Auctions

José M Vidal

Department of Computer Science and Engineering University of South Carolina

March 17, 2010

Abstract

We introduce auctions for multiagent systems. Chapter 7.

Valuation

- Private value.
- Common value.
- Correlated value.

Valuation

- Private value.
- Common value.
- Correlated value.

What type of value?

- Non-transferable tickets to a concert.
- Tickets to a concert.
- Collectible stamp.

Valuation

- Private value.
- Common value.
- Correlated value.

What type of value?

- Non-transferable tickets to a concert. Private
- Tickets to a concert. Correlated
- Collectible stamp. Common or Correlated

English Auction

- First-price open-cry ascending auction.
- Very common.
- Initial prices is the reservation price.
- Dominant strategy: bid current price plus epsilon until reservation price.
- Winner's curse if common or correlated.

First-price Sealed-bid Auction

- One bid per person in a sealed envelope are given to the auctioneer who then picks the highest bid.
- Whoever submitted the highest bit wins and pays that price.
- It has no dominant strategy.
- Leads to spying.

Dutch Auction


Ontario Flower Growers Co-op.

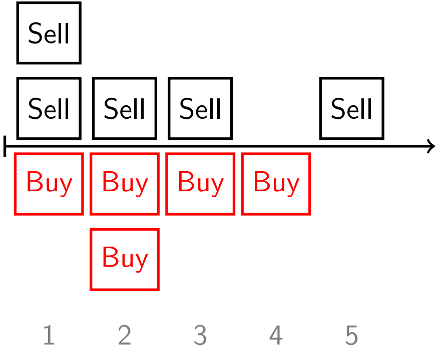
- Open-cry descending price.
- Equivalent to first-price sealed-bid auction.
- Real-time efficient.

Vickrey Auction

William Vickrey.
1914–1996.
Nobel Prize in
Economics, 1996.

- Second-price sealed-bid.
- Bidding true valuation is dominant strategy if private value.
- Eliminates strategizing.
- People don't like them.

Double Auction



Revenue Equivalence

- On which auction do sellers make more money?

Revenue Equivalence

- On which auction do sellers make more money?

Revenue Equivalence Theorem

All four auctions produce the same expected revenue in private value auctions and with bidders that are risk-neutral.

Revenue Equivalence

- On which auction do sellers make more money?

Revenue Equivalence Theorem

All four auctions produce the same expected revenue in private value auctions and with bidders that are risk-neutral.

- If risk-averse then Dutch and First-price are better.
- In common or correlated value English gives higher revenue.

Collusion

- Bidder collusion affects all 4 auctions.
- English and Vickrey auctions *self-enforce* collusion agreements.

Lying

- A lying auctioneer can make money from a Vickrey auction.
- He can also place shills in an English auction.

Inefficient Allocation

Costs of Doing Tasks

tasks	Agent 1	Agent 2
t1	2	1.5
t2	1	1.5
t1,t2	2	2.5

- Leads to inefficient allocation if auction t1 then t2.
- Implement full lookahead.
- Use a combinatorial auction.

Designing Auctions

- Decide what control you have.
  - Control only the agent.
  - Control only the mechanism.
  - Control both.
- If controlling the mechanism you must decide on:
  - Bidding rules.
  - Clearing rules.
  - Information rules.

Combinatorial Auction

- In a combinatorial auction agents can place bids for sets of goods.
  - M set of items for sale.
  - $b_i(S) = \aleph$ is agent i's bid for $S \subseteq M$.
  - $\overline{b}(S) = \max_{i \in bidders} b_i(S)$ is the set of relevant bids.

Example

Price	Bid items
$1	Beast Boy
$3	Robin
$5	Raven, Starfire
$6	Cyborg, Robin
$7	Cyborg, Beast Boy
$8	Raven, Beast Boy

Winner Determination

$$X^* = \arg\max_X \sum_{S \in X} \bar{b}(S)$$

where X is a set of sets of goods that allocates each good to only one bidder.

Problem Complexity

The number of ways to partition a set of n elements into k non-empty sets is the Stirling number of the second kind

$$S(n,k) = \frac{1}{k!}\sum_{i=0}^{k-1}(-1)^i \binom{k}{i}(k-i)^n$$

so the total number of allocations of m goods is

$$\sum_{i=1}^{m} S(m,i),$$

which is

$$O(m^m) \text{ and } \omega(m^{m/2}).$$

It is Hard

Theorem

Winner Determination in Combinatorial Auction is NP-hard. That is, finding the X^* that maximizes revenue is NP-hard.

It is Really Hard

Theorem

The decision version of the winner determination problem in combinatorial auctions is NP-complete, even if we restrict it to instances where every bid has a value equal to 1, every bidder submits only one bid, and every item is contained in exactly two bids.

Linear Programming?

If, there is a singleton bid for every item then:
Maximize:

$$\sum_{b \in B} x[b] b^{\text{value}}$$

Subject to:

$$\sum_{b \mid j \in b^{\text{items}}} x[b] \leq 1, \forall j \in M$$

$$x[b] \in \{0,1\}, \forall b \in B,$$

where $x[b]$ is a bit which denotes whether bid b is a winning bid.

Linear Programming?

If, there is a singleton bid for every item then:
Maximize:

$$\sum_{b \in B} x[b] b^{\text{value}}$$

Subject to:

$$\sum_{b \,|\, j \in b^{\text{items}}} x[b] \leq 1, \forall j \in M$$

$$x[b] \in \{0, 1\}, \forall b \in B,$$

where $x[b]$ is a bit which denotes whether bid b is a winning bid.
In practice, these general algorithms are much slower than specialized search algorithms.

More Linear Programming Cases

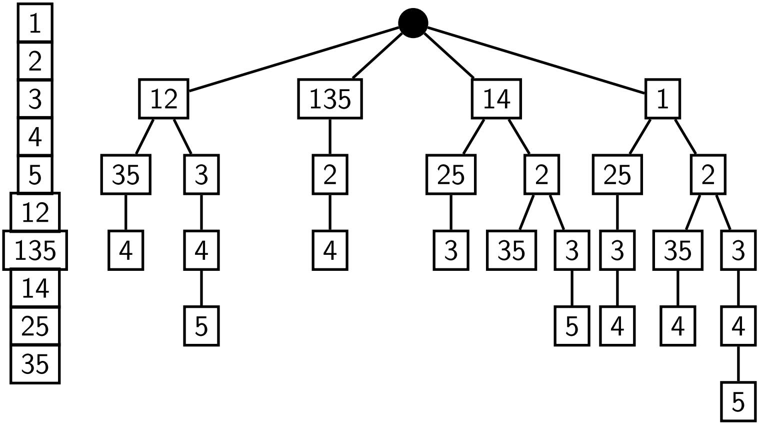
The LP problem will solve a combinatorial auction when the bids satisfy any one of the following criteria:

1. All bids are for consecutive sub-ranges of the goods.
2. The bids are hierarchical.
3. The bids are only OR-of-XORs of singleton bids.
4. The bids are all singleton bids.
5. The bids are downward sloping symmetric.

Search Algorithm

1. Number the goods from 1 to m.
2. Create an empty root node.
3. For each node, add as its children all the bids that
    1. include the smallest good that is not on the path and
    2. do not include any good on the path.

Search Tree



9 leafs versus 52 allocations

Branch and Bound on Branch on Items Tree

BRANCH-ON-ITEMS-CA()

1 $r* \leftarrow 0$ ▷ Max revenue found. Global variable.
2 $g* \leftarrow \emptyset$ ▷ Best solution found. Global variable.
3 BRANCH-ON-ITEMS-CA-HELPER$(1, \emptyset)$
4 **return** g^*

- We will use:

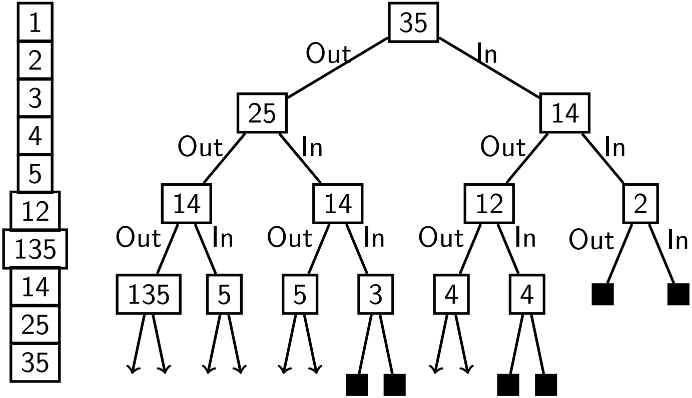
$$h(g) = \sum_{i \in \text{items not in } g} \max_{S|i \in S} \frac{\bar{b}(S)}{|S|}$$

Branch and Bound on Branch on Items Tree

$\textsc{Branch-on-Items-CA-Helper}(i, g)$

if $i = m$ $\triangleright$ g covers all items
  then if $\sum_{b \in g} b^{\text{value}} > r^*$ $\triangleright$ g has higher revenue than r^*
      then $g^* \leftarrow g$
        $r^* \leftarrow \sum_{b \in g} b^{\text{value}}$
    return
for $b \in \{b \in B \mid i \in b^{\text{items}} \wedge b^{\text{items}} \cap \bigcup_{b_1 \in g} b_1^{\text{items}} = \emptyset\}$
  do $g' \leftarrow g + b$ $\triangleright$ b^{items}don't overlap g
    if $\sum_{b_1 \in g'} b_1^{\text{value}} + h(g') > r^*$
      then $\textsc{Branch-on-Items-CA-Helper}(i + 1, g')$

Branch on Bids Search Tree



Branch and Bound on Branch on Bids Tree

BRANCH-ON-BIDS-CA()

1 $r* \leftarrow 0$ ▷ Max revenue found. Global variable.
2 $g* \leftarrow \emptyset$ ▷ Best solution found. Global variable.
3 BRANCH-ON-BIDS$(\emptyset, B)$
4 **return** g^*

Branch and Bound on Branch on Bids Tree

$\textsc{Branch-on-Bids}(g, \textit{available-bids})$

if $\textit{available-bids} = \emptyset$
    then return
if $\bigcup_{b \in g} b^{\text{items}} = M$ $\quad\triangleright$ g covers all items
    then if $\sum_{b \in g} b^{\text{value}} > r^*$ $\triangleright$ g has higher revenue than r^*
            then $g^* \leftarrow g$
                    $r^* \leftarrow \sum_{b \in g} b^{\text{value}}$
        return
$\textit{next} \leftarrow \textsc{First}(\textit{available-bids})$
if $\textit{next}^{\text{items}} \cap \bigcup_{b_1 \in g} b_1^{\text{items}} = \emptyset\}$ $\quad\triangleright$ $\textit{next}$'s items do not overlap g
    then $g' \leftarrow g + \textit{next}$
        if $\sum_{b_1 \in g'} b_1^{\text{value}} + h(g') > r^*$
            then $\textsc{Branch-on-Bids}(g', \textsc{Rest}(\textit{available-bids}))$
$\textsc{Branch-on-Bids-CA-Helper}(g, \textsc{Rest}(\textit{available-bids}))$

Other Improvements

- Keep only the highest bid for any set.
- Remove provably noncompetitive bids (i.e. they are dominated by another bid or sets of bids).
- Decompose bids into connected sets. Each solved independently.

Analysis and Updates

- Results depend a lot on bid distribution and correlation.
- Much better than testing allocations.
- Thousands of bids doable in seconds.

- Newer algorithms order bids for even better performance.
- Approximation algorithms are 1-2 orders faster but find only 99% of optimal solution.
- Does not divide payment among sellers.

Distribute over Bidders

The PAUSE auction.

```
Have simultaneous open-cry auctions for each individual item.
For (k ← 2; k ≤ m; k ← k + 1)
    do Each bidder must place a complete bidset
       (using his bids or others' bids)
       where bids are all of size less than k.
```

PAUSE Analysis

- Each bid is a complete bidset so auctioneer only needs to check that revenue increases.
- Bidders have an incentive to perform calculation as it means they could get a good deal if they win.
- We have published paper on myopically-optimal bidding algorithm: PAUSEBID

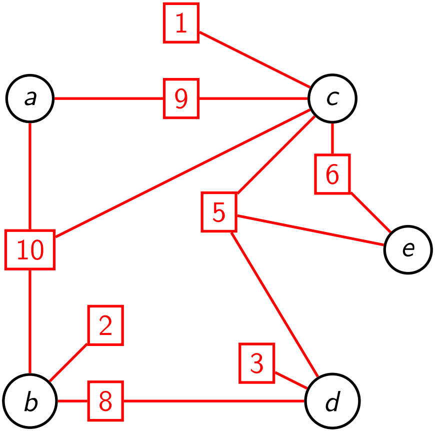
PAUSE Analysis

- Each bid is a complete bidset so auctioneer only needs to check that revenue increases.
- Bidders have an incentive to perform calculation as it means they could get a good deal if they win.
- We have published paper on myopically-optimal bidding algorithm: PAUSEBID
- Results: PAUSE+PAUSEBID $\rightarrow$ usually X^*

Distribute Over Sellers

- Each seller of an item negotiates with other sellers to clear a bid for himself.
- We then have a negotiation network, again.

Negotiation Network



Bidding Languages

- OR bids: b_1 OR b_2 OR... OR b_k.
  - Cannot represent subbaditive valuations

Bidding Languages

- OR bids: b_1 OR b_2 OR... OR b_k.
  - Cannot represent subbaditive valuations
- XOR bids: b_1 XOR b_2 XOR... XOR b_k
  - Can get very long for seemingly common valuations that can be more succintly expressed using the OR bids.
  - Most algorithms work with OR bids.

Bidding Languages

- OR bids: b_1 OR b_2 OR... OR b_k.
    - Cannot represent subbaditive valuations
- XOR bids: b_1 XOR b_2 XOR... XOR b_k
    - Can get very long for seemingly common valuations that can be more succintly expressed using the OR bids.
    - Most algorithms work with OR bids.
- OR* bids: user OR bids but create dummy items to express xor constraints.

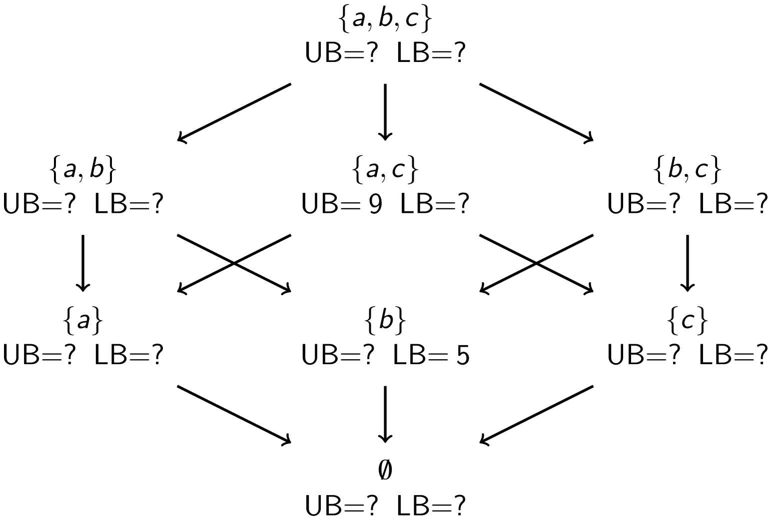
Preference Elicitation

- Idea: ask only about information we need to find solution.

Preference Elicitation

- Idea: ask only about information we need to find solution.
- Assume free disposal: if $S \subseteq T$ then $v(S) \leq v(T)$.

Constraint Network



PAR algorithm

PAR()

```
fringe ← {{1, ..., 1}}
while fringe ≠ ∅
      do c ← first(fringe)
         fringe ← rest(fringe)
         successors ← CHILDREN(c)
         if FEASIBLE(c)
            then pareto-solutions ← pareto-solutions ∪ c
                 fringe ← successors
            else for n ∈ successors
                     do if n ∉ fringe
                           ∧UNDOMINATED(n, pareto-solutions)
                           then fringe ← fringe ∪ n
```

PAR algorithm

$\textsc{Children}(\{k_1, \ldots, k_n\})$

for $i \in 1 \ldots n$

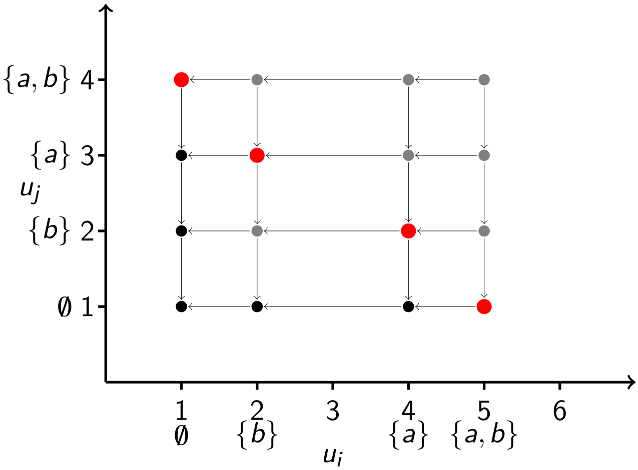
  do $c \leftarrow \{k_1, \ldots, k_n\}$

    $c[i] \leftarrow c[i] + 1$

    $result \leftarrow result \cup c$

return *result*

Rank Lattice



PAR Algorithm

- PAR algorithm only asks rank questions.
- PAR algorithm finds complete Pareto frontier.
- Cannot determine utilitarian solution since it only asks rank questions.

EBF Algorithm

- Same as PAR but also ask for the values and alwas expand allocation in fringe with highles value.
- Will find the utilitarian solution.

EBF Algorithm

$\textsc{EBF}()$

1 $\quad$ $\mathit{fringe} \leftarrow \{\{1,\ldots,1\}\}$
2 $\quad$ **if** $|\mathit{fringe}| = 1$
3 $\qquad$ **then** $c \leftarrow \mathrm{first}(\mathit{fringe})$
4 $\qquad$ **else** $M \leftarrow \{k \in \mathit{fringe} \mid v(k) = \max_{d \in \mathit{fringe}} v(d)\}$
5 $\qquad\qquad$ **if** $|M| \geq 1 \wedge \exists_{d \in M} \textsc{feasible}(d)$
6 $\qquad\qquad\quad$ **then return** d
7 $\qquad\qquad\quad$ **else** $c \leftarrow \textsc{pareto-solution}(M)$
8 $\quad$ **if** $\textsc{feasible}(c)$
9 $\qquad$ **then return** c
10 $\quad$ $\mathit{successors} \leftarrow \textsc{children}(c)$
11 $\quad$ **for** $n \in \mathit{successors}$
12 $\qquad$ **do if** $n \notin \mathit{successors}$
13 $\qquad\qquad$ **then** $\mathit{fringe} \leftarrow \mathit{fringe} \cup \{n\}$
14 $\quad$ **goto** 2

PAR and EBF Analysis

- Both have running times exponential on number of items.
- Bad performance in practice, ask too many questions.