Winner Determination in Combinatorial Auctions with Logic-based Bidding Languages

Joel Uckelman and Ulle Endriss
Institute for Logic, Language and Computation, University of Amsterdam

❶ Synopsis

We use 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, 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.

❷ Combinatorial Auctions

Combinatorial auctions are auctions in which the auctioneer is offering not just one, but a whole set of goods, for sale simultaneously. Potential buyers can make bids for different subsets of this set of goods.

Specifying bids for all 2^n subsets of n goods is infeasible. Bidders need a *bidding language*, as in ❸.

Logic-based bidding languages are attractive because their expressivity and succinctness can be tailored to be appropriate for the particular auction setting in which they are being used.

❸ Weighted Formulas

- $\mathcal{PS}$ is a fixed, finite set of propositional variables.
- $\mathcal{L}_{\mathcal{PS}}$ is the language of propositional logic over $\mathcal{PS}$.
- Each $p \in \mathcal{PS}$ represents one of the goods on auction.
- Bidders use formulas of $\mathcal{L}_{\mathcal{PS}}$ to express *goals*. E.g., $p_1 \wedge p_6$ expresses that a bidder wants goods p_1 and p_6 (*together*—each item on its own may represent no value at all), while $\neg p_2$ says that she would rather not get p_2.
- The set of goods M obtained by a bidder is a propositional *model*, such that $M \models p$ iff $p \in M$. Example: $M \models p_1 \wedge p_6$ 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 weighted goals where every φ_i is satisfiable.
- A goal base G generates a valuation (unique) function over sets of goods (models) M:

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

That is, **M is the sum of the weights of the goals that are satisfied by M.**

❹ Winner Determination

The Winner Determination Problem (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.

The *social welfare* of an allocation A is

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

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*.

❺ Languages

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'.

Examples: $\mathcal{L}(pcubes, pos)$ is the language of positive cubes (conjunctions) with positive weights. $\mathcal{L}(clauses, all)$ is the language of clauses (disjunctions) with arbitrary weights. Many more languages are possible.

❻ Branch & Bound

Branch & Bound is a search method which always finds optimal allocations: Start with an single-node initial tree with a single node $A^* = \emptyset$ where no goods have been allocated yet. g is the attained value, h is an (over)estimate of the remaining value.

1. Select a node (partial allocation) A from the frontier that still has a chance of beating the current top allocation A^*:
$$g(A^*) < g(A) + h(A).$$
Remove any A not meeting this condition.
2. Select a good not yet allocated in A: $p \in \text{und}(A)$.
3. Build as children of A all allocations A' which extend A by allocating p. Add all children to the frontier (and remove A from it).

Stop when there are no more viable partial allocations in the frontier to choose from (during step 1). Return (one of) the best (by now complete) allocations in the final frontier.

B&B involves three design choices: the *branching policy*, the *expansion policy*, and the *upper bound heuristic*.

❼ Branching Policies

- *Lexical*: The next item $b(A) = p$, where p is the lexically least good not allocated by partial allocation A. Used by our `PCubeLex` solver.
- *Best-estimate first*: The next item $b(A) = p$, 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 ❽. Used by our `PCubeBF` solver.

❽ Heuristic for $\mathcal{L}(pcubes, pos)$

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

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

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

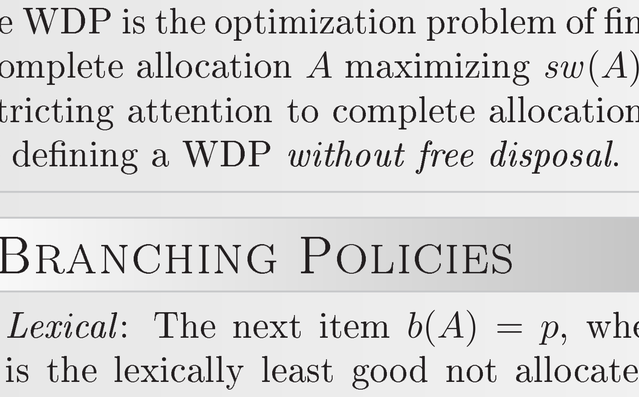
$$h^p_i(A, \varphi) = \begin{cases} \frac{w}{|\text{und}(A,\varphi)|} & \text{if } (\varphi, w) \in G_i, \\ & p \in \text{und}(A, \varphi), M_i^A \,?\, \varphi \\ 0 & \text{otherwise} \end{cases}$$

where $\mathcal{A}$ is the set of agents, A is a partial allocation, $\text{und}(A, \varphi)$ is the set of unallocated items from A occurring in φ, and M_i^A is the partial model defined by A from the point of view of agent i. $M_i^A \,?\, \varphi$ iff A leaves φ undecided.

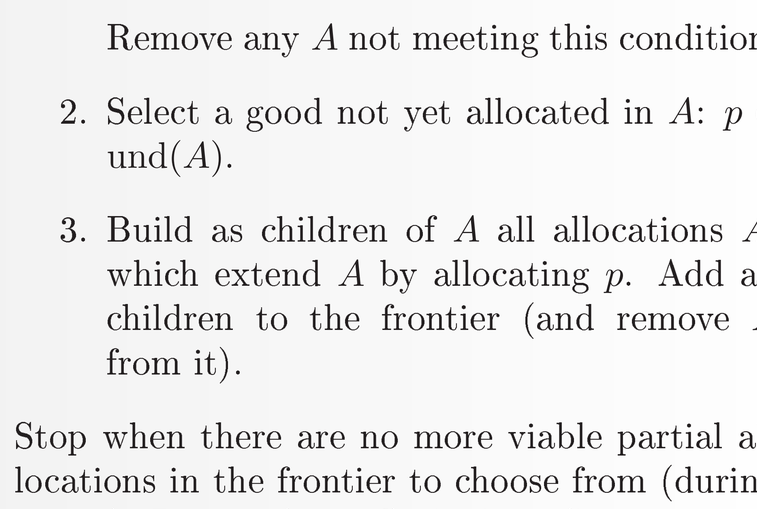
Intuition: **Divide the weight of a cube over its unallocated atoms.**

$h^+_\wedge$ is *correct*, as it is always an upper bound, but sacrifices *tightness* for *speed*.

❾ Selected Experimental Results for $\mathcal{L}(pcubes, pos)$



This plot shows the number of nodes built by `PCubeLex` with varying numbers of items and agents. The worst case is building every partial allocation, a complete n-ary tree of depth m, for n agents and m goods. At the extreme corner of the plot, $(20, 20)$, the full tree would have 5.5×10^{24} nodes. `PCubeLex` is quite parsimonious in building nodes.



This plot shows CPU time consumed by our Branch & Bound solver when using `PCubeBF` versus the number of items on auction, with the number agents fixed at 20. 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.