game came from testing and observing the results of different approaches during the qualifying and seeding rounds of the competition. The nature of the game makes testing and debugging a very time consuming activity, and far from optimal.

To be able to mainly concentrate on the strategies of the agent I decided to use the AgentWare, downloadable from the TAC-web-site [18], which handles the communication with the server, and maintains information about current bids and prices. A great deal of help and ideas were also received by the team at SICS that developed an agent for TAC-01, which nearly made it to the finals [1].

The competition offers a natural evaluation environment for testing the agent, in terms of how well it succeeds in bidding. The results from the competition can be used to give an indication of the competitiveness of the agent, and highlight the good and bad parts of the strategy. The hypothesis can therefore largely be tested by analyzing the outcome of the competition.

2 The Trading Agent Competition

2.1 Background

Inspired by the success of competitions in various other AI realms, such as RoboCup [12], the Trading Agent Competition (TAC) has been arranged with the intention of providing a similar challenge, particularly aimed towards researchers in the complex domain of e-marketplaces. The main goals of TAC, as stated by the initiators [21], are:

1. Spurring research on common problems within the domain.
2. Promoting definitions of benchmarks and standard problem descriptions.
3. Showcasing research in this area to the MAS community and beyond.
4. Having fun.

The first competition was held in July 2000 in Boston, in conjunction with the International Conference on Multi-Agent Systems (ICMAS-00). It ended up being a very successful event with 18 participants from six different countries [20], several of whom based on this experience contributed to the research literature on trading agents [3, 5, 6, 7, 8, 15, 16]. The participants' motivations illuminate the general applicability of the TAC setup and its relevance to today's research agendas.

The second TAC was held in October 2001 in Tampa, as part of the ACM Conference on Electronic Commerce (EC-01), with a small modification of the rules from the previous year. One positive result of the second TAC was the possibility of measuring actual progress, through performance and through the transfer of ideas from one competition to the next [20]. Many contributions to the literature was made this year as well [1, 9, 13, 20].

In 2002, TAC was hosted by the Swedish Institute of Computer Science (SICS), and the finals were held in July in Edmonton, as part of the Computers and Games Conference (CG-02), co-located with AAAI-02. The rules were not changed from TAC-01, but the server was re-implemented from scratch. A brief description of most of this and previous years' agent strategies, as well as most of the publications about TAC can be found at the TAC-01 web-page [19].

2.2 Game Description

Below follows a description of the TAC game, taken from the official TAC-02 site [18].

2.2.1 Game Overwiev

In the TAC shopping game, each "agent" (an entrant to the competition) is a travel agent, with the goal of assembling travel packages (from TAC-town to Tampa, during a notional 5-day period). Each agent is acting on behalf of eight clients, who express their preferences for various aspects of the trip. The objective of the travel agent is to maximize the total satisfaction of its clients (the sum of the client utilities).

Travel packages consists of the following:

- A round-trip flight
- A hotel reservation
- Tickets to some of the following entertainment events
  - Alligator wrestling
  - Amusement park
  - Museum

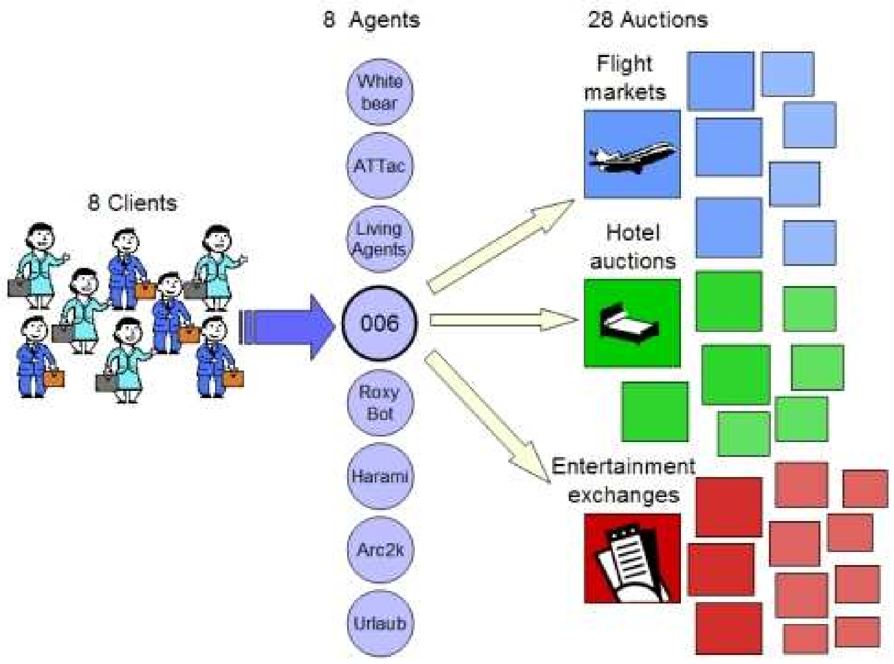


Figure 1: Illustration of the environment a TAC agent operates within. To the left are its eight clients and their preferences, in the middle all its competitors lined up (7 competitors/game), and to its right are all the auctions (28 simultaneous auctions of three different types).

There are obvious interdependencies, as the traveler needs a hotel for every night between arrival and departure of the flight, and can attend entertainment events only during that interval. In addition, the clients have individual preferences over which days they are in Tampa, the type of hotel, and which entertainment they want. All three types of goods (flights, hotels, entertainment) are traded in separate markets with different rules.

A run of the game is called an instance. Several instances of the game are played during each round of the competition in order to evaluate each agent's average performance and to smooth the variations in client preferences.

2.2.2 Goods

Flights. TACAIR is the only airline to fly between TACtown and Tampa. It operates only one flight each way per day. Tickets for these flights are sold in single seller auctions — one auction for each day and direction (in or out). Since all clients must stay at least one night in Tampa, there will be no in-flights on the last day, nor out-flights on the first day. The auctions will clear continuously. TACAIR is represented in the marketplace by an agent that sets prices according to a stochastic function. The process used to update flight prices is a random walk, starting between \$250 and \$400 and perturbed every 24 to 32 seconds by a value drawn uniformly from the range -10 to $x(t)$. The final upper bound x on perturbations is a random variable (not revealed directly to the agents), chosen for each flight independently from a uniform distribution on $[10, 90]$. The upper bound on perturbations at time t, $x(t)$, is a linear interpolation between 10 and the final bound $x : x(t) = 10 + (t/12 : 00) * (x - 10)$. Regardless of the perturbations, prices are constrained to remain in the range \$150 to \$800. The distribution of initial flight prices is also uniform. The supply of available seats on these flights is, as far as TAC agents are concerned, unlimited.

Hotels. There are two hotels in Tampa: the Tampa Towers (TT) and Shoreline Shanties (SS). TT is cleaner, more comfortable, and more convenient — all-around a nicer place to stay. For this reason, we would expect TT to cost more. Note that a client cannot move between hotels during its trip.

Rooms at the hotels are traded in ascending auctions: one auction for each combination of hotel and night, each with 16 rooms being auctioned off. Only the hotels can sell rooms. There is no minimum bid for either type of hotel. The price difference between the hotels is based on preference for Tampa Towers, as defined by the client utility functions (see next section).

Since clients need hotels only from the night of their arrival and through the night before their departure, no hotels will be available (or needed) on the last day.

Entertainment. At commencement of a game instance, each travel agent receives an allotment of entertainment tickets. There are a total of 8 tickets available for each event type on each day, and each agent receives 12 tickets partitioned as follows:

- One bundle of four of a particular type on day 1 or day 4
- One bundle of four of a particular type on day 2 or day 3
- One bundle of two of a particular type (different from above) on day 1 or day 4
- One bundle of two of a particular type (different from above) on day 2 or day 3

Agents exchange entertainment tickets through a continuous double auction (CDA). The travel agents can act both as buyers and sellers in these markets. There is one auction for each event-night combination.

As with the hotels, a client cannot use an entertainment ticket on the day of departure.

2.2.3 Auctions

All of the auctions run according to the following high-level protocol:

1. An agent submits a bid to the auction
2. The auction updates its price quote, indicating the current going prices

The rules for a particular auction specify when, or under what conditions the auction will match the bids and record the transactions.

Bid Format. A bid contains a bid string, representing an agent's willingness to buy and sell the goods in an auction. A bid string containing a list of bid points in the following form:

$$''((q_1\ p_1)(q_2\ p_2)\ldots(q_n\ p_n))''$$

where q_i is a quantity and p_i is a price. If there is a point $(q_i\ p_i)$ with $q_i > 0$, then it means that the agent is willing to buy q_i units of the goods at the auction for no more than p_i price units per unit of the goods. If there is a pair $(q_j\ p_j)$ with $q_j < 0$, then it means that the agent is willing to sell q_j units of the goods at the auction for no less than p_j price units per unit of the goods. The prices should always be nonnegative.

Given a list of quantity-point points that comprise the agent's bid string, say, $((-2\ 40)(3\ 20)(1\ 10))$, the semantics of the bid are "I am willing to sell 2 units if the price is \$40 or more; I am willing to buy 3 if the price is \$20 or less; and I am willing to buy 1 (in addition to the 3) if the price is \$10 or less."

Note that, it is illegal to place a bid in which the agent would sell goods to itself. For instance if an agent placed the bid $((-1\ 10)(1\ 20))$ it could potentially sell 1 unit to itself at a price between \$10 and \$20.

When a bid is matched at the AuctionBot, the bid string changes to reflect the match. For instance, if an agent places the bid $((-1\ 10)(-1\ 20))$ and sells one unit of the goods for \$10, then the bid string would become $((-1\ 20))$.

Flight Auctions. Flight auctions are continuous one-sided auctions, and close at the end of the game. Agents may submit buy bids, but not sell bids. Only the TACAIR seller may submit sell bids.

Price quotes are issued immediately in response to new bids. The price quote is specified as the ask price, which is simply the price of the current sell bid.

Flight auctions clear continually. Any buy bid points that are at least as high as the current ask price will match immediately at the ask price. Any buy bids that do not match immediatly remain in the auction as standing bids. A standing buy bid remains in the auction until it is matched by a sell bid with a price at or below the buy bid, or until the bid is withdrawn. A standing buy bid matches at the price of the buy bid. For instance, assume that TACAIR places a sell bid of $((-64\ 300))$, then:

- A bid of $((5\ 370))$ would match five units at \$300 each
- A bid of $((3\ 290))$ would not match, and would remain standing in the auction. A subsequent sell bid at \$290 or lower would match three units at \$290 each from the buy bid

- A bid of ((2 370)(3 290)) would match two units at $300 each. Since the entire bid would not match, the remaining portion, ((3 290)), would remain standing in the auction

Hotel Auctions. Hotel auctions are standard English ascending multi-unit, except that they close at randomly determined times. Specifically, at 04:00 (four minutes) into the game, one randomly chosen hotel auction will close. Another hotel auction closes each minute thereafter, on the minute, again chosen randomly, until 11:00 when the last hotel auction is closed. The agents cannot tell in advance which hotel auction will close at which time. A hotel auction clears and matches bids only once, when it closes. Price quotes are only generated once per minute, on the minute.

Agents may submit buy bids, but not sell bids. Only the hotel owners may submit sell bids. The hotel owners submit bids to provide up to 16 rooms of each hotel type on each night, for a minimum price of $0.

Price quotes are issued every minute, on the minute. The price quote is the ask price, calculated as the 16th highest price among all buy and sell bid units. For instance, if the following bids were in a hotel auction, the ask price would be $0:

$$((-16\ 0)), ((2\ 4)(6\ 6)), ((4\ 8)).$$

If however, the following bids were in a hotel auction, the ask price would be $6:

$$((-16\ 0)), ((2\ 4)(6\ 6)), ((4\ 8)), ((7\ 10)).$$

When agents submit new bids, they must “beat the quote”, according to the following rules:

> Let ASK be the current ask quote (16th highest price). Any new bid b must satisfy the following conditions to be admitted to the auction:
>
> 1. b must offer to buy at least one unit at a price of $ASK + 1$ or greater.
> 2. If the agent’s current bid *b'* would have resulted in a purchase of q units in the current state, then the new bid b must offer to buy at least q units at $ASK + 1$ or greater.

Agents may not withdraw bids from hotel auctions.

When a hotel auction clears, the 16 highest price buy units will be matched and the agents will pay the ask price for the hotel rooms. For instance, assume the following bids were in a hotel auction at the closing time:

- Hotel bid: ((-16 0))
- *Agent 1*: ((8 2))
- *Agent 2*: ((2 4)(6 6))
- *Agent 3*: ((4 8))
- *Agent 4*: ((7 10))

In this example, *Agent 4* would win 7 rooms, *Agent 3* would win 4 rooms, *Agent 2* would win 5 rooms, and *Agent 1* would not win any rooms. The price of the rooms would be $6.

Entertainment Ticket Auctions. The entertainment ticket auctions are standard continuous double auctions (much like a stock market) that close when the game ends. Agents may submit bids with buy and/or sell points (so long as a bid does not specify that the agent sell to itself).

Entertainment ticket auctions clear continuously. Bids match immediately, if possible. A bid that does not completely match remains standing in the auction.

Buy bid points will immediatly match the lowest price standing sell bid points that have prices at or below the price of the buy bid. Sell bid points will immediately match the highest price standing buy bid points that have prices above the price at or below the sell bid. Bids match at the price of the standing bid in the auction.

Price quotes are issued immediately in response to new bids. The price quote is specified as the bid and ask price. The bid price is the price of the highest standing buy point. The ask price is the price of the lowest standing sell point.

As an example, consider the following standing bids in an entertainment ticket auction:

- ((-1 100))
- ((-4 90)(-2 50))
- ((-6 60))
- ((1 40)(3 10))
- ((1 20))

The bid price is \$40 and the ask price is \$50. The following examples show what would happen if a new bid entered the auction:

- A new bid of ((-1 45)) would not match and would become a standing bid. The ask price would become \$45
- A new bid of ((3 48)) would not match and would become a standing bid. The bid price would become \$48
- A new bid of ((6 70)) would match two units at \$50 each and four units at \$60 each. The ask price would become \$60
- A new bid of ((-3 15)) would match one unit at \$40 each and one unit at \$20. Since this bid wouldn't match completely, the remaining portion would remain standing in the auction as ((-1 15)). The bid price would become \$10 and the ask price would become \$15

2.2.4 Client Utility Function

Clients specify their preferences by:

- A single preferred arrival date (PA)
- A single preferred departure date (PD)
  - The arrival and departure preferences for each client are chosen randomly such that all possible pairs of arrival/departure days are equally likely.
- A premium value for upgrading to the better hotel (HP)
  - Hotel premium values are chosen for each client uniformly in the range \$50 to \$150.
- A premium value for each type of entertainment event (AW, AP, MU)

- Entertainment premium values are chosen uniformly in the range \$0 to \$200 for every entertainment type.

Let a travel package be specified by:

- An actual arrival date (AA)
- An actual departure date (AD)
- Tampa Towers indicator ($TT?$ in $\{0, 1\}$)

A travel package is *feasible* if it contains rooms for every night between the arrival and departure dates. The soonest that any client can leave is one day after arrival. Note that it will never make sense to give a client a hotel for the day of departure. Also, the hotel bonus that a client gets for staying in the Tampa Towers is a bonus for the whole trip, not per night.

An entertainment package is a set of event tickets consisting of no more than one ticket of each type. For the purposes of computing the utility of an entertainment package, we use an abstract representation:

- A ticket indicator for each event type ($AW?$, $AP?$, $MU?$, each in $\{0, 1\}$)

An entertainment package is feasible if none of the tickets are for events on the same day, and all of the tickets coincide with nights the client is in town. Also note that the client does not get additional utility for attending the same type of entertainment more than once during their trip.

A client's utility, measured in dollars, from a feasible travel package and a feasible entertainment package is given by the following:

$$u = 1000 - travel\_penalty + hotel\_bonus + fun\_bonus$$

where

$$\begin{aligned} travel\_penalty &= 100 * (|AA - PA| + |AD - PD|) \\ hotel\_bonus &= TT? * HP \\ fun\_bonus &= AW? * AW + AP? * AP + MU? * MU \end{aligned}$$

A client receives zero utility for an infeasible package.

The following table shows the range of possible utilities for a feasible package assigned to a particular client.

```
                  Arr    Dep   Hotel   AW    AP    MU

Umin = 1000 - 300   - 300  +    0  +  0 +    0 +    0 =  400

Umax = 1000 -   0  -   0  + 150 + 200 + 200 + 200 = 1750
```

2.2.5 Final Score

At the end of the game, the travel agent holds several plane tickets, hotel rooms and event tickets. If it ends the game holding negative balances of any entertainment tickets (because it sold tickets it did not have), it is assessed a penalty of 200 for each ticket owed.

The TAC scorer allocates the agent's travel goods to its individual clients in order to construct feasible trips. Value for a particular allocation is the sum of the individual client utilities. The agent's final score is the value of the allocation of the goods to clients, minus the travel agent's expenses, minus a penalty for negative entertainment balances (if applicable). The scorer attempts to construct an optimal allocation, and usually succeeds or comes very close.

2.3 Competition Policies

Beyond the rules of the game itself, competition organizers reserve the right to disqualify agents violating the spirit of fair play. In particular, the following behaviors are strictly prohibited, as stated on the TAC web site [18]:

- Trading designed to benefit some other agent at the expense of the trader's own utility
- Communication with the agent during a game. Agents may obtain runtime game information only via the defined auction API
- Denial-of-service attacks. Agents may not employ API operations for the purpose of occupying or loading the game servers

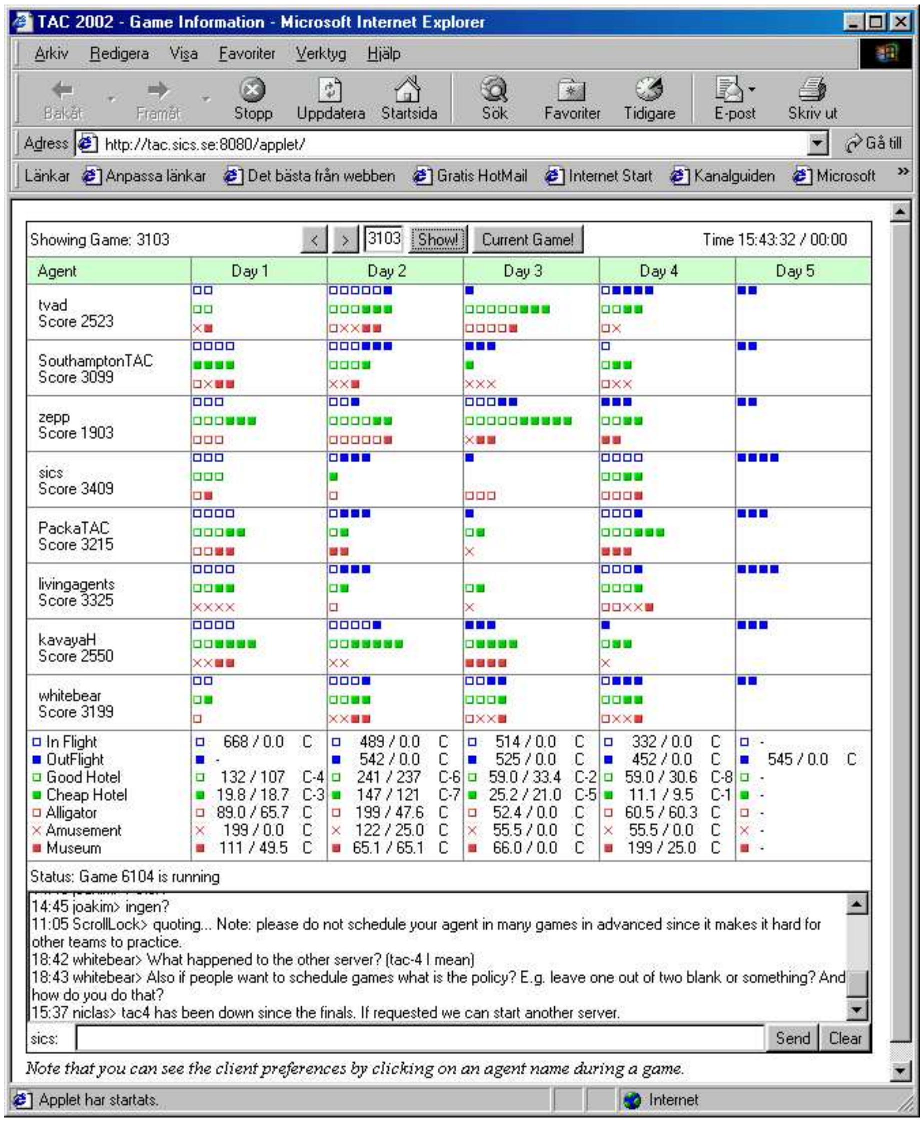
These policies clearly exclude all types of behavior where agents receive information from sources other than the TAC server. Examples of such behavior could be humans sending input to an agent based on the current status of the game, or agent that somehow receive runtime information in any other way than through the API, e.g., by extracting information sent to the *Game Viewer* applet. Although effective in theory, verifying that agents actually follow these policies is by no means an easy task, as agents participating in the competition are executed locally from all around the world. By not offering any form of prizes to winning agents, the willingness to use prohibited methods is minimized.

2.4 Visualization

The progress of a game can be monitored using the *Game Viewer*, see figure 2, which is an applet available at the TAC web site. The viewer displays current *ask* and *bid prices* for each auction, as well as the current status of the auction (closed or open). For each agent, their goods currently held in each auction are displayed as small squares, and their *Hypothetical Quantity Won* (HQW), which is what the agent would get in that auction if it had closed when the update was made, is indicated as blinking squares. The viewer is updated once every minute. The total net spenditure for each agent is also displayed, and when the game is over the score is displayed. The client preferences for a specific agent can be shown by clicking on the agents name. It is also possible to view a previously played game, instead of current. The viewer is very helpful during development of agents, since it makes it possible to visually monitor the progress of a game and the agents behavior. A countless number of hours has been spent, watching the viewer, by most of the participants in TAC.

2.5 AgentWare

To facilitate the development of TAC agents, an *AgentWare* for Java has been provided by the organizers, available for download at the web site. The AgentWare allow developers of TAC agents to stay focused on developing the strategy. Features of the AgentWare include:



TAC 2002 - Game Information - Microsoft Internet Explorer

Adress http://tac.sics.se:8080/applet/

Showing Game: 3103 — 3103 — Time 15:43:32 / 00:00

Agent	Day 1	Day 2	Day 3	Day 4	Day 5
tvad Score 2523	□□ □□ ×■	□□□□□■ □□□■■■ □××■■	■ □□□□□■■■ □□□□■	□■■■■ □□■■ □×	■■
SouthamptonTAC Score 3099	□□□□ ■■■■ □×■■	□□□■■■ □□□■ ××■	■■■ ■ ×××	□ □■■ □××	■■
zepp Score 1903	□□□ □□□■■■ □□□	□□■ □□□□■■ □□□□□■	□□□■■ □□□□□■■■■■ ×■■	■■■ □□■■ ■■	■■
sics Score 3409	□□□ □□□ □■	□■■■ ■ □	■ □□□	□□□□ □□■■ □□□■	■■■■
PackaTAC Score 3215	□□□□ □□□■■ □□■■	□■■■ □■ ■■	■ □■ ×	□□□■ □□□■■■ ■■■	■■■
livingagents Score 3325	□□□□ □□■■ ××××	□■■■ □■ □	 □■ ×	□□□■ □□□■ □□××■	■■■■
kavayaH Score 2550	□□□□ □□■■■■ ××■■	□□□□■ □□■■■■■ ××	■■■ □■■■■ ■■■■	■ □■■ ×	■■■
whitebear Score 3199	□□ □■ □	□□□■ □□■■ ××■■	□□■■ □□□■ □××■	□■■■ □□■■ □××■	■■
□ In Flight ■ OutFlight □ Good Hotel ■ Cheap Hotel □ Alligator × Amusement ■ Museum	□ 668 / 0.0 C ■ - □ 132 / 107 C-4 ■ 19.8 / 18.7 C-3 □ 89.0 / 65.7 C × 199 / 0.0 C ■ 111 / 49.5 C	□ 489 / 0.0 C ■ 542 / 0.0 C □ 241 / 237 C-6 ■ 147 / 121 C-7 □ 199 / 47.6 C × 122 / 25.0 C ■ 65.1 / 65.1 C	□ 514 / 0.0 C ■ 525 / 0.0 C □ 59.0 / 33.4 C-2 ■ 25.2 / 21.0 C-5 □ 52.4 / 0.0 C × 55.5 / 0.0 C ■ 66.0 / 0.0 C	□ 332 / 0.0 C ■ 452 / 0.0 C □ 59.0 / 30.6 C-8 ■ 11.1 / 9.5 C-1 □ 60.5 / 60.3 C × 55.5 / 0.0 C ■ 199 / 25.0 C	□ - ■ 545 / 0.0 C □ - ■ - □ - × - ■ -

Status: Game 6104 is running

```
14:45 joakim> ingen?
11:05 ScrollLock> quoting... Note: please do not schedule your agent in many games in advanced since it makes it hard for
other teams to practice.
18:42 whitebear> What happened to the other server? (tac-4 I mean)
18:43 whitebear> Also if people want to schedule games what is the policy? E.g. leave one out of two blank or something? And
how do you do that?
15:37 niclas> tac4 has been down since the finals. If requested we can start another server.
```

sics:

Note that you can see the client preferences by clicking on an agent name during a game.

Applet har startats.

Figure 2: The Game Viewer.

- Automatic connection, login and retrieval of game data
- Asynchronous communication with the TAC server
- Latest quotes from all auctions
- Bid monitoring of active bids
- Bookkeeping of transactions so that the agent knows what it own
- GUI for showing the internal state of the agent, bids, ownership, etc

The AgentWare minimizes the entry barrier for new participants in TAC. It was used by several entrants this year, including me.

3 The SICS Agent

3.1 Background

Last year, SICS entered TAC-01 with an agent, *006*, based on Constraint Programming [1]. The agent was somewhat quickly written during the qualifying and seeding rounds, and therefore far from fully explored with respect to strategies and price prediction. The main strategy of *006* was to only make repeated locally optimal adjustments to the goods it currently held, with the help of an optimizer, based on constraint programming. My ambition was to continue their work and to further explore the benefits of using the constraint programming approach.

3.2 Implementation and Design

The SICS agent was implemented completely in Java, using the TAC AgentWare. Design and coding started about one week before the first competition round, and continued until the end of the seeding rounds; in total no more than five weeks. The agent that played the semi-finals, emerged first during the last days of the seeding rounds. No further development was made after the seeding rounds due to the very limited testing opportunities.

One of the key elements when doing repeated locally optimal adjustments is to predict the closing prices as good as possible, which is where my focus came to be. The original idea was to use Machine Learning (ML). However, the shortage of time made a sophisticated ML approach far too ambitious, so instead a very simple form based on statistics and heuristics was used.

Since there are three different kinds of markets — flights, hotels and entertainment tickets — which differ in various ways, the strategy was divided into three parts, one for each kind of market.

To facilitate simple testing and use of different strategies, a plug-in interface was put between the AgentWare and the strategy classes. In this way, what strategies to use could be specified during startup, in a configuration file, without any need for recompilations. It would even be possible to change strategy during runtime, but this feature was never used. The design of the agent appears in figure 3.

3.2.1 Testing and Debugging

The final performance of SICS is a result of continuous fine-tuning during the development, based on the outcomes of the many TAC games played. In order to analyze the performance of the agent, all details of its behavior

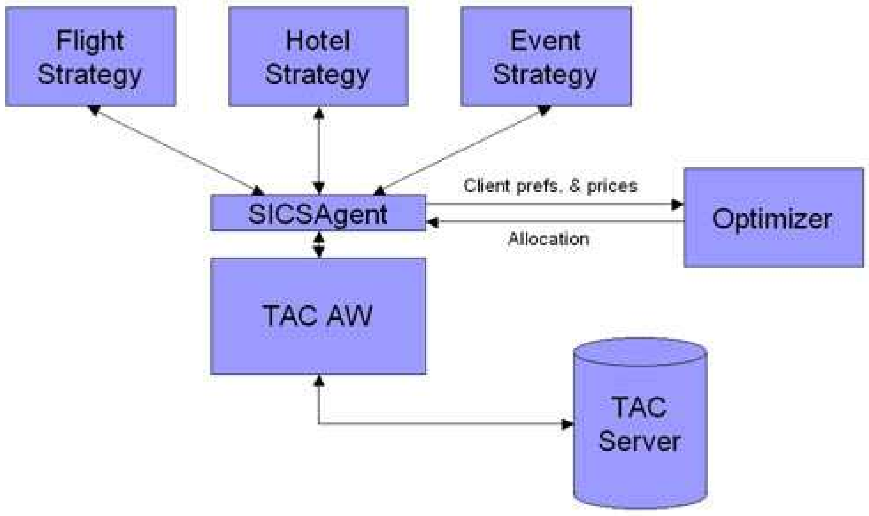


Figure 3: The agent design.

in each game is written to a log file. It is also possible to follow a game in real-time with the help of the Game Viewer and the graphical agent interface included in the TAC Agent Ware. There are also log files, containing all transactions made during a game, available on the TAC web site [18].

Developing and debugging a TAC agent is an extremely difficult and time-consuming process. The main reason is that evaluating even the smallest change in the code entails playing a 15-minute game instance, and most of the time even that is not enough, since the conditions for the game could be entirely different from game to game. If the behavior of the agent during some special kind of circumstances, for example expensive hotel rooms on day one, were to be tested it could take hours before that situation happened again. As a result, all parts of the agent's bidding strategy could not easily be tested or optimized, as the necessary conditions occurred only a few times in the games played before and during the competition.

To test the performance of an agent it needs to play against a representative composition of opponents. This can almost only be done during real competition rounds, unless all the opponents run their agents on the training server at the same time.

Another issue that complicated the development and testing was the fact that the TAC AgentWare was under development, and had never been used or properly tested before, which meant it did not always behave as expected. Numerous bugs were found in it, sometimes after hours and hours of debugging. Further was the optimizer sometimes very unreliable. It could under certain circumstances take up to several minutes, sometimes hours, before it came to an acceptable allocation. Since it was interrupted after only 20 seconds, the allocation could sometimes lead to disastrous behavior from the agent, or it could even fail to finish.

3.3 Execution Loop

The main strategy of SICS was, as for 006, to only make repeated locally optimal adjustments to the goods it currently held. The strategy can be summarized in the following steps:

1. **Collect price quotes**
   Get updated price quotes from the auctions.
2. **Compute Marginal Costs**
   For each type of goods, calculate a marginal cost vector $p = (p_0, \ldots, p_8)$, where p_i is the cost of changing from the current amount to i. If we already have i units of the goods, $p_i = 0$. If we need to buy goods to get i units the cost is positive, and if we have to sell it is negative. If the goods cannot be bought (perhaps because the auction has closed), the cost is set to infinity, and when it cannot be sold, it is set to zero. Since the agent does not know the true marginal costs p_i, the cost has to be estimated. The more accurate the estimates are, the better the agent performs.
3. **Compute Allocation**
   The optimizer computes the optimal amounts of goods (allocation) to hold, given that the goods can be acquired at the marginal costs above. The optimizer is an anytime algorithm that produces increasingly better allocations. It was run for 20 seconds.
4. **Bid**
   The bids are computed from what we already hold, what is currently bid for, and what is required from the allocation step above. The maximal price to bid for the resources is generally the amount that was specified in the marginal cost vector.
5. **Iterate**
   Repeat the steps above once every minute, right after the quote updates, until the game is over.

3.4 Flight Strategy

The flight strategy used is very simple. Bids on flight tickets are always placed according to the allocation returned by the optimizer, and bid price is always set to \$1000 to make sure to get the ticket. The price per ticket sent to the optimizer is simply current ask price.

Sometimes it would be favourable to postpone bying some tickets, since it could be very expensive if the hotel prices raise on a particular day. Bying all flight tickets at the start of the game means committing to buying certain hotel rooms, or buying extra tickets later on, which could both turn out to be expensive. On the other side, buying flight tickets later in the game generally means having to pay more for them, since flight prices tend to go up.

One approach could be to do a risk or cost/benefit analysis of buying flight tickets at the current stage versus at a later stage of the game. The tickets that are most likely to be used are bought immediately, and those that are not so likely to be used are postponed. In this way the trade-off between buying less expensive tickets early, and avoiding commitments at an early stage of the game, is more properly taken into account.

I never had the time to fully experiment with this approach, and at the same time I made the analysis that a badly designed approach would do more harm then good. Therefore, all flight tickets were bought as early as possible in the game, and occasionally some extra tickets had to be bought at a later stage in the game.

3.5 Hotel Strategy

The hotel strategy is the most important strategy, and therefore the one that the most effort was put into. The motivation is that the hotel rooms are the most limited type of goods, and at the same time of great importance. In total 64 clients compete for only 32 rooms each day, and without the required hotel rooms the package becomes infeasible and gives zero utility.

The strategy is to try to predict the closing prices for the hotel auctions as accurately as possible, and then place bids as if the prediction was correct. The prediction procedure is outlined below. Every time new quotes are available a new prediction is made and the bids are updated. The bidprice is set to be 20 percent above the predicted price, just as an extra safety marginal. This is justified by the fact that most of the time the price that the agent actually end up paying is lower than its bid, according to the 16th highest bid rule.

3.5.1 Price Prediction

The price prediction is the central part of the hotel strategy. Predictions are based on simple heuristics made after observing game data, and statistics from previous games. The predicted unit price for n rooms is made with the following formula:

$$newPrice(n) = max(currentPrice*incF(time), startPrice*compCons(n))$$

where

- *currentPrice* is the price that is valid at the moment, the ask price.
- *startPrice* is based on statistics from previous games.
- *incF* is the function that models the expected price increment.
- *compCons* is compensating for the raised competition that bidding for this many items will lead to.

Start Prices. The start prices are mainly used in the beginning of a game, before any useful information is revealed by the auction quotes. Most agents wait until right before the fourth minute to place their first bids, since that is when the first hotel auction closes. Placing bids earlier only means revealing useful information to the opponents. This means virtually that the only information available is game data from previously played games.

The start prices have two important roles. One is to make sure that the initial bid is high enough when the first hotel auction closes, and the other is to give a good estimate about the closing prices when the decision about the flight tickets is made. To take the first one into account, the price is chosen so that $F_{i,4thmin}(s_i) > 0.9$, where $F_{i,4thmin}$ is the density function for the auction i at the fourth minute, and s_i is the start price for auction i. This means that in 90 percent of the games the bid will be high enough. The other is taken into account by choosing the price to be as close to the average closing price as possible.

The statistics used for the start prices is based on the 100 last games. During the competition the start prices were updated manually now and

Auction	k
TT, day 1	0.7
TT, day 2	0.8
TT, day 3	1.0
TT, day 4	0.7
SS, day 1	0.5
SS, day 2	0.8
SS, day 3	0.8
SS, day 4	0.5

Table 1: k-value used in the increment function for each auction.

then, but support for automated update can easily be built into the agent. No update was made during the semi-finals, a decision made because of the limited number of rounds, and therefore limited supply of data.

Increment Function. Since hotel auction quotes are updated only once a minute, and one auction closes every minute, it is of great importance to place bids with a sufficient margin to the ask price. Otherwise the bid may fall outside the top 16 at the next update, which could be disastrous if the auction closes at the same time.

The first approach tried was simply to do exponential increments, e.g. always bid at least double the ask price. This was high enough to stay among the 16 highest, but a big problem was that the bidding price was most of the time much higher than the actual closing price — what the agent would end up paying — hence heavily overestimated. A problem with overestimated prices is that the optimizer sometimes is fooled to change allocation, due to the high price, when it would not be necessary if the real price was known.

Simply choosing a smaller exponential weight, or do linear increments instead of exponential will clearly not do, but a better model of the price development is needed. According to statistics from old games, prices normally rise quickly the first couple of minutes after the fourth minute, and about eight minutes into the game they more or less tend to stabilize. Therefore the model used in the agent is the sigmoid function:

$$incF(t) = k * \frac{1}{1 - e^{(t-8)}}$$

where t is the game time in minutes and k is a constant. The different values for k is shown in table 1. To prevent the increment function to be to low, a lower limit was set to 0.2, hence:

$$incF(t) = max(k * \frac{1}{1 - e^{(t-8)}}, 0.2)$$

Competition Consideration. The price in a hotel auction depends highly on the demand for that particular kind of goods. If many clients want to stay a particular day, the demand will be higher. If the demand is higher than 16 rooms in one auction, there will be competition amongst the agents, and the price will most likely rise quickly. How high depends on how highly the agents value the goods. To take this into account the predicted price per item depends on how many items the agent will bid for in the auction, and increases exponentially with respect to that, like

$$compCons(n) = k^{n-2}$$

where n is the number of items bid for, and k is a constant. The constant k was set to 1.15 for day 1 and 4, and 1.25 for day 2 and 3.

If we divide the available 32 rooms per day (16+16) amongst the eight agents, each is allotted 4 rooms per day (2+2). This means that it should be fairly safe to bid on 2 or less rooms per auction, since if the price rises high, there will always be other agents paying equally much or even more. Hence the term $n-2$.The competition consideration only affects the start price, since when real bids have been placed by all agents the ask price already reflects the demand.

Closed Auction. If the demand in an auction is higher than what is available, and the auction closes, then some of the agents that did not get the rooms they wanted will probably move to the other auction for same day. Without hotel rooms the clients will not be able to get an feasible package unless they change travel dates, which might involve buying extra flight tickets. When some agents move their clients from one hotel to the other, it will lead to higher demand, and consequently higher prices in the other auction. The prices will most likely be at least as high as they were willing to pay for the first hotel, the *bid price*. To take this into account, when an auction closes, the other auction for the same day will use the following price as *currentPrice* when the price prediction is made:

$$currentPrice = max(askP, bidP_c * \frac{(12-t)}{8})$$

where $bidP_c$ is the bid price from the closed auction, and t is the game time in minutes. As can be seen in the formula, the closed auctions bid price will have less impact in the end of the game. This is motivated by the fact that the more auctions that have closed, the less flexible the agents are to change from one type of hotel to another.

3.5.2 Hypothetical Quantity Won (HQW)

Every time the agents receive a new quote update, the quote contains a *Hypothetical Quantity Won* (HQW) for each auction. The HQW is the number of items the agent would have won in the auction if it had closed at the same time as the update was made. When the agent places a new bid, it is obligated to bid for at least the HQW, otherwise the bid will be rejected. Since no resale of hotel rooms is permitted, there is always a risk in changing from one allocation to another when the agent has HQW according to the first allocation. There is no guarantee that the agent will be able to drop out of the auction, as this can only be made if the other agents raise their bids. Therefore, the agent may end up with unused rooms if the allocation is changed.

To take this uncertainty into account the agent does not send the full price for rooms with HQW to the optimizer, since a fraction of the price can in a way, at least statistically, already be considered spent for the room. This effectively prevents the optimizer from changing allocation unless the benefit of changing is higher than the risk of doing so is.

3.6 Entertainment Ticket Strategy

Selling and buying entertainment tickets is an important way of increasing the client utilities and lowering the net expenditures, but far from trivial. The auctions clear continuously, which means predicting prices is hard.

A simple approach that was first tried was to use the current sell and bid price as predicted price. This did not work too well with the optimizer. Most agents use a linear price model, which means that they for example start selling for 200 and lower the ask price slowly until someone buys, and for buying they do the opposite. This means that using real prices prevents the optimizer from making a correct decision at an early stage, since most prices will change.

A far better approach, and even simpler, is to use fixed predictions based on the average clearing price. In this way the optimizer will base its decision on the same price all the time, the price that is most likely to be valid in the end. The predicted buy price was set a bit higher ($100) than the average, since there is no guarantee that someone will be willing to sell the ticket. For the same reason the predicted sell price was set a bit lower ($80).

Tickets are sold at a slowly decreasing price, starting fairly high, and bought at a slowly increasing price, starting fairly low. To make it harder for other agents to model the price, the prices are somewhat randomly chosen. Prices are set according to the following formulas:

$$\begin{aligned} sellPrice &= 50 + (150 - 100 * random()) * \frac{720 - t}{720} \\ buyPrice &= 50 + (70 - 50 * random()) * \frac{t}{720} \end{aligned}$$

where $random()$ generates a random number from the range $[0, 1)$ and t is the game time in seconds, cf. figure 4.

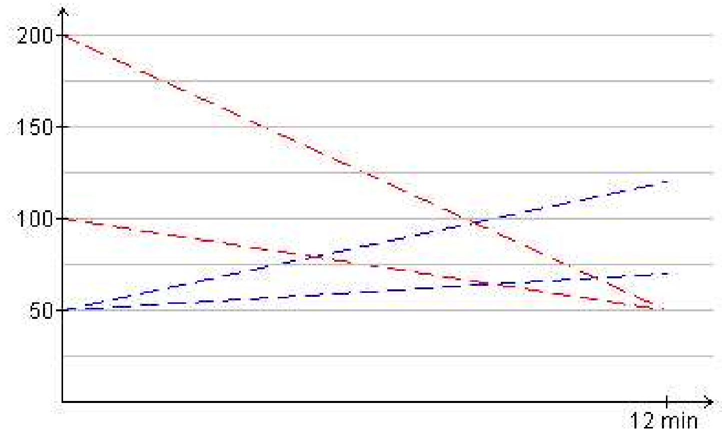


Figure 4: Upper and lower limit for sell (red) and buy (blue) prices for event tickets.

4 The Optimizer

4.1 Background

The task of the optimizer is to compose a trip package (allocation) for the clients so that the net agent utility is maximized. The net utility is the difference between the total client benefit and the total cost. In TAC'01, the *006* agent used an optimizer written in SICStus Prolog [14], based on constraint programming over finite domains (CLP(FD)) [1]. The exact same optimizer was initially used for the *SICS* agent in TAC'02. It however proved to be quite unreliable, sometimes returning unstable results or no result at all in a reasonable time. This resulted in that the agent sometimes bought flight tickets it had no use for, or did nothing at all, since no allocation was returned. The SICStus optimizer also seemed to concentrate too much on the client preferences, it sometimes did not change travel dates even though hotels were really expensive, as if it never had time to search deep enough to find that solution.

The solutions to the problems was a new optimizer, written in Java, by Joakim Eriksson. This optimizer was optimized with respect to the TAC problem and performed much better than the SICStus optimizer. It was not only faster, but also much more stable and more sensitive to prices, with respect to travel dates.

4.2 Communication

The optimizer was run as a separate process, and the communication between the agent and the optimizer was done using a TCP-based interface. The optimizer is started by a request from the agent, and then produces a stream of better and better approximations to the optimization problem, until either stopped or the optimum is found.

Input. The input to the optimizer consists of client preferences and price estimations in the form of marginal cost vectors, $p = (p_0, \ldots, p_8)$, one for each type of goods, where p_i is the cost of changing from the current amount to i. If we already have i units of the goods, $p_i = 0$. If we need to buy goods to get i units the cost is positive, and if we have to sell it is negative. If the goods cannot be bought (perhaps because the auction has closed), the cost is set to infinity ("sup"), and when it cannot be sold, it is set to zero.

Output. The output is a stream of increasingly improved allocations. The allocations have arrival and departure days for each client, which hotel to stay at, and what events to do. There is also the utility and the time spent by the optimizer to find the solution.

4.3 Search Procedure

For TAC, the search procedure should have the property that it quickly finds a first solution, and then keeps finding progressively better and better solutions. To accomplish this, the optimizer performs branch-and-bound search. All variables are assigned in a completely general way by the optimizer; there are no rules of thumb. Ordering heuristics are crucial for performance.

Variable order. The problem variables are assigned in the order: (i) arrival and departure dates, and hotel rooms; and (ii) entertainment tickets.

Client order. Before the search starts, clients are sorted with respect to arrival and departure dates (A, D) in the following order: $(1, 2)$, $(4, 5)$, $(2, 3)$, $(3, 4)$, $(1, 3)$, $(3, 5)$, $(2, 4)$, $(1, 4)$, $(2, 5)$, $(1, 5)$. That is, clients with the shortest preference of stay are handled first.

Value order. Let the *estimated value* $t(X = v)$ of an assignment $X = v$ be computed as the best possible benefit when that value is assigned. When arrival and departure dates (A, D) and hotel type H are going to be assigned, the optimizer computes the estimated value of each combination of values for A, D and H, and tries the value tuples by descending $t(A = a, D = d, H = h)$.

No special ordering is made when entertainment tickets are assigned.

5 Competition Results

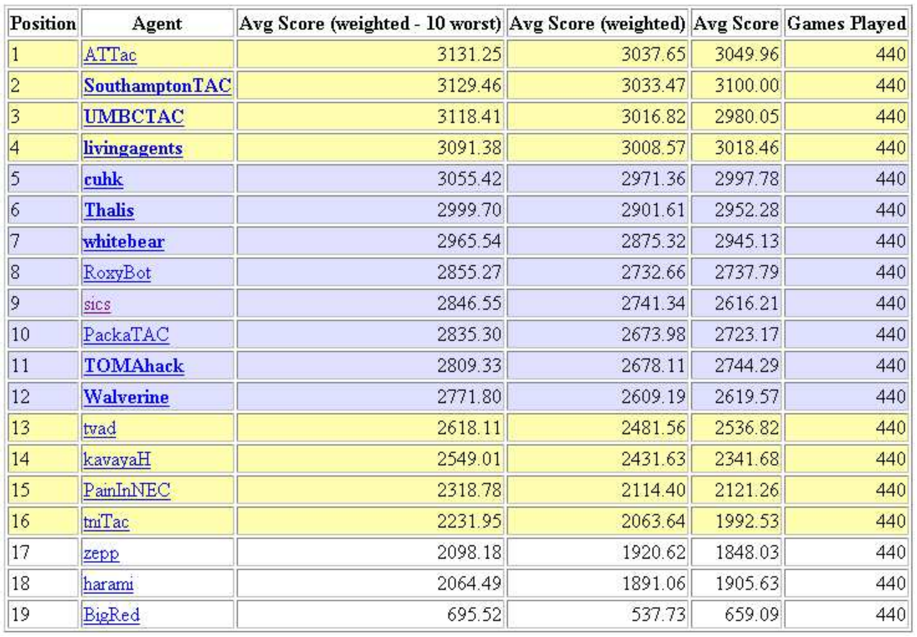
5.1 Introduction

This chapter presents the final results of TAC, and provides a general discussion regarding the results of SICS. The focus of the discussion is the evaluation of the strategy employed by SICS, and motivations for the outcome of the competition. Comparisons are made with other agents to highlight the good and bad parts of the strategy.

The final competition was held during the TAC workshop at the Computers and Games conference (CG'02) in Edmonton, on July 28, 2002. Several games were played during the day, including semi-finals and finals, and at the end of the day a winner was determined based on the cumulative scores for the games played in the finals. In addition, some of the participating teams also contributed with brief presentations of their agent designs throughout the day, revealing many different and very interesting approaches.

5.2 Seeding Rounds

The seeding rounds were held as an event over the Internet in the beginning of July, a couple of weeks before the final workshop in Edmonton. In total, 440 games were played by each agent, in the form of a round-robin tournament over a period of approximately twelve days. In total 19 agents were participating in the seeding rounds and ultimately the 16 best performing agents, based on the cumulative scores, were selected for the semi-finals at the TAC workshop. The scores were weighted with a weight that increased for each day during the seeding round, which enabled agents that were under development to climb in the results table by performing well during the last days. The SICS agent underwent a considerable change during the seeding rounds and was under constant development. From being one of the worst performing agents it finally reached a level of performance that was quite satisfying, which resulted in a climb from the 18th position to the 9th in the last days of the seeding rounds.



Position	Agent	Avg Score (weighted - 10 worst)	Avg Score (weighted)	Avg Score	Games Played
1	ATTac	3131.25	3037.65	3049.96	440
2	SouthamptonTAC	3129.46	3033.47	3100.00	440
3	UMBCTAC	3118.41	3016.82	2980.05	440
4	livingagents	3091.38	3008.57	3018.46	440
5	cuhk	3055.42	2971.36	2997.78	440
6	Thalis	2999.70	2901.61	2952.28	440
7	whitebear	2965.54	2875.32	2945.13	440
8	RoxyBot	2855.27	2732.66	2737.79	440
9	sics	2846.55	2741.34	2616.21	440
10	PackaTAC	2835.30	2673.98	2723.17	440
11	TOMAhack	2809.33	2678.11	2744.29	440
12	Walverine	2771.80	2609.19	2619.57	440
13	tvad	2618.11	2481.56	2536.82	440
14	kavayaH	2549.01	2431.63	2341.68	440
15	PainInNEC	2318.78	2114.40	2121.26	440
16	tniTac	2231.95	2063.64	1992.53	440
17	zepp	2098.18	1920.62	1848.03	440
18	harami	2064.49	1891.06	1905.63	440
19	BigRed	695.52	537.73	659.09	440

Table 2: The table shows the results from the seeding rounds. Final scores are calculated as weighted average minus the ten worst games. Agents with yellow background is in one semi-final group and blue in the other.

The results from the seeding round can be seen in table 2. They should be taken with a grain of salt since most agents were still under development during the seeding rounds.

5.3 Semi-finals and Finals

The semi-finals were played during the TAC workshop in Edmonton. Two heats were played, with eight agents in each, which competed in a total of 14 games each. After the semi-finals, the top four agents in each heat qualified to the finals that were played later the same day. The finals were played on two servers in parallel, and a total of 32 games.

SICS finished seventh in its semi-final heat, and therefore did not qualify to the finals, see table 3. As can be seen, the semi-final rankings clearly show the nearly negligible difference in the final score of the eight agents, with the average score of the one in the first place being only $320.43 higher than that of the eighth place. The result table suggests that there is no significant difference in the agents' performance.

A clear correlation between the average utility and the standard deviation can be seen among the agents in table 3. Generally a higher utility seems to give more unstable results, while a lower utility gives more stable results. A high utility means that the agent follows the client preferences very strictly, while a lower utility is the result of moving clients. SICS has the second lowest utility, and at the same time the second lowest standard deviation, wich is a result of the risk averse strategy. The Competition Consideration function used in the price prediction is effectively avoiding having to many clients staying the same day, which means having to move

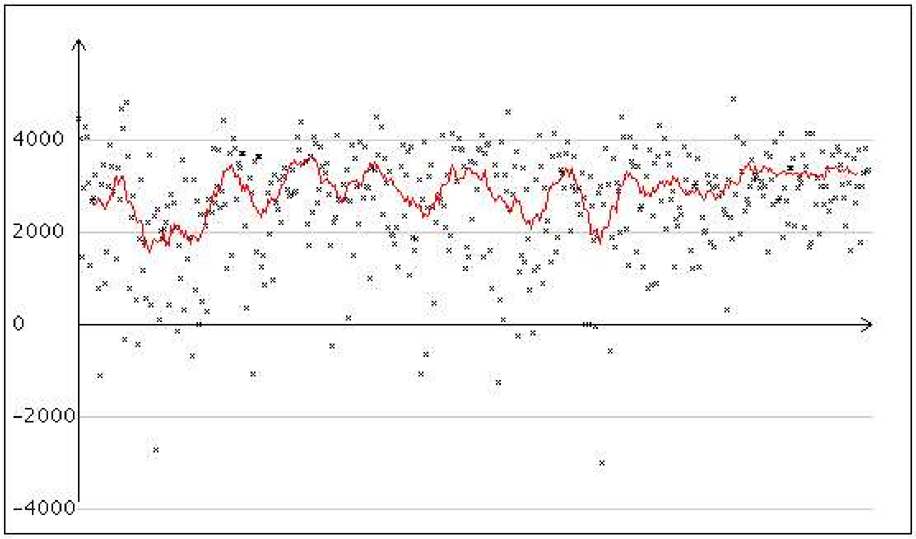


Figure 5: The graph shows the scores for SICS in the games played during seeding rounds. The red line is the moving average of the ten last games. As can be seen, the performance was considerably improved in the end.

Pos	**Agent**	**Score**	**Avg Utility**	**Std Dev**	**Min,Max**
1	whitebear	3441.66	9757.35	890.35	(1791.56,4775.91)
2	Walverine	3423.06	9692.42	1093.17	(1514.79,4383.37)
3	Thalis	3356.78	9687.85	901.32	(1153.77,4621.08)
4	cuhk	3353.54	9409	768.13	(2130.11,4244.12)
5	PackaTAC	3353.39	9580.85	893.71	(1911.88,4418.41)
6	RoxyBot	3309.46	9650.14	989.48	(1691.06,4373.52)
7	sics	3253.70	9225.21	717.47	(1749.78,4216.18)
8	TOMAhack[1]	3121.23	9192.92	615.95	(-777.92,4145.20)

Table 3: The results from semifinal heat 2.

Hotel	Seeding rounds	Semifinals
TT day 1	82	49
TT day 2	194	128
TT day 3	159	68
TT day 4	110	143
SS day 1	20	16
SS day 2	134	87
SS day 3	125	47
SS day 4	44	15

Table 4: Average closing prices for the hotel auctions in the last 100 games in the seeding rounds and the semi-finals heat 2. The closing prices from the seeding round were the ones that SICS was tuned for during the semifinals.

Pos	Agent	Score
1	SouthamptonTAC	3446.15
2	livingagents	3371.87
3	kavayaH	3300.02
4	UMBCTAC	3251.89
5	tniTac	3195.26
6	ATTac	3136.89
7	tvad	2923.43
8	PainInNEC	2279.27

Table 5: The results from semi-final heat 1.

some clients, and therefore getting a lower utility.

A study of the average closing prices from the seeding rounds compared to the ones from the semi-final heat 2, cf. table 4, reveals a significant difference. In the seeding rounds the prices were generally much higher, especially for the two middle days, with only one exception — good hotel, day four. This means that it would in fact pay off to, for example, move a client that wants to stay day one through day five, to only stay day one through day two, or day four through day five, since the price for a hotel on day two and three is higher than the penalty for moving clients ($100/day). In the semifinals however, the only hotels worth avoiding were the good hotel (TT) day two and four. SICS used the prices from the seeding rounds in its price prediction and therefore moved the clients too much, wich led to a lower utility, and eventually a lower score, than what would have been the case if the correct prices were used instead. A probable explanation for the lower prices in the semifinals is that those agents that actually did a rescheduling were the ones that played best in the seeding rounds, and therefore reached the semi-finals. Then, when all agents more or less rescheduled their clients, the competition for the hotel rooms became lower, which was reflected by the prices.

Pos	Agent	Score
1	whitebear	3556.48
2	SouthamptonTAC	3492.03
3	Thalis	3351.23
4	UMBCTAC	3320.65
5	Walverine	3315.62
6	livingagents	3309.83
7	kavayaH	3249.86
8	cuhk	3247.83

Table 6: The results from the finals. Livingagents had a crash in two of the games, of which only one was deleted. If the other game would be stroken as well, they would instead have the score 3423.96, which would place them third.

5.4 Results Summary

As can be seen in the results tables, there were no significant difference in the agents performance during the finals and semi-finals. To be able to draw any conclusions from the score table, many more games would have to be played.

The prices for hotel rooms were much lower in the semifinals than in the finals. This was probably a result of the strategies of the participating agents. Most agents learned from the seeding rounds that it was too expensive to let clients stay for many days, and therefore did some rescheduling in the semifinals and finals. This led to lower competition, and hence lower prices. The low prices could be an explanation for the small difference in scores among the agents.

6 Conclusions and Further Directions

6.1 General Conclusions

What is a good strategy in one kind of environment, could be a bad strategy in another. In the seeding rounds it was a good strategy to move clients, but in the semifinals, it was a bad strategy. What makes it hard when choosing a strategy is that, whether it is good or bad depends partly on who the opponents are, and when the opponents are not known the choice is far from trivial. A good idea is probably to wait with buying some of the flights, the ones that are least likely to be needed if the hotel prices are different from expected. In this way the agent avoids commiting to a travelplan too early and can wait until more information about the real demand is known, but at the price of offering some of its score since the flight prices tend to rise. Whitebear, the best scoring agent in the finals, among others, used this strategy. More experiments needs to be done in this area.

When many games are played, like in the seeding rounds, there are good opportunities to use machine learning. ATTack, which was the top-scoring agent in the seeding rounds, used machine learning in its price prediction [13]. When a simple kind of machine learning was used in SICS's price prediction, the performance was effectively improved. In the semi-finals, however, the games were very limited and there was simply not enough data to do machine learning. At the same time the games were very different from the seeding rounds in terms of hotel prices, which could be an explanation for that both ATTack and SICS did not qualify to the finals. They were tuned for the wrong type of game. The conclusion to be drawn is that using machine learning, or statistics, is a good idea when the number of games is big, but not as good when the number of games is small.

The very limited number of games in the semifinals and the finals, and the small differences in the scores, prevents that any real conclusions about which strategy was the best to use can be drawn from it. The problem was discussed during the workshop and some suggestions were to play it for several days, or to play on several servers in parallel, like in the finals.

6.2 Directions for Further Research

Most of the effort was put into the hotel price prediction when developing SICS. This was partly a result of the limited time before the competition, but also because of the problems with the optimizer, which took more time than expected. This led to a very simple flight strategy and event ticket strategy. As was discussed earlier, the flight strategy could be worth working more on, especially in terms of the bidding strategy. A good idea might be to use decision theory and risk theory to make decisions about what tickets to buy early in the game, and what tickets to buy later. This could lead to an agent that makes these decisions based on estimations and probabilities of the possible outcomes of different scenarios.

Something else that was discussed but never tested is using the optimizer several times, but with different price estimates. By doing that, more knowledge about how different prices effects the allocation could be gained, and maybe used by the bidding strategy. For example it could sometimes be worth placing extra bids to give more flexibility to the agent. Decision theory and risk theory could be used here.

Other ideas discussed, but never tried, were to instead of bidding the same price for all rooms that was bid for in one auction, bidding lower for some and higher for some, and in this way ensuring getting at least some of them. In general, no such bidding strategies were elaborated with at all in SICS, except from waiting until between the third and fourth minute to place the initial hotel bid.

More experimenting should be done with respect to how to use the statistics in the agent. How often, and how should the parameters be updated? Would it have been effective to update the agent during the semifinals? These are questions that still have not been answered properly.

6.3 The Future of TAC

After the third international Trading Agent Competition, the interest in the area is still very high. The dynamics of the problem domain is still not fully explored or understood, and the optimal strategy is still an open question. Discussion regarding suggestions and improvements for the next event were held among the participants on the day of the workshop, and have after that continued on the TAC mailing list. Whether there would be a next event or not was never really questioned. As it seems like now, there will be two different leagues in TAC next year. One is TAC-classic, which is TAC with unchanged rules, and the other will be a completely new game, with a suply-chain scenario. The details of the new game are not yet decided.

Acknowledgements

I would like to thank my supervisor at SICS, Magnus Boman, for devoted support, and inspiration throughout the writing of the thesis. I would also like to thank him for giving me the opportunity to attend the Trading Agent Competition and the workshop in Edmonton, Canada. A special thanks also to Joakim Eriksson and Niclas Finne at SICS, for valuable advise, devoted support, and inspiration throughout the writing of the thesis. Thanks also to Joakim and Niclas for their commited work on maintaining the performance of the TAC server troughout the competition. Final thanks go to the participants of TAC and the workshop for sharing their experiences and ideas.

References

[1] E. Aurell, M. Boman, M. Carlsson, J. Eriksson, N. Finne, S. Janson, P. Kreuger and L. Rasmusson, *A Trading Agent Built on Constraint Programming*, In Eighth International Conference of the Society for Computational Economics: Computing in Economics and Finance.

[2] M. Boman, *Trading Agents*, AgentLink news 6:15-17, 2001.

[3] S-E. Ceedigh: *Trading Agents*, Master's Thesis, DSV, Royal Institute of Technology, 2000.

[4] M. Fasli, *Trading Agent Competition 2001*, AgentLink news 9:13-14, 2002.

[5] N. Fornara and L. M. Gambardella, *An Autonomous Bidding Agent for Simultaneous Auctions*, Fifth International Workshop on Cooperative Information Agents (CIA-2001) Lecture Notes on Artificial Intelligence LNAI 2182, pages 130-141, 2001.

[6] A. Greenwald and J. Boyan, *Bid Determination in Simultaneous Auctions: An Agent Architecture*, In Proc. of the Third ACM Conference on Electronic Commerce, pages 210-212, 2001.

[7] A. Greenwald and J. Boyan, *Bid Determination in Simultaneous Auctions: A Case Study*, In Proc. of the Third ACM Conference on Electronic Commerce, pages 115-124, 2001.

[8] A. Greenwald and P. Stone, *Autonomous Bidding Agents in the Trading Agent Competition*, IEEE Internet Computing, 5(March-April,2), 2001.

[9] M. He and N. R. Jennings, *SouthhamptonTAC: Designing a Sucessful Trading Agent*, Fifteenth European Conference on Artificial Intelligence, Lyon, 2002.

[10] P. L. Lanzi and A. Strada, *A Statistical Analysis of the Trading Agent Competition 2001*, In SIGecom Exchanges 3(2):1-8, 2002

[11] T. McDonald, *Robot Agents: Coming Soon to Software Near You* NewsFactor Network, Oct 24, 2001,
`http://www.newsfactor.com/perl/story/14332.html`, 2002-09-03

[12] RoboCup Official Site, `http://www.robocup.org`

[13] R. E. Schapire, P. Stone, D. McAllester, M. L. Littman and J. A. Csirik, *Modelling Auction Price Uncertainty Using Boosting-Based Conditional Density Estimation*, Nineteenth International Conference on Machine Learning, Sydney, 2002.

[14] SICStus Prolog, `http://www.sics.se/isl/sicstus/`

[15] P. Stone and A. Greenwald, *The First International Trading Agent Competition: Autonomous Bidding Agents*, Journal of Electronic Commerce.

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

[17] TAC 2000, `http://tac.eecs.umich.edu`, 2002.

[18] TAC 2002, `http://www.sics.se/tac`, 2002.

[19] TAC Research Reports
`http://auction2.eecs.umich.edu/researchreport.html`,
2002-09-03.

[20] M. P. Wellman, A. Greenwald, P. Stone, and P. R. Wurman, *The 2001 Trading Agent Competition*, In Fourteenth Conference on Innovative Applications of Artificial Intelligence, pages 935-941, Edmonton, 2002.

[21] M. P. Wellman and P. R. Wurman, *A Trading Agent Competition for the Research Community*, IJCAI-99 Workshop on Agent-Mediated Electronic Commerce, Stockholm, 1999.

[22] M. P. Wellman, P. R. Wurman, K. O'Malley, R. Bangera, S. Lin, D. Reeves and W. E. Walsh, *Designing the market game for a trading agent competition*, IEEE Internet Computing, 5(2):43-51, 2001.