Winner Determination in Combinatorial Auctions with Logic-based Bidding Languages

(Short Paper)

Joel Uckelman*
Institute for Logic, Language and Computation
University of Amsterdam
juckelma@illc.uva.nl

Ulle Endriss
Institute for Logic, Language and Computation
University of Amsterdam
ulle@illc.uva.nl

ABSTRACT

We propose the use of logic-based preference representation languages based on weighted propositional formulas for specifying bids in a combinatorial auction. We then develop several heuristics for a branch-and-bound search algorithm for determining the winning bids in this framework and report on their empirical performance. The logic-based approach is attractive due to its high degree of flexibility in designing a range of different bidding languages within a single conceptual framework.

Categories and Subject Descriptors

F.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems; I.2.4 [**Computing Methodologies**]: Artificial Intelligence—*Knowledge Representation Formalisms and Methods*

General Terms

Algorithms, Experimentation, Economics

Keywords

Combinatorial Auctions, Preference Representation

1. INTRODUCTION

Combinatorial auctions are auctions in which the auctioneer is offering not just one but a whole set of goods for sale. Potential buyers can make bids for different subsets of this set of goods. The so-called *winner determination problem* faced by the auctioneer is then the problem of accepting a collection of bids that will maximize the sum of the prices offered, such that each good is sold at most once [4].

Bidding amounts to informing the auctioneer of one's valuation for the goods on auction. Early work on combinatorial auctions typically ignored this aspect and made *ad hoc* assumptions on how a bidder can transmit their bid to the auctioneer. But recently, several *bidding languages* with formal syntax and semantics have been proposed that allow bidders to describe their valuations in a uniform way [11, 9, 2]. For instance, if the OR-language is used then each bidder can submit any number of atomic bids consisting of a bundle of goods and a price, the auctioneer can accept any number of atomic bids such that the corresponding bundles do not overlap, and each bidder pays the sum of the prices of their accepted atomic bids. Another example is the XOR-language, where the auctioneer can accept at most one atomic bid per bidder. In this paper, we propose to use a particular class of logic-based preference representation languages as bidding languages. The basic idea is to identify goods with propositional variables and to let bidders express their goals in terms of propositional formulas over these variables. Each such goal is paired with a numerical weight, and we compute the value assigned to a bundle of goods by a bidder by summing up the weights of the goals that are "satisfied" by that bundle. These languages go back to early work on *penalty logic* [10] and have been studied in depth in the field of knowledge representation [7, 3]. By putting varying restrictions on the range of permissible (syntactic) forms of formulas, we can define bidding languages of varying expressive power. This is attractive because it allows us to design languages of the right expressive power for the application domain at hand.

Selecting a language which has the appropriate expressiveness for the domain is important: Too expressive and the WDP becomes very difficult, too inexpressive and bidders will be unable to provide their true valuations. The appeal of logic-based bidding languages is clear, then, as we can adjust the expressivity by selecting which formulas (or weights) are permitted. Furthermore, the succinctness of logic-based languages varies widely, so if concise representations are desired, a language having them may be available.

Our aim in this paper is to develop algorithms for the winner determination problem for combinatorial auctions where bids are expressed in terms of these logic-based languages. Our general approach follows ideas of Sandholm [11], Fujishima et al. [5], and others. Having to decide for each good which bidder to assign it to gives rise to a (very large) search tree, and we show how to define heuristics for pruning this search tree without removing the optimal solution(s). Concretely, we use a branch-and-bound (B&B) algorithm, although the same heuristics could also be used, for instance,

*This research was supported by a Marie Curie Early Stage Research fellowship from the GloRiClass project (MEST-CT-2005-020841).

Cite as: Winner Determination in Combinatorial Auctions with Logic-based Bidding Languages (Short Paper), Joel Uckelman and Ulle Endriss, *Proc. of 7th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2008)*, Padgham, Parkes, Müller and Parsons (eds.), May, 12-16., 2008, Estoril, Portugal, pp. XXX-XXX.

in an approach based on the A* search algorithm.

The remainder of this paper is organized as follows. Section 2 introduces the formal background of our logic-based bidding languages. In Section 3 we define the winner determination problem and outline the general framework for our algorithms. We report some of our results in Section 4. These results consist of the proposal for a B&B heuristic for the specific bidding language we consider here, followed by the use of a secondary heuristic guiding the order in which we explore branches. We then report on experimental results documenting the empirical performance of our algorithms.

2. LOGIC-BASED BIDDING LANGUAGES

We adopt the notation of Chevaleyre et al. [3]. Fix a finite set $\mathcal{PS}$ of propositional variables and let $\mathcal{L}_{\mathcal{PS}}$ denote the language of propositional logic over $\mathcal{PS}$. Each $p \in \mathcal{PS}$ represents one of the goods on auction. Each bidder has a *valuation* function $v : 2^{\mathcal{PS}} \to \mathbb{R}$ to model their preferences over alternative bundles of goods. Bidders can use formulas of $\mathcal{L}_{\mathcal{PS}}$ to express *goals*. For instance, $p_1 \wedge p_6$ expresses that our bidder would like to obtain goods p_1 and p_6 (*together*—each item on its own may represent no value at all), while $\neg p_2$ says that they would rather not get p_2. If $M \subseteq \mathcal{PS}$ is the set of goods obtained by a particular bidder, then we can also think of M as a *model* that will satisfy some of these formulas and falsify others. For instance, we have $M \models p_1 \wedge p_6$ and $M \not\models \neg p_2$ for $M = \{p_1, p_2, p_4, p_6\}$. A *weighted goal* is a pair (φ, w), where φ is a formula and $w \in \mathbb{R}$. A *goal base* $G = \{(\varphi_i, w_i)\}_i$ is a set of such weighted goals. Each φ_i is required to be satisfiable, and no (syntactically distinct) formula may appear more than once in G. G generates a valuation function over sets of goods (models) M as follows:

$$v(M) \quad = \quad \sum \{w_i \mid (\varphi_i, w_i) \in G \text{ and } M \models \varphi_i\}$$

That is, we obtain the valuation of M by adding up all the weights of the goals that are satisfied by M.

Let $H \subseteq \mathcal{L}_{\mathcal{PS}}$ be a syntactical restriction on formulas and $H' \subseteq \mathbb{R}$ a set of allowed weights. Then $\mathcal{L}(H, H')$ is defined as the bidding language given by the class of goal bases satisfying the restrictions H and H'. Here, we work with the language of $\mathcal{L}(\mathit{pcubes}, \mathit{pos})$, the language of positive cubes (conjunctions of positive literals) with positive weights. This language has similar (though not the same) expressivity as a language proposed by Hoos and Boutilier [6].

3. WINNER DETERMINATION

In this section we first introduce some notation and define the winner determination problem (WDP) for combinatorial auctions when bids are represented using weighted propositional formulas. We then outline our generic framework for B&B algorithms to solve the WDP.

3.1 Notation

Let $\mathcal{A}$ be the set of agents bidding in any given auction. Each agent $i \in \mathcal{A}$ has got a goal base G_i defining their valuation over the goods in $\mathcal{PS}$. An *allocation* $A : \mathcal{PS} \to \mathcal{A} \cup \{*\}$ is a function which maps goods to the agents to which they are given. We write $A(p) = *$ when A leaves good p unallocated, and in that case A is a (strictly) *partial* allocation. If A allocates all goods in $\mathcal{PS}$, then A is a *complete* allocation. The set $\text{und}(A) = \{p \in \mathcal{PS} \mid A(p) = *\}$ is the set of unallocated goods in allocation A, and we write $\text{und}(A, \varphi)$ for the set of unallocated goods appearing as propositional variables in the formula φ.

Next we introduce the notion of a (partial) allocation satisfying a given goal of a bidder. Define M_i^A as the set of goods assigned to bidder i in allocation A. As explained in Section 2, M_i^A defines a model for formulas of the language $\mathcal{L}_{\mathcal{PS}}$: a propositional variable $p \in \mathcal{PS}$ is *true* iff $p \in M_i^A$. We write $M_i^A \models \varphi$ if the goal φ is satisfied in M_i^A. Furthermore, we write $M_i^A \,?\, \varphi$ iff $M_i^A \not\models \varphi$ and $M_i^A \cup S \models \varphi$ for some set $S \subseteq \text{und}(A)$. That is, φ is a goal of agent i that is not (yet) satisfied in allocation A, but that could still be satisfied if we allocate more items.

3.2 The Winner Determination Problem

Intuitively, the WDP is the problem of deciding which goods to allocate to which bidder in such a way that maximizes the sum of the weights associated with the goals satisfied by the chosen allocation. To make this precise, we define the *social welfare* of an allocation A as follows:

$$sw(A) \quad = \quad \sum_{i \in \mathcal{A}} \sum_{\substack{(\varphi, w) \in G_i \\ M_i^A \models \varphi}} w$$

Then the WDP is the optimization problem of finding a complete allocation A maximizing $sw(A)$. By restricting attention to complete allocations we are defining a WDP *without free disposal*. If desired, we can easily model auctions with free disposal by adding a single bidder with an empty goal base to any given auction instance.

3.3 Winner Determination Algorithms

A *brute force* algorithm for solving the WDP enumerates all complete allocations, computes the social welfare for each, and picks the one with the highest value. Naturally, such an approach will not scale. The most common approach is to use Integer Programming, as done by Boutilier [1] for one logic-based language [2]. Instead, we use a B&B algorithm, similar to the work of Sandholm [11] and Fujishima et al. [5]. B&B consists of two parts: A procedure for *branching*—that is, dividing the solution space—and a procedure for *bounding* the quality of solutions contained on any branch.

Our search space is organized as follows. Initially, all goods are unallocated. We pick one of the unallocated goods and select an agent in $\mathcal{A}$ to receive it, and repeat until all items have been allocated. This gives rise to a large search tree, the depth of which is given by the number of goods on auction, and for which each internal node has $|\mathcal{A}|$ children. Each internal node corresponds to a strictly partial allocation, while the leaf nodes represent complete allocations.

Following standard conventions, we introduce two functions, g and h, mapping nodes (allocations) in the search tree to $\mathbb{R}$. The function g computes the social welfare already derived, *i.e.*, it is simply defined by $g(A) = sw(A)$. The *heuristic function* h estimates the additional social welfare achievable by allocating the remaining goods. B&B explores the search space as follows. We start with an initial tree consisting of a single node where no goods have been allocated yet. We maintain a *frontier* of leaf nodes and a pointer to the current *top allocation* A^* delivering the highest social welfare so far. The algorithm then repeatedly applies the following steps:

1. Select a node (partial allocation) A from the frontier that still has a chance of beating the current top allo-

cation A^*: $g(A^*) < g(A) + h(A)$. Any A not meeting this condition can be removed from the frontier.

2. Select a good not yet allocated in A: $p \in \text{und}(A)$.

3. Produce as children of A all allocations A' which extend A by allocating p. Thus each expanded node will have one child for each bidder in $\mathcal{A}$. Add all children to the frontier (and remove A from it).

We stop when there are no more viable partial allocations in the frontier to choose from (during step 1). As a solution we return (one of) the best (by now complete) allocations in the final frontier.

A heuristic function h is *admissible* iff it never underestimates the remaining social welfare still achievable: that is, iff $g(A) + h(A) \geq sw(A')$ for all partial allocations A and all complete allocations A'. As is well known (and easy to see), if h is admissible, then B&B is guaranteed to find an optimal solution.

When designing a winner determination algorithm for a particular bidding language within the general framework above, there are three choices to make: (*i*) the B&B heuristic h, (*ii*) a heuristic to decide which node to expand next (the *expansion policy*), and (*iii*) a heuristic to decide which good p to allocate next (the *branching policy*). In subsequent sections, we discuss several options for the B&B heuristic and for the branching policy (we have not experimented with different expansion polices, which we expect to be the least important factor in designing a WDP algorithm).

4. HEURISTICS FOR POSITIVE CUBES

In this section we develop heuristics for the B&B algorithm for the WDP using the bidding language $\mathcal{L}(pcubes, pos)$, which is based on positive cubes with positive weights. Intuitively, the weight given to a cube of the form $p_1 \wedge \cdots \wedge p_m$ may be regarded as the marginal utility associated with obtaining all of $p_1, \ldots, p_m$—beyond the utility associated with any subset of these.

4.1 A Branch-and-Bound Heuristic

We now define our upper-bound heuristic for $\mathcal{L}(pcubes, pos)$:

DEFINITION 1. *Define the heuristic function $h^+_\wedge$ as*

$$h^+_\wedge(A) = \sum_{p \in \mathcal{PS}} h^p(A) \quad \textit{where}$$

$$h^p(A) = \max_{i \in \mathcal{A}} h^p_i(A)$$

$$h^p_i(A) = \sum_{(\varphi, w) \in G_i} h^p_i(A, \varphi)$$

$$h^p_i(A, \varphi) = \begin{cases} \frac{w}{|\text{und}(A,\varphi)|} & \textit{if } (\varphi, w) \in G_i, \\ & p \in \text{und}(A, \varphi), M^A_i \ ? \ \varphi \\ 0 & \textit{otherwise} \end{cases}$$

The intuition embodied here is that we can estimate the value of an item for an agent by assigning to each item a share of the weight of each positive cube in which it appears. For example, suppose agent 1 bids $\{(a \wedge b, 6), (a \wedge c, 8)\}$, and agent 2 bids $\{(a, 6), (b \wedge c, 10)\}$. Under the empty partial allocation $\emptyset$, for agent 1 we have that $h^a_1(\emptyset) = \frac{6}{2} + \frac{8}{2} = 7$, $h^b_1(\emptyset) = \frac{6}{2} = 3$, and $h^c_1(\emptyset) = \frac{8}{2} = 4$. For agent 2, we have $h^a_2(\emptyset) = 6$, $h^b_2(\emptyset) = \frac{10}{2} = 5$, and $h^c_2(\emptyset) = \frac{10}{2} = 5$. Since each item may be allocated to only one agent, the atom-wise components of the heuristic "award" each item to the agent for whom that item contributes most: $h^a(\emptyset) = 7$ since $h^a_1(\emptyset) > h^a_2(\emptyset)$; $h^b(\emptyset) = 5$ since $h^b_2(\emptyset) > h^b_1(\emptyset)$; and $h^c(\emptyset) = 5$ since $h^c_2(\emptyset) > h^c_1(\emptyset)$. The upper-bound $h^+_\wedge(\emptyset)$ is the sum of the maximum contributions of the atoms: $h^+_\wedge(\emptyset) = 7 + 5 + 5 = 17$. Notice that in this case the optimal value is 16, which is attained when agent 2 receives all three items; the heuristic overestimates the optimal value to be 17 instead.

For lack of space, we omit the proof of the following result:

PROPOSITION 1. *The heuristic $h^+_\wedge$ is admissible for $\mathcal{L}(pcubes, pos)$.*

4.2 Two Branching Policies

Let $\mathfrak{A}$ be the set of strictly partial allocations. A function $b : \mathfrak{A} \to \mathcal{PS}$ is a *branching policy* if for all strictly partial allocations A, $b(A) = p$ for some p which A does not allocate.

DEFINITION 2. *We define two branching policies:*

- *The* lexical *branching policy is the branching policy b such that $b(A) = p$, where p is the lexically least good not allocated by partial allocation A.*
- *The* best-estimate first *branching policy is the branching policy b such that $b(A) = p$, where p is the lexically least good such that $h^p(A) = \max_{a \in \mathcal{PS}} h^a(A)$, where $h^p(A)$ is as in Definition 1.*

Each B&B solver used in our experiments implements one of these branching policies. Both are clearly correct, since neither rules out reaching any particular complete allocation.

4.3 Experimental Results[1]

For $\mathcal{L}(pcubes, pos)$ and a fixed number of goods m, we generated goal bases as follows: For each agent, we randomly chose an integer in $[1, 2m]$ to be the number of formulas in that agent's goal base, with each potential atom appearing in a given positive cube with probability 0.5. (This makes positive cubes of middling length more probable than very short or very long ones.) Each formula was given a random integer weight uniformly chosen from $[1, 10]$. The solvers used for $\mathcal{L}(pcubes, pos)$ were:

`BruteForce` A brute-force search. Iterates over all complete allocations, returning the lexically first optimal one.

`PCubeLex` A B&B solver, using the upper-bound heuristic $h^+_\wedge$ and the lexical branching policy.

`PCubeBF` A B&B solver, using the upper-bound heuristic $h^+_\wedge$ and the best-estimate-first branching policy.

Our first experiment tested the `BruteForce` solver against `PCubeLex` to establish a baseline for comparison. As expected, the size of the solution space rapidly overwhelms the `BruteForce` solver. Instances of size $(8, 8)$ (8 agents, 8 goods) which take nearly 54 seconds to solve by `BruteForce` can be solved by `PCubeLex` in less than 0.01 seconds.

[1]All experiments reported here were conducted on a Fedora 7 Linux system (kernel 2.6.23) with a 2GHz Intel Core 2 Duo T7300 CPU and 2GB of RAM, using Sun's Java SE JVM (version 1.6.0_02).

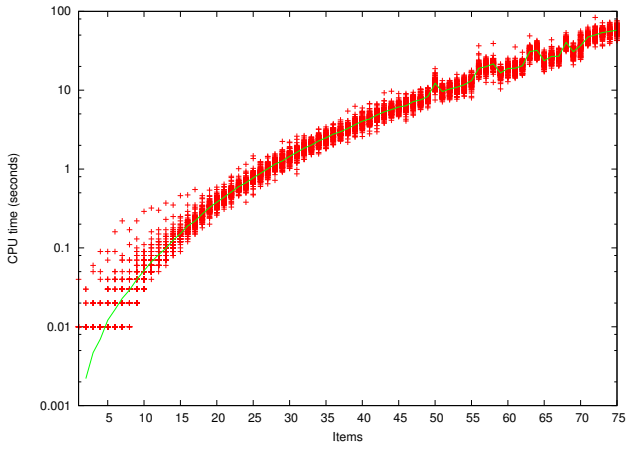

Figure 1: CPU time for WDP instances with 20 agents in $\mathcal{L}(\boldsymbol{pcubes}, \boldsymbol{pos})$ using the `PCubeBF` solver

Our second experiment tested the B&B solvers to see how many partial allocations (nodes) were created for each WDP instance. The number of nodes created is a useful measure of efficiency for a B&B solver. The worst case is building every partial allocation, a complete n-ary tree of depth m, for n agents and m goods. We found that `PCubeLex` is quite parsimonious in building nodes. At $(20, 20)$, `PCubeLex` builds (on average) only 401 of the 5.5×10^{24} possible nodes; even at $(20, 70)$, most instances generate around 1400 nodes.

Our third experiment tested the effect of varying the branching policy on solver runtime. Intuitively, one would expect that using a best-estimate-first branching policy would produce better performance than a lexical branching policy when using an upper-bound heuristic which is inexact. However, for `PCubeLex` and `PCubeBF` this turned out not to be the case—runtime saved in other parts of the solver by calculating which good to branch on next was usually consumed by the calculation itself, and so seldom was any gain realized this way.

In our fourth experiment, we fixed the number of agents at 20 and varied the number of goods from 1 to 75, and solved 100 randomly-generated (as above) WDP instances of each size using `PCubeBF`, the results of which can be seen in Figure 1. Due to our method of randomly generating goal bases, the number of atomic bids (*i.e.*, weighted formulas) present in any WDP instance is around $|\mathcal{A}| \cdot |\mathcal{PS}|$. For example, the average instance with 20 agents and 75 items contains around 1500 atomic bids. `PCubeBF` is capable of solving problems with nearly one hundred items and thousands of bids in under one minute.

5. CONCLUSIONS

Logic-based bidding languages, where bidders submit weighted propositional formulas over the names of the goods on auction to encode their valuations (*i.e.*, the prices they are offering for different bundles), allow bidders to specify the goals they wish to see satisfied (and their relative importance) in a natural manner. Furthermore, by putting restrictions of the syntax of formulas and the range of weights to be used, the expressive power and representational succinctness can be tailored to the problem domain at hand. In this paper, we have developed heuristics for use in branch-and-bound algorithms for solving the winner determination problem in combinatorial auctions when different such languages are used to encode bids.

Our experimental for results the language of positively-weighted positive cubes are encouraging. Considering that our implementation has been developed with a view of being able to conveniently specify and test different heuristics for different languages, significant improvements can be expected from future re-implementation. We stress that the language of positive cubes is very attractive for combinatorial auctions. It allows agents to specify their marginal valuations for obtaining a certain set of goods together, beyond the sum of the values associated with each of its subsets.

In the future, we plan to test our algorithms with more realistic auction instances. Unfortunately, work to date on generating such data, in particular the CATS system [8], cannot immediately be used for our purposes due to the differences between the XOR-language (used by CATS) and logic-based languages. As argued by Boutilier [1], a translation is possible in principle, but it is not clear how it would affect the hardness of an auction instance, nor whether a translated instance would still be a realistic instance.

6. REFERENCES

[1] C. Boutilier. Solving concisely expressed combinatorial auction problems. In *Proc. 19th National Conference on Artif. Intell.*, 2002.

[2] C. Boutilier and H. H. Hoos. Bidding languages for combinatorial auctions. In *Proc. 17th Intl. Joint Conference on Artif. Intell.*, 2001.

[3] Y. Chevaleyre, U. Endriss, and J. Lang. Expressive power of weighted propositional formulas for cardinal preference modelling. In *Proc. 10th Intl. Conference on Principles of Knowledge Representation and Reasoning*, 2006.

[4] P. Cramton, Y. Shoham, and R. Steinberg, editors. *Combinatorial Auctions*. MIT Press, 2006.

[5] Y. Fujishima, K. Leyton-Brown, and Y. Shoham. Taming the computational complexity of combinatorial auctions: Optimal and approximate approaches. In *Proc. 16th Intl. Joint Conference on Artif. Intell.*, 1999.

[6] H. H. Hoos and C. Boutilier. Solving combinatorial auctions using stochastic local search. In *Proc. 17th National Conference on Artif. Intell.*, 2000.

[7] J. Lang. Logical preference representation and combinatorial vote. *Annals of Mathematics and Artif. Intell.*, 42(1–3):37–71, 2004.

[8] K. Leyton-Brown, M. Pearson, and Y. Shoham. Towards a universal test suite for combinatorial auction algorithms. In *Proc. 2nd ACM Conference on Electronic Commerce*, 2000.

[9] N. Nisan. Bidding languages for combinatorial auctions. In P. Cramton et al., editors, *Combinatorial Auctions*. MIT Press, 2006.

[10] G. Pinkas. Propositional nonmonotonic reasoning and inconsistency in symmetric neural networks. In *Proc. 12th Intl. Joint Conference on Artif. Intell.*, 1991.

[11] T. W. Sandholm. Algorithm for optimal winner determination in combinatorial auctions. *Artif. Intell.*, 135(1–2):1–54, 2002.