

Bidding Algorithms for Simultaneous Auctions

Justin Boyan *ITA Software, One Kendall Square, Bldg 400, Cambridge, MA 02139*
justin@boyan.com

Amy Greenwald, R.M. Kirby, and Jon Reiter *Brown University, Providence, RI 02912*
amygreen@cs.brown.edu, kirby@cfm.brown.edu, jr@cfm.brown.edu

1. Introduction

Suppose you want to buy a used Canon AE-1 SLR camera and flash at an on-line auction. At last count, over 4000 links to on-line auction sites were available at advocacy-net.com. It would be quite a daunting task to manually monitor prices and make bidding decisions at all sites currently offering the camera—especially if the flash accessory is sometimes bundled with the camera, and sometimes auctioned separately. But for the next generation of automated trading agents, this will be a routine task.

Simultaneous auctions, or *parallel auctions*, present a challenge to bidders, particularly when complementary and substitutable goods are on offer. Complementary goods are items such as a flash, a tripod, and a case, that would complement a camera—but a bidder desires any of the former only if s/he is certain to acquire the latter. Substitutable goods are goods such as the Canon AE-1 and the Canon A-1—a bidder desires one or the other, but not both. In contrast to *combinatorial auctions*, in which bids may be placed for combinations of items (e.g., “camera and flash for \$295”), simultaneous auctions require separate bids to be placed for each individual item. In combinatorial auctions, the problem of choosing a set of winning bids that maximizes revenue—the so-called winner determination problem (WD)—falls in the hands of the auctioneer; in simultaneous auctions, however, the complexity burden lies with the bidders.

In this paper, we present an architecture for intelligent bidding in simultaneous auctions. Our architecture is centered on a class of problems we call *bid determination* (BD): determining what bids to place in simultaneous auctions, given current market conditions and utilities on combinations of items. We present two interesting theoretical results:

1. BD in double auctions, where goods can be sold as well as bought, can be formally reduced to the problem of BD in single-sided auctions.
2. BD problems in simultaneous auctions are isomorphic to common variants of WD in multi-unit combinatorial auctions.

In addition to our theoretical contributions, we implemented and tested two forms of BD solutions in the context of the First International Trading Agent Competition (TAC) [10]. A TAC agent is a simulated travel agent whose task is to organize itineraries for a group of clients who wish to travel from TACTown to Boston and back again during a five-day period. Travel goods, such as airline tickets and hotel reservations, are complementary, and tickets to entertainment events, such as the Boston Red Sox and the Boston Symphony Orchestra, are substitutable. The trading agent’s objective is to win those items that best satisfy its clients’ preferences as inexpensively as possible.

The BD bidding algorithms presented in this paper were inspired by the two top-scoring TAC agents. Our entry, **RoxyBot**, used an optimal search algorithm based on A^* to solve the allocation problem and heuristic beam search to find approximately optimal completions [3]. **ATTac** solved the allocation and acquisition problems using integer linear programming (ILP) [8]. We compare the performance of these techniques on data generated during the final round of the TAC competition, and on larger datasets built by merging TAC data. Our key empirical findings are as follows:

1. For the dimensions of TAC, optimal solutions to BD problems are tractable.
2. As the problem size increases, A^* scales poorly. ILP fares better on average, but has very high variance.
3. Heuristic approximation methods scale well, achieving near optimal solutions with predictable time and space requirements.

2. Bid Determination

The key challenge that bidding agents face in simultaneous auctions is determining how to bid on complementary and substitutable goods. Complementary goods are goods with superadditive utilities: *i.e.*, $u(A\bar{B}) + u(\bar{A}B) < u(AB)$. For example, in TAC, the utility of airline tickets without hotel reservations (or of hotel reservations without airline tickets) is zero, whereas the utility of complete travel packages is strictly positive. Substitutable goods are goods with subadditive utilities: $u(A\bar{B}) + u(\bar{A}B) > u(AB)$. For example, in TAC, the utility of both a theater ticket and a symphony ticket for the same night is bounded above by the higher of the individual utilities of the two separate events. It does not make sense to assign individual utilities to complementary goods (which are worthless in isolation) or substitutable goods (which are worthwhile only in isolation). Thus, simple bidding strategies such as “for each good x in auction x , bid up to its utility” are inapplicable in simultaneous auctions.

Instead, we argue that bidding agents must reason directly about *sets* of goods—the utilities of which are well-defined. While making bidding decisions in simultaneous auctions, a bidding agent may pose and solve questions such as the following:

- “Given only the set of goods I already hold, what is the maximum utility I can attain?” We call this the *allocation* problem, since the agent’s utility is determined by the choice of how to allocate its set of held goods into useful subsets.
- “Given the set of goods I already hold, and given market prices and supply in all open auctions, what set of additional goods should I acquire so as to maximize my utility less purchase costs?” This more general problem, which we term *acquisition*, provides a foundation for bidding strategies in single-sided auctions.
- “Given the set of goods I already hold, and given market prices, supply, and *demand*, on what set of additional goods should I place bids or *asks* so as to maximize my utility plus profits less costs?” This yet more general problem, *completion*, provides a foundation for bidding strategies in settings with simultaneous single-sided and double auctions.

2.1 Agent Architecture

An agent bidding in on-line simultaneous auctions must decide (i) on what goods to bid, (ii) for how many to bid, (iii) at what price to bid, and (iv) when to bid—and likewise for asks. A completion directly answers the first two of these questions, and it provides a partial answer to third question, namely bid (ask) no more (resp. less) than the market price. (The fourth question, which depends on the specific auction mechanisms, is addressed in [4].) Thus, a natural architecture for a bidding agent is to repeatedly compute estimates of market clearing prices, run a completion algorithm to determine target holdings, and bid/ask accordingly. This architecture is outlined in Table 1 for a simultaneous auction framework in which there is no advantage to allocating goods early.

The remainder of this paper focuses on the computational challenges of completion, acquisition (the special case of completion applicable to single-sided auctions), and allocation. Two other components of our architecture—estimation (step A2) and bid/ask tactics (step A4)—are equally challenging but problem-dependent, and are not addressed here. (For some insights, see [3, 4, 8]).

<p>(A) While some auction remains open, do:</p> <ol style="list-style-type: none"> 1. Update current prices and holdings for each auction. 2. Estimate clearing prices, supply, and demand of each good. 3. Run completion to determine the quantity of each good that is ultimately desired; compute the difference between the optimal solution and current holdings. 4. Place bids and asks strategically (with respect to current time and the auction mechanisms) to buy and sell goods to reach the desired quantities. <p>(B) After all auctions have closed, run allocation.</p>
--

Table 1: A high-level architecture for bidding agents in simultaneous auctions.

2.2 Formal Problem Statement

We now formally define allocation, acquisition, and completion, starting with acquisition. Consider an agent bidding in simultaneous auctions for goods g in the set G . For each good g , we are given the agent’s current holdings of g , the market supply, and the prices of additional copies of g . We represent these holdings and estimates by a single vector \vec{p}_g , which we call a *priceline*, in which the n th component, p_{gn} , stores the cost the agent would incur upon acquiring the n th copy of g .

For example, if the agent currently holds four units of a good \hat{g} , and predicts that two additional units of \hat{g} could be won at costs of \$20 and \$30, respectively, the corresponding priceline is given by $\vec{p}_{\hat{g}} = \langle 0, 0, 0, 0, 20, 30, \infty, \infty, \dots \rangle$. The leading zeroes indicate that the four goods the agent already holds may be “acquired” at no cost. The tail of infinite costs means that the agent may acquire no more than six units of \hat{g} in total. We assume priceline vectors are nonnegative and nondecreasing.

In acquisition, the agent’s goal is to determine how best to augment its holdings with new purchases so as to maximize the sum of the package utilities it can achieve less the total cost of the goods. Let a package be represented by a vector of quantities, one for each good, plus a unique identifier:¹ $\vec{q} = \langle q_1, \dots, q_{|G|}; q_{id} \rangle$. The package’s utility is denoted by $u(\vec{q})$.

Given a set of pricelines $P = \{\vec{p}_g \mid g \in G\}$ and a set of packages S , we define the utility and cost of S straightforwardly as follows:

$$\text{Util}(S) = \sum_{\vec{q} \in S} u(\vec{q}) \quad (1)$$

$$\forall g, \quad \text{Used}(S, g) = \sum_{\vec{q} \in S} q_g \quad (2)$$

$$\forall g, \quad \text{Cost}_g(S, P) = \sum_{n=1}^{\text{Used}(S, g)} p_{gn} \quad (3)$$

$$\text{Cost}(S, P) = \sum_{g \in G} \text{Cost}_g(S, P) \quad (4)$$

We can now formally define acquisition and allocation.

Definition 2.1 Acquisition(P, Q, u).

Inputs: a set of pricelines $P = \{\vec{p}_g \mid g \in G\}$; a set of packages Q ; a utility function $u : Q \rightarrow \mathbb{R}^+$.

Output: an optimal subset of packages $S^* \in \arg \max_{S \subseteq Q} (\text{Util}(S) - \text{Cost}(S, P))$.

1. The unique identifier serves to guarantee that the utility function is indeed a function even in the case where multiple packages contain the same goods but have different utility values.

Definition 2.2 *Allocation* is simply a special case of the acquisition problem in which no additional goods can be purchased. That is, all entries p_{gn} in the pricelines are assumed to be either 0 or ∞ .

Completion generalizes acquisition to double-auction scenarios in which the agent may sell some of the goods it holds, if it so chooses, rather than using them in packages. Let us represent the estimated market demand for each good g the agent owns as a vector $\vec{\pi}_g$, which we call a *profitline*. Much like a priceline, the n th component of a profitline stores the *profit* that the agent could earn by selling its n th copy of g on the open market.

For example, returning to the sample good \hat{g} above, suppose the agent estimated that it could find buyers for two of its four copies of \hat{g} at prices of \$10 and \$5, respectively. Its profitline would be given by: $\vec{\pi}_{\hat{g}} = \langle 10, 5, 0, 0 \rangle$ where the trailing zeroes reflect that there is no market demand for the third and fourth copies of \hat{g} . Note that profitlines have length equal to the quantity of the good held by the agent. We assume that profitlines are nonnegative and nonincreasing.

We now define profit analogously to cost, as above. Given a set of profitlines $\Pi = \{\vec{\pi}_g \mid g \in G\}$ and a set of packages S ,

$$\forall g, \quad \text{Unused}(S, g, \Pi) = \max\{|\vec{\pi}_g| - \text{Used}(S, g), 0\} \quad (5)$$

$$\forall g, \quad \text{Profit}_g(S, \Pi) = \sum_{i=1}^{\text{Unused}(S, g)} \pi_{gn} \quad (6)$$

$$\text{Profit}(S, \Pi) = \sum_{g \in G} \text{Profit}_g(S, \Pi) \quad (7)$$

We can now formally define completion.

Definition 2.3 *Completion*(P, Π, Q, u).

Inputs: a set of pricelines $P = \{\vec{p}_g \mid g \in G\}$; a set of profitlines $\Pi = \{\vec{\pi}_g \mid g \in G\}$;
a set of packages Q ; a utility function $u : Q \rightarrow \mathbb{R}^+$.

Output: an optimal subset of packages $S^* \in \arg \max_{S \subseteq Q} (\text{Util}(S) - \text{Cost}(S, P) + \text{Profit}(S, \Pi))$.

2.3 Reducing Completion to Acquisition

Perhaps surprisingly, completion problems (for double-sided simultaneous auctions) can be cast as acquisition (for single-sided auctions). This equivalence can be demonstrated in two ways: by mapping the profitlines into dummy packages, or by augmenting the pricelines with the information contained in the profitlines.

In the first mapping, we treat the market as an additional source of utility: each nonzero number π_{gn} on each profitline translates into a new package \vec{e}_{gn} containing only the single item g , with corresponding utility $u(\vec{e}_{gn}) = \pi_{gn}$. For example, given the example profitline $\vec{\pi}_{\hat{g}} = \langle 10, 5, 0, 0 \rangle$, we would create two dummy packages, $\vec{e}_{\hat{g}1}$ and $\vec{e}_{\hat{g}2}$, with utilities 10 and 5, respectively.

Theorem 2.4 For all P, Π, Q , and u , $\text{Completion}(P, \Pi, Q, u) = \text{Acquisition}(P, Q \cup Q', u \cup u') \cap Q$, where $Q' = \{\vec{e}_{gn} \mid g \in G, n = 1 \dots |\vec{\pi}_g|\}$ and $u' = \{\vec{e}_{gn} \mapsto \pi_{gn} \mid g \in G, n = 1 \dots |\vec{\pi}_g|\}$.

In the second mapping from completion to acquisition, the package-utility pairs are unchanged. Instead, we merge the profitlines into the pricelines. The intuition here is that in a double-auction scenario, an agent cannot use even the goods it holds at no cost; rather, the agent incurs *opportunity costs* for not selling those goods on the market.

Combining a priceline \vec{p}_g and a profitline $\vec{\pi}_g$ is a simple matter of replacing the leading $|\vec{\pi}_g|$ entries of \vec{p}_g —which are originally all zero, since they represent the cost of acquiring the agent's own holdings—with the contents of $\vec{\pi}_g$ in reverse. For example, consider the sample priceline $\vec{p}_{\hat{g}}$ and sample profitline $\vec{\pi}_{\hat{g}}$ shown earlier. Combining these two produces the following result: $\vec{p}_{\hat{g}} +$

$reverse(\vec{\pi}_{\hat{g}}) = \langle 0, 0, 5, 10, 20, 30, \infty, \infty, \infty, \dots \rangle$. The resulting priceline should again be nonnegative and nondecreasing. (If this were not so, there would exist an arbitrage opportunity, which the agent could exploit in a preprocessing step.)

The interpretation of the new priceline is as follows: the agent can use one or two copies of good \hat{g} without penalty, since in either case it will still be able to sell at most two further copies of \hat{g} . If it uses a third copy, though, it must charge itself \$5 in lost profits. If it uses all four of its held copies, it must charge itself \$5+\$10=\$15. If it uses five, it incurs the lost opportunity cost of \$15, plus the \$20 cost of purchasing an additional copy on the open market; similarly, for six, it incurs cost \$65. Finally, lack of market supply (represented by ∞) prevents the agent from using more than six copies of the good in total. We can now state our second theorem.

Theorem 2.5 *For all P, Π, Q , and u , $Completion(P, \Pi, Q, u) = Acquisition(P', Q, u)$, where $P' = \{\vec{p}_g + reverse(\vec{\pi}_g) \mid g \in G\}$.*

2.4 Winner Determination

The BD problems formulated above are isomorphic to variants of the winner determination (WD) problems that arise in combinatorial auctions. In combinatorial auctions (see, for example, [2, 7]), an auctioneer collects bids on combinations (*i.e.*, packages) of goods, and then seeks the most profitable subset of those bids—the “winners”—that can be fulfilled from the set of available goods. Combinatorial bids correspond exactly to our notion of package utilities in the allocation problem. Similarly, the bidding agent’s goal in allocation and the auctioneer’s goal in WD are identical: to choose a subset of bids/packages that maximizes utility while respecting the constraint on total goods available. Thus, allocation is isomorphic to WD in (multi-unit) combinatorial auctions.

More generally, the tasks of acquisition and completion are equivalent to *winner determination with reserve prices*. In this latter problem, it is assumed that there is some price below which the auctioneer prefers not to sell each good. Thus, the auctioneer seeks to maximize the difference between profits and reserve prices. Similarly, in acquisition and completion tasks, bidding agents seek to maximize the difference between utilities and market prices. WD problems are themselves equivalent to maximum weighted set-packing (and maximum weighted clique), and are therefore NP-complete [6]. Hence, allocation, acquisition, and completion are NP-complete.

3. TAC-2000 Market Game

Implementations of winner determination have typically been evaluated on randomly generated datasets [5], since data from large-scale combinatorial auctions is scarce. (One obvious exception is the FCC spectrum auction.) The equivalence between BD and WD makes new datasets available for testing by the combinatorial auction community—for example, the data generated by the Trading Agent Competition.

A TAC agent is a simulated travel agent whose task is to organize itineraries for a group of clients who wish to travel from TACTown to Boston and back during a five day period. Travel and entertainment goods are traded at 28 online auctions that run simultaneously for fifteen minutes. An agent’s objective is to secure the goods necessary to satisfy the particular desires of its clients, but to do so as inexpensively as possible. An agent’s score is the difference between the utilities it earns for its clients and the agent’s expenditures. For details, visit <http://tac.eecs.umich.edu>.

3.1 Supply

The market supply consists of three types of travel goods: (i) flights in and out of Boston (I and O, respectively), (ii) hotel room reservations at two competing hotels, namely, the Grand Hotel (G) and Le Fleabag Inn (F), and (iii) entertainment tickets for the Boston Red Sox (R), the Boston Symphony (S), and the theater (T). There is a separate auction for each combination of good and

day, yielding twenty-eight auctions in total: eight flight auctions (there are no inbound flights on the fifth day, and there are no outbound flights on the first day), eight hotel auctions (two hotel types and four nights), and twelve entertainment ticket auctions (three entertainment event types and four nights). All auctions are *simultaneous*. The rules of the various auctions are as follows:

- (i) An infinite supply of flights is sold by the “TAC seller”, a specially designated supplier, at continuously clearing auctions in which prices follow a random walk.
- (ii) The TAC seller also makes available sixteen hotel rooms per hotel per night, which are sold at open-cry, ascending, multi-unit, sixteenth-price auctions.
- (iii) Entertainment tickets are traded among TAC agents in continuous double auctions. Agents can act as either buyers or sellers, and transactions clear continuously.

3.2 Demand

A TAC game instance pits eight trading agents against one another, with each agent representing eight clients. The market demand is determined by the sixty-four clients’ preferences. Each client is characterized by a random set of preferences for ideal arrival and departure dates (IAD and IDD, respectively, which range over days 1 through 4),² a grand hotel room reservation value (HV, which takes integer values between 50 and 150), and reservation values for each of the three types of entertainment events (RV, SV, and TV—integers between 0 and 200—for Red Sox, symphony, and theater, respectively). A sample set of preferences appears in Table 2; these preferences were those of the clients assigned to *RoxyBot* during game 3065 of the competition.

Client	IAD	IDD	HV	RV	SV	TV
1	1	2	99	134	118	65
2	1	3	131	170	47	49
3	1	1	147	13	55	49
4	3	3	145	130	60	85
5	1	3	82	136	68	87
6	2	3	53	94	51	105
7	1	2	54	156	126	71
8	1	4	113	119	187	143

Table 2: *RoxyBot*’s client preferences in game 3065. These values jointly establish the bidding agent’s utility function over various combinations of goods.

The job of each TAC agent is to assemble a feasible package of goods for each of its clients. A *package* is characterized by arrival and departure dates (AD and DD, respectively, ranging over days 1 through 5), a hotel type (H, which takes on value G for Grand Hotel or F for Le Fleabag Inn), and entertainment tickets ($I(j, k)$ is an indicator variable that represents whether or not the package includes a ticket on night j to event $k \in \{r, s, t\}$; we also write R1, for example, to indicate that the package includes a Boston Red Sox ticket on night 1). In order to obtain positive utility for a client, an agent must construct a *feasible* package for that client; otherwise, the client’s utility is zero. A feasible package is one in which (i) the arrival date is strictly less than the departure date, (ii) the same hotel is reserved during all intermediate nights, (iii) at most one entertainment event per night is included, and (iv) at most one of each type of entertainment ticket is included. Given a feasible package, a client’s utility for that package is calculated as follows:

2. For notational convenience, we remap outbound flight j to $j - 1$, for $j \in \{2, 3, 4, 5\}$.

$$\text{utility} = 1000 - \text{travelPenalty} + \text{hotelBonus} + \text{funBonus} \quad (8)$$

where

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

3.3 Allocation

A TAC agent faces the allocation problem at the end of a game instance when it must find the assignment of goods to clients that maximizes total client utility. At the end of game 3065, **RoxyBot** held the set of goods listed in Table 3(a). The optimal allocation is depicted in Table 3(b), given the client preferences presented in Table 2. Packages were allocated to clients as shown, and the total utility obtained was 9999.

Good	1	2	3	4
R	2	2	1	2
S	2	0	2	0
T	1	1	1	0
G	4	3	3	1
F	2	2	0	1
I	6	0	2	0
O	1	4	2	1

Client	AD	DD	H	Tickets	Utility
1	1	2	G	S1, R2	1351
2	1	2	G	R1	1201
3	1	1	G	—	1147
4	3	3	G	R3	1275
5	1	2	F	R1, T2	1123
6	3	3	G	T3	1058
7	1	2	F	S1, R2	1282
8	1	4	G	T1, S3, R4	1562

Table 3: (a) **RoxyBot**’s final set of goods in game 3065. (b) **RoxyBot**’s final allocation in game 3065. The total utility is 9999.

4. Heuristic Solutions

In this section, we describe our heuristic search algorithm for solving bidding problems in the class BD as specified by the rules of TAC. (Our linear programming solution is described in a longer version of this paper.) We confine our attention to completion, since it is a simple matter to specialize any completion algorithm to an acquisition or allocation algorithm. Although we exploit some special structure of the TAC problem, we believe this approach is sufficiently generic to transfer to other practical problems in the realm of BD. For example, unlike linear programming solutions, heuristic search algorithms are not wedded to the assumption of linear utility functions.

An optimal completion involves assigning one (possibly null) travel package and one (possibly null) entertainment package to each of say I clients. This form of solution suggests formulating BD problems as search through a tree of depth $2I$, as shown in Figure 1. Search begins at the top of the tree with the set of available goods specified by the input pricelines. At each level of the tree, a package formed from the remaining goods is allocated to a client, and the costs indicated by the relevant pricelines are incurred. This setup corresponds to the “branch on bids” formulation of winner determination [7]. In this search tree, the maximum branching factor is 21 in the upper half and 73 in the lower half. The constraints of package availability and compatibility eliminate many branches at lower levels of the tree, but the full space of legal assignments is still enormous.

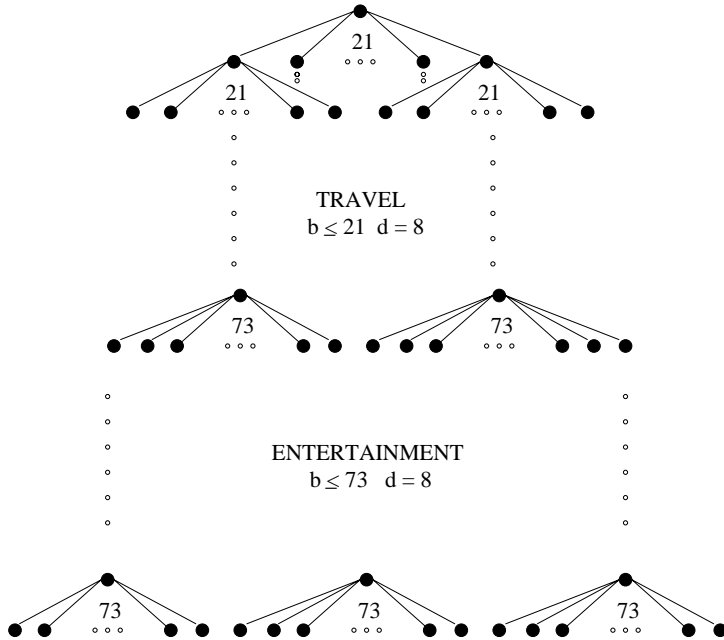


Figure 1: Search Tree Structure.

To tame the search space, we applied two varieties of heuristic AI search: optimal (A^* search) and approximate (beam search). Our A^* algorithm makes use of an intricate series of admissible heuristics, which we defer to a companion paper for lack of space [3]. Our beam search, however, which employs only one simple heuristic $f(x)$ to evaluate the promise of an intermediate search node x , proved to be extremely effective and scalable in practice. Recall that in beam search, search proceeds level by level with no backtracking; at each level, only the top B nodes according to the heuristic are expanded. The parameter B is called the “beam width.” Since our search tree is of fixed depth $2I$, beam search has the desirable property that it expands no more than $2IB$ nodes in total. Space and time requirements are therefore predictable.

Our heuristic $f(x)$ is inspired by the “rollout methods” that have been used in game-tree search (*e.g.*, [1, 9]). It works as follows: at each node x , we run a greedy algorithm to complete the assignment from x down to the bottom of the tree. Specifically, for a client that has thus far been assigned neither a travel package nor an entertainment package, s/he is assigned the travel and entertainment packages that (using only goods not already assigned) jointly maximize utility minus cost; for a client that already has a travel package, s/he is given the best entertainment package. The time to compute $f(x)$ is linear in the tree depth I . Therefore, the overall runtime of our beam search algorithm is quadratic in I . Since package assignments are made without replacement, this heuristic is inadmissible. It is, however, quite accurate, as we demonstrate in Section 5.

5. Experiments

We now present empirical results comparing heuristic search and integer linear programming on BD problems. We concentrate on allocation, since there is an objective testbed for allocation problems: the 16 games of the TAC finals. As there are eight agents per game, these games comprise client preferences and final holdings for 128 agents.

In our first series of tests we verified that the optimal utility values output by RoxyBot’s A^* algorithm were consistent with those of the ILP on our 128 datapoints. RoxyBot (which was designed

to report optimal allocations within the regulation four minutes) achieved a median run time of 0.59 seconds on a 600 MHz linux PC. Using CPLEX 6.5.3 on a 400 MHz SPARCstation with 2Gb of RAM, the ILP’s median run time was only 0.02 seconds. This series of tests confirmed what some TAC participants had already learned: optimal solutions are tractable for the dimensions of TAC.

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

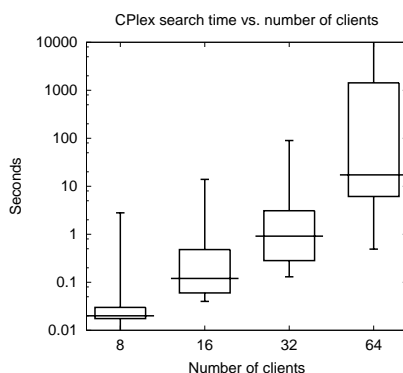


Figure 2: ILP running time at different problem sizes. (Note the log scale of the y -axis.)

By contrast, beam search scales predictably: its running time is quadratic in the number of clients and linear in the beam width. We therefore designed experiments to test its degree of suboptimality. We ran a series of tests with beam widths varying from 1 to 1280 on all our instances except the one whose optimal solution was unknown (because ILP was unable to solve it). We found that a beam width of only 1 (*i.e.*, best-first search) yielded a median accuracy of 99.4% in the 8 client case, with a median running time of less than 0.01 seconds. In the case of 64 clients, a beam width of 1 achieved a median accuracy of 97.9% in roughly 1 second. These figures are a strong testament to the power of the “rollout” heuristic used by our beam search. At the other extreme, a beam width of 1280 produced a median accuracy of 99.4% in the 64-client cases, but the median run-time was nearly 22 minutes. Figure 3 illustrates how search quality and running time, respectively, increase as a function of the beam width. Each datapoint is graphed as a box plot reflecting the minimum, 25th percentile, median, 75th percentile, and maximum value. Note that the running times have extremely low variance, while the performance is reliably above 96% of optimal for all but the smallest beam widths.

As allocation is the simplest of the BD problems defined, we do not expect either acquisition or completion to prove any less complex for the ILP, though this remains an open question. Beam search, however, solves allocation and completion in an identical way. We conclude that approximate heuristic search is a practical choice for use in the inner loop of a real-time bidding agent.

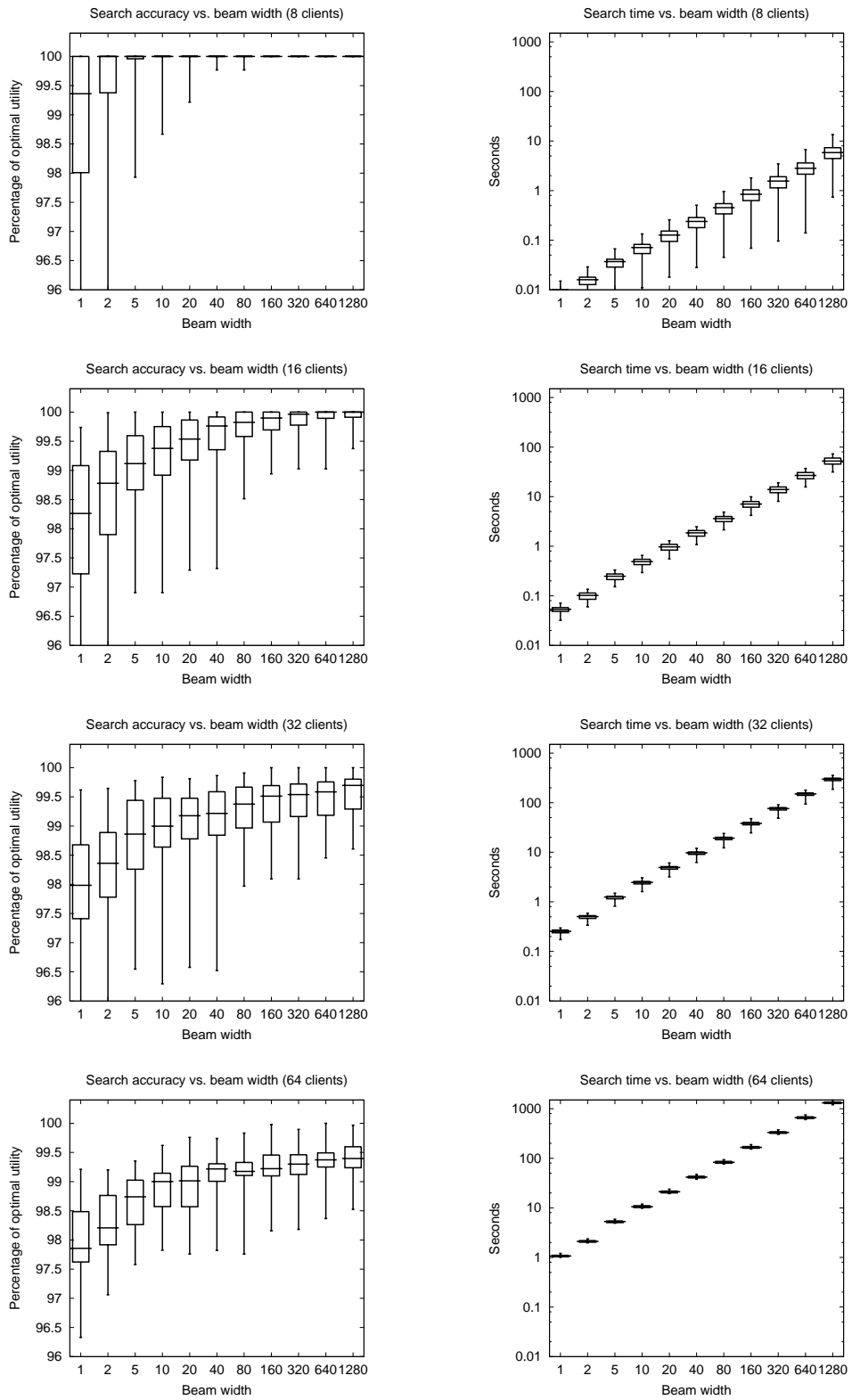


Figure 3: LEFT: Optimality of beam search, as a function of beam width and number of clients. (Note the offset on the y-axis.) RIGHT: Beam search runtimes, as a function of beam width and number of clients. (Note the log scale on the y-axis.)

6. Summary and Future Directions

In this paper we have achieved the following:

- proposed an architecture for intelligent bidding in simultaneous auctions
- defined a class of “bid determination” problems at the core of the proposed architecture
- described heuristic-search and integer-programming solutions to these problems on instances from the 2000 Trading Agent Competition

While BD problems form the core of the proposed architecture, the other major components—namely, estimation and bidding—are by no means trivial. Estimation requires predicting market dynamics on the basis of historical bid trajectories (taken over sets of goods, not only goods in isolation), and possibly modeling opponents’ behavior. Likewise, the strategic placement of bids and asks, particularly with regard to their timing, depends on analysis of the auction mechanism and opponent modeling. Moreover, even if all three major problems (estimation, completion, and strategic bidding/asking) could be solved optimally, the resulting behavior still need not be optimal. Most egregiously, the architecture as shown does not explicitly plan for uncertainty in the future dynamics of the auctions. An architecture that would be superior in this respect would bid based on distributions over estimated clearing prices, rather than on expected values. We see this as an exciting direction for future research in this area.

References

- [1] B. Abramson. Expected-outcome: A general model of static evaluation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(2):182–193, Feb. 1990.
- [2] Y. Fujishjima, K. Leyton-Brown, and Y. Shoham. Taming the computational complexity of combinatorial auctions: Optimal and approximate approaches. In *Proceedings of Sixteenth International Joint Conference on Artificial Intelligence*, pages 548–553, August 1999.
- [3] A. Greenwald and J. Boyan. Autonomous bidding agents in simultaneous auctions: A case study. <http://www.cs.brown.edu/amygreen/>, April 2001.
- [4] A. Greenwald and P. Stone. Autonomous bidding agents in the Trading Agent Competition. *IEEE Internet Computing*, April 2001.
- [5] K. Leyton-Brown, M. Pearson, and Y. Shoham. Towards a universal test suite for combinatorial auction algorithms. In *Proceedings of the ACM Conference on Electronic Commerce*, pages 66–76, October 2000.
- [6] M. H. Rothkopf, A. Pekeč, and R. M. Harstad. Computationally manageable combinatorial auctions. *Management Science*, 44(8), 1998.
- [7] T. Sandholm and S. Suri. Improved algorithms for optimal winner determination in combinatorial auctions and generalizations. In *Proceedings of AAAI*, pages 90–97, August 2000.
- [8] P. Stone, M. L. Littman, S. Singh, and M. Kearns. ATTac-2000: An adaptive autonomous bidding agent. In *Fifth International Conference on Autonomous Agents*, 2001.
- [9] G. Tesauro and G. R. Galperin. On-line policy improvement using Monte-Carlo search. In M. C. Mozer, M. I. Jordan, and T. Petsche, editors, *Advances in NIPS*, volume 9. MIT Press, 1997.
- [10] M. P. Wellman, P. R. Wurman, K. O’Malley, R. Bangerla, S.-d. Lin, D. Reeves, and W. E. Walsh. A trading agent competition. *IEEE Internet Computing*, April 2001.