Solving Concisely Expressed Combinatorial Auction Problems

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

Abstract

Combinatorial auctions provide a valuable mechanism for the allocation of goods in settings where buyer valuations exhibit complex structure with respect to substitutability and complementarity. Most algorithms are designed to work with explicit "flat" bids for concrete bundles of goods. However, logical bidding languages allow the expression of complex utility functions in a natural and concise way, and have recently attracted considerable attention.

Despite the power of logical languages, no current winner determination algorithms exploit the specific structure of logically specified bids to solve problems more effectively. In this paper, we describe techniques to do just this. Specifically, we propose a direct integer program (IP) formulation of the winner determination problem for bids in the $\mathcal{L}_{GB}$ logical language. This formulation is linear in the size of the problem and can be solved effectively using standard optimization packages. We compare this formulation and its solution time to those of the corresponding set of flat bids, demonstrating the immense utility of exploiting the structure of logically expressed bids. We also consider an extension of $\mathcal{L}_{GB}$ and show that these can also be solved using linear constraints.

1 Introduction

Combinatorial auctions (CAs) generalize traditional market mechanisms to allow the direct specification of bids over *bundles* of items [10; 11; 16]. When a bidder's preferences exhibit complex structure with respect to complementarity and substitutability, such combinatorial (or bundle) bids allow bidders to 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, equivalent to weighted set packing. As a result, *winner determination* for CAs is NP-complete [11].

By expressing her preferences, or prices, directly over bundles, a potential buyer can, in principle, very accurately reflect her utility function, regardless of its structure. In practice, however, specifying explicit "flat" bids over all relevant bundles may be difficult: many utility functions will require the specification of a number of bundle bids that is exponential in the number of goods of interest to the bidder. This is especially true for utility functions involving the complementarities and substitutability for which CAs are best-suited. To circumvent this, several researchers have proposed *logical bidding languages* that allow might allow complex utility functions to be expressed relatively concisely in a suitable language [12; 13; 5; 8; 2]. The recent $\mathcal{L}_{GB}$ language of Boutilier and Hoos [2], for example, allows goods to be "joined" using logical connectives, and prices to be attached to arbitrary subformulae. Despite their attractiveness, the computational aspects of logical bidding languages have received little attention. Indeed, no studies of which we are aware exploit the structure of logically specified bids in winner determination. Instead, a set of logical bids is usually converted to an equivalent set of flat bids and solved using methods designed for flat bids.

In this paper, we solve the winner determination problem for $\mathcal{L}_{GB}$ problems without conversion to flat bids. Rather we directly formulate the optimization problem in a way that exploits the structure of underlying bids. More precisely, we describe a very concise integer program (IP) formulation of the winner determination problem for $\mathcal{L}_{GB}$ that makes the logical structure explicit. The well-documented fact that the number of flat bids required to capture a particular problem may be exponentially larger than the set of logical bids suggests that this strategy could be useful. However, it could be that standard optimization techniques can discover the "lost" structure in a set of flat bids (and hence solve the flat problem effectively) or that the structure cannot be exploited (and hence the structured problem cannot be solved effectively). Our results show that neither is the case: the direct solution of structured problems offers immense computational savings in winner determination. Since logical languages generally, and $\mathcal{L}_{GB}$ specifically, offer advantages both in terms of the expression of bids and in winner determination, we expect that this approach will prove vital for handling large CAs.

The paper is organized as follows. We describe relevant background on CAs, winner determination, and bidding languages in Section 2. We focus on the $\mathcal{L}_{GB}$ language since it is fully expressive, and strictly more compact than any other lan-

guage in the literature. In Section 3 we describe the IP formulation of the winner determination problem for $\mathcal{L}_{GB}$. Through the introduction of several auxiliary variables, this formulation can be made very compact, linear in the size of the set of logical bids. We describe an extension of $\mathcal{L}_{GB}$ and how it also can be modeled using a concise set of constraints. We also show how an equivalent set of flat bids can be constructed and solved using the "standard" IP formulation. We present empirical results in Section 4 showing that conversion to flat bids cannot be competitive for problems of even moderate size.

2 CAs and Bidding Languages

In this section, we briefly review CAs and logical bidding languages.

2.1 Combinatorial Auctions

We suppose a seller has a set of goods $G = \{g_1, \ldots, g_n\}$ to be auctioned. Potential buyers value different subsets or *bundles* of goods, $b \subseteq G$, and offer bids of the form $\langle b, p\rangle$ where p is the amount the buyer is willing to pay for bundle b. We often use the term "flat bid" for such a bundle bid, to distinguish it from the structured bids we consider below. Given a collection of bids $B = \{\langle b_i, p_i\rangle : i \leq m\}$, the seller must find an allocation of goods to bids that maximizes revenue. We define an *allocation* to be any $L = \{\langle b_i, p_i\rangle\} \subseteq B$ such that the bundles b_i making up L are disjoint. The *value* of an allocation $v(L)$ is given by $\sum\{p_i : \langle b_i, p_i\rangle \in L\}$. An *optimal allocation* is any allocation L with maximal value (taken over the space of allocations). The *winner determination* problem is that of finding an optimal allocation given a bid set B. We sometimes consider *assignments* $A : G \rightarrow B$ of goods to bids. Assignment A induces allocation L_A whose bids are those that have been assigned all required goods (i.e., $b_i \subseteq A^{-1}(\langle b_i, p_i\rangle)$).

The winner determination problem is a straightforward combinatorial optimization problem, and can be formulated quite directly as an IP. Let x_i be a boolean variable indicating whether bid b_i is satisfied. Then we wish to solve the IP:

$$\textit{Maximize:} \sum_i p_i x_i \tag{1}$$

$$\textit{Subject to:} \sum\{x_i : g_k \in b_i\} \leq 1, \forall k \leq n \tag{2}$$

This formulation has m variables (one per bid) and n constraints (one per good), with constraints having z terms on average, where z is the average number of bids in which a good occurs. Winner determination is equivalent to the weighted set packing problem [11] and as such is NP-complete. Despite this, generic combinatorial optimization techniques seem to work quite well in practice. For example, results reported in [1; 15] suggest that using generic CPLEX IP solution techniques is reasonably competitive with recent algorithms designed specifically for CAs. Recent search algorithms—both complete methods [4; 12; 14] as well as stochastic techniques [5]—have been proposed in the AI literature and have also proven quite successful solving problems of reasonable size, often running faster than CPLEX.

2.2 Logical Bidding Languages

Most work on combinatorial auctions assumes that a bid is expressed using a simple bundle of goods associated with a price for that bundle. However, a buyer with a complex utility function will generally need to express multiple flat bids in order to accurately reflect her utility function. Logical bidding languages overcome this by allowing a bidder to express a single bid in which the logical structure of the utility function is captured. A number of different types of bidding languages have been proposed in the literature, among these languages that allow flat bids to be combined logically [12; 13; 8], and that allow goods to be combined logically [5].

The recent $\mathcal{L}_{GB}$ language of Boutilier and Hoos [2] generalizes these languages by allowing goods to be "joined" using logical connectives, and prices to be attached to arbitrary subformulae. $\mathcal{L}_{GB}$ is fully expressive (i.e., can express any utility function over goods) and is strictly more compact than existing languages (i.e., any bid expressible in these languages can be expressed at least as concisely in $\mathcal{L}_{GB}$). Indeed, for certain natural classes of utility functions, $\mathcal{L}_{GB}$ can express bids exponentially more compactly than any proposed languages [2]. For this reason, we focus on $\mathcal{L}_{GB}$.

Let G denote the set of goods, forming the atomic elements of the language. The syntax of $\mathcal{L}_{GB}$ is defined as follows:

- $\langle g, p\rangle \in \mathcal{L}_{GB}$, for any good $g \in G$ and any non-negative price $p \in \mathbf{R}_0^+$.
- If $b_1, b_2 \in \mathcal{L}_{GB}$, then $\langle b_1 \wedge b_2, p\rangle$, $\langle b_1 \vee b_2, p\rangle$, and $\langle b_1 \oplus b_2, p\rangle$ are all in $\mathcal{L}_{GB}$ for any non-negative price p.

Bids so-defined correspond to arbitrary propositional formulae over the goods, using connectives $\wedge$ (conjunction), $\vee$ (disjunction) and $\oplus$ (XOR), where each subformula is annotated with a price. We often don't mention the price for a subformula if $p = 0$, and loosely say that no price is associated with such a subformula. Examples of sentences include

$$(a : 1 \wedge b : 2) : 5 \quad \text{and} \quad (a \vee b) : 2 \oplus c : 3.$$

A sentence $b \in \mathcal{L}_{GB}$ is a *generalized logical bid (GLB)*. The *formula associated with* b, denoted $\Phi(b)$, is the logical formula obtained by removing all prices from subformulae.

The semantics of GLBs defines the price to be paid by a bidder given a particular assignment of goods to her GLB. Roughly, the underlying idea is that the *value* of a GLB b is given by summing the prices associated with all satisfied subformulae (with one exception). We first define what it means for an assignment to satisfy a (priceless) formula.

Let A be an assignment $A : G \rightarrow B$ of goods to GLBs. Let $\Phi(b)$ be the formula associated with b. We write $\sigma(\Phi(b), A) = 1$ to denote that A *satisfies* b, and $\sigma(b, A) = 0$ to denote that A does not satisfy b. The relation is defined as follows:

- If $\Phi(b) = g$ for some $g \in G$ then $\sigma(\Phi(b), A) = 1$ iff $A(g) = b$.
- If $\Phi(b) = \Phi_1 \vee \Phi_2$ or $\Phi(b) = \Phi_1 \oplus \Phi_2$ then $\sigma(\Phi(b), A) = \max(\sigma(\Phi_1, A), \sigma(\Phi_2, A))$
- If $\Phi(b) = \Phi_1 \wedge \Phi_2$ then $\sigma(\Phi(b), A) = \min(\sigma(\Phi_1, A), \sigma(\Phi_2, A))$

Given a bid b and assignment A of goods to bids, we define the *value of* b *under* A, denoted $\Psi(b, A)$, recursively. If g is a good, b_1, b_2 are bids, and p is a price:

$$\Psi(\langle g, p\rangle, A) = p \cdot \sigma(g, A)$$
$$\Psi(\langle b_1 \wedge b_2, p\rangle, A) =$$
$$\Psi(b_1, A) + \Psi(b_2, A) + p \cdot \sigma(\Phi(b_1) \wedge \Phi(b_2), A)$$
$$\Psi(\langle b_1 \vee b_2, p\rangle, A) =$$
$$\Psi(b_1, A) + \Psi(b_2, A) + p \cdot \sigma(\Phi(b_1) \vee \Phi(b_2), A)$$
$$\Psi(\langle b_1 \oplus b_2, p\rangle, A) =$$
$$\max\{\Psi(b_1, A), \Psi(b_2, A)\} + p \cdot \sigma(\Phi(b_1) \vee \Phi(b_2), A)$$

Intuitively, the value of a bid is the value of its components, together with the additional price p if certain logical conditions are met. $\langle b_1 \wedge b_2, p\rangle$ pays price p if the formulae associated with both b_1 and b_2 are both satisfied; $\langle b_1 \vee b_2, p\rangle$ and $\langle b_1 \oplus b_2, p\rangle$ both pay price p if either (or both) of b_1 or b_2 are satisfied. The semantics of $\vee$ and $\oplus$ differ in how subformula value is used. Specifically, the value of a disjunctive bid given an assignment is the sum of the values of the subformulae: in this sense, both subformulae are of value to the bidder. In contrast, a "valuative XOR" bid allows only the maximum value of its subformulae to be paid: thus the subformulae are viewed as substitutes.[1] It is important to realize that the valuative XOR connective does not have a logical XOR interpretation; rather it refers to the valuation of the formula, stating that the bidder is willing to pay for the satisfaction of at most one subformula. Notice that an assumption of free disposal is built in to the semantics.

We refer to [2] for further details of the language and examples of its expressive power. We give three examples here to illustrate the intuitions. Consider the bid

$$\langle\langle a, 1\rangle \wedge \langle b, 1\rangle \wedge \langle c, 3\rangle \wedge \langle d, 5\rangle, 50\rangle.$$

This might reflect that a, b, c, and d are complementary goods with joint value 50, and that the individual goods have some intrinsic (e.g., salvage) value over and above that of their role within the group. As a second example, consider

$$\langle\langle a, 1\rangle \vee \langle b, 1\rangle \vee \langle 3, c\rangle \vee \langle d, 5\rangle, 50\rangle.$$

Here the individual goods are substitutes: they provide a basic functionality of value 50, but perhaps do so with differing quality (or each has different intrinsic value) reflected in the "bonus" associated with each good.

As a final example, consider a scenario in which we have a number of goods $\{r_1, \cdots, r_k\}$ whose utilities/prices p_i are conditionally *dependent* on the presence of another good m but are (conditionally) additive *independent* of each other [2]. For instance, think of the r_i as resources or raw materials, and of m as a machine used for processing those resources. This situation can be captured using a single GLB of the form:

$$\langle m \wedge r_1, p_1\rangle \vee \langle m \wedge r_2, p_2\rangle \vee \cdots \vee \langle m \wedge r_k, p_k\rangle$$

To express the same utility function using other languages would require a number of bids exponential in k (essentially requiring the enumeration of all subsets of resources). For example, with one machine m and four resources r_1, r_2, r_3, r_4 (worth 1, 2, 3, and 4, respectively), we'd need the following bid:

$$\langle mr_1, 1\rangle \vee \langle mr_2, 2\rangle \vee \langle mr_3, 3\rangle \vee \langle mr_4, 4\rangle$$
$$\vee \langle mr_1r_2, 3\rangle \vee \langle mr_1r_3, 4\rangle \vee \langle mr_1r_4, 5\rangle \vee \langle mr_2r_3, 5\rangle$$
$$\vee \langle mr_2r_4, 6\rangle \vee \langle mr_3r_4, 7\rangle \vee \langle mr_1r_2r_3, 6\rangle \vee \langle mr_1r_2r_4, 7\rangle$$
$$\vee \langle mr_1r_3r_4, 8\rangle \vee \langle mr_2r_3r_4, 9\rangle \vee \langle mr_1r_2r_3r_4, 10\rangle$$

We note that each of the connectives is commutative and associative, so we can safely treat them as having more than two operands (e.g., it is legitimate to refer to the conjunction of $k > 2$ bids).

The notion of a *k-of* bid, explored in the context of logical bids without priced subformulae [5], can be extended to $\mathcal{L}_{GB}$ quite readily. Let $\mathcal{L}_{GB}^{k\text{-}of}$ denote the extension of $\mathcal{L}_{GB}$ with the k-of operator. Intuitively, $\langle k\text{-}of(b_1, b_2, \ldots, b_d), p\rangle$ is satisfied if any k of the d bids $b_1, \ldots, b_d$ is satisfied (and a price of p is associated with its satisfaction). As in the semantics above, the value of a *k-of* bid is determined by the price p as well as the values of any satisfied subformulae.[2]

Since combinatorial auctions are still relatively rare in practice, it is difficult to say whether $\mathcal{L}_{GB}$ can naturally and concisely express utility functions that are likely to arise in practice. However, the examples above suggest that it does capture a lot of the natural structure in utility functions. In addition, since it can directly "emulate" any existing bidding language, it should be considered state of the art at this point.

3 Winner Determination for LGB

The expressive advantages of logical bidding languages are readily apparent. One might also hope that such languages permit CAs to be solved more effectively as well. If one can express bids concisely, there must be structure in the underlying utility function. If this is so, we should be able to exploit this structure computationally in winner determination. Unfortunately, to date there has been no serious investigation of this possibility.

There are several ways to exploit logical structure computationally. First, one might convert the logical bids to a set of flat bids and hope that existing algorithms discover the "hidden" structure. Evidence that this might work was described in [5], but we will show that for realistic sized problems this approach is doomed. Second, one might devise special purpose procedures for winner determination that exploit the logical structure, such as the stochastic local search procedure suggested in [2].

Finally, one could simply formulate the optimization problem directly in terms of $\mathcal{L}_{GB}$ bids and use generic IP solvers to solve the problem. It is this final approach that we now consider. Expressing logical relationships among goods directly in an IP is reminiscent of the use of optimization techniques to solve problems in logical inference, as proposed by Chandru and Hooker [3].

[1] This semantics of XOR is just one of several natural interpretations. The practical use of XOR may determine other semantics.

[2] This extends the treatment of *k-of* bids in [5], which allowed choosing any k of n *goods* rather than *bids*.

3.1 A Direct IP Formulation for LGB

Our aim is to formulate an IP that directly expresses the winner determination problem for a set of $\mathcal{L}_{GB}$ bids. We first consider the objective function and then the constraints. We assume a set of n goods $\{g_i : i \leq n\}$ and m bids $\{b_i : i \leq m\}$ expressed in $\mathcal{L}_{GB}$. We use the following variables:

- $x_{ij} \in \{0, 1\}$ for each good g_i that occurs in bid b_j: true (1) if g_i is assigned to b_j.
- $s_\beta \in \{0, 1\}$ for each subformula β of any bid: true (1) if β is satisfied by the optimal assignment.
- v_β for each subformula β of any bid: this denotes the value of β under the optimal assignment.[3]
- $t_\beta \in \{0, 1\}$ for each subformula β of any bid that is an *immediate* subformula of an XOR: true (1) if β is the (unique) formula that contributes value to the encompassing XOR.

As a trivial example, consider two bids:

$$b_1 = \langle\langle a, 1\rangle \vee \langle b, 2\rangle, 3\rangle \oplus \langle c, 3\rangle, 7\rangle \quad (3)$$

$$b_2 = \langle\langle a, 1\rangle \wedge \langle b, 2\rangle \wedge \langle d, 1\rangle, 3\rangle \vee \langle c, 4\rangle, 8\rangle \quad (4)$$

There are seven variables x_{ij} corresponding to the assignment of (relevant) goods to each bid. b_1 has five s-variables, one per subformula (a, b, $a \vee b$, c, $(a \vee b) \oplus c$), while b_2 also has five s-variables (note that we view $\wedge$ as a ternary connective in this example). There is also a corresponding v-variable for each subformula of each bid. Finally, b_1 has two t-variables, one for subformula $a \vee b$ and one for c, since these are the immediate subformulae of an XOR. The number of variables in linear in the size of the logical formulation of the bids.

The objective function is straightforward:

$$\textit{Maximize: } \sum \{v_\beta : \beta \textit{ corresponds to a top-level bid}\}$$

In our example, the objective function is $v_{\beta_1} + v_{\beta_2}$, where v_{β_1} is the v-variable for b_1's formula, $(a \vee b) \oplus c$, and similarly for b_2. There is one term in the objective for each bid.

A set of constraints is imposed for each subformula of each bid. The constraints will vary with the main connective. The constraints place upper bounds on the values of all variables, since the objective value can only increase with increasing variable values. For each atomic subformula β of the form $\langle g_i, p\rangle$ in bid b_j, we impose two constraints:

$$s_\beta \leq x_{ij}; \qquad v_\beta \leq p \cdot s_\beta$$

Thus the formula is satisfied only if g_i is assigned to b_j (and value is determined correspondingly).

For each subformula $\beta = \langle \beta_1 \vee \cdots \vee \beta_d, p\rangle$, we impose two constraints:

$$s_\beta \leq \sum_{i \leq d} s_{\beta_i}; \qquad v_\beta \leq p \cdot s_\beta + \sum_{i \leq d} v_{\beta_i}$$

This ensures β is considered satisfied if any subformula is, and assigns value as dictated by our semantics.

For each subformula $\beta = \langle \beta_1 \wedge \cdots \wedge \beta_d, p\rangle$, we impose two constraints:

$$d \cdot s_\beta \leq \sum_{i \leq d} s_{\beta_i}; \qquad v_\beta \leq p \cdot s_\beta + \sum_{i \leq d} v_{\beta_i}$$

This ensures β is considered satisfied if all subformula are.

Finally, for each subformula $\beta = \langle \beta_1 \oplus \cdots \oplus \beta_d, p\rangle$, we impose four constraints:

$$s_\beta \leq \sum_{i \leq d} s_{\beta_i}; \qquad v_\beta \leq p \cdot s_\beta + \sum_{i \leq d} v_{\beta_i}$$

$$\sum_{i \leq d} t_{\beta_i} \leq 1; \qquad v_{\beta_i} \leq \textit{maxval} \cdot t_{\beta_i}, \ \forall i \leq d$$

The penultimate constraint ensures that only one subformula of the XOR is *selected* for contribution of value to the XOR as a whole, while the final constraint ensures that only the selected subformula has positive value. *maxval* is a large constant assured to be larger than the value of any formula.[4]

The number of constraints is linear in the number of subformulae (hence in the size of the bid specification), and the size of each constraint is bounded by the "actual" arity of the connective involved. Thus, the IP formulation is very compact.

The IP formulation also extends naturally to $\mathcal{L}_{GB}^{k\text{-}of}$. Let β be a subformula of the form $\langle k\text{-}of(\beta_1, \beta_2, \ldots, \beta_d), p\rangle$. We introduce a new variable n_β for each *k-of* bid denoting the number of satisfied subformulae. We then impose the following three linear constraints:

$$n_\beta \leq \sum_{i \leq d} s_{\beta_i}; \qquad s_\beta \cdot k \leq n_\beta$$

$$v_\beta \leq p \cdot s_\beta + \sum_{i \leq d} v_{\beta_i}$$

The first constraint ensures that number n_β of subbids counted as satisfied is legitimate, while the second ensures the *k-of* bid is satisfied only if at least k of the subbids are satisfied.

3.2 Converting LGB to Flat Bids

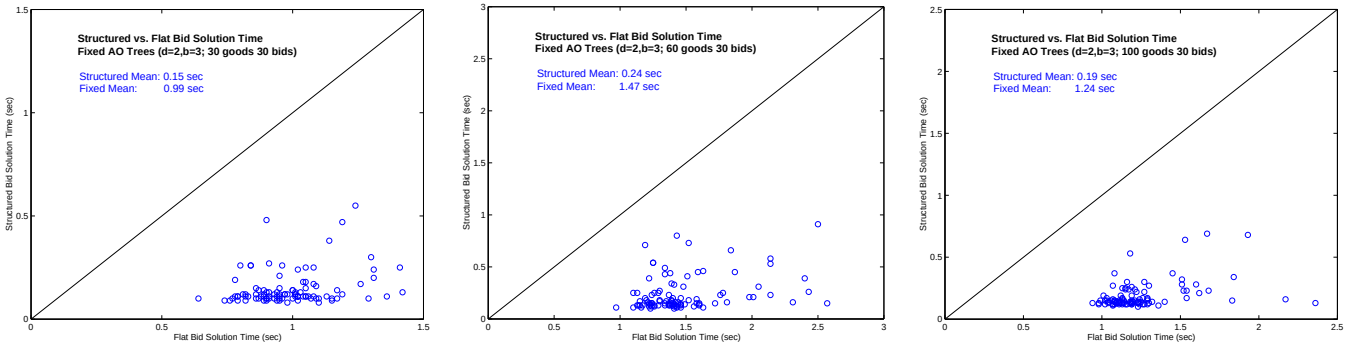
The utility function represented by a GLB b can be captured using an equivalent set of flat bids. Let $G(b)$ denote the set of goods occurring in b. The required set of flat bids $f(b)$ can be generated using a very simple strategy: since each good mentioned in b may contribute to value, every subset $s \subseteq G(b)$ can be viewed as a potential flat bid having some utility to the customer, and this utility can be determined by calculating the value of the assignment s to b. Of course, only one such subset is of interest, so we insert a single dummy good into each flat bid (subset) to ensure that only one such bid can be satisfied. More precisely:

$$f(b) = \{\langle s \cup \{d_b\}, \Psi(b, s)\rangle : s \subseteq G(b)\}$$

where d_b is a dummy good associated with GLB b. The winner determination problem for $\mathcal{L}_{GB}$ can be solved by converting each GLB b into a set of flat bids, and solving the corresponding "flat" problem using these.

[3] For simplicity, we treat this as an integer, which is valid if all prices are integral. Allowing a mixed formulation is not problematic.

[4] This constraint can be formulated without such a constant through the introduction of additional variables.



(a) $n = 30$ (b) $n = 60$ (c) $n = 100$

Figure 1: Flat vs. structured solution times with varying number of goods

A number of flat bids generated in this way may be "redundant," in the sense that some smaller subset could generate equivalent value. In our experiments below, we in fact use a more sophisticated, bottom-up strategy for generating equivalent flat bid sets from a GLB to ensure that the set of flat bids is in fact a minimal representation of the utility function.

4 Empirical Results

In this section we report on experiments run to compare the relative effectiveness of solving the direct IP formulation of an $\mathcal{L}_{GB}$ problem with the IP formulation for the corresponding set of flat bids. In all experiments, the CPLEX optimization package (Version 7.1.0) was used to solve the IP. CPLEX has a number of strategies for solving IPs, and algorithm choice was left to the software. Running times reported include presolve times, but do not include read times (which would put the large flat bid formulations at a disadvantage). All experiments were run under Linux with a 933MHz, PIII, 512Mb PC.

A number of researchers have proposed candidate problem distributions for CAs in order to facilitate the comparison of different evaluation techniques. Many of these problems are very abstract and it is unclear how these might arise in practice. In an effort to alleviate this problem, a suite of test problems—or more precisely a suite of schemes for generating random test problems—has been proposed that draws on somewhat more realistic intuitions [7]. This collection of problems, CATS, arguably reflects structure that is more likely to arise in practical problems. Unfortunately, the problems in this suite are largely designed to generate structured "subsets" of goods, and hence reflect little of the natural structure suited to a logical language such as $\mathcal{L}_{GB}$. For this reason, we consider the generation of logical bids directly. We first consider some abstract problems, and then consider a class of problems that exhibit the same type of "natural" structure that motivated the development of CATS. The development of a suite of realistic "logical" test problems is an important future goal.

Our first set of experiments focus on randomly generated GLBs with conjunction and disjunction.[5] Bids are generated using randomly constructed parse trees of a given depth and branching factor. One parameterized distribution we consider is RandAO-d-b-m-n-p: these problems have m bids over n goods, with each bid having a parse tree of depth d and branching factor b. At each interior node a connective $\wedge$ or $\vee$ is inserted (with equal probability), while at each leaf a random good is inserted (drawn uniformly). At each node (interior or leaf), a price is included, drawn uniformly from the range $[0, p]$. For example, the bid

$$\langle\langle\langle a, 2\rangle \wedge \langle b, 3\rangle, 0\rangle \wedge \langle\langle a, 2\rangle \wedge \langle c, 0\rangle, 1\rangle, 20\rangle$$

is a bid with depth $d = 2$ and branching factor $b = 2$. We also consider variants AltAO-d-b-m-n-p and AltOA-d-b-m-n-p, where the connectives $\wedge$ and $\vee$ strictly alternate at each level of the tree (starting with $\wedge$ at the root of AO-trees, and $\vee$ at the root of OA-trees).

We start with the RandAO distributions with $m = n = 30$.[6] On very small GLBs, with $b = 2$ and $d = 2$ (thus inducing a tree with four leaves, and at most 15 flat bids), the IP solution of the flat bids dominates that of the structured bids, with mean times of 0.02s and 0.06s, respectively. However, if we increase the branching factor to 3 (thus each GLB corresponds to as many as 511 flat bids), structured solutions dominate flat solutions, with mean times of 0.15s and 0.99s, respectively. The scatterplot of solution times shown in Figure 1(a) shows that the the structured solution time is less than the flat time on each problem instance. Figures 1(b) and (c) show the relative solution times with larger numbers of goods: with $n = 60$, the average solution times are 0.24s and 1.47s, respectively, while with $n = 100$, average times are 0.19s and 1.24s, respectively.

The advantages of solving structured CAs directly is even more apparent with only slightly larger problems.

[5] We report on XOR and *k-of* bids in the longer version of the paper. Results are similar.

[6] In all experiments, $p = 50$. All results are averages over 100 random instances except where noted.

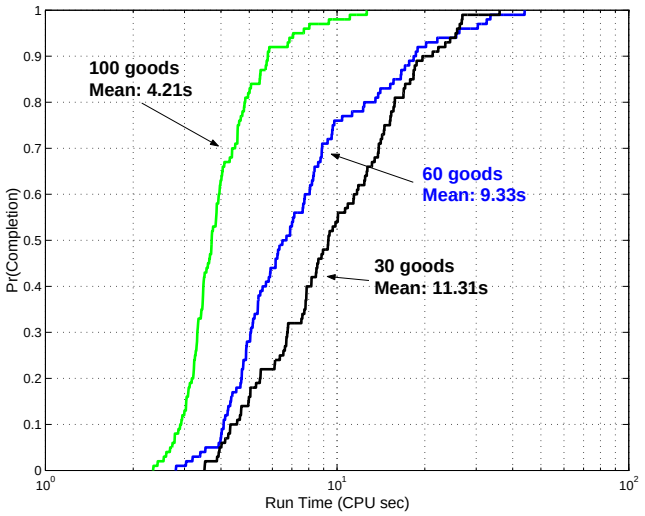


Figure 2: Cumulative runtime distributions for different numbers of goods. RandAO distributions, with $d = 3, b = 4$ and 30 bids. Number of goods: 30, 60, 100. Each distribution generated from 100 problem instances.

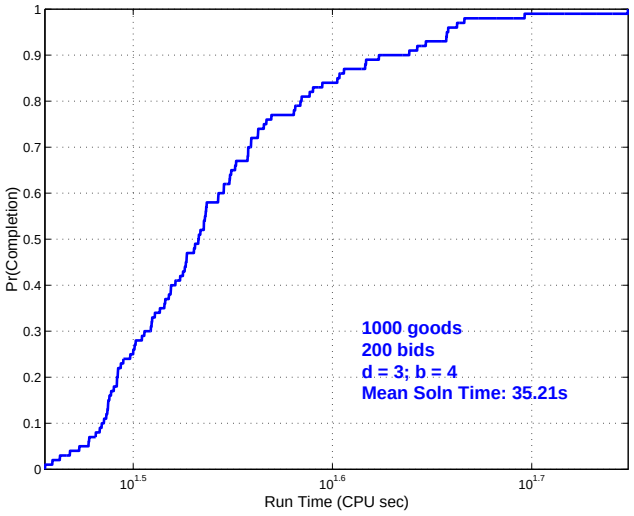


Figure 3: Cumulative runtime distribution for large problems ($d = 3, b = 4, n = 1000, m = 200$). Generated from 100 problem instances.

The following table shows the solution times (in seconds) for five random instances with $d = 2, b = 4$ (each GLB corresponds to 65535 flat bids), 20 goods and 40 bids:

Instance	I1	I2	I3	I4	I5	Mean
Structured	0.14	0.10	0.12	0.22	0.14	**0.14**
Flat	364.9	172.2	218.3	184.8	169.4	**221.9**

Even though these structured bids are of fairly small size (with only 16 leaves in the parse tree), solving the flat version of the problem takes at least three orders of magnitude longer.

The next results illustrate run times on larger problems, for which solving flat versions of the problem proved infeasible. Figure 2 shows the change in runtime distributions as the good:bid ratio varies. In these problems $d = 3, b = 4$ (each GLB thus corresponds to as many as 2^{64} flat bids). In each instance, 30 bids are present. Each line shows the cumulative runtime distribution for a different number of goods (hence the x-axis shows the run time, while the y-axis shows the probability that an instance will be solved by that time). Note that as the number of goods increases, random problems become less constrained and hence somewhat easier to solve. Figure 3 shows the runtime distribution for similar problems but with a much larger number of bids (200) and goods (1000). The mean solution time of 35.21s is very encouraging for such large problems, where the corresponding flat bids sets could scarcely be enumerated.

Finally, Figure 4 shows the runtime distribution for 10 problem instances for GLBs with $d = 4$ and $b = 4$: the set of flat bids for each such GLB could be as large as 2^{256} (if we have at least 256 goods from which to draw). Each problem has 30 GLBs over 100 goods. The mean solution time is 472.9 seconds. It scarcely needs to be mentioned that using flat bids can't even be contemplated for problems of this magnitude.

Further empirical study is needed of different structured bid distributions. While for problems involving GLBs of more than depth 2 and branching factor 3, flat solution methods will unlikely be feasible for any distribution, for "small" GLBs, the specific problem distributions may prove more or less advantageous for flat formulations. Studies of AltAO and AltOA distributions, for instance, with $d = 2$ and $b = 3$ (these bids are the same "size" as those evaluated in Figure 1), reveal that the flat IP is competitive with the structured IP for AltOA: over 100 instances, the flat mean solution time is 1.08s, while the structured mean is 1.07s; furthermore the flat solution time has much lower variance. In contrast, the advantage of the structured over the flat IP is even greater in AltAO problems than for RandAO: the structured technique takes on average 0.15s, while flat takes 1.07s.

We have not reported on XOR or *k-of* bids. The structured IP retains its extreme advantage over the flat IP, naturally; but it is worth pointing out that different semantics for XOR have fairly dramatic impact on the structured solution times, while seeming to have less impact on the flat technique. We have done only preliminary experimentation with *k-of* bids, but these suggest that the structured IP can handle problems of the same order of magnitude reported above.

Other variants of these problem distributions need to be considered as well. The abstract distributions above assign prices randomly to subformulae, without regard to the number of items required to satisfy them. We plan to study more biased (and realistic) price distributions in the future.

The second set of problems we consider are motivated by more realistic considerations. The parameterized distribution Mach-n-m-r-b-p captures the resource allocation problems discussed in Section 2.2. This distribution assumes a set of m machines and r resources available for auction. Each of the b bidders wants one (specific) machine from this collection and n of the r resources. The form of the bid is exactly as

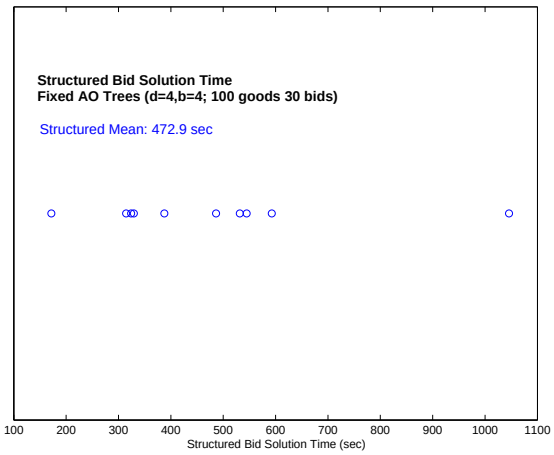


Figure 4: Runtimes for very large problems ($d = 4, b = 4, n = 100, m = 30$), over 10 instances.

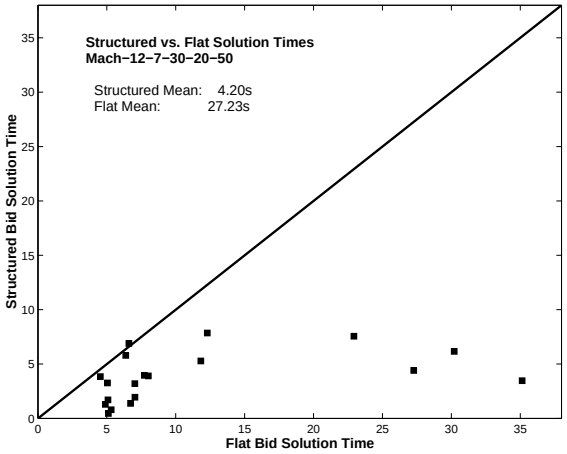


Figure 5: Flat vs. structured solution times for small Mach problems ($n = 12, m = 7, r = 30, b = 20$).

specified in Section 2.2: a bidder is willing to pay some price p_i for the conjunction $m \wedge r_i$ for each of its requested resources r_i. The price p_i is drawn uniformly from the range $[0, p]$. The machine and resources needed by each bidder are also drawn uniformly from the set of machines and resources.

We first compare the structured and flat solution methods on the Mach distribution with $m = 7$, $r = 30$, each bidder requesting $n = 12$ distinct resources, and $b = 20$ bidders.[7] The scatterplot of solution times shown in Figure 5 shows that the structured solution time is considerably less than the flat time on each of 20 problem instances, even for such small problems.[8] The mean solution times are 4.2s and 27.2s for the structured and flat methods, respectively.

For even slightly larger problems, solving the set of flat instances becomes infeasible. A systematic test of the Mach-15-8-50-20-50 distribution on flat bids is impractical. For one typical instance, the solution time was 1933.14s (approximately half an hour). By contrast, the runtime distribution based on 100 random problem instances for the harder problem distribution Mach-20-10-50-20-50 is shown in Figure 6. The mean solution time over these instances is 7.55s. Note that in each instance, 20 bidders are competing for 10 machines. Furthermore, since each bidder requests 20 resources from the set of 50, each of the 20 GLBs in these instances would correspond to 2^{20} flat bids.

[7] In all Mach-distribution experiments, $p = 50$.

[8] One outlying point is removed: for this problem, the structured solution time was 10.9s, while the flat solution time was 325.4s.

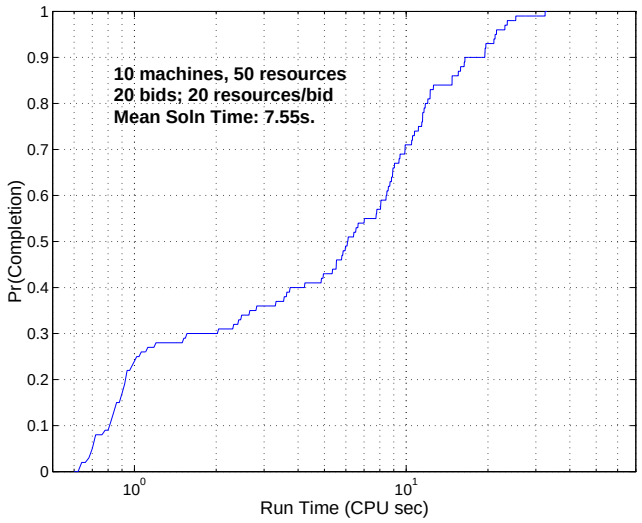


Figure 6: Cumulative runtime distribution for medium sized Mach problems ($n = 20, m = 10, r = 50, b = 20$). Generated from 100 problem instances.

Finally, the following table shows the structured solution times (in seconds) for five random instances with drawn from Mach-30-10-200-100-50:

Instance	I1	I2	I3	I4	I5	Mean
Time	114.0	25.3	85.9	148.6	157.2	**106.2**

Again we see that the structured formulation offers considerable advantages, allowing very large resource allocation problems to be solved effectively.

5 Concluding Remarks

We have described a technique for producing a compact IP reflecting the winner determination problem for CAs involving the generalized logical bidding language $\mathcal{L}_{GB}$ and its extension $\mathcal{L}_{GB}^{k\text{-}of}$. Apart from the expressive advantages of $\mathcal{L}_{GB}$, our empirical results demonstrate the unequivocal superiority of computational methods that directly exploit the logical structure of these bids in winner determination. We have provided evidence for several representative problem distributions, though the combinatorics alone imply that these advantages will obtain for any distribution over GLBs of moderate size.

A number of extensions of this work are being pursued. One is the extension of $\mathcal{L}_{GB}$ and the IP formulation to multi-

unit CAs. This extension is straightforward; we expect the same computational advantages to persist. We are also currently exploring the use of stochastic local search techniques for solving CAs expressed using $\mathcal{L}_{GB}$. Specifically, the procedure proposed in [2] seems to provide a useful anytime method for solving bids expressed in $\mathcal{L}_{GB}$ in a way that exploits their logical structure.

The development of realistic problem distributions for logically structured utility functions remains an important task. The Mach distributions proposed here seem to reflect natural intuitions about certain types of resource allocation problems, but additional problem classes are needed to fully verify the usefulness of our technique. A test suite for logically specified CAs, similar to CATS [7], would be a great step in this direction.

Finally, the problem of sharing partial solutions across related CAs might be one that can readily exploit logical structure. Related CAs arise, for instance, in the implementation of generalized Vickrey-Clarke-Groves mechanisms [9; 6], where multiple CAs are solved with different bidders removed. Logical structure in utility functions could be used to facilitate the "transfer" of partial solutions.

Acknowledgements

Thanks to Holger Hoos and Tuomas Sandholm for their helpful discussions and the anonymous referees for their suggestions. This research was supported by CombineNet, Inc.

References

[1] Arne Andersson, Mattias Tenhunen, and Fredrik Ygge. Integer programming for combinatorial auction winner determination. In *Proceedings of the Fourth International Conference on Multiagent Systems*, pages 39–46, Boston, 2000.

[2] Craig Boutilier and Holger H. Hoos. Bidding languages for combinatorial auctions. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, pages 1211–1217, Seattle, 2001.

[3] Vijay Chandru and John N. Hooker. *Optimization Methods for Logical Inference*. Wiley, New York, 1999.

[4] Yuzo Fujisima, Kevin Leyton-Brown, and Yoav Shoham. Taming the computational complexity of combinatorial auctions. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pages 548–553, Stockholm, 1999.

[5] Holger H. Hoos and Craig Boutilier. Solving combinatorial auctions using stochastic local search. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence*, pages 22–29, Austin, TX, 2000.

[6] Noa E. Kfir-Dahav, Dov Monderer, and Moshe Tennenholtz. Mechanism design for resource bounded agents. In *Proceedings of the Fourth International Conference on Multiagent Systems*, Boston, 2000.

[7] Kevin Leyton-Brown, Mark Pearson, and Yoav Shoham. Towards a universal test suite for combinatorial auction algorithms. In *ACM Conference on Electronic Commerce (EC-2000)*, pages 66–76, Minneapolis, MI, 2000.

[8] Noam Nisan. Bidding and allocations in combinatorial auctions. In *ACM Conference on Electronic Commerce (EC-2000)*, pages 1–12, Minneapolis, MI, 2000.

[9] Noam Nisan and Amir Ronen. Computationally feasible VCG mechanisms. In *ACM Conference on Electronic Commerce (EC-2000)*, pages 242–252, Minneapolis, MI, 2000.

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

[11] Michael H. Rothkopf, Aleksander Pekeč, and Ronald M. Harstad. Computationally manageable combinatorial auctions. *Management Science*, 44(8):1131–1147, 1998.

[12] Tuomas Sandholm. An algorithm for optimal winner determination in combinatorial auctions. In *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence*, pages 542–547, Stockholm, 1999. Extended version, Washington Univ. Report WUCS-99-01.

[13] Tuomas Sandholm. eMediator: a next generation electronic commerce server. In *Proceedings of the Fourth International Conference on Autonomous Agents*, pages 341–348, Barcelona, 2000.

[14] Tuomas Sandholm, Subash Suri, Andrew Gilpin, and David Levine. CABOB: A fast optimal algorithm for combinatorial auctions. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, pages 1102–1108, Seattle, 2001.

[15] Dale Schuurmans, Finnegan Southey, and Robert C. Holte. The exponentiated subgradient algorithm for heuristic boolean programming. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, pages 334–341, Seattle, 2001.

[16] Michael P. Wellman, William E. Walsh, Peter R. Wurman, and Jeffrey K. MacKie-Mason. Auction protocols for decentralized scheduling. *Games and Economic Behavior*, 35:271–303, 2001.