Chapter 1

Bidding Languages

Noam Nisan

1.1 Introduction

This chapter concerns the issue of the *representation* of bids in combinatorial auctions. Theoretically speaking, bids are simply abstract elements drawn from some space of strategies defined by the auction. Every implementation of a combinatorial auction (and, in fact, every other game) must define how each possible bid is actually represented using the underlying communication technology. As the representation details have no strategic implications, the issue of representation is often ignored. This issue, however, may not be ignored in combinatorial auctions due to the underlying complexity of bids: the space of possible bids in combinatorial auctions is usually huge, as large as the space of possible valuations. Specifying a valuation in a combinatorial auction of m items, requires providing a value for each of the possible $2^m - 1$ non-empty subsets. A naive representation would thus require $2^m - 1$ real numbers to represent each possible bid. It is clear that this would be very inconvenient in practice even for 10-item auctions, and completely impractical for more than about two dozen items.

We will thus look for more convenient *bidding languages* that will allow us to encode "common" bids more succinctly. The exact details of the communication technology are not important, the only point being that communication is always achieved using finite strings of characters. A bidding language is thus simply a mapping from the abstract mathematical space of possible bids into the set of finite strings of characters. The exact choice of the set of characters does not matter much as long as it is finite, and may be taken practically to be the set of 100-or-so ASCII characters, or more theoretically, to be the boolean set $\{0, 1\}$.

This puts us in a "syntax vs. semantics" situation common in logic and in computer science. A *language* assigns a semantic meaning (in our case, a bid) to each well-formed syntactic element (in our case, word in the bidding language). The language specification matters and needs to be studied to the extent that it affects our ability to constructively handle the semantic items of interest. Bidding languages for combinatorial auctions have always been implicit in any implementation of combinatorial auctions. They were first studied explicitly and in a general and formal way in (Nisan 2000).

Representation of real numbers

A standard technical detail that needs to be taken care of before we proceed is the fact that infinite-precision real numbers cannot be represented by finite strings of characters. We will ignore this issue by actually assuming the opposite: i.e. that a single real number *may* be represented in a finite number of characters. One may formalize this in various mathematical ways, e.g. by assuming that all real numbers in bids are given in *finite* precision, e.g. using 64 bits. (This is certainly enough for all practical applications, allowing a micro-penny precision on trillion-dollar auctions.)

Bids vs. Valuations

Each combinatorial auction mechanism may define an arbitrary space of possible bids. This space of bids is identical to the space of valuations when the mechanism is a direct revelation mechanism, as is commonly the case. However this is not mandatory and, in principle, the space of bids may be quite different from the space of valuations. Of course, a player's bid will always be a function of his valuation, but still not necessarily equivalent to it. However, in almost all studied single-round auctions, the bid space is essentially equivalent to the valuation space[1]. In such cases, bids *are* valuations. Indeed in all studied bidding languages, the bidding language is really a *valuation language* – a syntactic representation of valuations. We use the name "bidding language" rather than the more precise "valuation language" due to the main intended use of such languages – bidding[2].

A bidding language can now provide a side benefit: classify valuations according to their complexity: "simple valuations" have short representations while "complex valuations" require long representations. A collection of bidding languages may yield a qualitative classification according to the set of bidding languages in which the valuation has sufficiently succinct representation. Such syntactic classifications may be useful alongside the various semantic classifications of valuations.

Information-theoretic limitations

From the onset we should notice that it is impossible to allow encoding *all* possible valuations succinctly. This is due to simple information theoretic

[1] Iterative auctions (Parkes, chapter 3), on the other hand, rely on incremental bidding rather than on the bidders providing a single bid in some bidding language.

[2] It is quite hard to think of useful bidding languages that are not valuation languages, since once players' functions from valuations to bids are fixed (as they will be in any studied equilibrium point), it is possible to associate every bid with a valuation that results in this bid. Still, this possibility remains an interesting research direction.

reasons: there are less than 2^t strings of bits of length strictly less than t, and thus not all elements from a space of size 2^t can be represented using strings of length less than t. The size of the space of all valuations on m items, (with each real number limited to finite precision) is exponential in 2^m. It follows that in any bidding language some valuation (actually, almost all valuations) requires encoding length that is exponential in m. Our interest would thus be not in attempting to succinctly represent all valuations but rather only to succinctly represent *interesting* ones.

Expressiveness vs. Simplicity

When attempting to choose or design a bidding language we are faced with the same types of trade offs common to all language design tasks: *expressiveness vs. simplicity*. On one hand we would like our language to express important valuations well, and on the other hand we would like it to be as simple as possible. Let us look closer at these two requirements.

We would first want our valuation language to be *fully expressive*, i.e. to be able to express *every* valuation. We would then want to ensure that "important" valuations have succinct representations. In any intended use there are some valuations that may be realistically those of some bidder – these are the important ones for this application. Other valuations are unlikely to appear in this application, hence their encoding length is not an issue. Unfortunately it is quite hard to characterize this set of "realistic" valuations: not only is it application-specific and not general, but also there is very little empirical evidence from real combinatorial auctions.

We are thus left with two possible ways to choose the "important" valuations for which we aim to give short representations. The "soft" approach attempts guessing various types of valuations that would seem to be natural and realistic in many settings. A language that can more succinctly represent more of these valuations would be considered better. The "hard" approach attempts comparing the expressive power of various bidding languages, formally proving relations between representation lengths in different languages. One language would be better than another if it can succinctly express any valuation that the other may.

The notion of the simplicity of a language has two components: an intuitive one and a technical one. First, the language must "look" simple to humans. Humans should be able to intuitively understand what a bid in the language means, as well as how to represent a valuation. This requirement may be somewhat relaxed, allowing such simplicity to emerge only with the use of "software crutches" in the form of agents. Technically, simplicity should mean that all computational tasks associated with the valuations should be as easy as possible. The most obvious such task being winner determination when bids are presented in this language.

One would expect the goals of expressiveness and simplicity to be relatively conflicting, as the more expressive a language is, the harder it becomes to handle it. A well chosen bidding language should aim to strike a good balance between these two goals.

The rest of the chapter

We follow the standard notations used in most of this book: our combinatorial auction is on a set of m items. A valuation v provides a real value $v(S)$ for each subset S of the items. We assume that $v(\emptyset) = 0$ and that for $S \subseteq T$ we have that $v(S) \leq v(T)$.

We start, in section 2, by providing some examples of valuations that we consider natural. In no way is this meant to be a comprehensive list of types of valuations that we wish our bidding languages to express, but rather just a representative sample. Section 3 discusses the main paradigm for bidding languages: exclusive and non-exclusive combinations of package bids. Section 4 describes some extensions and special cases. Section 5 shortly describes the associated computational complexity issues. Section 6 contains the technical proofs for the theorems in section 3. Finally, section 7 concludes.

1.2 Some Examples of Valuations

We present here a few examples of valuations that seem to be natural and are used below as examples. Some of these valuations are symmetric, i.e. all items are identical (at least from the point of view of the bidder), while other valuations are not. For symmetric valuations v, the value $v(S)$ depends only on the size of the set of items obtained S.

1.2.1 Symmetric Valuations

The simple additive valuation

The bidder values any subset of k items at value k. I.e. $v(S) = |S|$.

The simple unit demand valuation

The bidder desires any single item, and only a single item, and values it at 1. Thus $v(S) = 1$ for all $S \neq \emptyset$.

The simple K-budget valuation

Each set of k items is valued at k, as long as no more than K items are obtained. I.e. $v(S) = min(K, |S|)$.

Intuitively, each single item is valued at 1, but the total budget available to the bidder is K.

The Majority-valuation

The bidder values at 1 any majority (i.e. set of size at least $m/2$) of the items and at 0 any smaller number of items.

The General Symmetric valuation

Let $p_1, p_2, ..., p_m$ be arbitrary non-negative numbers. The price p_j specifies how much the bidder is willing to pay for the j'th item won. Thus $v(S) = \sum_{j=1}^{|S|} p_j$.

In this notation the simple additive valuation is specified by $p_j = 1$ for all j. The simple unit demand valuation is specified by $p_1 = 1$ and $p_j = 0$ for all $j > 1$. The simple K-budget valuation is specified by $p_j = 1$ for $j \leq K$ and $p_j = 0$ for $j > K$. The majority valuation is specified by $p_{m/2} = 1$ and $p_j = 0$ for $j \neq m/2$.

A Downward sloping Symmetric Valuation

A symmetric valuation is called downward sloping if $p_1 \geq p_2 ... \geq p_m$.

1.2.2 Asymmetric Valuations

An additive valuation

The bidder has a value v^j for each item j, and he values each subset as the sum of the items in it. I.e. $v(S) = \sum_{j \in S} v^j$.

The unit demand valuation

The bidder has a value v^j for each item j, but desires only a single item. Thus $v(S) = \max_{j \in S} v^j$.

The monochromatic valuation

There are $m/2$ red items and $m/2$ blue items for sale. The bidder requires items of the same color (be it red or blue), and values each item of that color at 1. Thus the valuation of any set of k blue items and l red items ($|S| = k + l$) is $max(k, l)$.

The one-of-each-kind valuation

There are $m/2$ pairs of items. The bidder wants one item from each pair and values it at 1. Thus, the valuation of a set S that contains k complete pairs and l singletons ($|S| = 2k + l$) is $k + l$.

1.3 Exclusive and non-Exclusive Bundle-bids

In this section we present the leading paradigm for bidding in combinatorial auctions: making sets of bids for "bundles" of items, where the bid for the different bundles can be either exclusive or non-exclusive. This paradigm seems to be intuitive and is often used without even formally defining the underlying bidding language. Basic bidding languages of this form were explicitly used in (Sandholm 1999; Fujishima et. al. 1999), and the general classification was given in (Nisan 2000).

1.3.1 Basic Bidding Languages

We first start by defining the most basic types of bids – those that desire a single bundle of items.

Atomic bids

Each bidder can submit a pair (S, p) where S is a subset of the items and p is the price that he is willing to pay for S. Thus $v(T) = p$ for $S \subseteq T$ and $v(T) = 0$ otherwise. Such a bid is called an atomic bid.

Atomic bids were called single-minded bids in (Lehmann et. al. 1999). It is clear that many simple bids cannot be represented at all in this language, e.g. it is easy to verify that an atomic bid cannot represent even the simple additive valuation on two items.

OR bids

Each bidder can submit an arbitrary number of atomic bids, i.e. a collection of pairs (S_i, p_i), where each S_i is a subset of the items, and p_i is the maximum price that he is willing to pay for that subset. Implicit here is that he is willing to obtain any number of disjoint atomic bids for the sum of their respective prices. Thus an OR bid is equivalent to a set of separate atomic bids from different bidders. More formally, for a valuation $v = (S_1, p_1)\, OR ... OR\, (S_k, p_k)$, the value of $v(S)$ is defined to be the maximum over all possible valid collections W, of the value of $\sum_{i \in W} p_i$, where W is valid if for all $i \neq j \in W$, $S_i \cap S_j = \emptyset$.

Not all valuations be represented in the OR bidding language. It is easy to verify that the following proposition completely characterizes the descriptive power of OR bids.

Proposition 1.3.1 *OR bids can represent all bids that don't have any substitutabilities, i.e., those where for all $S \cap T = \emptyset$, $v(S \cup T) \geq v(S) + v(T)$, and only them.*

In particular OR bids cannot represent the simple unit demand valuation on two items.

XOR bids

Each bidder can submit an arbitrary number of pairs (S_i, p_i), where S_i is a subset of the items, and p_i is the maximum price that he is willing to pay for that subset. Implicit here is that he is willing to obtain at most one of these bids. More formally, for a valuation $v = (S_1, p_1) XOR ... XOR (S_k, p_k)$, the value of $v(S)$ is defined to be $max_{i | S_i \subseteq S} p_i$.

Proposition 1.3.2 *XOR-bids can represent all valuations.*

The term XOR bids is taken from (Sandholm 1999), and is commonly used despite the fact that purists may object to the confusion with the totally different boolean exclusive-or function. As we have seen, XOR bids can represent everything that can be represented by OR bids, as well as some valuations that can not be represented by OR bids – those with substitutabilities. Yet, the representation need not be succinct: there are valuations that can be represented by very short OR bids and yet the representation by XOR bids requires exponential size.

Definition 1 *The size of a bid is the number of atomic bids in it.*

Proposition 1.3.3 *Any additive valuation on m items can be represented by OR bids of size m. The simple additive valuation requires XOR bids of size 2^m.*

1.3.2 Combinations of OR and XOR

While both the OR and XOR bidding languages are appealing in their simplicity, it seems that each of them is not strong enough to succinctly represent many desirable simple valuations. A natural attempt is to combine the power of OR bids and XOR bids. In this subsection we investigate such combinations.

OR-of-XORs bids

Each bidder can submit an arbitrary number of XOR-bids, as defined above. Implicit here is that he is willing to obtain any number of these bids, each for its respectively offered price.

OR-of-XORs bids were called OR-XOR bids in (Sandholm 1999). OR-of-XORs bids generalize both plain XOR bids and plain OR bids. The following example is a non-obvious example of their power and demonstrates that OR-of-XOR bids can succinctly express some valuations that cannot be succinctly represented by either OR-bids or XOR-bids alone.

Lemma 1.3.4 *(Nisan 2000) OR-of-XORs bids can express any downward sloping symmetric valuation on m items in size m^2.*

Proof: For each $j = 1..m$ we will have a clause that offers p_j for any single item. Such a clause is a simple XOR-bid, and the m different clauses are all connected by an OR. Since the p_j's are decreasing we are assured that the first allocated item will be taken from the first clause, the second item from the second clause, etc. []

The fact that the valuation is downward sloping, $p_1 \geq p_2 \geq ... \geq p_n$, is important: The majority valuation (which is not downward sloping) requires exponential size OR-of-XORs bids. This exponential lower bound will be shown below in theorem 1.3.8 for an even more general language, $OR*$-bids.

One may also consider the dual combination: XOR-of-OR bids. Intuitively these types of bids seem somewhat less natural than OR-of-XOR bids, but, as we will see, they can also be useful.

XOR-of-ORs bids

Each bidder can submit an arbitrary number of OR-bids, as defined above. Implicit here is that he is willing to obtain just one of these bids.

An example where XOR-of-ORs bids are more powerful than OR-of-XORs bids is the monochromatic valuation. XOR-of-ORs bids require only size m: "(OR of all blues) XOR (OR of all reds)". On the other hand, OR-of-XORs require exponential size for this, as the following theorem, whose proof appears in section 6, shows.

Theorem 1.3.5 *(Nisan 2000) The monochromatic valuation requires size of at least $2 \cdot 2^{m/2}$ in the OR-of-XORs bidding language.*

On the other hand, for other valuations the OR-of-XORs language may be exponentially more succinct. We have already seen that OR-of-XORs bids can succinctly represent any downward sloping symmetric valuation, and, so in particular, the simple K-budget valuation. In contrast, we have the following theorem, whose proof is postponed to section 6.

Theorem 1.3.6 *(Nisan 2000) Fix $K = \sqrt{m}/2$. The K-budget valuation requires size of at least $2^{m^{1/4}}$ in the XOR-of-ORs bid language.*

It is natural at this point to consider not just OR-of-XORs and XOR-of-ORs, but also arbitrarily complex combinations of ORs and XORs. We will define valuations obtained from general OR/XOR formulae over atomic bids. This general definition also formalizes the semantics of the OR-of-XORs and XOR-of-ORs languages discussed above. The general definition treats OR and XOR as operators on valuations. The XOR operator allows taking any one of the two operands, and the OR operator allows partitioning the items between the two operands. Formally,

Definition 2 *Let v and u be valuations, then $(v\,XOR\,u)$ and $(v\,OR\,u)$ are valuations and are defined as follows:*

- $(v\,XOR\,u)(S) = \max(v(S), u(S))$.
- $(v\,OR\,u)(S) = \max_{R,T \subseteq S,\ R \cap T = \emptyset}\ v(R) + u(T)$

OR/XOR formulae bids

Each bidder may submit a OR/XOR formula specifying his bid. OR/XOR formula are defined recursively in the usual way using OR and XOR operators, where the operands are either atomic bids (specifying a price p for a subset S) o,r recursively, OR/XOR formulae.

It should be noticed that all previously mentioned bidding languages are special types of OR/XOR formulae bids, where the formulae are restricted to a specific syntactic form. In particular, the previous two lemmas provide examples of valuations that can be represented succinctly by general OR/XOR formulae but require exponential size OR-of-XOR bids or alternatively XOR-of-OR bids. A single valuation which requires exponential size for both OR-of-XORs bids and XOR-of-OR bids can be obtained by defining a valuation that combines the monochromatic valuation on the first $m/2$ items and the K-budget valuation on the second $m/2$ items.

1.3.3 OR Bids with Dummy Items

We now introduce the bidding language of our choice. This language was introduced in (Fujishima et al 1999) in order to allow XOR bids to be expressed as a variant of OR bids, and its full power was exposed in (Nisan 2000). The idea in this bidding language is to allow the bidders to introduce "dummy" items into the bidding. These items will have no intrinsic value to any of the participants, but they will be indirectly used to express constraints. The idea is that a XOR bid $(S_1, p_1)\, XOR\, (S_2, p_2)$ can be represented as $(S_1 \cup \{d\}, p_1)\, OR\, (S_2 \cup \{d\}, p_2)$, where d is a dummy item.

Let M be the set of items for sale, we let each bidder i have its own set of dummy items D_i, which only he can bid on.

OR* bids

Each bidder i can submit an arbitrary number of pairs (S_l, p_l), where each $S_l \subseteq M \cup D_i$, and p_l is the maximum price that he is willing to pay for that subset. Implicit here is that he is willing to obtain any number of disjoint bids for the sum of their respective prices.

An equivalent but more appealing "user interface" may be put on the OR* language as follows: users may enter atomic bids together with "constraints" that signify which bids are mutually exclusive. Each one of these constraints is them simply interpreted as a dummy item that is added to the pair of mutually exclusive bids. Despite its apparent simplicity, this language turns out to be the most efficient one so far and can simulate all bidding languages presented so far, including general OR/XOR formulae.

Theorem 1.3.7 *(Nisan 2000) Any valuation that can be represented by OR/XOR formula of size s, can be represented by OR* bids of size s, using at most s^2 dummy items.*

The proof is postponed to section 6 and may be directly converted into a "compiler" that takes OR/XOR formulae bids and converts them to OR* bids. The following theorem shows that even the OR* language has its limitations. The proof is postponed to section 6.

Theorem 1.3.8 *The majority valuation requires size of at least $\binom{m}{m/2}$ in the OR* bid language.*

One of the appealing features of OR* bids is that despite their power, they "look just like" regular OR-bids, on a larger set of items. Winner determination algorithms that expect to see OR-bids, will directly thus be able to handle OR* bids as well.

1.4 Extensions and Special cases

So far we have discussed the basic standard constructs used in bidding languages. In this section we first shortly describe various possible extensions, and then shortly discuss some important special cases.

1.4.1 Additional Constructs

We have so far started with simple atomic bids and used two "combining operators" on valuations: OR and XOR. One may think of many other ways of constructing valuations from combinations of simpler valuations. We now shortly mention some of those that seem to "make sense".

Logical languages

We start by considering the basic building blocks. We have so far used only the very simple atomic bids. In general we should be able to take any monotone boolean predicate $f : \{0,1\}^m \rightarrow \{0,1\}$, and assign a value p to any bundle that satisfies this predicate (identifying a boolean vector $b \in \{0,1\}^m$ with the set of items $\{j | b_j = 1\}$). Formally, the logical valuation $v = (f, p)$ is given by $v(S) = p$ if $f(S) = 1$ and $v(S) = 0$ otherwise. These valuations can then further be combined by OR and XOR operations as well as other constructs presented below.

Naturally, at this point one might ask how can f be represented. The most natural representation would be to use monotone boolean formulae (Hoos and Boutlier 2000). The most general representation would be to allow general boolean circuits. The comparison between the many various possibilities puts us well into the field of boolean complexity theory, and the interested reader may refer e.g., to (Wegner 1987).

Budget limits

Given a valuation, we can think of limiting it up to some given "budget" k – bundles that used to receive a higher valuation are now valued at the budget limit. Formally: the valuation $v' = Budget_k(v)$ is defined by $v'(S) = min(v(S), k)$.

Limits on number of items

Given a valuation we can think of limiting it up to some maximum number k of obtained items. Valuations of larger sets of items are truncated by ignoring the extra items. Formally: the k-satiated valuation $v' = Sat_k(v)$ is defined by $v'(S) = max_{S' \subseteq S,\, |S'| \leq k} v(S')$.

Semantics for AND: ALL and MIN

Certainly, the name of the "OR" construct suggests that we should have an "AND" construct as well. There is probably no truly satisfying definition of an AND construct on valuations. The most natural definition, called ALL in (Zinkevich et al 2003) takes value 0 (zero) to mean failure, and uses addition to combine non-failed values. Formally, the valuation $v = v_1\, ALL\, v_2$ is defined by $v(S) = 0$ if $v_1(S) = 0$ or if $v_2(S) = 0$, and otherwise $v(S) = v_1(S) + v_2(S)$. An alternative semantics is to simply take point-wise minimum. Formally: the valuation $v = v_1\, MIN\, v_2$ is defined by $v(S) = min(v_1(S), v_2(S))$.

k-OR

The OR and XOR operators can be viewed as extreme cases of a general construct that allows combinations of at most k valuations from a given set of t valuations. In this respect, XOR is the case $k = 1$, while OR is the case with $k = t$. Formally: the valuation $v = OR_k(v_1 ... v_t)$ is defined by $v(S) = max_{S_1 ... S_k} \sum_{j=1}^{k} v_{i_j}(S_j)$, where $S_1 ... S_k$ form a partition of the items, and $i_1 <$

$i_2 ... < i_k$.

Operators with associated prices

It is sometimes natural to "factor out" parts of the values of bundles in a way that represents how the value is derived. For example, consider the valuation giving value 101 to item A and 102 to item B (lets ignore the value of the pair AB). This valuation could have been obtained by the following logic "any item will give me a benefit of 100, and I also get a side benefit of 1 from A and 2 from B". One can think of suggesting this using a syntax like $([(A,1)OR(B,2)],100)$. A general language using such constructs was suggested in (Boutlier and Hoos 2001).

1.4.2 Special Cases

Symmetric Valuations

Clearly the case where all items are identical (at least for the bidder) is of special interest. In this case the value of a subset is fully determined by the size of the subset, i.e. by the number of items won. Auctions where this is guaranteed to be the case for all bidders are in fact just multi-unit (single-good) auctions. Symmetric valuations can be fully represented by a vector of m numbers $v_1 \leq ... \leq v_m$, with $v(S) = v_{|S|}$. Equivalently and more intuitively they can be represented by the vector of marginal values $p_i = v_i - v_{i-1} \geq 0$ as was done in section 2 above.

When the symmetric valuation is downward sloping, i.e. $p_i \geq p_{i+1}$ for all i, this same information may be provided in a natural economic format by a *demand curve* d, where $d(p)$ denotes the number of items desired at price p per item, i.e. the largest value of i such that $p_i < p$. This demand curve may be fully specified by giving the value of d only for the finite set of values p where $d(p)$ changes. This representation may be more concise than providing the full vector of marginal values in cases where the p_i's are not strictly decreasing but rather remain constant for long intervals.

Combinations of Singleton Valuations

An interesting class of valuations is those obtained by restricting the atomic bids to be *singleton bids*, i.e., atomic bids (S,p) where $|S| = 1$. ORs of such singleton valuations are exactly the additive valuations, and XORs of singleton valuations are the unit demand valuations. In (Lehman et al. 2001) more complex combinations of singleton bids were studied: OR of XORs of singletons, termed OXS bids, and XOR of ORs of singletons, termed XOS bids. None of these languages is fully expressive, as all XOR and OR combinations of singleton bids are complement-free (i.e. satisfy $v(S \cup T) \leq v(S) + v(T)$). The expressive power of these classes (without any limitations on succinctness) was shown to form a proper hierarchy within the class of complement-free valuations.

Theorem 1.4.1 *(Lehman et al. 2001) The following sequence of strict containments holds: $GS \subset OXS \subset SM \subset XOS \subset CF$. Here, GS is the class* *of (gross-)substitute valuations, SM the class of submodular valuations, and CF the class complement-free valuations.*

Network valuations

In many cases, the items for sale are network resources. Specifically and formally, they may be viewed as edges in some underlying graph. In many such cases, bidders will pay a given price for any bundle of items that has some specific property in the underlying graph. In such cases, bidding language can simply allow describing the desired property as well as price offered. for example, consider a bidder that requires to send a message from vertex s to vertex t in the underlying graph. For this, he must acquire a set of edges that consists of a directed path from s to t. In this case, the bidding language should simply allow specifying s, t, and a price p. Similar examples includes a bidder that desires a spanning tree of a particular subset of vertices, a set of edges with enough flow capacity, etc.

1.5 Computational Complexities of Bidding Languages

Once a bidding language has been fixed, various desired operations on valuations become computations over strings, and their computational complexity can be studied. There are various such operations of which at least three should be mentioned:

1. **Expression:** Given another representation of a valuation v, express v in the bidding language. This task of course depends on the "other representation", of which the most interesting one is the intuitive non-formal way by which people *know* their preferences.

2. **Winner Determination:** Given a set of valuations $\{v_i\}$, find the allocation (i.e., a partition of the items) $\{S_i\}$ that maximizes the total value: $\sum_i v_i(S_i)$.

3. **Evaluation:** Given a valuation v, and a "query" about it, answer the query. The most basic type of query to be considered is the *value query*: Given a set S, determine $v(S)$. Another import query is the *demand query*: Given a set of item prices $\{p_i\}$, find the set that maximizes $v(S) - \sum_{i \in S} p_i$. Other types of queries may be considered as well, of course.

We will not discuss further the question of Expression complexity due to its non-formal nature. It is certainly an important topic for experimental evaluation.

1.5.1 Complexity of Winner Determination

The question of winner determination is studied at length in part III (chapters 13-17) of this book. Here we only wish to point out the basic computational characteristics: Even the simplest bidding language, one allowing atomic bids only, gives an NP-complete optimization problem. Even the strongest language that we have studied still gives "just" an NP-complete problem. This can be easily seen by the fact that from the point of view of winner determination, bids in the OR^* language look exactly like collections of simple atomic bids. Thus from a "high-level" point of view winner determination complexity is basically independent of the bidding language choice. Of course, in real applications we do expect stronger bidding languages to require more effort, and more refined analysis will uncover such differences.

Important exceptions that must be mentioned are those bidding languages that give rise to polynomial-time solvable winner determination problem. These cases are studied in chapter 14, and we mention here just two cases: XOR-bids when there a constant number of bidders (using simple exhaustive search) and OXS-bids (using bipartite maximum-weight matching).

1.5.2 Complexity of Evaluation

We would now like to address the question of the algorithmic difficulty of interpreting a bid. Let us start with the simplest queries, value queries: given a valuation v in one of the bidding languages defined above, how difficult is it to calculate, for a given set of items S, the value $v(S)$? One would certainly hope that this algorithmic problem (just of interpreting the bid – not of any type of allocation) is easy.

This is indeed the case for the atomic bidding language as well as the XOR bidding language. Unfortunately, this is not true of all other bidding languages presented above, starting from the OR bidding language. Given an OR bid:

$$v = (S_1, p_1)\, OR\, (S_2, p_2)\, OR\, ...\, OR\, (S_k, p_k),$$

computing the value $v(S)$ for a subset S is exactly the same optimization problem as allocating S among many different bidders (S_i, p_i), a problem which is NP-complete. Thus theoretically speaking almost all languages considered are "too strong" – even a simple value query cannot be answered on them. However, it seems that practically, this is not a problem: either allocation between the different clauses is practically easy, or else the difficulty gets merged into the hardness of winner determination.

Now let us consider demand queries: given a valuation v and a set of "item prices" $p_1 ... p_m$, find the set that maximizes $v(S) - \sum_{i \in S} p_i$. It is again easy to verify that for the atomic bidding language as well as for the XOR bidding language this is still solvable in polynomial time. This problem is NP-complete for stronger languages, starting with the OR language. In general, one may observe that demand queries are always at least as hard as value queries:

Lemma 1.5.1 *In any bidding language, a value query may be reduced to demand queries (via a polynomial-time Turing reduction).*

Proof: First, we can reduce a value query to a sequence of "marginal value" queries: given a subset S and an item $i \notin S$, evaluate $v(i|S) = v(S \cup \{i\}) - v(S)$. The reduction is simply using the recursive algorithm: $v(\emptyset) = 0$ and $v(S \cup \{i\}) = v(S) + v(i|S)$.

A marginal value query $v(i|S)$ can be reduced to demand queries using the item prices: $p_j = 0$ for $j \in S$ and $p_j = \infty$ for $j \notin S \cup \{i\}$. The demand query will answer S for every $p_i > v(i|S)$ and will answer $S \cup \{i\}$ for $p_i < v(i|S)$. The value of $v(i|S)$ can thus be found using binary search (assuming finite precision). []

1.5.3 Complete Bidding Languages

At this point it may become natural to consider "complete" bidding languages, i.e. languages that can efficiently simulate "all" bidding languages from some class. The natural choice would be a bidding language that allows submitting arbitrary polynomial-time computable *programs* as valuations. What is not clear is what these programs should compute. The natural answer is that such "program bids" should answer value queries: accept as input a set S and return, in polynomial time, the value $v(S)$. This natural choice is not entirely satisfying for two reasons: (1) This does not give enough power to a winner determination algorithm that accepts several such bids: almost nothing non-trivial can be done with value queries alone. (2) As previously mentioned, in most bidding languages, value queries are NP-complete. It follows that such a polynomial-time computable program will *not* be able to simulate, e.g., even the OR language.

Two alternatives may then be contemplated, each attempting to address one of these issues. The first would be to require these program-bids to answer other queries as well, e.g. demand queries. This may enable winner determination algorithms to function using better information, as in (Bartal et. al. 2003). On the other hand, this would only further make simulation of other bidding languages harder. The other approach, suggested in (Nisan 2000), is to weaken the query semantics and make it "non-deterministic": such a query will accept a set S and a "witness" w, and must output some value $v(S, w)$. The only requirement is that $v(S) = \max_w v(S, w)$. It may be verified that all bidding languages considered in this chapter have such non-deterministic value queries, and may thus be simulated by such program-bid language.

1.6 Proofs of Theorems From Section 3

Proof: (of Theorem 1.3.5) Consider a fixed OR-of-XORs bid representing the monochromatic valuation. Let us call each of the XOR expressions in the bid, a clause. Each clause is composed of atomic bids to be called in short, atoms.

Thus an atom is of the form (S, p) where S is a subset of the items. We can assume without loss of generality that each atom in the bid is monochromatic, since otherwise removing such non-monochromatic atoms cannot distort the representation of the monochromatic valuation. Also, for no atom can we have $p > |S|$ since otherwise the valuation of S will be too high. Thus we can also assume without loss of generality that for all atoms $p = |S|$, since otherwise that atom can never be "used" in the valuation of any set T, since that would imply too low a valuation for T.

We can now show that all atoms must be in the same clause. Assume not, if two blue atoms are in different clauses, then any red atom cannot be in the same clause with both of them. If a red atom $(p = |T|, T)$ and a blue atom $(p = |S|, S)$ are in different clauses, then since the clauses are connected by an OR, then the valuation given by the bid to $S \cup T$ must be at least $|S| + |T|$, in contradiction to the definition of the monochromatic valuation.

Thus the whole bid is just a single XOR clause, and since every subset S of the red items and every subset S of the blue items must get the valuation $|S|$, it must have its own atom in the XOR clause. Since there are $2^{m/2}$ blue subsets and $2^{m/2}$ red subsets the theorem follows. []

Proof:(of Theorem 1.3.6) Consider a XOR-of-ORs bid representing the K-budget valuation. Let us call each of the OR expressions in the bid, a clause. Each clause is composed of atomic bids, in short, atoms. Thus an atom is of the form (S, p) where S is a subset of the items. Let us call a set of size K, a K-set, and an atom of a K-set, a K-atom. Let t be the number of clauses in the bid and l the number of atoms in the largest bid.

First, for no atom (p, S) can we have $p > |S|$ since otherwise the valuation of S will be too high. Thus we can also assume without loss of generality that for all atoms $p = |S|$, since otherwise that atom can never be "used" in the valuation of any set T, since that would imply too low a valuation for T.

For every K-set S, the valuation for S must be obtained from one of the clauses. This valuation is obtained as an OR of disjoint atoms $S = \cup_i S_i$. We will show that if the clause size $l \leq 2^{m^{1/4}}$, then each clause can yield the valuation of at most a fraction $2^{-m^{1/4}}$ of all K-sets. Thus the number of clauses must satisfy $t \geq 2^{m^{1/4}}$, obtaining the bound of the lemma.

Fix a single clause. Now consider whether some K-set S is obtained in that clause via a disjoint union $S = \cup_i S_i$, of at least $m^{1/4}$ atoms.

If such a set S does not exist then all K-sets S whose valuation is obtained by this clause are obtained by a union of at most $m^{1/4}$ atoms. Thus the total number of K-sets S obtained by this clause is bounded from above by $\binom{l}{m^{1/4}}$. As a fraction of all possible K-sets this is $\binom{l}{m^{1/4}}/\binom{m}{K}$, which, with the choice of parameters we have, is bounded from above by $2^{-m^{1/4}}$.

Otherwise, fix an arbitrary K-set S that obtains its valuation as a disjoint union of at least $m^{1/4}$ atoms $S = \cup_i S_i$. The main observation is that any other

K-set T whose valuation is obtained by this clause, must intersect each one of these sets S_i, since otherwise the valuation for $T \cup S_i$ would be too high! Let us bound from above the number of possible such sets T. A randomly chosen K-set T has probability of at most Kc/m to intersect any given set S_i of size c. For all $S_i \subseteq S$, this is at most $1/4$, since $c \le K = \sqrt{m}/2$. The probability that it intersects all these $m^{1/4}$ sets S_i is thus at most $4^{-m^{1/4}}$ (the events of intersecting with each S_i are not independent, but they are negatively correlated, hence the upper bound holds). Thus the fraction of K-sets T that can be obtained by this clause is at most $4^{-m^{1/4}} \le 2^{-m^{1/4}}$. []

Proof: (of Theorem 1.3.7) We prove by induction on the formula structure that a formula of size s can be represented by an OR* bid with s atomic bids. We then show that each atomic bid, in the final resulting OR* bid, can be modified as to not to include more than s dummy items in it.

Induction: The basis of the induction is an atomic bid, which is clearly an OR* bid with a single atomic bid. The induction step requires handling the two separate cases: OR and XOR.

To represent the XOR of several OR* bids as a single OR* bid we introduce a new dummy item x_j for each original bid j. For each atomic bid (S_i, p_i) in the j'th OR* bid, we add an atomic bid $(S_i \cup \{x_j\}, p_i)$ to the generated OR* bid.

To represent the OR of several OR* bids as a single OR* bid, we introduce a new dummy item x_{ST} for each pair of atomic bids (S, p) and (T, q) that are in two different original OR* bids. For each bid (S, p) in any of the original OR* bids, we add to the generated OR* bid an atomic bid $(S \cup \{x_{ST} | T\}, p)$, where T ranges over all atomic bids in all of the other original OR* bids.

It is clear that the inductive construction constructs a OR* bid with exactly s clauses in it, where s is the number of clauses in the original OR/XOR formula. The number of dummy items in it, however, may be exponentially large. However, we can remove most of these dummy items. One can see that the only significance of a dummy item in an OR* bid is to disallow some two (or more) atomic bids to be taken concurrently. Thus we may replace all the existing dummy items with at most $\binom{s}{2}$ new dummy items, one for each pair of atomic bids that cannot be taken together (according to the current set of dummy items). This dummy item will be added to both of the atomic bids in this pair. []

Proof:(of Theorem 1.3.8) First note that no subset of the real items of size smaller than $m/2$ can appear with a non-zero valuation in the bid (whatever the associated phantom items are). Now, since every set of the real items of size $m/2$ should have valuation 1, and they cannot get that valuation indirectly from subsets, then this set must appear as one of the atoms in the OR* bid. []

1.7 Conclusion

The core of this chapter was the presentation and formal definition of several bidding languages. These languages were formally studied in terms of their expressive power. The main formal results are of two types: either a simulation result showing that one language is at least as good as another, or a separation result exhibiting a valuation that can be described succinctly in one language, but not in another one. Most attention was given to bidding languages that use combinations of bids on bundles of items; several extensions and important special cases were also shortly mentioned.

The most important computational task associated with a bidding language is solving the winner determination problem among such bidders. This issue was hardly touched in this chapter, as it is the subject of part III of this book. Other computational issues were shortly discussed.

Bibliography

[1] C. Boutilier and H. H. Hoos (2001). Bidding Languages for Combinatorial Auctions. In *Seventeenth International Joint Conference on Artificial Intelligence (IJCAI-01).*

[2] Y. Bartal, R. Gonen, and N. Nisan (2003). Incentive Compatible Multi-Unit Combinatorial Auctions. In *conference of Theoretical Aspects of Knowledge and Reasoning.*

[3] Y. Fujishima, K. Leyton-Brown, and Y. Shoham (1999). Taming the computational complexity of combinatorial auctions: Optimal and approximate approaches. In *Proceedings of IJCAI'99*, Stockholm, Sweden, Morgan Kaufmann.

[4] H. H. Hoos and C. Boutlier (2000). Solving combinatorial auctions using stochastic local search. In *Proceedings of the 7th National Conference on Artificial Intelligence.*

[5] D. Lehmann, L. I. O'Callaghan, and Y. Shoham (1999). Truth revelation in rapid, approximately efficient combinatorial auctions. In *1st ACM conference on electronic commerce.*

[6] B. Lehmann, D. Lehmann, and N. Nisan (2001). Combinatorial Auctions with Decreasing Marginal Utilities. In *3rd ACM conference on electronic commerce.*

[7] N. Nisan (2000). Bidding and Allocation in Combinatorial Auctions. In *2nd ACM conference on electronic commerce.*

[8] T. Sandholm (1999). An algorithm for optimal winner determination in combinatorial auctions. In *IJCAI-99.*

[9] I. Wegner (1987). *The Complexity of Boolean Functions.* John Wiley and Sons.

[10] M. Zinkevich, A. Blum, and T. Sandholm, (2003). On Polynomial-Time Preference Elicitation with Value Queries. In Proceedings of the ACM Conference on Electronic Commerce (ACM-EC), 176-185.