Bidding Languages for Combinatorial Auctions

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

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

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 bids* for concrete bundles of goods. However, logical bidding languages allow the expression of complex utility functions in a natural and concise way. We introduce a new, generalized language where bids are given by propositional formulae whose subformulae can be annotated with prices. This language allows bidder utilities to be formulated more naturally and concisely than existing languages. Furthermore, we outline a general algorithmic technique for winner determination for auctions that use this bidding language.

1 Introduction

Combinatorial auctions (CAs) have been proposed as a means of dealing with the allocation of goods to buyers whose preferences exhibit complex structure with respect to complementarity and substitutability [Rassenti *et al.*, 1982; Rothkopf *et al.*, 1998; Wellman *et al.*, 2001]. 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, equivalent to weighted set packing. As a result, *optimal winner determination* for CAs is NP-complete [Rothkopf *et al.*, 1998].

By expressing her preferences (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 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. In contrast, the *logical structure* of a complex utility function might allow such preferences to be expressed relatively concisely in a suitable language. Several researchers have proposed mechanisms for expressing bids logically [Sandholm, 1999; 2000; Hoos and Boutilier, 2000; Nisan, 2000].

In this paper, we describe a generalized language for expressing bids that captures the most important elements of existing bidding languages. In our *logical combination of bids and goods* model, one specifies a bid using a logical formula, but is allowed to associate prices with arbitrary subformulae. For example, suppose in an auction for shipping capacity a bidder can send her shipment using two standard containers, a and b, or one oversized container c. The shipment has an inherent value of 50, but the convenience of using an oversize container is worth 5. We can express a suitable bid for services as $\langle\langle a \wedge b, 0\rangle \vee \langle c, 5\rangle, 50\rangle$ in our language, capturing the overall value of 50 for satisfying the requirement $a \wedge b \vee c$, as well as the premium of 5 for the oversized container. We will see examples like this below where our language allows the logical structure of a utility function to be expressed *directly* within a bid. Furthermore, our language allows one to make the very important distinctions between *sharable* and *consumable* goods, unlike existing bidding languages. We show that our language affords complete expressiveness, and that for certain natural classes of utility functions, it can express bids exponentially more compactly than existing languages. In addition, we argue that it provides a natural and concise mechanism for expressing complex bids.

We also propose an algorithmic framework for solving the winner determination problem for a set of bids expressed in our generalized logical language. We formulate a stochastic search procedure that works directly with our logical bids, sidestepping the problem of converting a logical bid into a (potentially, exponentially) large number of explicit bids. Though we have yet to study its computational properties, we expect this approach to offer a significant advance over existing algorithms.

We briefly review CAs and logical bidding languages in Section 2. In Section 3 we present our generalized logical bidding language, describing its syntax and semantics, discuss various properties of this language, and illustrate its ability to handle certain types of utility functions much more naturally and concisely than existing logical languages. In Section 4, we describe a stochastic local search procedure that exploits the structure of our logical bids to search through the space of bid allocations to solve the winner determination problem. We conclude in Section 5 with a discussion of future work.

2 Logical Languages for Schematic Bids

In this section, we briefly review CAs and prior proposals for 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. Given a collection of bids $B = \{\langle b_i, p_i \rangle\}$, 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* $A : G \to B$ of goods to bids. Assignment A induces allocation L_A whose bids are those that have been assigned all goods (i.e., $b_i \subseteq A^{-1}(\langle b_i, p_i \rangle)$).

The winner determination problem is equivalent to the weighted set packing problem [Rothkopf *et al.*, 1998] and as such is NP-complete. Algorithms for weighted set packing and related combinatorial problems can be used for winner determination. Search algorithms—both complete methods [Fujisima *et al.*, 1999; Sandholm, 1999] as well as stochastic techniques [Hoos and Boutilier, 2000]—have been proposed in the AI literature and have proven quite successful at solving medium-sized problems. Though the problem is known not to be approximable in polynomial time, the afore-mentioned stochastic approximation technique tends to find optimal solutions very quickly for problems that can be handled by the complete methods.

2.2 Logical 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. Such a bundle naturally captures the complementarities among the goods within that bundle. However, a buyer with a complex utility function will often need to express multiple bundle bids in order to accurately reflect her utility function.

Logical bidding languages can overcome this by allowing a bidder to express complex bids in which the logical structure of the utility function is captured. There are two distinct classes of logical bidding languages in the literature: languages that allow *logical combinations of goods* as formulae, and associate a price with each such formula (we call this the $\mathcal{L}_G$ family of languages); and languages that allow *logical combinations of bundle bids* as formulae, where the subformulae (or atomic bids) themselves have prices associated with them (we call this the $\mathcal{L}_B$ family of languages). We discuss these briefly in turn in this section, but refer to the cited papers for further details. In what follows we assume a set of goods G over which bids are expressed.

An $\mathcal{L}_G$ language is one in which logical formulae are constructed from goods; that is, goods are taken as atomic propositions and are combined using logical connectives to express a bid. A price is attached to this formula expressing the amount the bidder is prepared to offer for the *satisfaction* of that formula. Such languages can be used to capture some of the logical structure of a utility function. The language $\mathcal{L}_G^{pos}$ proposed by Hoos and Boutilier [2000] is of this type, with the restriction that only positive formulae (i.e., without negation) are considered. Formally, if $g \in G$ then $g \in \mathcal{L}_G^{pos}$; and if $\alpha_1, \alpha_2 \in \mathcal{L}_G^{pos}$, then $\alpha_1 \vee \alpha_2 \in \mathcal{L}_G^{pos}$ and $\alpha_1 \wedge \alpha_2 \in \mathcal{L}_G^{pos}$. A *logical bid* $\langle \alpha, p \rangle$ is simply a formula $\alpha \in \mathcal{L}_G^{pos}$ and an associated price p. Semantically, an assignment of goods to this bid *satisfies* the bid if the corresponding logical formula is satisfied viewing the assignment as a truth assignment (i.e., those goods assigned to the bid are "true" and those not are "false"). As an example, should a bidder desire (for price p) either g_1 or h_1, *and* g_2 or h_2, *and* g_3 or h_3, *and* g_4 or h_4, she must formulate sixteen explicit bids of the form $\{g_1, g_2, g_3, g_4\}$, $\{g_1, g_2, g_3, h_4\}$, etc. In contrast, $\mathcal{L}_G^{pos}$ allows such preferences to be expressed relatively concisely using a logical bid of the form:

$$\langle (g_1 \vee h_1) \wedge (g_2 \vee h_2) \wedge (g_3 \vee h_3) \wedge (g_4 \vee h_4), p \rangle$$

Variants of this language have been proposed by Hoos and Boutilier [2000], including the use of *k-of* clauses, expressing a desire to have any k goods from a given set, and focusing on special forms such as CNF. It is important to note that a bidder generally must express a number of logical bids in $\mathcal{L}_G^{pos}$ to capture her utility function: the fact that only one price can be attached to a formula means that independent preferences are captured by independent bids (e.g., the same bidder might bid both $\langle a, 1 \rangle$ and $\langle b, 2 \rangle$). While perfect substitution is captured by disjunction in $\mathcal{L}_G^{pos}$, imperfect substitutes must be dealt with using multiple bids and *dummy goods* [Fujisima *et al.*, 1999]. For example, if an agent wants only one of $a \wedge b$ or $c \wedge d$, and slightly prefers $a \wedge b$, she could specify two bids,

$$\langle a \wedge b \wedge g, p \rangle \quad \text{and} \quad \langle c \wedge d \wedge g, p' \rangle$$

with $p > p'$. The insertion of dummy good g prevents both bids from being satisfied. There is an implicit assumption of free disposal in the semantics of $\mathcal{L}_G^{pos}$. If a bid $\langle a \vee b, p \rangle$ is offered, the same price is paid if the bid is assigned a alone, b alone, or both a and b. If this assumption is violated, this bid must be broken into multiple (exclusive) bids.

A different approach is taken by Sandholm [1999; 2000] and Nisan [2000], who use $\mathcal{L}_B$ languages. Intuitively, these languages take bundle bids as their atomic elements and combine these using various logical connectives. For instance, Sandholm proposed the use of $\mathcal{L}_B^{or}$, combining atomic bids using disjunction, as in $\langle \{a, b\}, 1 \rangle \vee \langle \{a, c\}, 2 \rangle$. Semantically, $\mathcal{L}_B$ languages are interpreted by assigning goods to the *component atomic bids* (e.g., to $\{a, b\}$ and $\{a, c\}$ in the example above), rather than to the formula as a whole (in contrast with the $\mathcal{L}_G$ model). The price paid is determined by the logical relationship of the component bids. In $\mathcal{L}_B^{or}$, for instance, the sum of the prices of satisfied atomic bids is paid.

Several interesting varieties of $\mathcal{L}_B$ languages are studied by both Sandholm and Nisan, who consider languages using OR, XOR ($\mathcal{L}_B^{xor}$), and two-level nesting of such connectives (OR-of-XOR and XOR-of-OR). The use of bundle bids as atomic elements allows one to express complementarities; OR (as in $\mathcal{L}_B^{or}$) allows one to capture independent preferences; and XOR allows the expression of substitutability. Nisan also proposes (and favors) the language $\mathcal{L}_B^{or*}$, essentially $\mathcal{L}_B^{or}$ with dummy goods allowed within atomic bids. $\mathcal{L}_B^{or*}$ is fully expressive and is generally more compact than the other $\mathcal{L}_B$ languages for many types of utility functions. We refer to Nisan [2000] for a discussion of the relative merits of these languages. Because multiple prices occur within a single formula, a bidder can express her preferences completely using a single bid (in contrast with the $\mathcal{L}_G$ model). However, preferences involving disjunction are often expressible much more compactly within $\mathcal{L}_G$ than $\mathcal{L}_B$. For instance, the preference function above involving multiple clauses of the form $g_i \vee h_i$ requires a bid of exponential size in any $\mathcal{L}_B$ language.

3 A Generalized Language for Logical Bids

Both language families $\mathcal{L}_G$ and $\mathcal{L}_B$ have certain drawbacks. In $\mathcal{L}_G$ languages a bidder who wants to offer different prices for related logical combinations of goods is forced to specify distinct bids for those combinations. This prevents the bids from exploiting any logically common substructure. $\mathcal{L}_B$ languages are unable to exploit the logical structure of disjunctive combinations of goods that exhibit perfect substitutability. Furthermore, $\mathcal{L}_B$ languages are unable to exploit the fact that a good may be "sharable," that is, it may contribute to the satisfaction of *multiple* atomic bids within a single bidder's logical bid. This is due to the fact that each good is assigned to an atomic bid within the $\mathcal{L}_B$ semantics and cannot be shared. This can be a severe drawback. Consider an example in which a bidder desires a single reusable resource, say a machine m, and some number of consumable resources, say raw materials r_1, r_2, etc. to be processed on m. The bidder may value each of the r_i independently, but only if the machine is available on which to process these materials. In such a case, it is most natural to express preferences for the $\langle m, r_i \rangle$ pairs, allowing m to contribute to the satisfaction of *each* such bid or subformula.

In this section, we introduce the language $\mathcal{L}_{GB}$ of *generalized logical bids* that allows for the logical combination of both goods and bids within a single formula. Specifically, a positive propositional formula over goods may have prices associated with arbitrary subformulae. As such, both goods and bids can be combined in arbitrary ways. We will see that the expressive power afforded by this approach offers a number of advantages, both in terms of the naturalness and conciseness of the representation of certain classes of utility functions. It inherits the fundamental advantages of both $\mathcal{L}_G$ and $\mathcal{L}_B$ languages.

Our new language will allow us to formulate logical bids that reflect such structured utility functions directly and concisely. We first describe the syntax and semantics of our language in fairly abstract terms. We then discuss various formal and informal properties of our language, comparing it to the languages mentioned above in terms of expressiveness, naturalness, and conciseness.

3.1 Syntax

Let G denote the set of goods, forming the atomic elements of our language. The language 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 $\oplus$ (*valuative XOR*, the naming of which will become clear below), where each subformula is annotated with a price. We often don't mention the price for a subformula if $p = 0$, and call such a subformula *priceless*. A sentence $b \in \mathcal{L}_{GB}$ is called a *generalized logical bid (GLB)*. Examples of GLBs include

$$\langle \langle a, 1 \rangle \wedge \langle b, 2 \rangle, 5 \rangle \quad \text{and} \quad \langle a \vee b, 2 \rangle \oplus \langle c, 3 \rangle.$$

The *formula associated with* b, denoted $\Phi(b)$, is the logical formula obtained by removing all prices from subformulae.

3.2 Semantics

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 $A : G \to B$ 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 satisfaction relation σ is defined as follows:

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

Notice that the satisfaction relation is identical for the connectives $\vee$ and $\oplus$. The difference between the connectives will become evident when we define the value of a bid.

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, or VXOR bid, allows only the maximum value of its subformulae to be paid: thus the subformulae are viewed as substitutes (see below).

Note how, under this definition, zero prices can be used to represent subformulae to which no price is attached (such as for the conjunctive bids defined in the previous section). Let '$\equiv$' denote semantic equivalence (i.e., two GLBs have exactly the same value under any assignment of goods). We summarize the intuitive semantics of $\mathcal{L}_{GB}$ (examples are provided in the next section):

- Bids $b_1 \wedge b_2$ and $b_1 \vee b_2$ are both valued as the sum of component values for b_1 and b_2 (i.e., their utilities are independent). Without prices, conjunction and disjunction, when they appear at the top-level of a GLB, are semantically equivalent. When such formulae are priced (e.g., $\langle b_1 \wedge b_2, p\rangle$), or when they appear as subformulae in a more complex GLB, however, the meaning is quite different.
- Bid $\langle b_1 \wedge b_2, p\rangle$ expresses the complementarity of b_1 and b_2 (with value p). However, it allows intrinsic value to be expressed within the subbids (see below).
- Bid $\langle b_1 \vee b_2, p\rangle$ expresses the *partial* substitutability of b_1, b_2 (with value p). However, it allows intrinsic value to be expressed within the subbids, so satisfying both may have greater value than satisfying either one alone. If b_1, b_2 are completely priceless, then disjunction represents full and perfect substitutability.
- Bid $(b_1 \oplus b_2)$ expresses the *complete* substitutability of b_1, b_2. Only one of the values of b_1 or b_2 can be paid. If the values are distinct then they are imperfect substitutes. Perfect substitutes can be captured using $\oplus$ or $\vee$ (see above).
- Bid $\langle b_1 \oplus b_2, p\rangle$ is much like $(b_1 \oplus b_2)$, but with price p paid if either *or both* b_1, b_2 are satisfied. No "penalty" is paid if both are satisfied, so there is an implicit assumption of free disposal. A variation we do not pursue here would pay p iff *one* of the subbids held. In what follows, we assume $\oplus$ formulae have no prices, since $\langle\langle\alpha_1, p_1\rangle \oplus \langle\alpha_2, p_2\rangle, p\rangle \equiv (\langle\alpha_1, p_1 + p\rangle \oplus \langle\alpha_2, p_2 + p\rangle)$.

3.3 Properties and Examples

We begin by illustrating the key features of the generalized language $\mathcal{L}_{GB}$ with several examples. We then describe some of the formal properties of our language.

The ability to associate prices with subformulae gives $\mathcal{L}_{GB}$ the ability to express certain complex preferences much more concisely and naturally than existing languages in either the $\mathcal{L}_G$ of $\mathcal{L}_B$ families. And complex preferences often exhibit considerable structure, as studied in multiattribute utility theory [Keeney and Raiffa, 1976], that can be exploited by $\mathcal{L}_{GB}$. To illustrate, consider the bid

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

Intuitively, 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. The use of subformula prices allows the direct expression of the natural decomposition of the underlying utility function. This bid can be expressed reasonably concisely in $\mathcal{L}_B$ (e.g., OR*) and, hence, $\mathcal{L}_G$: the disjunction of five atomic bids—$\langle a, 1\rangle$, $\langle b, 1\rangle$, $\langle c, 3\rangle$, $\langle d, 4\rangle$, $\langle abcd, 60\rangle$—would suffice. However, this set of bids disguises the true structure of the utility function. Moreover, if the atoms were replaced by disjunctive formulae, the required size of the $\mathcal{L}_B$ or $\mathcal{L}_G$ formulae would blow up, since we would need to distribute the disjunction across each of the conjuncts to form suitable atomic bids.

A related bid is $\langle\langle a, 1\rangle \vee \langle b, 1\rangle \vee \langle 3, c\rangle \vee \langle d, 5\rangle, 50\rangle$. 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. Once again the use of subformula prices allows one to express this preference naturally. The most natural way to express this bid in $\mathcal{L}_B^{or*}$ would be as the disjunction of all 15 combinations of the four goods (in $\mathcal{L}_G$ this would require 15 bids instead of 15 disjuncts). In general, this would require an exponential blowup of the bid. It turns out one can express this bid more concisely as the disjunction of the following 8 bids (where g is a dummy good):[1] $\langle ag, 51\rangle$, $\langle bg, 51\rangle$, $\langle cg, 53\rangle$, $\langle dg, 55\rangle$, $\langle a, 1\rangle$, $\langle b, 1\rangle$, $\langle c, 3\rangle$, $\langle d, 5\rangle$. The dummy good is needed to ensure that 50 is not paid more than once. While the blowup is only linear, any natural structure in the utility function is buried. Furthermore, this conversion only applies when the disjuncts are goods; if they are arbitrary GLBs, then the $\mathcal{L}_B$ (or $\mathcal{L}_G$) expression will blow up.

An important feature of the semantics of $\mathcal{L}_{GB}$ is that goods are assigned to logical bids (as in $\mathcal{L}_G$) as opposed to component subformulae (as in $\mathcal{L}_B$). This means that a good assigned to a logical bid makes all occurrences of that good "true". This allows the natural distinction between "sharable" resources that complement multiple goods, and "consumable" resources whose utility can only be "counted" once. 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. For instance, think of the r_i as raw materials, and of m as a machine used for processing those raw materials. 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 any $\mathcal{L}_B$ language would require a number of bids exponential in k (essentially requiring the enumeration of all subsets of consumable

[1] Here and in some of the following examples, in order to enhance readability, we use the notation $g_1 g_2 \ldots g_k$ instead of $\{g_1, g_2, \ldots, g_k\}$ for bundles of goods.

goods). For example, with one sharable m and four consumables r_1, r_2, r_3, r_4 (worth 1, 2, 3, and 4, respectively), we'd need the following bid (in $\mathcal{L}_B^{or}$ or $\mathcal{L}_B^{or*}$):

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

Again we see that $\mathcal{L}_{GB}$ allows the natural and concise expression of certain types of utility functions.

We observe that goods that complement multiple goods in a *nonsharable* fashion can be captured either using $\oplus$ or by using multiple GLBs. For instance, in the example above, if the machine m is to be treated as consumable, replacing the disjunction with VXOR:

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

would ensure that the machine was not shared across the r_i (or at least no *value* was associated with more than one r_i). Similarly, we could break up each disjunct into a separate GLB (all belonging to the same bidder): since goods are assigned to only one GLB, this approach too prevents m from being shared.

Since a bidder can offer multiple GLBs, it is important to note the distinction between the appearance of a good g in *multiple* bids and its appearance it multiple subformulae within a *single* bid. In the former case, g is treated as nonsharable, since it can be assigned to only one bid. In the latter case, each occurrence of g is satisfied by the assignment of g to that bid, hence g can be viewed as being shared by each of the component subformulae containing it. This distinction arises precisely because our notion of satisfaction is defined with respect to the assignment of goods to bids rather than bidders. We note that other *purely logical* means for distinguishing sharable and nonsharable resources may be possible, rather than relying on whether a multiple good occurrences lie "above the bid level" or below it. For instance, resource-oriented logics (e.g., linear logic [Girard, 1987]) are designed primarily to deal with the issue of resource consumption and sharing. The connections to this work seem worthy of deeper exploration.[2]

When a bidder offers multiple GLBs, we must enforce substitutability constraints by using dummy goods. This is not necessary when the bid is contained within a single GLB: VXOR can be used to ensure that only a single good from some set of (perfect or imperfect) substitutable goods is valued (assuming free disposal).

It is not hard to show that the connectives in $\mathcal{L}_{GB}$ are commutative and (in a certain sense) associative.

Proposition 1 *Let $b_1, b_2, b_3 \in \mathcal{L}_{GB}$. Then*

(a) $\langle b_2 \wedge b_1, p\rangle \equiv \langle b_1 \wedge b_2, p\rangle$ (similarly for $\vee$, $\oplus$).

(b) $\langle (b_1 \wedge b_2) \wedge b_3, p\rangle \equiv \langle b_1 \wedge (b_2 \wedge b_3), p\rangle$; note that the inner conjunctions have no price associated with them (similarly for $\vee$, $\oplus$).

[2]We note that existing resource-oriented logics do not seem to able to handle complementarities.

This justifies the informal use of conjunction (etc.) of a set of GLBs, with a price paid for the conjunction. Certain distribution laws can be derived as well for price-free subformulae, and for priced subformulae if we allow manipulation (e.g., addition and subtraction) of prices. We conjecture that several useful normal forms for $\mathcal{L}_{GB}$ exist.

While there are preferences for which $\mathcal{L}_{GB}$ offers much more concise expression than either $\mathcal{L}_G$ or $\mathcal{L}_B$, the converse is not true. Any bid expressed in $\mathcal{L}_B^{or}$, $\mathcal{L}_B^{or*}$, $\mathcal{L}_B^{xor}$, or $\mathcal{L}_G$ can be expressed equally concisely in $\mathcal{L}_{GB}$. $\mathcal{L}_G$ bids are simply special cases of GLBs. Similarly, $\mathcal{L}_{GB}$ can represent each disjunct in an $\mathcal{L}_B^{or}$ or $\mathcal{L}_B^{or*}$ bid as a separate GLB for the specified bidder, resulting in a collection of smaller bids whose total size, structure, and meaning is the same. As a corollary to the results of Sandholm and Nisan, that show that $\mathcal{L}_B^{xor}$ and $\mathcal{L}_B^{or*}$ are both fully expressive, we have:

Proposition 2 *$\mathcal{L}_{GB}$ can represent any utility function over a set of goods G.*

4 Stochastic Search for GLBs

As we have seen logical languages for bid *expression* have been considered by several authors. However, the logical structure of bids formulated in these languages has not been directly exploited computationally in winner determination. For instance, in the the computational study of $\mathcal{L}_G$ languages for winner determination undertaken in [Hoos and Boutilier, 2000], compact logical bids were converted into a (large) set of explicit bids and the behavior of winner determination was examined. Despite the conversion, the stochastic local search algorithm, Casanova, proposed in that study proved to work extremely well. In this section, we formulate a stochastic search procedure for the winner determination problem for GLBs that works directly with logical bids, sidestepping the problem of converting a logical bid into a (potentially exponentially) large number of explicit bids. Though we have yet to study its computational properties, we expect this approach to offer a significant advance over existing algorithms.

Given a set B of GLBs, our aim is to find an assignment $A : G \rightarrow B$ of goods to bids that maximizes revenue; that is, whose value $V(A) = \sum_{b \in B} \Psi(b, A^{-1}(b))$ is maximal. In the spirit of the Casanova algorithm for standard combinatorial auctions, and motivated by the success of stochastic local search (SLS) techniques for a broad range of hard combinatorial problems, we devise an SLS procedure that operates directly on the space $\mathcal{A}$ of assignments and uses the objective function $V(A)$ to guide the search. In this search space, intuitively, we want to consider two assignments $A, A' \in \mathcal{A}$ to be neighbors if we can construct A from A' by shifting certain goods from some bids to others. For example, one could imagine defining the neighborhood relation as follows: a neighbor of A is reached by moving exactly one good from its assigned bid b in A to a new bid b'. While this would render each $A \in \mathcal{A}$ reachable from any other assignment, selecting good moves based on the objective function $V(A)$ would be difficult, as many single-good moves are unlikely to cause a change in $V(A)$, leading to large plateaus in the searchscape. Alternatively, one could follow the CASLS approach [Hoos and Boutilier, 2000], and consider assignments A and A' to be neighbors if A' can be reached by selecting an unsatisfied bid b in A and shifting goods from some other bid to b so that it becomes (maximally) satisfied. We feel, however, that this approach would not adequately reflect the fact that GLBs have *degrees of satisfaction* (i.e., values). Furthermore, revenue maximizing assignments need not necessarily *maximally* satisfy any bid; thus this neighborhood relation would not necessarily allow one to reach optimal solutions from arbitrary points in search space.

The neighborhood relation we propose can be seen as a compromise between these two extremes; it is based on the observation that the existence of priced subformulae in GLBs provides the means to improve the value of a bid in natural increments by moving goods from bid to bid in *price-improving bundles*. The value of a GLB under an assignment A is determined precisely by the *priced subformulae* that are satisfied by A. We then define assignment A' to be a neighbor of A, if it can be reached from A by selecting an unsatisfied priced subformula in some bid b and by moving just enough goods to that bid to satisfy this subformula.

To complete the definition of an SLS procedure, we need to specify methods for selecting an initial assignment and for choosing a neighboring assignment at each search step. Analogous to the CASLS scheme, we propose to start the search at an empty assignment where all goods are unassigned and all the bids are fully unsatisfied (i.e., their value is zero).

To formally define the method for selecting neighbors, we use the notion of a *logical bid tree*: Each GLB b can be represented as a logical bid tree whose subtrees correspond to the priced subformulae of b; this tree is simply the parse tree for $\Phi(b)$ with prices attached to each node. A subformula without a (top-level) price attached is semantically equivalent to the same subformula with price zero; we call such subformulae and the corresponding nodes *priceless*, while all other subformulae (and nodes) are called *priced*. Since each subtree of a logical bid tree is itself a GLB, notions of value under a given assignment and maximal value under any assignment are well-defined for subtrees; the *(maximal) value of a node* is the (maximal) value of the subtree rooted at that node.

Our method for selecting a neighbor of the current assignment in each search step proceeds in two stages: First, choose from some partially unsatisfied bid b an unsatisfied price node n whose ancestors do not have maximal value.[3] Then, select a set G' of goods that, when assigned to bid b, will satisfy node n. Together, these two choices determine a neighboring assignment, which is reached by reassigning all goods in G' to bid b. More precisely, we restrict this second selection to minimal satisfying sets G', (i.e., to sets that contain only goods that are necessary to satisfy n). Finding such a minimal set satisfying n is rather straightforward using the procedure *Satisfy*(n), recursively computed as follows:

(a) Let n be a leaf node labeled with good g. Then return g.

(b) Let $n = \wedge(n_1, n_2, \cdots, n_k)$, and w.l.o.g. assume that $n_1, \cdots, n_j$ $(1 \leq j \leq k)$ are unsatisfied, and $n_{j+1}, \cdots, n_k$ are satisfied. Since n is not satisfied, at least one subnode must be unsatisfied. (Satisfaction information is recorded for each node.) Then return $\cup_{i \leq j}$*Satisfy*(n_i), calling the *Satisfy*(n_i) in random order.

(c) Let $n = \vee(n_1, n_2, \cdots, n_k)$. (Note that since n is not satisfied, each subnode must be unsatisfied.) Randomly choose one n_i, $i \leq k$. Then return *Satisfy*(n_i).

(d) Let $n = \oplus(n_1, n_2, \cdots, n_k)$. (Note that the satisfaction of such a subformula is defined exactly as in the case of $\vee$. Hence, since n is not satisfied, each subnode must be unsatisfied.) Randomly choose one n_i, $i \leq k$. Then return *Satisfy*(n_i).

To obtain minimal assignments, we update the assignment of goods incrementally as we work recursively through the tree. This way, once a good is assigned in one part of the tree this fact will be reflected in the other parts of the tree. Due to the stochastic choice of subformula to satisfy within ORs and VXORs, and the random order in which subformulae beneath AND nodes are visited, *Satisfy*(n) can find any minimal assignment of goods to bid b that will satisfy node n.

The schematic stochastic local search algorithm we propose initializes the search at an empty assignment and then iteratively moves from the current assignment to a neighboring assignment by transferring a set of goods between bids as described above. After each such search step, the satisfaction information for all bids (and all subformulae within bids) is updated based on the new assignment. In practice, to deal with premature stagnation, this SLS technique will be extended with standard restart mechanisms such that the search is reinitialized from an empty assignment after a fixed number of steps have been performed since the last initialization (*fixed cutoff restart*) or whenever no improvement in revenue has been achieved for a given number of steps (*soft restart*).

Concrete instantiations of this algorithmic framework are obtained by specifying mechanisms for the various selections in each search step: the choice of a subformula to be satisfied, and the choice of a minimal assignment (as implemented by *Satisfy*(n), possibly extended by an additional selection from a number of minimal assignments that satisfy n). There is a broad range of possibly suitable and effective mechanisms for these selections; which of many strategies will work best will have to be determined based on empirical analyses. However, it seems clear that the choices should be made in a biased randomized fashion such that alternatives that lead to higher direct increases in $V(A)$ are selected with higher probability, while any possible alternative can be chosen with some small, lower-bounded probability. The former criterion is based on analogous results for standard combinatorial auctions [Hoos and Boutilier, 2000] and other well-known combinatorial problems such as propositional satisfiability, while the latter is a sufficient condition to ensure that for arbitrarily long runs, the SLS procedure will find an optimal solution with probability approaching one (i.e., it ensures probabilistic approximate completeness, see [Hoos, 1999] for details).

[3] To implement the search procedure efficiently, for each bid, we compute the maximum value $\max(n)$ for each node n in the logical bid tree (or for each subformula) prior to commencing the search. This computation is simple and requires just a single bottom-up sweep of each logical bid tree.

The general approach we propose here can also be used to obtain a systematic search algorithm capable of finding optimal solutions and proving their optimality. The overall search method could be very similar, starting with an empty assignment and selecting subformulae that are satisfied by assigning a set of goods in each step. To guarantee completeness of the algorithm, all choices would have to be done in a systematic fashion such that when using a backtracking mechanism, the full search space of a given problem instance will be explored after a finitely bounded amount of time. Notice that this is possible even for randomized choices. The practical efficiency of such an algorithm would depend on suitable heuristics for ordering the alternatives to be explored at each choice point, and on sufficiently powerful pruning or bounding techniques. This approach could be very useful for solving relatively small problem instances provably optimally. However, considering the NP-hardness of the given problem and well-known results for other hard combinatorial optimization problems, such as MAX-SAT, TSP, or standard combinatorial auctions, we believe that SLS techniques like the one outlined above will most likely show better absolute performance and anytime behavior on large and complex problem instances.

5 Concluding Remarks

We have proposed a new logical bidding language for CAs that exploits structure in utility functions, thereby facilitating the natural and concise expression of bids. By associating prices with subformulae and adequately dealing with sharable resources, $\mathcal{L}_{GB}$ can express certain bids exponentially more compactly than existing languages. We have also sketched a search procedure for solving CAs that does not require the conversion of logical bids to atomic bids. Though this procedure has not been tested empirically, we are confident that it will work well. We are currently developing an implementation suitable for extensive experimentation.

Apart from empirical work, we are also exploring extensions of $\mathcal{L}_{GB}$ to deal with *k-of* expressions and multiunit CAs. The distinction between sharable and consumable goods also deserves further exploration. Clearly an important concept for CAs, $\mathcal{L}_{GB}$'s ability to make this distinction implicitly is very desirable for the natural, concise expression of preferences. Finally, we are currently pursuing the connection to work in resource-oriented logics (e.g., linear logic [Girard, 1987]), though existing logics do not seem to able to handle complementarities.

References

[Fujisima *et al.*, 1999] 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.

[Girard, 1987] Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

[Hoos and Boutilier, 2000] 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.

[Hoos, 1999] Holger H. Hoos. *Stochastic Local Search – Methods, Models, Applications*. infix-Verlag, Sankt Augustin, Germany, 1999.

[Keeney and Raiffa, 1976] R. L. Keeney and H. Raiffa. *Decisions with Multiple Objectives: Preferences and Value Trade-offs*. Wiley, New York, 1976.

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

[Rassenti *et al.*, 1982] 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.

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

[Sandholm, 1999] 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.

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

[Wellman *et al.*, 2001] Michael P. Wellman, William E. Walsh, Peter R. Wurman, and Jeffrey K. MacKie-Mason. Auction protocols for decentralized scheduling. *Games and Economic Behavior*, 2001. To appear.