# RoxyBot-06: An (SAA)$^2$ TAC Travel Agent

**Seong Jae Lee, Amy Greenwald, and Victor Naroditskiy**
Department of Computer Science, Brown University, Providence, RI 02912
{sjlee,amy,vnarodit}@cs.brown.edu

## Abstract

In this paper, we describe our entrant in the travel division of the 2006 Trading Agent Competition (TAC). At a high level, the design of many successful autonomous trading agents can be summarized as follows: (i) price prediction: build a model of market prices; and (ii) optimization: solve for an approximately optimal set of bids, given this model. To predict, we simulate *simultaneous ascending auctions*. To optimize, we apply the *sample average approximation* method. Both of these procedures might naturally be abbreviated SAA; hence the title of this paper. Our agent dominated the preliminary and seeding rounds of TAC Travel in 2006, and emerged as champion in the finals in a photo finish.

## 1 Introduction

A TAC Travel agent is a simulated travel agent whose task is to organize itineraries for a group of clients to travel to and from TACTown. The agent's objective is to procure travel goods that satisfy its clients' preferences as inexpensively as possible. Travel goods are sold in simultaneous auctions:

- flights are sold by the "TAC seller" in dynamic posted-pricing environments; no resale is permitted
- hotel reservations are sold by the "TAC seller" in multi-unit ascending call markets; specifically, 16 hotel reservations are sold in each hotel auction to the 16 highest bidders at the $16th$ highest price; no resale is permitted
- agents trade tickets to entertainment events among themselves in continuous double auctions; resale is permitted

Flights and hotel reservations are complementary goods: flights are not useful to a client without the complementary hotel reservations, nor vice versa. Tickets to entertainment events, e.g., the Boston Red Sox and the Boston Symphony Orchestra, are substitutable.

The TAC Travel environment models the problem faced by an agent bidding in simultaneous auctions for complementary and substitutable goods, e.g., an agent bidding on eBay. At a high-level, the design of many successful TAC Travel agents (e.g., Walverine [Cheng *et al.*, 2004] and ATTac [Stone *et al.*, 2003]) can be summarized as: (i) *price prediction*: build a model of the auctions' clearing prices, and (ii) *optimization*: solve for a near-optimal set of bids, given this model.

In this paper, we describe RoxyBot–06, the top-scoring TAC Travel agent in 2006. Here, we reveal RoxyBot's secrets. One feature that distinguishes RoxyBot–06 from most other TAC agents is that it builds noisy (i.e., stochastic) models of the auctions' clearing prices, rather than predicting clearing prices via point estimates. Given stochastic price predictions, stochastic optimization is at the heart of RoxyBot–06. Our approach is decision-theoretic rather than game-theoretic.

```
REPEAT
    {start bid interval}
  0. Download current prices and winnings from server
  1. predict: build stochastic models
         a. flights: Bayesian updating/learning
         b. hotels: simultaneous ascending auctions
         c. entertainment: sample historical data
  2. optimize: sample average approximation
  3. Upload current bids to server
     (three separate threads)
    {end bid interval}
UNTIL game over
```

Table 1: A high-level view of RoxyBot-06's architecture.

## 2 RoxyBot-06's Architecture

Table 1 depicts the high-level architecture of RoxyBot–06. After current prices and winnings are downloaded from the server, the key prediction and optimization routines are run. Output from the optimization routine is a bidding policy, that is, a mapping from auctions to bids. Finally, current bids are uploaded to the server by three separate threads, one for flights, one for hotels, and one for entertainment.

TAC Travel games last 9 minutes. Flight price updates are broadcast every ten seconds. The eight hotel auctions clear on the minute at each of minutes 1 through 8, with exactly one auction closing. (The precise auction to close is decided at random, with all open auctions equally likely to be selected.) For the others, the server reports the "hypothetical quantity won" by each agent as well as the current ask price. Although the entertainment auctions clear continuously, price updates are broadcast only every 30 seconds.

RoxyBot–06 discretizes time into bid intervals. Since server updates are received only every ten seconds, it suffices for TAC Travel agents to reason about intervals of this length.

For simplicity, assume the prediction and optimization steps are instantaneous. Under this assumption, based on the current bidding policy, (i) the flight thread bids on a flight only if its price is near its predicted minimum; (ii) the hotel thread bids on a hotel only if it is moments before the end of a minute; and (iii) the entertainment thread places bids immediately. In practice, the prediction and optimization steps are time-consuming, so the timing of bid placement in TAC Travel games is often complex.

## 3 Price Prediction

In this section, we describe how RoxyBot-06 builds its stochastic models of flight, hotel, and event prices. Each model is a discrete probability distribution, represented by a weighted set of "scenarios." Each scenario is comprised of a vector of "current" prices—prices at which goods can be bought and sold during the current stage—and a vector of "future" prices—prices at which goods can be bought and sold after the current stage. For flights, the current (or future) buy price is RoxyBot-06's prediction of the expected minimum price during (or after) the current stage. For hotels, the current buy prices are predicted by simulating simultaneous ascending auctions to approximate competitive equilibrium prices. There are no future buy prices for hotels. For entertainment, RoxyBot-06 predicts current and future buy and sell prices based on historical data.

### 3.1 Flights

Flight prices follow a biased random walk. They are initialized uniformly in the range $[250, 400]$, and constrained to remain in the range $[150, 800]$. At the start of each TAC game instance, a bound $z$ on the final perturbation value is selected for each flight. These bounds are not revealed to the agents. What is revealed to the agents is a sequence of random flight prices. Every ten seconds, TACAir perturbs the price of each flight by a random value that depends on the hidden parameter $z$ and the current time $t$ as follows: given constants $c \in \mathbb{R}$ and $T > 0$, each (intermediate) bound on the perturbation value is a linear function of $t$:

$$x(t, z) = c + \frac{t}{T}(z - c) \qquad (1)$$

The perturbation value at time $t$ is drawn uniformly from one of the following ranges:

- $U[-c, x(t, z)]$, if $x(t, z) > 0$
- $U[-c, +c]$, if $x(t, z) = 0$
- $U[x(t, z), +c]$, if $x(t, z) < 0$

For TAC's parameter settings, namely $c = 10$ and $T = 540$, with $z$ uniformly distributed in the range $R = [-10, 30]$, given no further information about $z$, flight prices are expected to increase (i.e., the expected perturbation is positive). Conditioned on $z$, however, flight prices may increase *or* decrease (i.e., the expected perturbation can be positive or negative). To facilitate their flight deliberations, one of the tasks faced by TAC agents is to model the probability distribution $P_t[z]$ associated with $z$ at time $t$ for use in predicting current and future flight prices. Models of probability distributions can be built using Bayesian updating.

Given a probability distribution $P_t[z]$, to predict a flight price, RoxyBot could simulate a random walk from time $t+1, \ldots, t'$ and select the minimum price. In practice, however, only RoxyBot-06's hotel and event price predictions are stochastic; its flight price predictions are point estimates (i.e., constant across scenarios). For each flight and for each possible value of the hidden parameter $z$, RoxyBot-06 simulates an "expected" random walk (see Algorithm 1), selects the minimum price, and then outputs as its prediction the expectation of these minima, averaging according to $P_t[z]$. Alternative scenario generation procedures are also possible: e.g., an agent could sample instead of calculating expected perturbations. Our choice of flight price prediction method was guided by time constraints.

**Algorithm 1** Expected_Minimum_Price$(t, t', p_t, P_t)$

```
for all $z \in R$ do
  $\min[z] = +\infty$
  for $\tau = t+1, \ldots, t'$ do
    $[a, b] = \text{getRange}(\tau, z)$
    $\Delta = \frac{b-a}{2}$ {expected perturbation}
    $p_\tau = p_{\tau-1} + \Delta$ {perturb price}
    $p_\tau = \max(150, \min(800, p_\tau))$
    if $p_\tau < \min[z]$ then
      $\min[z] = p_\tau$
    end if
  end for
end for
return $\sum_z P_t[z] \min[z]$
```

### 3.2 Hotels

RoxyBot-06's approach to hotel price prediction is inspired by Walverine's [Cheng *et al.*, 2004], in which the tâtonnement method is used to approximate competitive equilibrium prices. In a competitive market where each individual's effect on prices is negligible, equilibrium prices are prices at which supply equals demand, assuming all producers are profit-maximizing and all consumers are utility-maximizing.

Formally, let $\vec{p}$ denote a vector of prices. If $\vec{y}(\vec{p})$ denotes the cumulative supply of all producers, and if $\vec{x}(\vec{p})$ denotes the cumulative demand of all consumers, then $\vec{z}(\vec{p}) = \vec{x}(\vec{p}) - \vec{y}(\vec{p})$ denotes the excess demand in the market. The tâtonnement process adjusts the price vector at iteration $n+1$, given the price vector at iteration $n$ and a sequence $\{\alpha_n\}$ of adjustment rates: $\vec{p}_{n+1} = \vec{p}_n + \alpha_n \vec{z}(\vec{p}_n)$.

In the TAC game context, tâtonnement is not guaranteed to converge. Walverine forces convergence by letting $\alpha_n \to 0$. We force convergence by modifying the adjustment process to simulate *simultaneous ascending auctions* (SimAA) [Cramton, 2005]. In SimAAs prices increase as long as there is excess demand but they can never decrease: $\vec{p}_{n+1} = \vec{p}_n + \alpha_n \max\{\vec{z}(\vec{p}_n), 0\}$.

Following [Wellman *et al.*, 2004], we evaluate these hotel price predictions using two metrics: Euclidean distance and "expected value of perfect prediction" (EVPP). Euclidean distance is a standard way of measuring the difference between two vectors, in this case the actual and the predicted prices. The value of perfect prediction (VPP) for a client is the difference between the value of the best package for the client based on the actual prices and the value of the best package for the client based on the predicted prices. EVPP is the expected VPP averaged over the client distribution.

In Figure 1 (left), we reproduce a scatter plot generated by the Walverine team that evaluates the hotel price prediction methods of TAC 2002 agents at the beginning of the game. We add two versions of SimAA and tâtonnement to the plot. Following Wellman *et al.* [Cheng *et al.*, 2003], one uses 56 "expected" clients; the other samples 56 random clients. Eight agents play the TAC game, each with eight clients. Each agent knows the preferences of its own clients, but must simulate the demand of the 56 others. An expected client corresponds to one of ten different arrival/departure pairs, with average hotel and entertainment values.

We interpret each prediction with 56 random clients as a sample scenario, so that a set of such scenarios represents draws from a probability distribution over competitive equilibrium prices. The vector of predicted prices that is evaluated and plotted is the average of multiple (40) such predictions. The predictions generated using sets of random clients are not as good as the predictions with expected clients, although with more than 40 sets of random clients, the results might improve. Still, using random clients helps us make better interim predictions later in the game as we explain next.

As hotel auctions close, RoxyBot-06 updates the predicted clearing prices of the open hotel auctions. We experimented with two ways of constructing interim price predictions. The first is to fix the prices of the closed auctions at their clearing prices and then to run SimAA or tâtonnement with expected or random clients. The second is to distribute goods from the closed auctions to the clients who want them the most, and then to exclude any closed auctions in further runs of SimAA or tâtonnement. (NB: We determine which clients want which goods most by running SimAA or tâtonnement as usual.) Note that we can only distribute goods to random clients. It is not clear how to distribute goods to "expected clients," which are aggregate clients rather than real clients. Figure 1 (center and right) shows that the predictions based on the distribution method are better than the others. Hotels that close early tend to sell for less than hotels that close late; hence, any method that makes relatively constant predictions all throughout the game is bound to suffer.

### 3.3 Entertainment

During each bid interval, RoxyBot-06 predicts current and future buy and sell prices for tickets to all entertainment events. These price predictions are optimistic: the agent assumes it can buy (or sell) goods at the least (or most) expensive prices that it expects to see before the end of the game. More specifically, each current (or future) price prediction is the best predicted price during (or after) the current bid interval.

RoxyBot-06's estimates of entertainment ticket prices are based on historical data from the past 40 games. To generate a scenario, a sample game is drawn at random from this collection, and the sequences of entertainment bid, ask, and transaction prices are extracted. Given such a history, for each auction $a$, let $trade_{ai}$ denote the price at which the last trade before time $i$ transacted; this value is initialized to 200 for buying and 0 for selling. In addition, let $bid_{ai}$ denote the bid price at time $i$, and let $ask_{ai}$ denote the ask price at time $i$.

To predict current buy price in auction $a$ at time $t$, RoxyBot-06 first computes the minimum among the historical trade and ask prices at time $t$ and the current ask price in the present game. The current buy price is then constrained to be above the current bid price in the present game. Without this latter constraint, the agent might be inclined to buy a good at a price that is lower than the outstanding bid, which is impossible. Formally,

$$curr\_buy_{at} = \max\{currBid, \min\{trade_{at}, ask_{at}, currAsk\}\} \quad (2)$$

The current sell price is predicted analogously:

$$curr\_sell_{at} = \min\{currAsk, \max\{trade_{at}, bid_{at}, currBid\}\} \quad (3)$$

RoxyBot-06 predicts the future buy price in auction $a$ after time $t$ as follows:

$$future\_buy_{at} = \min_{i=t+1,\ldots,T} \min\{trade_{ai}, ask_{ai}\} \quad (4)$$

In words, the future buy price at each time $i = t+1, \ldots, T$ is the minimum of the ask price after time $i$ and the most recent trade price. The future buy price at time $t$ is the minimum across the future buy prices at all later times. As above, the future sell price after time $t$ is predicted analogously:

$$future\_sell_{at} = \max_{i=t+1,\ldots,T} \max\{trade_{ai}, bid_{ai}\} \quad (5)$$

## 4 Optimization

We characterize RoxyBot-06's optimization routine as (i) stochastic, (ii) global, and (iii) dynamic. It takes as input stochastic price predictions; it simultaneously considers flight, hotel, and entertainment bids in unison; and it simultaneously reasons about bids to be placed in both current and future stages of the game.

We assume that our agent's decisions do not affect prices and express the game-theoretic bidding problem as a decision-theoretic optimization problem. The influence of other agents on prices is represented by a set of scenarios or, more generally, a distribution over prices.

Hence, RoxyBot-06 is confronted with a dynamic stochastic optimization problem. It solves this problem by collapsing the future into only two relevant stages—"current" and "future". This approach is reasonable in TAC, and other similar combinations of sealed-bid and continuously-clearing simultaneous auction environments, as we now explain.

The key bidding decisions are: what goods to bid on, at what price, and when? Since hotel auctions close in an unknown random order, RoxyBot-06, like most TAC agents, operates under the assumption that all hotel auctions close at the

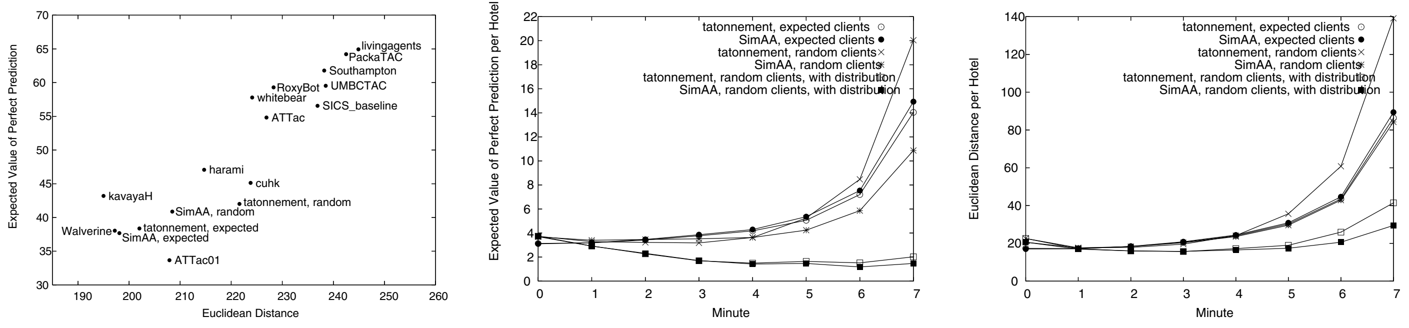

Figure 1: Left: EVPP and Euc. Dist. for TAC 2002 agents as well as our prediction methods: Tât. and SimAA; expected and random. Center and Right: EVPP and Euc. Dist. in TAC 2006 of our hotel price prediction methods with distribution as the game progresses.

end of the current stage. Hence, the only pressing decisions regarding hotels are: what goods to bid on and at what price? There is no need to consider the timing of bids. Accordingly, the only model of hotel prices is the current one.

In contrast, since flight and entertainment auctions are continuous, a trading agent should reason about the timing of its bids for these goods. Still, it suffices to consider only two pricing models. The current model predicts the best good prices during the current bid interval, whereas the future model predicts the best good prices after the current bid interval, conditioned on current prices.

We express the bidding problem as a stochastic program where current and future prices are given by corresponding stochastic models. The sample average approximation (SAA) method is a numerical means of approximating solutions to stochastic programs via Monte Carlo simulation [Ahmed and Shapiro, 2002]. The main idea is simple: (i) sample a set of scenarios from the input distribution, and (ii) approximate the solution to the stochastic program by solving an approximation of the stochastic optimization problem which incorporates only the sample scenarios. RoxyBot–06 samples scenarios by simulating SimAA as described in Section 3.2.

The ILP formulation of SAA applied to the TAC bidding problem is included in Appendix A. The power of this formulation is in its ability to make globally optimal decisions. Flight, hotel, and entertainment bids are optimized together to produce a consistent bidding policy. The first stage decisions are the bids that are submitted now. The second stage decisions are the bids that can be submitted later. All hotel bids are submitted now; flight and entertainment bids can be submitted now or later.

The formulation of the problem facilitates reasoning not only about what bids to place but also about when to place them. The fundamental issue regarding TAC flight decisions is a common one: balancing concern about future price increases with the benefit of delaying commitment to travel on particular days. We model this tradeoff explicitly by giving the agent the option of buying flight and entertainment tickets now at the current prices or later at the scenario's (i.e., future) prices. If the agent decides to buy a flight or entertainment ticket later, it can submit different bids in different scenarios.

One of the weaknesses of the ILP formulation is that its hotel bids can only be as high (or as low) as the maximum (minimum) price in the scenarios although an agent may be willing to pay more (less). We overcome this problem by adding to the set of scenarios generated by our stochatic model additional scenarios in which one of the hotels is priced higher/lower than in any other scenario. This way the agent may bid higher or lower than predictions in case the actual clearing prices are unexpectedly high or low. RoxyBot-06 optimized with respect to 30 scenarios generated as described in Section 3 and up to 30 extra scenarios with high and low prices. These choices were guided by time constraints.

## 5 Competition Results

We present the results of the last day of the TAC-06 Finals (80 games). We omit the first two days because agents can vary across days, but cannot vary within. Presumaly, the entries on the last day are the teams' preferred versions of the agents. Mean scores are plotted in Figure 2 and detailed statistics are tabulated in Table 2.

There is no single metric such as low hotel or flight costs that is responsible for RoxyBot's success. Rather its success derives from the right balance of contradictory goals. In particular, RoxyBot incurs high hotel and mid-range flight costs while achieving mid-range trip penalty and high event profit.

We compare RoxyBot with two closest rivals: Walverine and WhiteDolphin. Comparing to Walverine first, Walverine bids lower prices (by 55) on fewer hotels (49 less), yet wins more (0.8) and wastes less (0.42). It would appear that Walverine's hotel bidding strategy outperforms RoxyBot's, except that RoxyBot earns a higher hotel bonus (15 more). RoxyBot also gains an advantage by spending 40 less on flights and earning 24 more in total entertainment profit.

A very different competition takes place between RoxyBot and WhiteDolphin. WhiteDolphin bids lower prices (120 less) on more hotels (by 52) than RoxyBot. RoxyBot spends much more (220) on hotels than WhiteDolphin but makes up for it by earning a higher hotel bonus (by 96) and a lower trip penalty (by 153). It seems that WhiteDolphin's strategy is to minimize costs even if that means sacrificing utility.

|  | Rox | Wal | Whi | SIC | Mer | L-A | kin | UTT |
|---|---|---|---|---|---|---|---|---|
| # of Hotel Bids | 130 | 81 | 182 | 33 | 94 | 58 | 15 | 24 |
| Average of Hotel Bids | 170 | 115 | 50 | 513 | 147 | 88 | 356 | 498 |
| # of Hotels Won | 15.99 | 16.79 | 23.21 | 13.68 | 18.44 | 14.89 | 15.05 | 9.39 |
| Hotel Costs | 1102 | 1065 | 882 | 1031 | 902 | 987 | 1185 | 786 |
| # of Unused Hotels | 2.24 | 1.82 | 9.48 | 0.49 | 4.86 | 1.89 | 0.00 | 0.48 |
| Hotel Bonus | 613 | 598 | 517 | 617 | 590 | 592 | 601 | 424 |
| Trip Penalty | 296 | 281 | 449 | 340 | 380 | 388 | 145 | 213 |
| Flight Costs | 4615 | 4655 | 4592 | 4729 | 4834 | 4525 | 4867 | 3199 |
| Event Profits | 110 | 26 | 6 | -6 | 123 | -93 | -162 | -4 |
| Event Bonus | 1470 | 1530 | 1529 | 1498 | 1369 | 1399 | 1619 | 996 |
| Total Event Profits | 1580 | 1556 | 1535 | 1492 | 1492 | 1306 | 1457 | 992 |
| Average Utility | 9787 | 9847 | 9597 | 9775 | 9579 | 9604 | 10075 | 6607 |
| Average Cost | 5608 | 5693 | 5468 | 5765 | 5628 | 5605 | 6213 | 3989 |
| Average Score | 4179 | 4154 | 4130 | 4010 | 3951 | 3999 | 3862 | 2618 |

Table 2: Last day of Final 2006. 80 games.

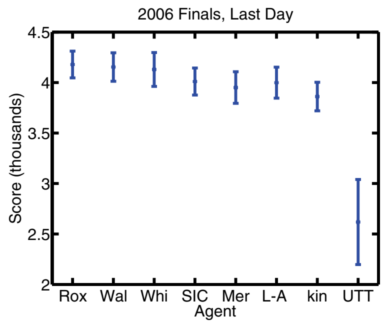

Figure 2: 2006 Finals' Scores and 95% Confidence Intervals.

## 6 Conclusion

The success of an autonomous trading agent, particularly TAC agents, often hinges upon two key modules: (i) *price prediction*, in which the agent builds a model of market prices; and (ii) *optimization*, in which the agent solves for an approximately optimal set of bids, given this model. For example, at the core of RoxyBot's 2000 architecture [Greenwald and Boyan, 2005] was a *deterministic* optimization problem, namely how to bid given price predictions in the form of point estimates. In spite of its effectiveness in the TAC-00 tournament, a weakness of the 2000 design was that RoxyBot could not explicitly reason about variance within prices. In the years since 2000, we recast the key challenges faced by TAC agents as several different *stochastic* bidding problems (see [Greenwald and Boyan, 2004]), whose solutions exploit price predictions in the form of distributions. In spite of our perseverance, RoxyBot fared unimpressively in tournament conditions year after year…until 2006. Half a decade in the laboratory spent searching for bidding heuristics that can exploit stochastic information at reasonable computational expense finally bore fruit, as RoxyBot emerged victorious in TAC-06. In a nutshell, the secret of RoxyBot-06's success is: hotel price prediction by simulating simultaneous ascending auctions, and optimization based on the sample average approximation method.

## References

[Ahmed and Shapiro, 2002] Shabbir Ahmed and Alexander Shapiro. The sample average approximation method for stochastic programs with integer recourse. Optimization Online, `http://www.optimization-online.org`, 2002.

[Cheng *et al.*, 2003] S.F. Cheng, E. Leung, K.M. Lochner, K.O'Malley, D.M. Reeves, L.J. Schvartzman, and M.P. Wellman. Walverine: A Walrasian trading agent. In *Proceedings of the Second International Joint Conference on Autonomous Agents and Multi-Agent Systems*, pages 465–472, July 2003.

[Cheng *et al.*, 2004] S.F. Cheng, E. Leung, K.M. Lochner, K.O'Malley, D.M. Reeves, L.J. Schvartzman, and M.P. Wellman. Walverine: A Walrasian trading agent. *Decision Support Systems*, page To Appear, 2004.

[Cramton, 2005] Peter Cramton. Simultaneous ascending auctions. In Peter Cramton, Yoav Shoham, and Richard Steinberg, editors, *Combinatorial Auctions*. MIT Press, 2005.

[Greenwald and Boyan, 2004] A. Greenwald and J. Boyan. Bidding under uncertainty: Theory and experiments. In *Proceedings of the 20th Conference on Uncertainty in Artificial Intelligence*, pages 209–216, July 2004.

[Greenwald and Boyan, 2005] Amy Greenwald and Justin Boyan. Bidding algorithms for simultaneous auctions: A case study. *Journal of Autonomous Agents and Multiagent Systems*, 10(1):67–89, 2005.

[Stone *et al.*, 2003] P. Stone, R.E. Schapire, M.L. Littman, J.A. Csirik, and D. McAllester. Decision-theoretic bidding based on learned density models in simultaneous, interacting auctions. *Journal of Artificial Intelligence Research*, 19:209–242, 2003.

[Wellman *et al.*, 2004] M.P. Wellman, D.M. Reeves, K.M. Lochner, and Y. Vorobeychik. Price prediction in a Trading Agent Competition. *Artificial Intelligence Research*, 21:19–36, 2004.

## A TAC Bidding Problem: SAA

The problem of bidding in the simultaneous auctions that characterize TAC can be formulated as a two-stage stochastic program, in which bids are placed in the first stage and winnings are allocated in the second stage. In this appendix, we present the implementation details of the integer linear program (ILP) encoded in RoxyBot-06 that approximates an optimal solution to this stochastic program.

This ILP formulation of the TAC bidding problem assumes linear prices, that is, constant economies of scale. Table 3 lists the the price constants that comprise scenarios. Each scenario includes a "current" price, which is the best price expected within the stage, and a "future" price, which is the best price expected from the start of the next stage until the end of the game, for each good (except hotels, which do not have future buy prices because they are treated as one-shot auctions).

Table 3 also lists the decision variables that pertain to each auction type. For hotels, the only decisions pertain to buy offers, right now; for flights, the agent chooses buy offers for now and for later; for entertainment events, both buy and sell offers are made now and later. "Now" decisions are taken under uncertainty. They are the bids to be placed on each good, necessarily constant across scenarios. "Later" decisions are scenario dependent. Uncertainty is resolved (i.e., prices are known) in scenarios; there, it suffices to simply choose the quantity of each good to trade.

| Flights | Scenario | Decision Type |
|---|---|---|
| buy now | current price $\mathcal{M}_{as}$ | bid $\mu_{apq}$ |
| buy later | future price $\mathcal{Y}_{as}$ | quantity $\upsilon_{as}$ |

| Hotels | Scenario | Decision Type |
|---|---|---|
| buy now | current price $\mathcal{M}_{as}$ | bid $\mu_{apq}$ |

| Events | Scenario | Decision Type |
|---|---|---|
| buy now | current price $\mathcal{M}_{as}$ | bid $\mu_{apq}$ |
| buy later | future price $\mathcal{Y}_{as}$ | quantity $\upsilon_{as}$ |
| sell now | current price $\mathcal{N}_{as}$ | bid $\nu_{apq}$ |
| sell later | future price $\mathcal{Z}_{as}$ | quantity $\zeta_{as}$ |

Table 3: Auction types and associated scenario prices and decision types.

## A.1 Index Sets

$a \in A$ indexes the set of goods, or auctions.

$a_f \in A_f$ indexes the set of flight auctions.

$a_h \in A_h$ indexes the set of hotel auctions.

$a_e \in A_e$ indexes the set of event auctions.

$c \in C$ indexes the set of clients.

$p \in P$ indexes the set of prices.

$q \in Q$ indexes the set of quantities (i.e., the units of each good in each auction).

$s \in S$ indexes the set of scenarios.

$t \in T$ indexes the set of trips.

## A.2 Constants

$\mathcal{G}_{at}$ indicates the quantity of good $a$ needed to complete trip $t$.

$\mathcal{M}_{as}$ indicates the current buy price of $a_f, a_h, a_e$ in $s$.

$\mathcal{N}_{as}$ indicates the current sell price of $a_e$ in scenario $s$.

$\mathcal{Y}_{as}$ indicates the future buy price of $a_f, a_e$ in scenario $s$.

$\mathcal{Z}_{as}$ indicates the future sell price of $a_e$ in scenario $s$.

$\mathcal{H}_a$ indicates the hypothetical quantity won of hotel $a_h$.

$\mathcal{O}_a$ indicates the quantity of good $a$ the agent owns.

$\mathcal{U}_{ct}$ indicates client $c$'s value for trip $t$.

## A.3 Decision Variables

$\Gamma = \{\gamma_{cst}\}$ is a set of boolean variables indicating whether or not client $c$ is allocated trip $t$ in scenario $s$.

$M = \{\mu_{apq}\}$ is a set of boolean variables indicating whether to bid price $p$ to buy the $q$th unit of $a_f, a_h, a_e$ now.

$N = \{\nu_{apq}\}$ is a set of boolean variables indicating whether to bid price $p$ to sell the $q$th unit of $a_e$ now.

$Y = \{\upsilon_{as}\}$ is a set of integer variables indicating how many units of $a_f, a_e$ to buy later in scenario $s$.

$Z = \{\zeta_{as}\}$ is a set of integer variables indicating how many units of $a_e$ to sell later in scenario $s$.

## A.4 Objective Function

$$\max_{\Gamma,M,N,Y,Z} \sum_S \Bigg[ \overbrace{\sum_{C,T} \mathcal{U}_{ct}\gamma_{cts}}^{trip\ value} - event - \overbrace{\sum_{A_f}\left(\overbrace{\sum_{Q,p\geq\mathcal{M}_{as}} \mathcal{M}_{as}\mu_{apq}}^{current} + \overbrace{\mathcal{Y}_{as}\upsilon_{as}}^{future}\right)}^{flight\ cost} - \overbrace{\sum_{A_h,Q,p\geq\mathcal{M}_{as}} \mathcal{M}_{as}\mu_{apq}}^{hotel\ cost}\Bigg] \tag{6}$$

$$event = \sum_{A_e}\left( \overbrace{\overbrace{\sum_{Q,p\geq\mathcal{M}_{as}} \mathcal{M}_{as}\mu_{apq}}^{current} + \overbrace{\mathcal{Y}_{as}\upsilon_{as}}^{future}}^{event\ cost} \overbrace{- \overbrace{\sum_{Q,p\leq\mathcal{N}_{as}} \mathcal{N}_{as}\nu_{apq}}^{current} + \overbrace{\mathcal{Z}_{as}\zeta_{as}}^{future}}^{event\ revenue} \right) \tag{7}$$

## A.5 Constraints

$$\sum_T \gamma_{cst} \leq 1 \quad \forall c \in C, s \in S \tag{8}$$

$$\overbrace{\sum_{C,T} \gamma_{cst}\mathcal{G}_{at}}^{allocation} \leq \overbrace{\mathcal{O}_a}^{own} + \overbrace{\upsilon_{as} + \sum_{Q,p\geq\mathcal{M}_{as}} \mu_{apq}}^{buy} \quad \forall a \in A_f, s \in S \tag{9}$$

$$\overbrace{\sum_{C,T} \gamma_{cst}\mathcal{G}_{at}}^{allocation} \leq \overbrace{\mathcal{O}_a}^{own} + \overbrace{\sum_{Q,p\geq\mathcal{M}_{as}} \mu_{apq}}^{buy} \quad \forall a \in A_h, s \in S \tag{10}$$

$$\overbrace{\sum_{C,T} \gamma_{cst}\mathcal{G}_{at}}^{allocation} \leq \overbrace{\mathcal{O}_a}^{own} + \overbrace{\upsilon_{as} + \sum_{Q,p\geq\mathcal{M}_{as}} \mu_{apq}}^{buy} - \overbrace{\zeta_{as} + \sum_{Q,p\leq\mathcal{N}_{as}} \nu_{apq}}^{sell} \quad \forall a \in A_e, s \in S \tag{11}$$

$$\sum_{P,Q} \mu_{apq} \geq \mathcal{H}_a \quad \forall a \in A_h \tag{12}$$

$$\sum_P \mu_{apq} \leq 1 \quad \forall a \in A_f \cup A_h \cup A_e, q \in Q \tag{13}$$

$$\sum_P \nu_{apq} \leq 1 \quad \forall a \in A_e, q \in Q \tag{14}$$

Equation (8) limits each client to one trip in each scenario. Equation (9) prevents the agent from allocating flights that it does not own or buy. Equation (10) prevents the agent from allocating hotels that it does not own or buy. Equation (11) prevents the agent from allocating event tickets that it does not own or buy and not sell. Equation (12) ensures the agent bids on at least HQW units in each hotel auction. Equation (13) prevents the agent from placing more than one buy offer per unit in each flight, hotel, or event auction. Equation (14) prevents the agent from placing more than one sell offer per unit in each event auction. An agent might also be constrained not to place sell offers on more units of each good than it owns, and/or not to place buy (sell) offers for more units of each good than the market supplies (demands).