# Multiagent Coordination Using a Distributed Combinatorial Auction

**José M. Vidal**

Computer Science and Engineering
University of South Carolina, Columbia, SC 29208
vidal@sc.edu

## Abstract

COMBINATORIAL AUCTIONS are a great way to represent and solve distributed allocation problems. Unfortunately, most of the winner determination solutions that exists are centralized. They require all agents to send their bids to a centralized auctioneer who then determines the winners. The PAUSE auction, in contrast, is an increasing-price combinatorial auction in which the problem of winner determination is naturally distributed amongst the bidders. We present PAUSEBID, a bidding algorithm for agents in a PAUSE auction which always returns the bid that maximizes the bidder's utility. Our test results show that a system where all agents use PAUSEBID finds the revenue-maximizing solution at least 95% of the time. Run time, as expected since this is an NP-complete problem, remains exponential on the number of items.

## 1. Bidding in the PAUSE Auction

A PAUSE AUCTION for $m$ items has $m$ **stages**. Stage 1 consists of having simultaneous ascending price open-cry auctions for each individual item. During this stage the bidders can only place individual bids on items. At the end of this stage we know what is the highest bid for each individual item and who placed that bid. In each successive stage $k = 2,3,\ldots,m$ we hold an ascending price auction where the *bidders must submit sets of bids that cover all goods but each one of the bids must be for k goods or less*. The bidders are allowed to use bids that other agents have placed in previous rounds when placing their bid, thus allowing them to find better solutions. Also, any new bidset has to have a sum of bid prices which is bigger than the currently winning bidset. That is, revenue must increase monotonically.

Formally, let each **bid** $b$ be composed of $b^{\text{items}}$ which is the set of items the bid is over, $b^{\text{value}}$ the value or price of the bid, and $b^{\text{agent}}$ the agent that placed the bid. The agents maintain a set $B$ of the **current best bids**, one for each set of items of size $\leq k$. At any point in the auction, after the first round, there will also be a set $W \subseteq B$ of **currently winning bids**. This is the set of bids that currently maximizes the revenue, where the **revenue** of $W$ is given by

$$r(W) = \sum_{b \in W} b^{\text{value}}. \tag{1}$$

Agent $i$'s **value function** is given by $v_i : S \rightarrow \Re$ where $S$ is a subset of the items. Given an agent's value function and the current set of winning bids $W$ we can calculate the agent's **utility** from $W$ as

$$u_i(W) = \sum_{b \in W \,|\, b^{\text{agent}} = i} v_i(b^{\text{items}}) - b^{\text{value}}. \tag{2}$$

Given that $W$ is the current set of winning bids, agent $i$ must find a $g^*$ such that $r(g^*) \geq r(W) + \varepsilon$ and

$$g^* = \arg\max_{g \subseteq 2^B} u_i(g), \tag{3}$$

where each $g$ is a set of bids all taken from $B$ and $g$ covers all items. The goal of the PAUSEBID algorithm is to find this $g^*$.

## 2. Pausebid

PAUSEBID$(i,k)$

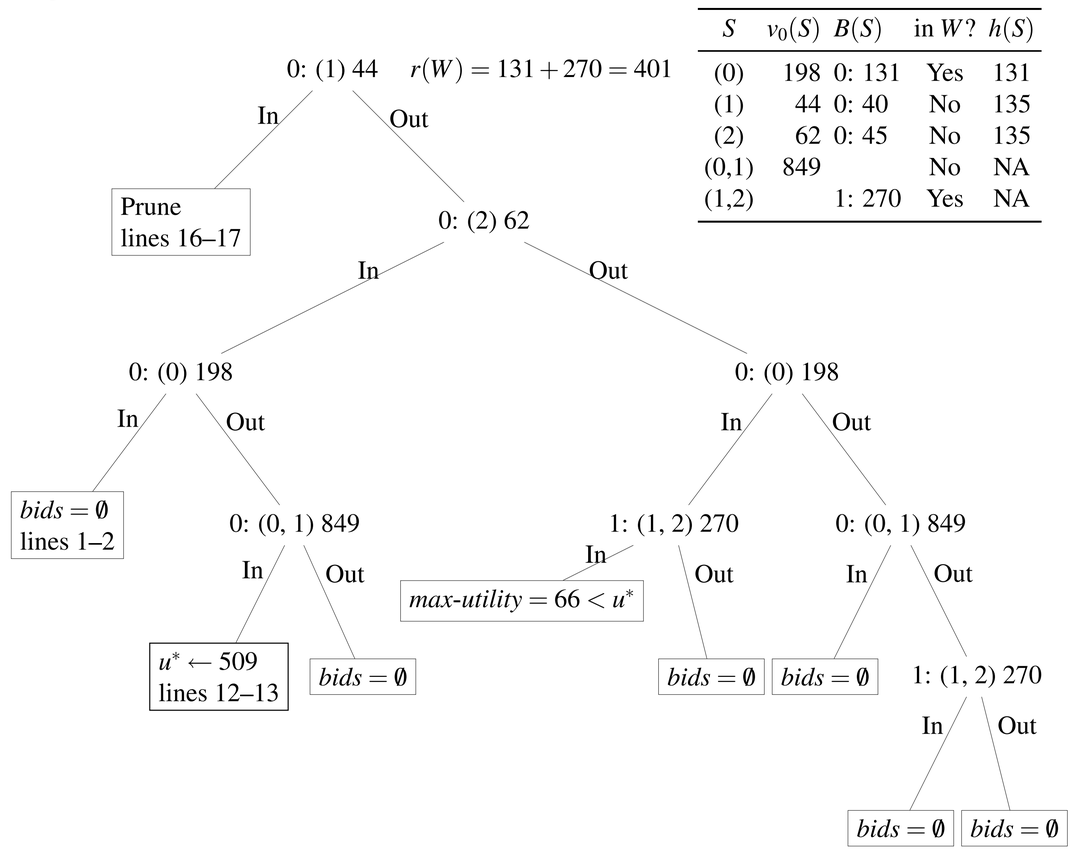
```
1  my-bids ← ∅    ▷ Will contain a bid for every S for which v_i(S) > corresponding bid in B.
2  their-bids ← ∅ ▷ Will contain rest of bids from B.
3  for b ∈ B
4       do if b^agent = i or v_i(b^items) > b^value
5            then my-bids ← my-bids + new Bid(i, b^items, v_i(b^items))
6            else their-bids ← their-bids + b
7  for S ∈ subsets of k or fewer items such that
           v_i(S) > 0 and ¬∃_{b∈B} b^items = S
8       do my-bids ← my-bids + new Bid(i, S, v_i(S)) ▷ Set my-bids to my valuation.
9  bids ← my-bids + their-bids   ▷ bids has best bid for every set, or my valuation if greater.
10 g* ← ∅                        ▷ Global variable.
11 u* ← u_i(W)                   ▷ Global variable.
12 h(S) ← max revenue on items from S given B, for all S.
13 PAUSEBIDSEARCH(bids, ∅)       ▷ The function that does the actual search.
14 surplus ← Σ_{b∈g* | b^agent=i} b^value − W(b^items)
15 if surplus = 0
16    then return g*
17 my-payment ← v_i(g*) − u*     ▷ How much I need to pay to make g* win.
18 for b ∈ g* | b^agent = i      ▷ Distribute surplus among my bids.
19      do if my-payment ≤ 0
20           then b^value ← 0
21           else b^value ← W(b^items) + my-payment · (b^value − W(b^items)) / surplus
22 return g*
```

**Figure 1:** *The* PAUSEBID *algorithm which implements a branch and bound search. i is the agent and k is the current stage of the auction, for* $k \geq 2$.

PAUSEBIDSEARCH$(bids, g)$

```
1  if bids = ∅                          ▷ A branch-and-bound algorithm.
2     then return
3  b ← first(bids)
4  bids ← bids − b
5  g ← g + b
6  if g does not contain a bid from i   ▷ None of my bids means no utility for me, so quit.
7     then return
8  if g includes all items              ▷ We are at a leaf node: feasible bidset.
9     then min-payment ← max(0, r(W) + ε − (r(g) − r_i(g)), Σ_{b∈g | b^agent=i} B(b^items))
10         max-utility ← v_i(g) − min-payment
11         if r(g) > r(W) and max-utility ≥ u*
12            then g* ← g                ▷ Found a new best.
13                 u* ← max-utility      ▷ Set our new bound.
14         PAUSEBIDSEARCH(bids, g − b)   ▷ b is Out.
15    else max-revenue ← r(g) + h(items not in g)
16         if max-revenue ≤ r(W)         ▷ Can't beat current best so b is Out.
17           then PAUSEBIDSEARCH(bids, g − b)
18         elseif b^agent ≠ i            ▷ All bids left belong to someone else. Is it still worth it?
19           then min-payment ← r(W) + ε − (r(g) − r_i(g)) − h(items not in g)
20                max-utility ← v_i(g) − min-payment
21                if max-utility > u*    ▷ Seems promising so b is In.
22                  then PAUSEBIDSEARCH({x ∈ bids | x^items ∩ b^items = ∅}, g)
23                PAUSEBIDSEARCH(bids, g − b)   ▷ b is Out.
24           else ▷ Do complete search over my bids.
25                PAUSEBIDSEARCH({x ∈ bids | x^items ∩ b^items = ∅}, g)   ▷ b is In.
26                PAUSEBIDSEARCH(bids, g − b)   ▷ b is Out.
27 return
```

**Figure 2:** *The* PAUSEBIDSEARCH *recursive procedure where bids is the set of available bids and g is the current partial solution.*




| $S$ | $v_0(S)$ | $B(S)$ | in $W$? | $h(S)$ |
|---|---|---|---|---|
| (0) | 198 | 0: 131 | Yes | 131 |
| (1) | 44 | 0: 40 | No | 135 |
| (2) | 62 | 0: 45 | No | 135 |
| (0,1) | 849 | | No | NA |
| (1,2) | | 1: 270 | Yes | NA |

**Figure 3:** *Sample search tree produced by* PAUSEBIDSEARCH *for agent 0 given the values on the table at the top right. We assume that* $\varepsilon = 1$. *The nodes are bids of the form "agentid : (items) price".*

## 3. Tests

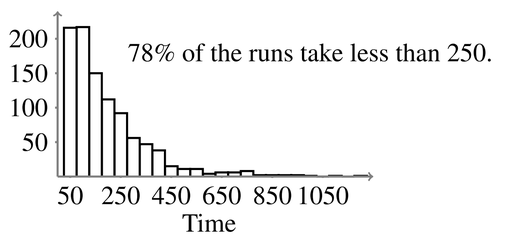



**Figure 4:** *Distribution of the times it took to run each auction, for 1000 runs with 6 agents and 5 items. The y-axis is the number of runs that took at most x time units.*

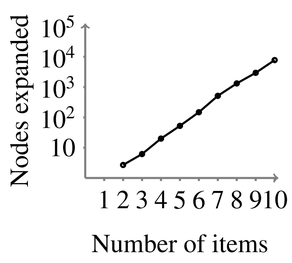



**Figure 5:** *Average number of nodes expanded as a function of the number of items in the auction. Number of bids is exponential on items.*

In at least 95% of the cases the solution found was the revenue-maximizing solution.