Bidding and Allocation in Combinatorial Auctions

Noam Nisan*

April 17, 2000

Abstract

When an auction of multiple items is performed, it is often desirable to allow bids on combinations of items, as opposed to only on single items. Such an auction is often called "combinatorial", and the exponential number of possible combinations results in computational intractability of many aspects regarding such an auction. This paper considers two of these aspects: the bidding language and the allocation algorithm.

First we consider which kinds of bids on combinations are allowed and how, i.e. in what language, they are specified. The basic tradeoff is the expressibility of the language versus its simplicity. We consider and formalize several bidding languages and compare their strengths. We prove exponential separations between the expressive power of different languages, and show that one language, "OR-bids with phantom items", can polynomially simulate the others.

We then consider the problem of determining the best allocation – a problem known to be computationally intractable. We suggest an approach based on Linear Programming (LP) and motivate it. We prove that the LP approach finds an optimal allocation if and only if prices can be attached to single items in the auction. We pinpoint several classes of auctions where this is the case, and suggest greedy and branch-and-bound heuristics based on LP for other cases.

1 Introduction

1.1 Motivation

Auctions are an important market mechanism from times immemorial. They allow selling rare and unusual goods and apply in situations where a more conventional "market", in which buyers consider the price as given, does not exist. A large informal body of knowledge on auctions has been in existence for hundreds of years, and a more formal, game theoretic, analysis of auctions began in the 1960's with the pioneering work of Vickrey [27]. The field in micro-economics and game theory that studies these and related issues is often call mechanism design or implementation theory. An introduction to this field can be found in the text books [20, 12], and an influential web site in [17].

Recently, we have seen a great rise in the popularity of auctions of various types. This development occurred in many settings: in government privatizations and rights allocation (most famously, the FCC spectrum rights auctions) [17, 14]; in the many Internet auction sites [1, 2]; in the usage of techniques from auction theory for computational resource allocation [28, 9, 19]; in computerized

*Institute of Computer Science, Hebrew U., Jerusalem. This research was supported by grants from the Israeli ministry of Science and the Israeli academy of sciences.

agent systems [25, 22]; and in current trends in business-to-business electronic commerce [21]. With this new dominance of auctions, also comes a quantum leap in the complexity of these auctions.

In many cases, auctions of multiple non-identical items is performed. The price that a bidder may offer for one item may depend in complicated ways on what other items he wins. This may be due to several items complementing each other (e.g. a left sock and a right sock; or rights to identical frequency bands in geographically adjacent areas), or they may be partial substitutes of each other (e.g. a red T-shirt and a green T-shirt; or rights to different frequency bands in the same geographic area). Such cases also occur frequently when dealing with computational or communication resources, where each resource is only part of a possible solution and thus only bundles of resources have true value (e.g. a set of communication links forming a complete communication path). Values of items in business to business auctions may also turn out to be a function of other items obtained due to the internal pricing models of each business.

In such cases it is often desirable to allow bids on *combinations of objects*, as opposed to only on single objects. Such an auction is often called "combinatorial" or "combinational". A combinatorial auction is desirable since it allows the bidders to express their true preferences, and thus may lead to better allocations. However, the exponential number of possible combinations usually results in computational intractability of dealing with such auctions. Due to these difficulties, only a small number of combinatorial auctions have been implemented to date.

1.2 Basic Issues in Combinatorial Auctions

There are basically four issues one must deal with in a combinatorial auction:

- **Bidding:** Each bidder must be able to express a bid on combinations of items. The bid may be interpreted as a half-contract, committing to the maximum amount of money that the bidder is willing to pay for each possible combinations of items that he may get. Since there are an exponential number of combinations, this information will usually be implicitly coded in a succinct way by the auction communication. The auction protocol determines how this bidding communication is done, and in particular which bids can be expressed, and how efficiently.

- **Allocation:** Once the bids are all in, the items in the auction must be allocated among the different bidders. The allocation will attempt the optimization of some target function, usually the auctioneer's revenue or the total economic efficiency. However, in most cases, this optimization problem is computationally intractable (NP-complete) and heuristics must be used.

- **Payment:** How much does each winner of a set of items pay? In some cases this may not be identical to what he bid. The payment rules determines the bidders' strategies as well as the auctioneers' revenue.

- **Strategy:** Once the auction's protocol, allocation, and payment rules are fixed, each bidder is free to choose an arbitrary bidding strategy. A well designed auction will ensure that the intended goals of the auction are met when all bidders act according to their selfishly chosen strategies.

In this paper we consider only the first two issues: bidding and allocation. Since we want to isolate these issues, we will not consider the payment rules and the bidders' strategies (which are at the heart of what is usually studied in mechanism design.) We will assume that the bids are simply the bidders' valuations, and we will try to optimize the total economic efficiency according to the declared bids. The reader who would rather think in game-theoretic terms and consider strategic bidders, may alternatively assume that the pricing mechanism is defined as to make the auction incentive compatible (e.g. using the Vickrey-Clarke-Groves (VCG) mechanism [27, 7, 5]), in which case we are assured that all bidders will act non-strategically.

We will concentrate on sealed-bid auctions. In these cases, the bidding rules amount to no more than a bidding language – i.e. syntax and semantics for bids. The simplest bidding language would be to let each bidder place a vector of numbers specifying his bid for each possible subset of items he can win. This is clearly un-reasonable as there are an exponential number of combinations, and the bid itself would be exponentially long. Thus the bidding language must provide some short-hand for placing these bids. The basic tradeoff in specifying a bidding language is expressiveness vs. simplicity.

- **Expressiveness:** The bidding language should be able to express any desired vector of bids, and furthermore "important" ones should be easily expressed. This ease should be both technical, i.e. the expression of the bid should be short, and human, i.e. human bidder should find it easy to express their bid in this language (directly or via agent programs).

- **Simplicity:** Dealing with bids expressed in the bidding language should be simple. This simplicity should be both in a technical sense, i.e. it should be computationally easy to handle the allocation when bids are in the language, and in a human sense, i.e. humans should find it simple to understand and work with this bidding language.

One would expect these to goals 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 to goals.

1.3 Previous Work

Clearly much work on auction design exists, and an introduction can be found in the textbooks [20, 12]. Most of this work considered single item auctions and concentrated on issues of bidders' strategies.

Situations where the value of an item to a bidder strongly depends on other items he wins have been identified by economists in situations like airline landing slot auctions, spectrum auctions, electricity markets and others. In most cases, combinatorial auctions were not employed due to the complexities involved. In such cases usually one of the following was attempted:

- Ignore the problem, and let bidders do their speculation about whether they will win future items.

- Add a re-sale market for the items, which hopefully can help the creation of the economically desirable combinations.

- Sell multiple items in parallel in a multi-round auction. Hopefully feedback from previous rounds (about all items) can provide bidders with information for bidding in the next round. The most famous auction of this type is the simultaneous ascending auction (see the many references in [17]) and it seems that many high-value auctions of multiple goods follow this route.
- Provide a mechanism by which two or more bidders can "team up" and combine their bids. Hopefully, they will be able to find the optimal way of teaming up.

All of these approaches merely try to improve on single-item auctions, and indeed are very far from providing a general way to bid on combinations. Recently, several researchers started considering actually providing combinatorial auctions and providing algorithms to perform the allocation.

These papers have introduced several types of bidding languages, specifically single-minded bids (which we call atomic bids) [10], OR-bids, XOR-bids, and OR-of-XOR bids [24], and OR-bids with dummy items (which we call phantom items) [6], and several restricted classes of bids [23]. None of these papers formally studied the power of the bidding language they used.

The allocation algorithms suggested may be classified into three types:

- Provably optimal and polynomial time allocation algorithms for restricted classes of bids [23, 26].
- Provably optimal allocations that are obtained by effectively doing an exponential search over the allocation space. The idea in these papers is to cleverly trim the search space as much as possible [6, 24]. A recent paper suggests using existing integer programming software [3].
- Heuristics that run quickly (in polynomial time), and yet hopefully provide rather good allocations [10].

It is well known that if the payments in a combinatorial auction with a perfect allocation algorithm are defined according to the Vickrey-Clarke-Groves (VCG) mechanism [27, 7, 5] then the auction is incentive compatible, and it is a dominant strategy for each bidder to bid his true valuation. This is no longer true when the allocation algorithm is not optimal [18]. An incentive compatible combinatorial auction with a computationally efficient (but not optimal) allocation algorithm appears in [10].

1.4 Results Obtained

1.4.1 Analysis of Bidding Languages

We formally define six bidding languages including, as far as we know, all those that were considered in the literature. These languages are: OR-bids, XOR-bids, OR-of-XORs, XOR-of-ORs, OR/XOR-formulae, and OR-bids with phantom items – called OR*-bids. We analyze the expressiveness of each of them, and compare their relative power. We prove two major results here:

- The OR-of-XORs and XOR-of-ORs languages are incomparable in their expressive power: there are bids that can be expressed succinctly in the OR-of-XORs language but require exponential size XOR-of-ORs bids, and conversely, there are bids that can be expressed succinctly in the XOR-of-ORs language but require exponential size OR-of-XORs bids.

- The OR* bidding language is strictly more expressive than both OR-of-XORs and XOR-of-ORs: it can efficiently simulate these two languages and even more general OR/XOR formulae, and can succinctly express some valuations that require exponential size in any of the former two languages.

As it was shown to be the most expressive, the rest of the paper uses the OR* bidding language. Allocation algorithms are presented for this bidding language, thus demonstrating the technical simplicity of this language.

1.4.2 Allocation using Linear Programming

The allocation problem in combinatorial auctions is easily and commonly formalized as an integer programming problem – which is, unfortunately, computationally intractable. A common approach for solving integer programming problems is to start by relaxing them to a linear programming problem which can be efficiently solved. This paper advocates using this approach for combinatorial auctions, and provides motivation, algorithms, and theorems supporting this.

Our first observation is that the linear program is intimately related to the question of whether prices can be attached to individual items in the auction. Such prices for individual items are certainly desirable and it is known that for some combinatorial auctions they exist [28, 29, 4] and in other cases they can not be defined in any reasonable manner [16, 13, 28, 29]. When the bids are presented, as this paper assumes, in the OR* language, we completely characterize the cases where individual item prices exist:

- A combinatorial auction admits individual item prices if and only if the linear program solution is integral and hence directly optimizes the allocation.

We then identify a host of situations where we can prove that the linear program does directly optimize the allocation. In any of these cases, our algorithm provides an optimal allocation. These situations include most of those for which a polynomial time algorithm appears in the literature, specifically in [8].

- If any one of the following restrictions apply to all bids in the auction then an optimal allocation is obtained: Hierarchical bids, Linear order bids, Single item bids, OR-of-XOR-of-singletons bids, and downward sloping symmetric bids. An optimal allocation is also obtained if a "direct sum" of such auctions is taken or if "majorized" bids are added to such auctions.

We should emphasize that none of these restrictions need to be enforced a-priori by the combinatorial auction rules. It is just that if bids do happen to fall into any one of these categories then an optimal solution is directly found by the linear program. Otherwise, one of the following two heuristics that we present should be used:

- A greedy heuristic that provably runs in polynomial time, however, we will not be able to guarantee that it always produces the optimal allocation.

- A branch-and-bound heuristic that provably produces the optimal allocation, however, we will not be able to guarantee that it always runs in polynomial time.

Both of these heuristics take approaches that are rather standard in the combinatorial optimization literature, and in particular for integer programming. The variants proposed here, though, are intended specifically for combinatorial auctions and we believe that they will perform especially well in cases of real combinatorial auctions. As they are heuristics we cannot, of course, prove this last statement. The basic intuition behind this belief is that in real auction settings items will "almost" have individual item prices, and that in these cases the LP solution will "almost" provide an optimal allocation.

2 The Model

This paper considers only sealed bid auctions, assumes the private value model, and does not address the issue of bidders' strategies. Specifically, there are m items for sale by a single auctioneer in a single combinatorial auction. There are n bidders who all desire these items. Each bidder i has its own private valuation function, v_i, that specifies his valuation for each possible subset of items that he may get. I.e. for a subset S of items, $v_i(S)$ is the amount of money that bidder i is willing to pay in order to win this subset S of items. We assume that each bidder i knows (or can determine) his valuation function. In this paper we assume that these valuations all satisfy the following conditions:

- **No Externalities:** The bidders' valuation depends only on the set of items he wins. I.e. the valuation function v_i is $v_i = v_i(S)$, where S is the set of items won by bidder i.
- **Free disposal:** Items have non-negative value. Thus v_i satisfies $v_i(S) \leq v_i(T)$, whenever $S \subseteq T$.
- **Normalization:** $v_i(\emptyset) = 0$.

These valuation functions may exhibit subsets of items that are complementary to each other or substitutes of each other:

Definition 1 *For a bidder i, we will say that two disjoint sets of items S and T are:*

- **Complementary** *if* $v_i(S \cup T) > v_i(S) + v_i(T)$.
- **Substitutes** *if* $v_i(S \cup T) < v_i(S) + v_i(T)$.

We will say that a valuation function v_i has no substitutabilities if no two sets are substitutes for i and has no complementarities if no two sets are complementary for i.

The auction proceeds in a single round where each bidder presents his bid in a "sealed envelope" to the auctioneer. A bid simply represents a valuation v_i, and is a commitment from the bidder to pay at most $v_i(S)$ if he is allocated the subset S of items. The auctioneer reads all bids and then invokes the allocation algorithm deciding the allocations of all items. The auctioneers' aim is to find an allocation, i.e. pair-wise disjoint sets $S_1...S_n$, that maximize the total declared economic efficiency, i.e. $\sum_i v_i(S_i)$.

3 Bidding Languages

A bidding language is a formalism for expressing valuations. So before we start considering bidding languages, let us mention some of the valuations that they should be able to represent.

3.1 Examples of Valuations

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

3.1.1 Symmetric Valuations

The additive valuation
The bidder values any subset of k items with a valuation of k. I.e. $v(S) = |S|$.

This valuation exhibits no substitutes or complementarities.

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

In this valuation all items are perfect substitutes of each other.

The 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 additive valuation is specified by $p_j = 1$ for all j. The single item valuation is specified by $p_1 = 1$ and $p_j = 0$ for all $j > 1$. The 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 $p_1 \geq p_2 ... \geq p_m$.

A downward sloping valuation captures the case of decreasing marginal values: i.e. each additional item is worth less than the previous one. It may be viewed as the "normal" economic case, as it corresponds to a “normal” demand curve.

3.1.2 Some Asymmetric Valuations

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 (thus $|S| = k + l$) 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 (thus $|S| = 2k + l$) is $k + l$.

3.2 Basic Bidding Languages

We first start by mentioning the most basic types of bidding languages.

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 [10]. 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 can not represent even the 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.

OR-bids were used in [24]. While the semantics of OR bids should be clear from this definition, the reader who would like more formality may find it in section 3.5. Not all valuations can even be represented at all in the OR-bid language. It is easy to verify that the following proposition completely characterizes the descriptive power of OR bids:

Proposition 3.1 *OR bids can represent all bids that don't have any substitutabilities, and only them.*

In particular OR bids can not represent at all the single item valuation even 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. Thus the bid for a set of items that contains several of the S_i's is the maximum p_i from these bids.

It is easy to observe that XOR bids can describe all valuations:

Proposition 3.2 *XOR-bids can represent all valuations.*

The term XOR bids is taken from [24]. As we have seen XOR bids can represent everything that can be represented by OR bids, as well as some valuations can not be represented by OR bids – those with substitutabilities. Yet, the representation may not be language: there are valuations that can be represented by very short OR bids and yet the representation by XOR bids requires exponential size.

Definition 2 *The size of a bid is the number of atomic bids in it.*

The following separation is left to the reader:

Proposition 3.3 *The additive valuation on m items can be represented by OR bids of size m, but requires XOR bids of size 2^m.*

3.3 OR-of-XORs bids

While both the OR and XOR bidding languages are appealing in their simplicity, it seems that the 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 the rest of this section 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 [24]. Like XOR bids, they can describe any valuation. However, the required size may be exponentially smaller in some cases. Specifically, OR-of-XORs bids generalize, in particular, plain OR bids, which, as we have seen, may be exponentially more language than XOR bids. Here is a non-obvious example of their power:

Lemma 3.4 *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 can be 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 later in lemma 4.4 for an even more general language, $OR*$-bids.

We will now give another example of a valuation that can not be succinctly represented in the OR-of-XORs language.

Theorem 3.5 *The monochromatic valuation requires size of at least $2 \cdot 2^{m/2}$ in the OR-of-XORs bidding.*

Proof: Consider a 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 can not 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. ♠

3.4 XOR-of-ORs bids

One may also consider 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.

XOR-of-ORs bids can efficiently express the monochromatic valuation, by "(OR of all blues) XOR (OR of all reds)", so for some valuations this language is more succinct than OR-of-XORs bids. We will now show that, on the other hand, for other valuations, it may also be less efficient. We have already seen that OR-of-XORs bids can succinctly represent any downward sloping symmetric valuation, and, so in particular, the K-budget valuation.

Theorem 3.6 *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.*

Proof: 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 $c \leq K = \sqrt{m}/2$. 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 $4^{-m^{1/4}} \leq 2^{-m^{1/4}}$. ♠

3.5 OR/XOR formulae

It is natural at this point to consider not just OR-of-XORs and XOR-of-ORs, but also arbitrary combinations of ORs and XORs. This is indeed possible, and we can define general OR/XOR formulae. Such a general definition may also appeal to the reader who wishes more formality in the definitions of bidding languages presented above. First we must formally define the meaning of the valuations $(v \, OR \, u)$ and $(v \, XOR \, u)$, where v and u are arbitrary valuations.

Definition 3 *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 formula are defined recursively in the usual way using these OR and XOR operators, where the operands are either atomic bids (specifying a price p for a subset S) or recursively OR/XOR formulae.

OR/XOR formulae bids

Each bidder may submit a OR/XOR formula specifying his bid.

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 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 either OR-of-XORs bids or XOR-of-OR bids can easily be obtained by defining a valuation that combines the monochromatic valuation on the first m items and the K-budget valuation on the second m items.

3.6 Bidding Languages and Polynomial Time Computation

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

Definition 4 *A bidding language is called polynomially interpretable if there exists a polynomial time algorithm that given as input a bid in the language b and a subset of items S, computes the value $b(S)$.*

It is easy to verify that the atomic bidding language as well as the XOR bidding language are polynomially interpretable. Unfortunately, this is not true of all other bidding languages presented above, including in particular the OR bidding language. Given an OR bid $b = (S_1, p_1)\ OR\ (S_2, p_2)\ OR\ ...\ OR\ (S_k, p_k)$, computing the value $b(S)$ 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. Yet, all authors that have considered combinatorial auctions have not limited themselves to, what we call, polynomially interpretable bidding languages. This is probably due to two reasons: first, polynomially interpretable languages are simply not expressive enough; second, in an auction, the optimization problem of interpreting the bid is never left to the bidder, as the allocation algorithm anyway has to solve it.

This leads us to a more relaxed definition of the required algorithmic efficiency: that the allocation algorithm will be able to "prove" to the bidder or to a referee the value of $b(S)$. This is similar to an NP-type proof.

Definition 5 *A function V, that accept three inputs, a bid b, a subset S, and a proof w, is said to verify a bidding language if, for all b and S, $\max_w V(b, S, w) = b(S)$. A bidding language is called polynomially verifiable if there exists a polynomial time verifying algorithm for it.*

It is easy to see that all bidding languages that we defined above (as well the OR* language to be defined below) are polynomially verifiable. In all cases, the "proof" is a description of the allocation of the set of items won into the different components of the bid. All allocation algorithms that we will suggest below, naturally obtain this information and can offer it to the bidders.

At this point it may become natural to consider "complete" bidding languages, i.e. languages that can simulate all polynomial time verifiable bidding languages (while still being polynomial time verifiable themselves). It is easy to verify that the following language is indeed complete in this sense:

Applet bids

A bid is a polynomial time computer program b that accepts two inputs, a subset S and a string w, and outputs a number. The value of a bid b on a subset S is $\max_w b(S, w)$.

(These are called "applet" bids, after the Java programming language name for mobile computer programs. More traditionally they would be called Universal Turing Machine bids.)

While such a complete language can certainly represent all reasonable valuations succinctly, there seems to be no hope for an allocation algorithm that aims to do any reasonable allocation with such bids.

4 OR bids with phantom items

We now introduce the bidding language of our choice. This language was introduced in [6] in order to allow XOR bids to be expressed as a variant of OR bids. The idea in this bidding language is to allow the bidders to introduce "phantom" 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. Let G be the set of items for sale, we let each bidder i have its own set of phantom items G_i, which only it can bid on.

OR* bids

Each bidder i can submit an arbitrary number of pairs (S_l, p_l), where each $S_l \subseteq G \cup G_i$, 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.

The idea is that a XOR bid $(S_1, p_1)XOR(S_2, p_2)$ can be represented as $(S_1 \cup \{g\}, p_1)OR(S_2 \cup \{g\}, p_2)$, where g is a phantom item. This language turns out to be the most efficient one so far. We will see that is can simulate all bidding languages presented so far (except applet bids, of course). We shall start be observing that OR* bids are at least as strong as OR-of-XORs bids, as well as XOR-of-ORs bids.

It is not clear how intuitive these phantom items are for human users. A more appealing "user interface" may be put on the OR* language, by allowing users to enter atomic bids as well as "constraints" that signify which bids are mutually exclusive. Each one of these constraints is them simply interpreted as a phantom item that is added to these bids.

Lemma 4.1 *(implicit in [6]) Any valuation that can be represented by OR-of-XORs bids of size s, can be represented by OR* bids of total size s, using at most s dummy items.*

Proof: For each XOR clause we introduce a phantom item x, and for each atomic bid (S, p) in that clause we add an atomic bid $(S \cup \{x\}, p)$ to the OR* bid. ♠

Lemma 4.2 *Any valuation that can be represented by XOR-of-ORs bids of size s, can be represented by OR* bids of size s using at most s^2 phantom items.*

Proof: For each pair of atomic bids $(S, p), (T, q)$ (in the XOR-of-ORs bid) that are in different OR-clauses, we introduce a phantom item x_{SQ}. For each bid (S, q), we add an OR* bid $(S \cup \{x_{ST}|T\}, p)$, where T ranges over all atomic bids (in the original XOR-of-ORs bid) in different OR-clauses. ♠

We can now prove the general theorem that, the $OR*$ language can even simulate general OR/XOR formulae:

Theorem 4.3 *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 phantom items.*

Proof: 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 phantom 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 is exactly like in the previous two lemmas, but counting only the size and ignoring the blow-up in the number of phantom items.

Finally, given any OR* bid with s atomic bids in it, one can see that the only significance of a phantom item is to disallow some two (or more) atomic bids to be taken concurrently. Thus we may replace all the existing phantom items with at most $\binom{s}{2}$ new phantom items, a new phantom item for each pair of atomic bids that can not be taken together (according to the current set of phantom items). This phantom item will be added to both of the atomic bids in this pair. The total number of phantom items added to each atomic bid is thus at most $s-1$. ♠

Thus the OR* bid language is at least as strong as all previous ones considered. Immediately it follows that it is strictly stronger than both the OR-of-XORs language and the XOR-of-ORs one. We do not know whether it is strictly stronger than general OR/XOR formulae.

The proof above may be directly converted into a "compiler" that takes OR/XOR formulae bids and converts them to OR* bids.

We now observe that even the OR* language has its limitations.

Lemma 4.4 *The majority valuation requires size of at least $\binom{m}{m/2}$ in the OR* bid language.*

Proof: 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 can not get that valuation indirectly from subsets, then this set must appear as one of the atoms in the OR* bid. ♠

The OR* bid language can be extended to express valuations with externalities by allowing phantom items that are common to several bidders. E.g. if bidder 1 wants to bid on item 1 only if bidder 2 does *not* win item 2, then this can be achieved by introducing a common phantom item x, forcing bidder 2 to add x to any of his bids that includes item 2, and letting bidder 1 add x to his bids that depend on bidder 2 not winning item 2. We will not further develop these possibilities in this paper.

5 The Allocation Problem and Linear Programming

Once all bidders have bid (using the OR* bidding language), we can now attempt the allocation of all items among all bidders. This section formalizes this problem and relates it to the issue of linear programming.

5.1 A Formalization of the Allocation Problem

Conventions and Notation

We assume that each bidder placed a bid in the OR* bidding language. From the point of view of the allocation algorithm, the identity of the bidder of each of the atomic bids does not matter so we will simply consider the total set of atomic bids: $\{B_i | i = 1...n\}$, where each bid B_i is given by $B_i = (S_i, p_i)$, where S_i is a subset of the items (both real and phantom), $S_i \subseteq G = \{g_1...g_m\}$, and $p_i \geq 0$ is the bid for that subset. Thus the distinction between real and phantom goods also does not matter for the algorithm.

The allocation we desire is to choose a subset of winning bids, i.e., a partition of G to disjoint subsets $\{S_i | i \in Winners\}$, such that $\sum_{i \in Winners} p_i$ is maximized. This may be formalized by the following integer programming problem (which is actually a packing problem). In this formalization, x_i is a bit that denotes whether the bid B_i is a winner.

Integer Programming Formalization

Maximize:
$\sum_{i=1}^{n} x_i p_i$
Subject to:
$\sum_{i|j \in S_i} x_i \leq 1$, for each $j = 1...m$.
$x_i \in \{0, 1\}$, for each $i = 1...n$.

5.2 The Linear Programming Relaxation

A classic technique in combinatorial optimization theory is to relax an integer program to a linear one. The following is the natural relaxation of the packing problem to a, so called, fractional packing problem. In this relaxation we allow what was previously a bit, x_i, to become any fraction. Intuitively, this corresponds to bids that are partial winners – winning an x_j fraction of the bid. Clearly the combinatorial auction problem statement does not allow such taking of fractions of bids, and this is what we will have to deal with in the rest of the paper.

Linear Programming Relaxation

Maximize:
$\sum_{i=1}^{n} x_i p_i$
Subject to:
$\sum_{i|j \in S_i} x_i \leq 1$, for each $j = 1...m$.
$x_i \geq 0$ for each $i = 1...n$.

5.3 The Meaning of the LP Solution

In this subsection we will discuss the basic connection between the solution of the linear program and the solution to the original combinatorial auction problem. We will identify situations where no gap exists.

5.3.1 Fractional Combinatorial Auctions

In many cases the auction setting itself may allow fractions of bids to be won, as opposed to only complete bids. Taking a fraction f of a bid (S_i, p_i) means that the bidder gets f fraction of each item in S_i and pays for this bundle the price $f \cdot p_i$. Possible examples of such auctions are for raw materials (like oil, or gold), for electricity, or for bandwidth (e.g. network bandwidth). E.g. a bid for the connection bundle $\{TelAviv - Paris, Paris - NewYork\}$, may be only 2/3-winning meaning that both connections are available at 2/3 of the required bandwidth. If another bundle, e.g. $\{TelAviv - London, London - NewYork\}$ is 1/3-won by the same bidder, then this bidder now has enough bandwidth for his purposes (assuming that messages can be split between different channels – a situation which is true in many communication settings).

Philosophically, this clarifies the source of difficulty in combinatorial auctions. The main difficulty does *not* come from the effects of substitutabilities or complementarities – these are handled by the linear program, as long as OR*-bids are used. The difficulty lies in the indivisibility of the goods which is what can not be handled well.

5.3.2 A Characterization by Single Item Prices

Any combinatorial auction yields prices for the various winning *bundles*. I.e. one can associate the price p_i with the winning bundle S_i. A more desirable outcome would be to associate prices y_j with specific *items* in accordance to the bundle bids. It is not completely clear how such item prices can be defined, but the following definition is certainly a minimum requirement.

Definition 6 *We say that the allocation $x_1 ... x_n \in \{0,1\}^n$ is supported by single-item prices $y_1 ... y_m$ if for every winning bid, $x_i = 1$, we have $p_i \geq \sum_{j \in S_i} y_j$, and for every losing bid, $x_i = 0$, we have $p_i \leq \sum_{j \in S_i} y_j$. We say that the allocation is exactly supported if, in fact, $p_i = \sum_{j \in S_i} y_j$ for all winning bids.*

We say an allocation x is full if all items are allocated in it, i.e. for each j there exists $x_i = 1$ such that $j \in S_i$. We say that the auction admits single-item prices if it has any full allocation x that is supported by any vector of single item prices y.

Thus when single item prices exists, bids win because they offer at least the sum of prices of the items in the bundle they desire, while bids lose since they did not offer the required sum of the item prices. Perhaps somewhat surprisingly, not all combinatorial auctions do admit such single item prices. It turns out that the combinatorial auctions that do admit single-item prices are exactly those where the linear programming relaxation produces a non-fractional solution. This is a simple corollary of LP duality theory.

Theorem 5.1 *A combinatorial auction admits single-item prices if and only if the solution of the linear programming relaxation above produces integer $\{0,1\}$ solutions. In this case the full allocation is the solution of the linear program, the supporting prices are the solutions to the dual linear program, and the allocation is supported exactly.*

This theorem is slightly more subtle than it may seem. Let us inspect what happens if the valuations in the auction had substitutes, e.g. bidder 1 bidding *(5 for A) XOR (6 for B)*. In such a case, representing the bid in our formalization requires the introduction of phantom items (in this example as: *(5 for [A,P]) OR (6 for [B,P]))*. The theorem attaches item prices to all items – including phantom items. These prices on phantom items are essential for the correctness of this theorem.

In fact, there are known examples [15] where single item prices do not exist only due to this lack of phantom items or an equivalent construct. In the example above, if additionally to bidder 1 bidding *(5 for A) XOR (6 for B)*, bidder 2 bids *(3 for B)* then individual item prices cannot be found without attaching a price also to the implicit phantom item "P": the optimal allocation is to give "A" to bidder 1 and "B" to bidder 2. Since bidder 2 wins "B", the price attached to "B" cannot be more than 3, but if no phantom item "P" exists, since bidder 1 does not obtain "B", "B" must be priced at least at 6 – a contradiction. If phantom items are assigned prices, then we can get the following supporting prices: "P" is given a price of 4, "A" is valued at 1, and "B" at 3.
Proof:

Let us first consider the dual linear program:

The Dual Linear program

Minimize:
$\sum_{j=1}^{m} y_j$
Subject to:
$\sum_{j \in S_i} y_j \geq p_i$, for each $i = 1...n$
$y_j \geq 0$ for each $j = 1...m$.

Let $x_1^*...x_n^*$ be the optimal solution of the primal program, and $y_1^*...y_m^*$ be the optimal solution of the dual program. From LP-duality theory we know the following:

1. The primal and dual problems have the same value, i.e. $\sum_{j=1}^{m} y_j^* = \sum_{i=1}^{n} x_i^* p_i$.
2. For each j such that $y_j^* > 0$, we have the j'th constraint of the primal problem saturated, i.e. $\sum_{i|j \in S_i} x_i^* = 1$.
3. For each i such that $x_i^* > 0$, we have the i'th constraint of the dual problem saturated, i.e., $\sum_{j \in S_i} y_j^* = p_i$.

We can now proceed with the proof. First we show that if the linear program has integer solutions, i.e. $x_i^* \in \{0, 1\}$ for all i, then the dual solution y^* is a vector of single-item prices exactly supporting the allocation: The fact that for every losing bid, (S_i, p_i), we have $p_i \leq \sum_{j \in S_i} y_j^*$ is a direct corollary of the fact that y^* is a dual-feasible solution. The fact that for every winning bid, (S_i, p_i), we have $p_i = \sum_{j \in S_i} y_j^*$, follows directly from the third property from LP duality theory mentioned above.

Assume now that there exists any vector of individual item prices y that supports the a full allocation $x \in \{0, 1\}^n$. We can assume without loss of generality that y exactly supports the allocation, since otherwise, for every winning bid i where $p_i > \sum_{j \in S_i} y_j$, we can choose an arbitrary $j \in S_i$ and increase y_j until we get equality. We are now in a situation where x is primal-feasible and y is dual-feasible. Since for every winning bid i (i.e. $x_i = 1$) we have that $p_i = \sum_{j \in S_i} y_j$, we can represent

$$\sum_{i=1}^{n} x_i p_i = \sum_{i|x_i=1} p_i = \sum_{i|x_i=1} \sum_{j \in S_i} y_j = \sum_{j=1}^{m} y_j$$

Thus the values of the primal and dual solutions are equal and thus they must be optimal solutions of the primal and dual linear programs respectively. ♠

6 Cases where the LP relaxation is optimal

There are many restricted classes of combinatorial auctions where the LP relaxation will provably yield integer results. In such cases, we obtain the provably optimal allocation directly. These cases include many of the cases for which polynomial time allocation algorithms were presented in [23, 26]. It should be emphasized though that unlike the previous allocation algorithms, here one does not need to limit the bidding language in advance to allow only these types of bids. Instead, arbitrary bids may be allowed, and if the bids happen to conform to one of the categories mentioned below, an optimal solution is guaranteed.

Below we list several classes of auctions where we can prove that the LP solution is integral, and hence provides an optimal allocation directly. We roughly divide these classes into three types of situations, according to the basic combinatorial reason for the integrality of the solution.

6.1 Order

Linear Order Bids

The sets of items can be linearly ordered $G = \{g_1 ... g_n\}$ such that all bids S are for a consecutive sub-range $S = \{g_k ... g_l\}$ of items [23].

Lemma 6.1 *If the auction has only Linear Order bids then the LP solution provides an optimal allocation.*

The correctness of this lemma is based on the well-known "consecutive 1's" property that ensures integral solutions for linear problems. Since the proof in this case is simple we bring it here.

Proof: We will consider the best fractional allocation in an auction with linear order bids, and prove that the best fractional allocation is integral. The LP program finds the best fractional allocation, hence its solution is integral.

The proof is by induction on the number of items, m. the basis, $m = 1$, is trivial, so let us assume the lemma for less than m items, and prove it for the case of m items. For each $j = 1..m$, Denote the value of the optimal fractional allocation of the subset $\{g_1 ... g_j\}$ by v_j. Consider the optimal fractional allocation of all m goods, and look at all allocated sets that include g_m. There can be at most m such sets (according to the starting point in the linear order), and let us denote that for each j the bid $(\{g_j ... g_m\}, p_j)$ was allocated to a fraction of f_j (where $\sum_j f_j \leq 1$). It is clear that the value of this fractional allocation is bounded from above by $\sum_{j=1}^{m} f_j \cdot (p_j + v_{j-1})$. This is a weighted average, and let j^* denote the j with the largest value of $p_j + v_{j-1}$. But this value, which is at least as large as the value of the optimal fractional allocation can be achieved by an integral allocation that lets the bid $(\{g_{j^*} ... g_m\}, p_{j^*})$ win, and allocates the first $j^* - 1$ items according to the best allocation – which is integral. ♠

An important special case is the following type of bids:

Hierarchical Bids

The sets of goods in all bids form a nested hierarchy, i.e., for every two subsets of items S, T that appear as part of any bid, we have that either they are disjoint or that one contains the other [23].

Corollary 6.2 *If the auction has only Hierarchical bids then the LP solution provides an optimal allocation.*

These type of bids may be generalized as follows: assume now that the items are somehow organized as vertices in a tree, and that each bid may be for a connected sub-tree of the tree. One may verify that such auctions are also optimally solved by the LP solution.

6.2 Mutual Exclusion

OR-of-XORs of Singletons Bids

The bid was presented in the OR-of-XORs language, where each atomic bid was for a singleton set (and was later translated to the OR* language in the standard way.)

Lemma 6.3 *If the auction has only OR-of-XORs of singletons bids then the LP solution provides an optimal allocation.*

Proof: When OR-of-XORs of singleton bids are converted to OR* bids, each atomic bid contains one non-phantom item and at most one phantom item. We can view all atomic bids as edges in a bipartite graph of phantom vs. real items. The allocation goal in these terms is to find the best fractional matching in the graph, which is well known to be integral [11]. ♠

Two important special cases are exhibited below.

Single Item Bids

Each bidder can only win a single item at most. I.e. all atomic bids that are bid by bidder i are of the form $\{j, g_i\}$, where j is some item and g_i is a unique phantom item for bidder i.

Corollary 6.4 *If an auction has only single item bids then the LP solution provides an optimal allocation.*

Downward Slopping Symmetric Bids

Each bidder values all items as if they are identical, and has a sequence of valuations $p_1 \geq p_2 \geq ... \geq p_m$, where p_j specifies his valuation of the j'th item he wins.

The proof of lemma 3.4 shows that downward sloping symmetric bids can be represented as OR-of-XORs of singletons. We thus have:

Corollary 6.5 *If the auction has only downward sloping symmetric bids then the LP solution provides an optimal allocation.*

6.3 Substructure

In this subsection we show how auctions that have integral LP solutions can be combined and modified while still retaining this property.

Definition 7 *We say that an auction is a direct sum of auctions over the partition of items Q, R, if all valuation functions v can be represented as the sum of valuations over these subsets. I.e. for all i and S, $v_i(S) = v_i^Q(S \cap Q) + v_i^R(S \cap R)$. The functions v_i^Q and v_i^R are called the projected valuation functions.*

It is easy to verify that the direct sum of two auctions that each have integral LP solutions, still has this property.

Proposition 6.6 *If the auction is a direct sum over a partition Q, R, and the projected auctions on Q and R each have an integral solution of their respective linear programs, then the LP solution of the combined auction provides an optimal allocation.*

Definition 8 *A bid (S_i, p_i) in an auction is said to be majorized by the other bids if $p_i < \sum_{j \in S_i} y_j$, where the y_j's are the solution of the dual linear program.*

It is easy to see that majorized bids "do not matter".

Proposition 6.7 *If the auction obtained after removing the non-majorized bids has an integral LP solution then the LP solution of the whole auction also provides an optimal allocation.*

7 Handling a non Integer solution

Although, as we have seen, in many cases the LP relaxation will provide integer results and thus yield optimal allocations, in other cases this will not turn out to be the case. In these situations we must transform the LP solution, somehow, to an integer allocation. Due to the NP-completeness of finding the perfect allocation, this can not be efficiently done in an optimal way. In fact, even finding an approximation to the optimal solution is known to be NP-complete. So we must rely on heuristics, and we are faced with two options:

1. Introduce a heuristic that is provably computationally efficient. In this case, however, we will not be able to guarantee that it always produces the optimal allocation.

2. Introduce a heuristic that provably produces the optimal allocation. In this case, we will not be able to guarantee that it always runs in polynomial time.

In the next two subsections we will present heuristics taking each of these directions. Both of these substitutes take approaches that are rather standard in the combinatorial optimization community, and that are commonly applied to integer programming problems. The variants proposed here, though, are intended specifically for combinatorial auctions and we believe that they will perform especially well in cases of real combinatorial auctions. As they are heuristics we cannot, of course, prove this last statement. The basic intuition behind this belief is that in real settings items will "almost" have individual item prices, and that in these cases the LP solution will "almost" provide an optimal allocation.

7.1 A Greedy algorithm

Once we believe that the dual solution produces some notion of item prices, we can see which bids should be allocated given these prices. The fractional allocation x produced by the linear program assigns non-zero weight $x_i > 0$ only to bids (S_i, p_i) that exactly match the sum of the single item prices of the bundle – the "fair price": $p_i = \sum_{j \in S_i} y_j$. All other bids offer less than the fair price and are not given any weight in the fractional allocation, $x_i = 0$. An important quantity to consider about such bids would thus be how far they are from the fair price: $p_i/(\sum_{j \in S_i} y_j)$.

One would naturally try to take bids that are as close as possible to the fair price, starting with the ones that offer it fully. Among these bids, a natural intuition is that the higher the value of x_i, the higher its contribution to the final allocation quality. With this in mind, we suggest the following greedy algorithm.

Greedy Allocation Algorithm

1. Run the LP relaxation obtaining a fractional allocation $x_1...x_n$, and the dual "prices" $y_1...y_m$.
2. Re-order the bids by decreasing value of $p_i/(\sum_{j \in S_i} y_j)$. Bids with $x_i \neq 0$ (and thus $p_i/(\sum_{j \in S_i} y_j) = 1$) are ordered by decreasing value of x_i.
3. $WinningBids \leftarrow \emptyset$, $AllocatedItems \leftarrow \emptyset$.
4. For $i = 1..n$ (in the new order), if $(S_i \cap AllocatedItems = \emptyset)$ then
   - Add i to $WinningBids$.
   - Add all elements in S_i to $AllocatedItems$.

7.2 A Branch-and-bound Allocation Algorithm

A well-known technique to trim the search space in exponential searches is the branch-and-bound technique. This technique requires a good upper bound on partial solution quality. The LP relaxation is such a good bound, and we suggest an algorithm based on it. We first presents two components of the branch and bound algorithm: an upper bound algorithm and a lower bound algorithm. We then present the recursive branch-and-bound algorithm that uses them.

Upper Bound Algorithm

1. Run the LP relaxation.
2. Return the value of the LP solution as the upper bound.

Lower Bound Algorithm

1. Run the previous greedy algorithm.
2. Return the value of the solution as a lower bound (with the actual solution attached).

Branch-n-Bound Allocation Algorithm

Parameter: *LowValue*. The lowest value solution that is of interest. If the algorithm cannot find a solution above this threshold then it returns with failure. At the top of the recursion, LowValue is initialized to 0, and in further calls it is set by the caller.

1. Obtain *UpperBound* via the Upper Bound algorithm.
2. If ($UpperBound \leq LowValue$) then return with failure.
3. Obtain *LowerBound* via the Lower Bound algorithm.
4. if ($LowerBound > LowValue$), then $LowValue \leftarrow LowerBound$, (and remember the solution itself).
5. Let (S, p) be the first bid *according to the order specified in the greedy algorithm.*
6. Try Letting (S, p) win. I.e:
   - Remove the bid (S, p) and all items in S from the auction. Remove all other bids that contained any element of S.
   - Recursively find a solution using the Branch-n-Bound algorithm, with *LowValue* decreased by p.
   - If the recursive call returned with success, then update *LowValue*, by adding p to the returned solution value (and remember the solution itself).
7. Try letting (S, p) loose. I.e:
   - Remove the bid (S, p) from the original auction, but do not remove the items in S.
   - Recursively find a solution using the Branch-n-Bound algorithm, with current *LowValue*.
   - If the recursive call returned with success, then update *LowValue* to be the returned solution value (and remember the solution itself).
8. If *LowValue* was updated at any point, then return the solution with success; else return with failure.

The correctness of this algorithm is trivial:

Proposition 7.1 *This algorithm always returns the optimal allocation.*

The efficiency of this algorithm follows from the following, informal, analysis: This algorithm first explores the most promising directions as is done in the greedy algorithm. This will hopefully provide very good lower bounds quickly. We expect that the upper bounds obtained using the LP relaxation will usually be close enough to the truth. Together with good lower bounds, this should suffice to severely trim further search.

Perhaps the highest computational cost in this algorithm is the repeated solution of a linear program. A very large number of variants and optimizations to this algorithm may be proposed which help reduce this cost.

Optimizations:

1. The correctness of the algorithm does not depend on the Upper Bound algorithm providing an optimal solution to the LP. Any upper bound for the LP solution, in particular, any feasible solution of the dual problem, suffices for this. Many LP algorithms, in particular primal/dual ones, can obtain high-quality upper-bounds much faster than the true solution. Specifically, whenever an upper bound that is smaller than $LowValue$ is reached during the LP solution process, the LP algorithm can be immediately terminated.

2. If the goal of finding an optimal allocation is relaxed to ϵ-optimality (i.e. finding a solution whose value is within a factor of $1+\epsilon$ from optimal), then the test in step 2 can be relaxed to: $UpperBound < LowValue \cdot (1+\epsilon)$. This may provide additional trimming of the search space

3. Memoization (i.e. storing the solutions of previously solved sub-problems in a hash-table, and later recalling them), can eliminate the repeated solution of the same sub-problem that occurs here. This is just a well-known implicit form of dynamic programming, and has the usual price in space. A clever tradeoff would be to selectively store only "heavy" sub problems – e.g. ones where the sub problem solution required a very wide search.

4. The LP that is solved each time is rather similar to previous ones solved. Many LP algorithms will run faster if they start with a previous solution, which is not optimal (or even feasible) but that may turn out to be close to such.

8 Acknowledgments

I thank Amir Ronen, Ori Regev, Danniel Lehman Dov Monderer, Motty Perry and Moshe Tennenholtz for helpful discussions.

References

[1] ebay. Web Page: http://www.ebay.com.

[2] Moai. Web Page: http://www.moai.com.

[3] Arne Andersson, Mattias Tenhunen, and Fredrik Ygge. Integer programming for combinatorial auction winner determination. In *ICMAS*, 2000.

[4] Sushil Bikhchandani and John W. Mamer. Competitive equilibrium in an exchange economy with indivisibilities. *Journal of economic theory*, 74:385 – 413, 1997.

[5] E. H. Clarke. Multipart pricing of public goods. *Public Choice*, pages 17–33, 1971.

[6] Yuzo Fujishima, Kevin Leyton-Brown, and Yoav Shoham. Taming the computational complexity of combinatorial auctions: Optimal and approximate approaches. In *Proceedings of IJCAI'99*, Stockholm, Sweden, July 1999. Morgan Kaufmann.

[7] T. Groves. Incentives in teams. *Econometrica*, pages 617–631, 1973.

[8] R. M. Harstad, Rothkopf M. H., and Pekec A. Computationally manageable combinatorial auctions. Technical Report 95-09, DIMACS, Rutgers university, 1995.

[9] A.A. Lazar and N. Semret. The progressive second price auction mechanism for network resource sharing. In *8th International Symposium on Dynamic Games*, Maastricht, The Netherlands, July 1998.

[10] Daniel Lehmann, Liadan Ita O'Callaghan, and Yoav Shoham. Truth revelation in rapid, approximately efficient combinatorial auctions. In *1st ACM conference on electronic commerce*, 1999.

[11] L. Lovasz and M.D. Plummer. *Matching Theory*. North Holland, 1986.

[12] A. Mas-Collel, W. Whinston, and J. Green. *Microeconomic Theory*. Oxford university press, 1995.

[13] R. Preston McAfee and John McMillan. Analyzing the airwaves auction. *Journal of economic perspecives*, 10(1):159 – 175, 1996.

[14] P. Milgrom. *Auction Theory for Privatization*. Cambridge university press, 1998.

[15] Paul Milgrom. Auctions and bidding: a primer. *Journal of economic perspectives*, 3(3):3 – 22, 1989.

[16] Paul Milgrom. Putting auction theory to work: the simultaneous ascending auction. Technical Report Working Paper 98-002, Dept. of Economics, Stanford University, 1998.

[17] Market design inc. Web Page: http://www.market-design.com.

[18] Noam Nisan and Amir Ronen. Computationally feasible vcg-based mechanisms. Manuscript.

[19] Noam Nisan and Amir Ronen. Algorithmic mechanism design. In *STOC*, 1999. Avilable at http://www.cs.huji.ac.il/~amiry.

[20] M. J. Osborne and A. Rubistein. *A Course in Game Theory*. MIT press, 1994.

[21] Forrester research. Dynamic trade research brief. Web Page: http://www.forrester.com/ER/Research/Brief/0,1317,1481,FF.html.

[22] Jeffrey S. Rosenschein and Gilad Zlotkin. *Rules of Encounter: Designing Conventions for Automated Negotiation Among Computers*. MIT Press, 1994.

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

[24] Tuomas Sandholm. An algorithm for optimal winner determination in combinatorial auctions. In *IJCAI-99*, 1999.

[25] Tuomas W. Sandholm. Limitations of the vickrey auction in computational multiagent systems. In *Proceedings of the Second International Conference on Multiagent Systems (ICMAS-96)*, pages 299–306, Keihanna Plaza, Kyoto, Japan, December 1996.

[26] Moshe Tennenholtz. Some tractable combinatorial auctions. Technical report, 2000.

[27] W. Vickrey. Counterspeculation, auctions and competitive sealed tenders. *Journal of Finance*, pages 8–37, 1961.

[28] W.E. Walsh, M.P. Wellman, P.R. Wurman, and J.K. MacKie-Mason. Auction protocols for decentralized scheduling. In *Proceedings of The Eighteenth International Conference on Distributed Computing Systems (ICDCS-98)*, Amsterdam, The Netherlands, 1998.

[29] Peter R. Wurman. Market structure and multidimensional auction design for computational economies, ph.d. dissertation, 1999.