Solving Combinatorial Auctions using Stochastic Local Search

Holger H. Hoos
Department of Computer Science
University of British Columbia
Vancouver, BC V6T 1Z4
hoos@cs.ubc.ca

Craig Boutilier
Department of Computer Science
University of Toronto
Toronto, ON M5S 3H5
cebly@cs.toronto.edu

Abstract

Combinatorial auctions (CAs) have emerged as an important model in economics and show promise as a useful tool for tackling resource allocation in AI. Unfortunately, winner determination for CAs is NP-hard and recent algorithms have difficulty with problems involving goods and bids beyond the hundreds. We apply a new stochastic local search algorithm, Casanova, to this problem, and demonstrate that it finds high quality (even optimal) solutions much faster than recently proposed methods (up to several orders of magnitude), particularly for large problems. We also propose a logical language for naturally expressing combinatorial bids in which a single logical bid corresponds to a large (often exponential) number of explicit bids. We show that Casanova performs much better than systematic methods on such problems.

1 Introduction

Auctions have been the focus of increasing study in AI. Certainly the emergence of E-commerce has made market mechanisms an attractive means for conducting business transactions and sales online. Furthermore, as nontrivial multiagent systems become more prevalent, researchers are looking to market protocols such as auctions as the basis for the coordination of agent activities or for resource allocation [5, 18].

When multiple items need to be sold, standard "single-item" auction protocols may be inappropriate, particularly when items exhibit *complementarities*. Specifically, when a bidder attaches a value to a *collection* of goods, associating a "value" with the individual elements is problematic. For example, if an agent requires two adjacent gates at an airport at a specific time—such that obtaining one slot is useless without the other—attaching independent values to each is difficult. Furthermore, bidding for them individually (e.g., in sequence [3, 6, 8] or in parallel [2, 15]) exposes the agent to certain risks (e.g., obtaining one item without the other). *Combinatorial auctions (CAs)* have been proposed as a means of dealing with such problems [14, 16, 18]. Instead of selling items individually, the seller allows bids on *bundles* of items, allowing bidders to deal with the entities of direct interest and avoid the risk of obtaining incomplete bundles. Given a set of combinatorial bids, the seller then decides how best to allocate individual goods to those bundles for which bids were placed, with the aim of maximizing revenue. Because bundles generally overlap, this is—conceptually—a straightforward optimization problem, and is in fact equivalent to weighted set packing. As a result, *optimal winner determination* for CAs is NP-complete [16].

A number of complete algorithms for winner determination have been proposed, including dynamic programming models [16], and algorithms for dealing with problems with special structure.[1] More recently, two proposals for applying AI-style search techniques have been used with some success for winner determination [7, 17]. In these proposals, the structure of bids is exploited to restrict the search—if the number of bids received is relatively sparse compared to the space of possible bids, these approaches perform much better than dynamic programming, and despite the computational complexity of the problem, have been shown to perform reasonably well on problems of moderate size.

When problem instances are large or when solutions are needed quickly, existing algorithms are likely to prove inadequate. In many, if not most, resource-allocation or E-commerce problems that are most readily modeled as CAs, it seems apparent that real-time response to very large problems will be expected. Complete algorithms—those designed to guarantee optimality—necessarily spend *considerable* time "proving" that the solution they produce is optimal at the expense of providing high-quality (though perhaps suboptimal) solutions quickly. Furthermore, as instances become larger, complete algorithms will, in most cases, become infeasible. While certain domains may require optimal solutions (e.g., for legal reasons), we expect that typical applications of CAs will be ideally suited for techniques that produce high-quality, approximate solutions quickly (or within a suitable time frame). Drawing an analogy to scheduling research, for example, large scheduling problems are invariably solved heuristically: even though the smallest improvements in schedule quality can have large economic consequences, the problems are simply to hard to be solved ex-

[1] Generally the structures investigated (e.g., restricting the size or structure of bids) are of interest because they allow one to obtain polynomial time algorithms; the existence of such structure in practice is often questionable (see Sandholm [17] for an overview of some special cases).

actly. For the same reasons, heuristic and approximation techniques for large CAs must be viewed as critical.

For these reasons, in this paper we consider the use of incomplete methods for winner determination. Specifically, we develop the *CASLS framework* for studying the solution of combinatorial auctions using *stochastic local search (SLS)* techniques. SLS has been used in AI and operations research for many decision and optimization problems with great success, and has generally proven more successful than systematic methods on a wide range of combinatorial problems. As we demonstrate in this paper, SLS can be applied with great success to the winner determination problem, finding high quality solutions much more quickly than systematic techniques and often finding optimal solutions. We also show that our techniques can tackle problem instances of considerably larger size than existing systematic methods. The nature of SLS does not permit one to offer solution quality or performance guarantees.[2] Instead we adopt the empirical methodology proposed by Hoos [9] to evaluate the success of SLS.

We also consider the use of logical languages to specify *schematic bids*. The CA problem is traditionally formulated by supposing that each bid is a bundle of items together with a bid value. However, there are many circumstances in which a bidder is indifferent between any of a number of different items or even different bundles of items. When such *substitutability* exists, requiring *explicit* bids imposes an undue burden on the bidder. To take one example, suppose a bidder wants any five of a collection of twenty items (e.g., five airport gate slots). The number of concrete bundles the user must bid on is over 15,000. By formulating the bid using a logical language such requirements can be expressed very concisely. We devise two languages for the logical specification of bids and examine the performance of SLS on bids so specified.

2 Combinatorial Auctions

2.1 Basic Model

We suppose a seller has a set of goods $G = \{g_1, ...g_{|G|}\}$ to be auctioned. Potential buyers value different subsets or *bundles* of goods, $b \subseteq G$, and offer bids of the form $\langle b, v\rangle$ where v is the amount the buyer is willing to pay for bundle b. Given a collection of bids $B = \{\langle b_i, v_i\rangle\}$, the seller must find an allocation of goods to bids that maximizes revenue. We define an *allocation* to be any $A = \{\langle b_i, v_i\rangle\} \subseteq B$ such that the bundles b_i making up A are disjoint. The *value* of an allocation $v(A)$ is given by $\sum\{v_i : \langle b_i, v_i\rangle \in A\}$. An *optimal allocation* is any allocation A with maximal value (taken over the space of allocations). The *optimal winner determination* problem is that of finding an optimal allocation given a bid set B. We call any algorithm that constructs some allocation, not necessarily optimal, a winner determination algorithm.

Notice that complementarities are naturally taken care of in this type of auction by allowing bidders to bid on collections of goods. Substitutability can be dealt with easily as well by allowing each bidder one dummy good that is inserted into each of her bids. If a bidder wants only one of several subsets of goods, she can bid on each subset but add the dummy good so that only one bid can be accepted. Because of this, winner determination need not rely on the identities of buyers, but only on the bids themselves.

We can view allocations in a slightly different way. Given a bid set B, an *assignment* is any function $f : G \to B$, assigning goods to specific bids. An assignment f induces an allocation $A_f = \{\langle b_i, v_i\rangle : f^{-1}(\langle b_i, v_i\rangle) \supseteq b_i\}$. Intuitively, given an assignment f, we consider allocated those bids that are "satisfied" by f. Unsatisfied bids (assigned less than their full complement of goods) are ignored. We can generally restrict our attention to assignments that only assign goods $g \in b$ to a bid $\langle b, v\rangle$ if we insert a dummy bid of value zero containing all goods.

The winner determination problem is equivalent to the weighted set packing problem [16] and as such is NP-complete. Algorithms for weighted set packing and related combinatorial problems can be used for winner determinations. Dynamic programming has been proposed for winner determination [16] but requires that the space of possible bids be enumerated, and thus is impractical for problems with a large number of goods (its complexity is independent of the number of actual bids). Search techniques have recently been proposed that exploit the fact that one need really only consider combinations of *actual bids*: if the set of actual bids is relatively sparse, such methods can work quite well.

The CASS algorithm developed by Fujishima, Leyton-Brown and Shoham [7] is good example of the effectiveness of search techniques. CASS uses a depth-first search to find optimal allocations; but clever structuring of the search space, preprocessing, heuristic ordering methods and pruning techniques allow the search to find optimal allocations rather effectively. Not surprisingly, CASS exhibits reasonable anytime performance as well, providing good allocations prior to finding optimal allocations. Sandholm [17] has also explored the use of search, developing an A*-formulation of the problem with good heuristics and pruning/preprocessing techniques. In both works, suitably structured search has proven to be quite computationally effective.

A number of approximation algorithms for weighted set packing have been developed in the literature, some based on local search. However, the emphasis in much of this work is on developing search strategies—or, more accurately, *local improvement strategies* for suboptimal solutions—that have provable quality guarantees rather than good practical applicability.[3] See [1, 4] for examples of such results. More practical stochastic search techniques such as tabu search and simulated annealing have been applied to related problems, but apparently not directly to weighted set packing.

[2] Indeed, as shown in [17], optimal winner determination is not even approximable in polytime.

[3] Specifically, none of the work cited here on approximation algorithms provides any empirical study of the actual approximation quality obtained in practice, only worst-case quality bounds.

2.2 A Language for Schematic Bids

In many cases buyers will have complex valuations for bundles of goods, reflecting the fact that certain goods or bundles can be substituted for one another. When combined with the natural complementarities captured by CAs, the set of explicit bids a buyer may need to reflect her true utility function may be very large. For example, should she desire either g_1 or h_1, *and* g_2 or h_2, *and* g_3 or h_3, she must formulate eight explicit bids (i.e., $\{g_1, g_2, g_3\}$, etc.). Complex requirements corresponding to a large number of explicit bids can often be expressed very compactly using a logical language. These bids, for example, can be captured using the logical formula $(g_1 \vee h_1) \wedge (g_2 \vee h_2) \wedge (g_3 \vee h_3)$.

To capture the logical structure of a set of bids, we introduce two logical languages for combinatorial bid specification. Given a set of goods G, a *clause over* G is any nonempty subset of G. Clauses over a set of goods are interpreted "disjunctively:" when a clause is part of a combinatorial bid, it expresses the fact that one (or more) of the goods in the clause is desired. A *clause set* is any (possibly empty) set of clauses over G. Clause sets are interpreted conjunctively: as part of a bid, a clause set is satisfied if each of its clauses is satisfied. Thus, a clause set expresses the fact that at least one good from each of its clauses is desired.[4] We can think of a clause set as a logical formula in conjunctive normal form (CNF) involving only positive literals (viewing each good as a logical atom). A *CNF bid* is any clause set f (positive CNF formula) together with an associated valuation v. Intuitively, such a bid means an agent is willing to pay v for an allocation of goods that "satisfies" the formula. We call the language of CNF bids $\mathcal{L}_{CA}^{cnf}$. Formally, we say a CNF bid $\langle c, v\rangle$ is *satisfied* by an assignment $f : G \rightarrow B$ (of goods to bids) iff $f^{-1}(c_i) \neq \emptyset$ for each $c_i \in c$; that is, if at least one good $g \in c_i$ from each clause $c_i \in c$ has been assigned to $\langle c, v\rangle$. The value of an assignment, or the allocation induced by an assignment, given a set of CNF bids is defined as the sum of the bid values of satisfied bids. Notice that simple bundles bids can be expressed trivially in this language; however, substitutability is expressible far more naturally using this logical language, obviating the need for dummy goods.

In many practical settings, a bidder will desire a subset of a set of "identical" goods offered for auction. For instance, a bidder may accept any five airport gates from a collection of twenty offered for lease. Expressing bids of this type in CNF can be cumbersome; furthermore, the size of the required set of explicit bids grows factorially with the size of the good collection of interest. For this reason, we consider an extended language, $\mathcal{L}_{CA}^{k\text{-}of}$, that allows *$k$-of clauses* having the form *k-of*(S), where $k > 0$ and $S \subset G$ is such that $|S| \geq k$. An *extended CNF bid* is any set of clauses or *k-of* clauses. Satisfaction of an extended bid is defined in the obvious way.

[4] If obtaining more than one of these goods increases value, then the bid should be expressed differently. The fact that obtaining multiple items from a set does not *decrease* value can be justified by assuming free disposal. Our algorithms will not assign more than one good to a clause in any case, though our approach could be extended to deal with undesirable items (i.e., "bads" along with goods).

3 Stochastic Local Search Applied to CAs

We now sketch a model for applying stochastic local search methods to the winner determination problem. There are several ways SLS techniques can be applied to CAs. Here we focus on the *CASLS* family of algorithms, which searches the space of feasible allocations (nonoverlapping subsets of bids) by selecting in each step a bid which is currently unsatisfied and modifying the current allocation such that this bid becomes satisfied.[5] Searching through feasible allocations has the advantage that the search steps can be easily scored, obviating the need to assign scores to partially-satisfied bids based on their "potential."

Formally, the neighborhood relation for CASLS algorithms is defined as follows: a_j is reachable from a_i iff a_j is determined by adding a new bid b to a_i and assigning the required goods to b; this may entail removing the goods from other bids in a_i. As a consequence, a_j is *adjacent* to a_i iff $a_j = a_i \cup \{b\} \setminus \{b' \in a_i : b' \cap b \neq \emptyset\}$ for some $b \notin a_i$. Thus a_j will generally consist of some subset of a_i together with a new bid b. Note that the adjacency relation is not symmetric (e.g., to return to a_i from a_j may require several steps); but any valid (nondominated) allocation a can be reached from any other in no more than $|a|$ steps. The neighborhood relation we use is analogous to that used for set packing in [4].

3.1 Casanova

Casanova is a CASLS algorithm that bears a strong resemblance to the Novelty$^+$ algorithm for SAT defined by Hoos [10], one of the best-performing algorithms for solving hard SAT problems known to date (see also the Novelty algorithm of [13]). It is based on scoring each search state using the "revenue per good" of the corresponding allocation. Since each neighbor can be reached by adding a bid (and adjusting), we write *sc*(b) to denote the increase in revenue obtained by adding b. The scoring function *score*(b) $=$ *sc*$(b)/$*length*(b) normalizes the revenue by the number of goods the bid "consumes".[6] During the search process, we define the *age* of each bid to be the number of steps since that bid was last selected (since initializing the search) to be added to a candidate solution.

Casanova starts with an empty allocation, where all goods assigned to a dummy bid and all real bids are unsatisfied. Then at each step, with probability *wp* (walk probability), a random unsatisfied bid is selected; with probability $1 -$ *wp* we select a bid "greedily" by ranking all bids according to their score. Then either the highest ranked bid b_1 or the second-highest b_2 is inserted into the solution as follows: if *age*$(b_1) \geq$ *age*(b_2), insert b_1; otherwise insert b_2 with probability *np* (novelty probability) and b_1 with probability $1 -$

[5] Note how this search scheme is analogous to the WalkSAT algorithm family for propositional satisfiability [13], where a currently satisfied clause is selected and satisfied in each search step; however, for CA there is no secondary selection involved in choosing how to satisfy a bid (there is only one way), unlike literal selection *within* the clause in WalkSAT.

[6] Revenue per good is commonly used to measure the quality of a bid in search approaches to CAs.

np. The search proceeds for *maxSteps* steps and is restarted with the empty allocation for a total of *maxTries* independent searches, with the best allocation found at any step of any search reported as the solution. Optionally, we also use a *soft restart strategy*, which reinitializes the search if at least θ_r search steps have occurred since the last initialization, but no improvement in revenue has been achieved within the last $\theta_r/2$ steps.

3.2 Explicit Bids: Empirical Evaluation

We tested Casanova on several random problem distributions, and compared its performance to CASS, the systematic search technique described in [7]. CASS is a complete algorithm which, given enough time, will find an optimal solution and prove its optimality. Casanova, like most SLS algorithms, is incomplete. In practice, given enough time, it may find optimal solutions, but it cannot be used to prove the optimality of any solution it finds. Both algorithms have useful anytime properties, as they generate and report intermediate solutions. But while CASS is deterministic, Casanova is a highly stochastic algorithm. Therefore, for Casanova, the time to achieve a given solution quality as well as the solution quality obtained after a fixed cutoff time are random variables. Both aspects have to be taken into consideration when comparing these two algorithms.

Generally, we performed two types of experiments: For large problem instances—those where CASS could not prove optimality of the best solution it found within 60 CPU seconds[7]—we measured the best solution obtained by CASS within the given cutoff time for each problem instance, while for Casanova, we measured a solution quality distribution over 10 runs of the algorithm. The cutoff times were chosen such that the experimental analysis could be conducted on a sufficient number of instances and in a reasonable amount of time; for bigger problems, we had to allow higher cutoff times to make sure that CASS would at least report the revenue for one candidate solution. For small problem instances, where CASS could prove the optimality of the solutions it found, we measured for each problem instance CASS's time to find the optimal solution. In order to ensure a fair comparison, we did not measure the total running time of CASS, which includes the time needed to "prove" the solution is optimal: we ran CASS to completion to ensure an optimal solution was found, and then determined the time at which the solution was *first* enumerated. For Casanova, we measured the distribution of the run time required to find the optimal solution. These run-time distributions (RTDs) were estimated from 100 runs of the algorithm for each given problem instance.

The CASS implementation we used is highly optimized and relies heavily on caching and pruning techniques. Likewise, Sandholm's bidtree algorithm [17], another systematic algorithm for winner determination, makes use of various preprocessing techniques. Casanova, on the other hand, has not been optimized for speed or memory, and did not have its parameters fine-tuned. Furthermore, we did not apply any pruning, preprocessing, or caching techniques.

Our test sets were generated according to several problem instance distributions known from the literature [17, 7]. These distributions are: UNI-p-g-b, Sandholm's uniform distribution where each instance comprises g goods, b bids, and each bid consists of p goods; DEC-p-g-b, Sandholm's decay distribution; EXP-p-g-b, the exponential distribution introduced in [7]; and BIN-p-g-b, the binomial distribution from [7]. Each of our test sets contains either 10 or 100 problem instances drawn from the same distribution, using identical parameter values.

The results for our first series of experiments, large problem instances with a fixed cutoff time, are reported in Table 1. For Casanova, we estimated the mean revenue from the distributions measured for each instance, while CASS gives a unique revenue for each instance. We report the median, the 90% percentile, and the 90%/10% percentile ratio as a measure of variation between instances. The variation in revenue over different runs of Casanova on the same instance was generally found to be very small (variation coefficient ≤ 0.01). Our results indicate clearly that Casanova gives superior solution quality for most of the test sets (the differences are up to 5.7% in median solution quality and up to 4.8% in the 90% percentile). The only exception is test set EXP-5-100-1000, where CASS gives a median solution quality which is 1.7% better than the mean revenue achieved by Casanova. However, the data suggests as the number of goods increase, Casanova's improvement relative to CASS also increases (see, e.g., the results for UNI-3-100-1000 and UNI-3-200-2000); in particular, for the larger EXP-5-500-5000 instances, Casanova outperforms CASS. Finally, it should be noted that the variation of solution quality over the individual test sets is generally smaller for Casanova than for CASS; this indicates that Casanova finds good solutions more consistently.

While Table 1 summarizes our results, the underlying analyses we performed are much more detailed. Figure 1 shows a typical scatter plot of the correlation between the (mean) revenue obtained by running CASS vs. Casanova on each instance across a test set (here we illustrate the results for UNI-3-200-10000). The data shows clearly that for almost all instances Casanova finds better solutions than CASS. Furthermore, the variation in solution quality is significantly smaller for Casanova than for CASS. Finally, there is no apparent correlation between the solution quality achieved by the two algorithms, suggesting that there are no differences in the intrinsic hardness of the instances of the test set. We also analyzed the dependence of these results on the cutoff time chosen. Figure 2 shows the revenue for Casanova vs. CASS for a typical instance of test set UNI-3-200-2000. Clearly, Casanova gives consistently better solution quality in this (typical) case, even when basing the comparison on the worst performance observed for Casanova over 10 runs. It should also be noted that for Casanova, the solution quality increases steadily over time, while for CASS, short series of rapid improvements are typically followed by long quiescent phases.

[7] All experiments were performed on a Pentium II 400Mhz with 512KB CPU cache and 128MB RAM, running Linux 2.2.15.

test set	# inst	cutoff	CASS			Casanova			np	wp	θ_r
			median	Q_{90}	Q_{10}/Q_{90}	median	Q_{90}	Q_{10}/Q_{90}			
UNI-3-100-1000	100	10s	130396	133838	1.05	**134216**	136203	1.03	0.5	0.15	–
UNI-3-200-2000	100	10s	252084	257643	1.04	**264814**	267573	1.02	0.5	0.15	–
UNI-3-100-5000	100	30s	142947	144015	1.02	**143886**	144666	1.01	0.5	0.02	–
UNI-3-200-10000	100	60s	281413	284033	1.02	**286164**	287632	1.01	0.5	0.02	–
BIN-0.01-500-5000	10	60s	583279	594931	1.04	**616708**	623624	1.04	0.1	0.01	1000
DEC-0.75-500-5000	10	60s	668458	678830	1.04	**675198**	279919	1.01	0.5	0.02	1000
EXP-5-100-1000	10	30s	**135027**	135658	1.03	132705	134412	1.03	0.05	0.02	1000
EXP-5-500-5000	10	60s	647629	650302	1.02	**655329**	659238	1.02	0.05	0.02	1000

Table 1: Regular bids: Comparison of solution quality (revenue) achieved by CASS and Casanova when using the same fixed cutoff time. We report statistics of the distribution across the test set, the Q_x are the $x\%$ percentiles. For Casanova, our analysis is based on the mean solution quality measured over 10 runs for each instance.

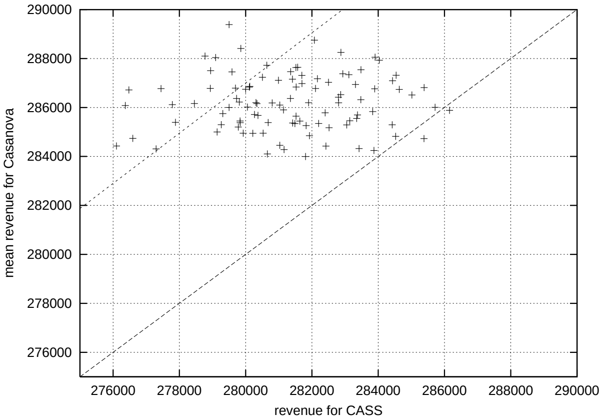

Figure 1: Regular bids, test set UNI-3-200-10000: Correlation of (mean) revenue obtained by CASS and Casanova within a fixed cutoff time of 60 CPU sec. The two lines show ratios of 1 and 1.025 when comparing the revenue obtained by Casanova to that of CASS.

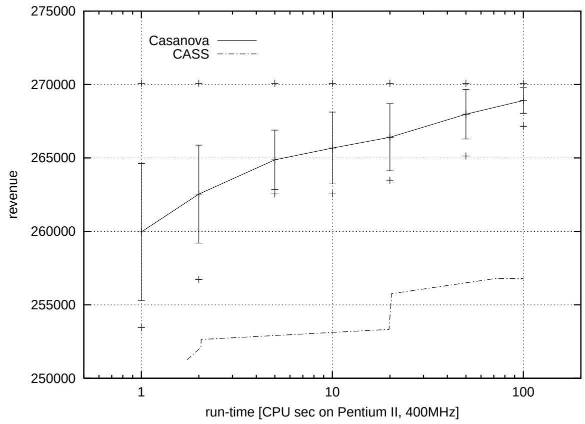

Figure 2: Regular bids, typical instance from test set UNI-3-200-2000: revenue over run time for CASS vs. Casanova. For Casanova, the solid line represents mean revenue and the error bars indicate ± 1 stddev. The data points above and below the error bars show the min and max revenue found over 10 runs. CASS does not report any revenue before reaching 1.73 CPU seconds.

Finally, it can be seen that for Casanova, the variability of the solution quality over multiple runs decreases over time. Together with the fact that the maximal revenue remains constant, this suggests that the best solution found by Casanova (revenue=270075) might be the optimal solution to the problem. Overall, these observations illustrate the superior anytime behavior of the Casanova algorithm.

In our second series of experiments, we compared the time required by Casanova vs. CASS to find optimal solutions. The results are reported in Table 2. For each instance, we measured the time required by CASS to find an optimal solution, and estimated the time to obtain the same revenue with Casanova from an RTD constructed from 100 runs. The results indicate clearly that for certain types of problems (particularly UNI$*$, but also DEC$*$), Casanova is dramatically faster than CASS in finding optimal solutions, while for others (EXP$*$ and BIN$*$) CASS is clearly superior for the small instances tested here. However, it is remarkable that Casanova, although incomplete, finds optimal solutions for all instances tested.

It should be noted that for systematic search algorithms, like CASS and Sandholm's bidtree procedure, the UNI$*$ instances are extremely hard [17]. For these, Casanova finds optimal solutions between one and three orders of magnitude faster than CASS.[8] Furthermore, our results clearly indicate that Casanova's search time increases with problem size at a significantly lower rate than CASS's.

As we did with the larger instances when using fixed cutoff times, we studied the correlation between the performance of CASS and Casanova. Figure 3 shows a typical result; each point corresponds to the data obtained for one instance from the UNI-3-100-100 test set. Clearly, the mean run time required by Casanova to find an optimal solution is generally much lower than for CASS, typically by about a factor of

[8] Comparing the results reported here for CASS with those for bidtree from [17] strongly suggests that for the instance distributions tested here, CASS is up to one order of magnitude faster in finding optimal solutions.

test set	# inst	CASS			Casanova			np	wp	θ_r
		median	Q_{90}	Q_{10}/Q_{90}	median	Q_{90}	Q_{10}/Q_{90}			
UNI-3-50-50	100	0.058	0.125	9.09	**0.0092**	0.029	7.61	0.5	0.15	–
UNI-3-75-75	100	2.211	6.222	10.91	**0.030**	0.197	24.99	0.5	0.15	–
UNI-3-100-100	100	96.41	446.50	27.17	**0.136**	0.964	36.24	0.5	0.15	–
UNI-3-50-100	100	0.487	1.40	15.20	**0.091**	0.543	21.29	0.5	0.15	–
UNI-3-75-150	100	125.76	409.95	17.24	**1.078**	3.974	25.59	0.5	0.15	–
UNI-3-20-2000	100	33.99	140.55	462.86	**1.725**	5.160	6.911	0.5	0.15	–
UNI-10-200-200	100	147.99	308.92	15.72	**1.677**	6.051	10.54	0.5	0.15	–
BIN-0.2-20-500	100	**0.051**	0.066	1.48	7.980	31.447	11.08	0.2	0.02	–
DEC-0.75-200-200	10	252.82	1061.04	44.62	**6.236**	632.35	800.747	0.5	0.02	–
EXP-5-20-500	100	**0.0282**	0.0315	1.21	0.852	8.689	749.01	0.5	0.05	–

Table 2: Regular bids: Comparison of time (in CPU seconds) required by CASS and Casanova for finding optimal solutions. We report statistics of the distribution across the test set, the Q_x are the $x\%$ percentiles. For Casanova, the performance measure is the mean time for finding an optimal solution.

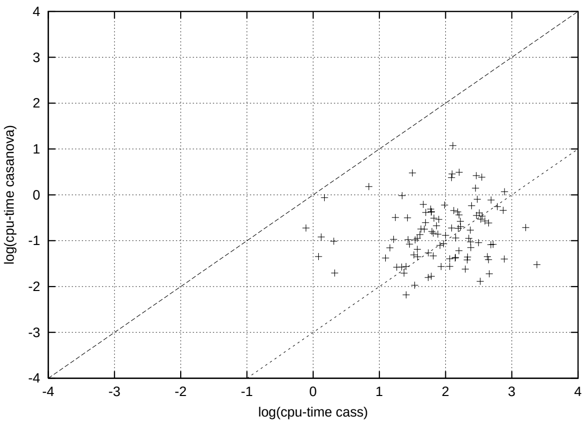

Figure 3: Regular bids, test set UNI-3-100-100: Correlation of CPU time (in CPU seconds) required by CASS and Casanova for finding optimal solutions. The two lines show factors of 1 and 1000 between the solution time for CASS and Casanova.

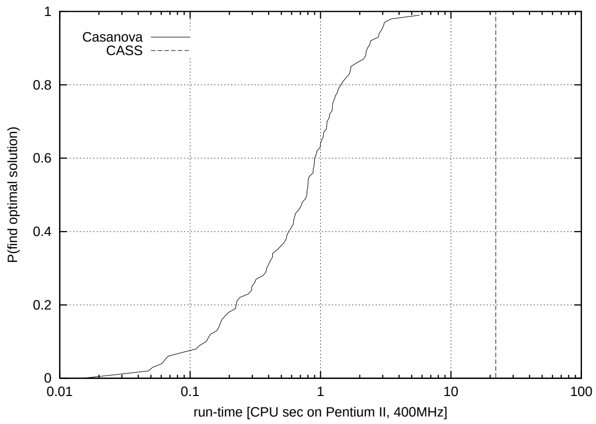

Figure 4: Regular bids, typical instance from test set UNI-3-100-100: Empirical RTDs for finding the optimal solution using Casanova vs. CASS, based on 100 runs for Casanova, one deterministic run for CASS.

1000; in fact, Casanova's mean solution time is better than CASS's in *all* instances in this test set. Again, there is no evidence of significant correlation between the performance of the two algorithms, suggesting that the features which render instances of this test set difficult for either of these algorithms are different and independent.

For Casanova, we are interested not only in the distribution of mean search cost within each test set, but also in the variation between multiple runs on a single instance. We therefore measured RTDs for all individual instances of each test set and characterized these by fitting functional models to the empirical data, using the methodology developed by Hoos and Stützle [9]. Figure 4 shows a rather typical example of the RTDs for Casanova and CASS (the latter of which is a step function due to the deterministic nature of the algorithm) for an instance from test set UNI-3-100-100. Clearly, Casanova is not only superior to CASS when comparing the mean solution time, but also when comparing higher percentiles of the RTDs. Surprisingly, we found that for all instances we checked (except for some extremely easy ones, where the data is too discrete due to the limited precision of CPU timing) the RTDs for Casanova could be successfully approximated with exponential distributions.[9] Based on this characterization we can conclude that with a probability of about 0.63 Casanova's actual search cost will be smaller than the mean solution time reported here.

3.3 Schematic Bids: Empirical Evaluation

To compare CASS and Casanova on schematic bids, we generated schematic bids randomly and then converted the schematic bids into a set of explicit bids on which the algorithms were run. Thus, a CNF bid b with c clauses and d

[9] These approximations were obtained by fitting the empirical RTD-data with the cdf of an exponential distribution and validated using the χ^2-test, a standard statistical test.

disjuncts per clause, for example, would generate d^c explicit bids, corresponding to the selection of one good from each clause. Each explicit bid generated from a CNF bid b also includes a dummy good g_b that prevents more than one explicit bid from being satisfied. Thus each explicit bid will have $c+1$ goods. Note that the number of explicit bids is exponential in the size of the CNF bid (and has a factorial component for *k-of* bids). This restricts the number of schematic bids we can handle. For CNF bids, we use two problem distributions. The CUNI-c-d-b-g problem distribution involves g goods and b bids, each bid consisting of c clauses with d disjuncts each. The CPOIS-m-n-b-g distribution generates bids where the number of clauses, and number of disjuncts in each clause, are generated using a Poisson distribution (with means m and n, respectively).[10] Thus the bids are variable length; but unlike the EXP-distribution, we do not have a mode at 1 (we expect the Poisson to be a more realistic model of bidding behavior). For *k-of* bids, we conducted preliminary experiments on a version of the CUNI$*$ bids, where each CNF clause was turned into a *k-of* clause. These problem distributions are denoted KUNI-c-d-k-b-g.

We performed two experimental series analogous to those for explicit bids described above. The results are reported in Tables 3 and 4. For the CUNI$*$ test sets, Casanova achieved between 15% and 35% more revenue than CASS for an identical cutoff time, with a clear trend for this difference in performance to increase with problem size. Casanova's performance for the CPOIS$*$ test sets is equally impressive. For the second series of experiments, the problem size was restricted by the time required by CASS to find optimal solutions and prove optimality.[11] Our results show that the CUNI$*$ instances—which are solved by Casanova in less than one CPU second—are extremely hard for CASS, which requires a median time of more than 10 CPU minutes. For the CPOIS$*$ as well as for the KUNI$*$ instances we observe a similar advantage of Casanova over CASS. Generally, the variation of search cost across the test sets is significantly lower for Casanova than for CASS, indicating a more robust performance.

3.4 Interpretation

Casanova outperforms CASS on large problem instances with fixed cutoff times (over various distributions); and on smaller instances, though incomplete, Casanova generally finds optimal solutions. On uniform problems, Casanova finds optimal solutions much faster than CASS. For other problem types, such as the one based on exponential bid-length distributions introduced in [7], the improvement shown by Casanova is less significant, and for smaller problem instances, CASS clearly has an advantage. These performance differences seem to be explained by the distribution of bid lengths and prices. While the UNI$*$ instances have no variation in bid length and relatively low variation of prices, the EXP$*$ instances are characterised by an extreme variation in both bid length and prices. We found that the variation coefficient of bid prices from all distributions is correlated with the performance difference: increased efficiency of Casanova is observed for lower values of the variation coefficient. This observation is confirmed by the results we obtained for the CNF instances, which are characterised by a low variation in bid prices and clusters of bids with identical price—here Casanova outperforms CASS significantly. This fact is likely to be of great practical import: large allocation problems will often be characterized by large numbers of bids with identical prices (corresponding to large-scale substitution effects) and many prices with reasonably low variability.

The experimental results presented here also indicate that Casanova's performance improves relative to CASS's with growing problem size. This suggests that the preprocessing and pruning techniques which are crucial for the efficiency of the systematic search algorithms are more adversely affected by growing problem size than the stochastic local search heuristics used by Casanova. Overall, our results suggest that for solving large problem instances with several hundred goods and thousands of bids, SLS algorithms like Casanova offers considerable advantages over current systematic search procedures. Finally, SLS algorithms like Casanova offer another important advantage over deterministic systematic search methods in that they can be parallelized easily with significant speedup (given the approximately exponentially-distributed running time).

4 Concluding Remarks

We have developed the CASLS framework for applying stochastic local search to combinatorial auction winner determination and have demonstrated the effectiveness of Casanova, a specific instantiation of this model. While the initial results presented here are very encouraging, we believe that CASLS offers hope for much better performance—through both the solution of larger problem instances and the provision of better anytime behavior—than demonstrated here. We have investigated only one, relatively straightforward SLS method in this paper, and have done very little parameter tuning. Our future investigations will include the examination of better scoring functions, different problem distributions and the use of more sophisticated SLS techniques. In particular, Iterated Local Search algorithms [12] appears to hold significant promise.

We also intend to explore techniques for solving problems involving schematic bids without explicit conversion to explicit form. To do this we will exploit the strong analogy between CAs with schematic bids and propositional satisfiability problems. This holds promise for significantly extending the scope and scale of problems that can be effectively dealt with. We are also currently investigating new classes of bidding languages which offer natural ways of expressing common utility functions, at the same time offering structure that can be exploited computationally.

Finally, we hope to extend our approach to deal with more

[10] Specifically, the number of clauses $c \sim 1 + Pois(m)$ to ensure a positive c; similarly for the number of goods per clause.

[11] Again, we emphasize that we compare to the time required by CASS to *find* the optimal solution, not to run to completion and prove optimality.

test set	# inst	cutoff	CASS			Casanova			np	wp	θ_r
			median	Q_{90}	Q_{10}/Q_{90}	median	Q_{90}	Q_{10}/Q_{90}			
CUNI-3-50-50	100	10s	55015	58479	1.15	**63360**	65745	1.09	0.5	0.15	–
CUNI-3-100-100	100	60s	104868	108687	1.10	**127011**	130440	1.06	0.5	0.15	–
CUNI-3-50-250	100	60s	52245	56943	1.20	**70158**	70551	1.02	0.5	0.15	–
CPOIS-2-50-50	100	10s	53204	56397	1.15	**60398**	63115	1.10	0.5	0.15	–
CPOIS-2-100-100	100	60s	99238	105275	1.13	**117889**	122673	1.0691	0.5	0.15	–
CPOIS-2-50-250	100	60s	53066	56094	1.13	**69608**	70755	1.03	0.5	0.15	–
CPOIS-2-100-500	100	60s	101568	105941	1.10	**135973**	138266	1.03	0.5	0.15	–
KUNI-2-4-2-100-100	10	60s	48812	50608	1.20	**59938**	63194	1.09	0.5	0.15	–

Table 3: CNF and *k-of* bids: Comparison of solution quality (revenue) achieved by CASS and Casanova when using the same fixed cutoff time.

test set	# inst	CASS			Casanova			np	wp	θ_r
		median	Q_{90}	Q_{10}/Q_{90}	median	Q_{90}	Q_{10}/Q_{90}			
CUNI-3-20-20	100	791.11	2904.89	180.58	**0.050**	0.138	5.24	0.5	0.15	–
CPOIS-2-20-20	100	1.855	9.355	41.15	**0.240**	1.048	17.74	0.5	0.15	–
KUNI-2-4-2-20-20	100	24.364	48.690	72.40	**0.474**	4.678	24.40	0.5	0.15	–

Table 4: CNF and *k-of* bids: Comparison of time (in CPU seconds) required by CASS and Casanova for finding optimal solutions.

sophisticated domains, for example, those involving a temporal component such as scheduling tasks [18], and multi-item combinatorial auctions of the type explored in [11].

Acknowledgements

Thanks are due to Kevin Leyton-Brown, Tuomas Sandholm, Yoav Shoham, and Moshe Tennenholtz for their comments and general discussion of these issues. We are especially indebted to Kevin for his CASS software and for his very generous and helpful support in adapting the code to our needs. This research was supported by IRIS Phase-III Grant "Preference Elicitation and Interactive Optimization."

References

[1] E. Arkin and R. Hassin. On local search for weighted k-set packing. *ESA-97*, pp.13–22, Graz, 1997.

[2] S. Bikhchandani and J. Mamer. Competitive equilibria in and exchange economy with indivisibilities. *J. Econ. Th.*, 74:385–413, 1997.

[3] C. Boutilier, M. Goldszmidt, and B. Sabata. Sequential auctions for allocation of resources with complementarities. *IJCAI-99*, pp.527–534, Stockholm, 1999.

[4] B. Chandra and M. Halldórsson. Greedy local improvement and weighted set packing approximation. *SODA-99*, pp.169–176, Baltimore, 1999.

[5] S. Clearwater, ed. *Market-based Control: A Paradigm for Distributed Resource Allocation*. World Scientific, 1995.

[6] R. Engelbrecht-Wiggans and R. Weber. A sequential auction involving assymetrically informed bidders. *Int. J. Game Th.*, 12:123–127, 1983.

[7] Y. Fujisima, K. Leyton-Brown, and Y. Shoham. Taming the computational complexity of combinatorial auctions. *IJCAI-99*, pp.548–553, Stockholm, 1999.

[8] D. Hausch. Multi-object auctions: Sequential vs. simultaneous sales. *Mgt. Sci.*, 32(12):1599–1610, 1986.

[9] H. Hoos and T. Stützle. Evaluating Las Vegas Algorithms–Pitfalls and Remedies. *UAI-98*, pp.238–245, Madison, 1998.

[10] H. Hoos. On the Run-time Behaviour of Stochastic Local Search Algorithms for SAT. *AAAI-99*, pp.661–666, Orlando, FL, 1999.

[11] K. Leyton-Brown and Y. Shoham and M. Tennenholtz. An algorithm for multi-unit combinatorial auctions. to appear, *AAAI-2000*, Austin, TX, 2000.

[12] O. Martin, S. Otto, E. Felten. Large-step Markov chains for the traveling salesman problem. *Compl. Sys.*, 5:299–326, 1991.

[13] D. McAllester, H. Kautz, B. Selman. Evidence for invariants in local search. *AAAI-97*, 321–326, Providence, RI, 1997.

[14] S. Rassenti, V. Smith, and R. Bulfin. A combinatorial auction mechanism for airport time slot allocation. *Bell J. Econ.*, 13:402–417, 1982.

[15] M. Rothkopf. Bidding in simultaneous auctions with a constraint on exposure. *Op. Res.*, 25:620–629, 1977.

[16] M. Rothkopf, A. Pekeč, and R. Harstad. Computationally manageable combinatorial auctions. *Mgt. Sci.*, 44(8):1131–1147, 1998.

[17] T. Sandholm. An algorithm for optimal winner determination in combinatorial auctions. *IJCAI-99*, pp.542–547, Stockholm, 1999. Extended , Washington Univ. Report WUCS-99-01.

[18] M. Wellman, W. Walsh, P. Wurman, and J. MacKie-Mason. Auction protocols for decentralized scheduling. *Games and Econ. Behavior*, 1999. To appear.