Combinatorial Auctions: Winner Determination

Sandholm 2002, Rothkopf et al. 1998

Peter Olsar

October 4, 2004

Outline

- Motivation and problem definition
- Dynamic programming solution
- Solution using a search tree
- Making the algorithm practical using heuristic search
- Extensions

Motivation for combinatorial auctions

Several items to be auctioned

Agent's valuations are not additive $\Longrightarrow$ agent needs to estimate what other items it will get—difficult, does not guarantee efficient allocation

Partial solutions:

- Parallel auction
- Aftermarket
- Progressive auction with bid retraction

A better solution: combinatorial auction

Allows bids on combinations of items

M = set of items to be auctioned

Agent i places bid of value $\$_i(S)$ on a subset $S \subseteq M$;
$\$_i(S) = 0$ if agent i does not place bid on S

Highest bid on combination S is $\$(S) = \max_i \$_i(S)$

Goal: maximize auctioneer's revenue:

$$\max_W \sum_{S \in W} \$(S)$$

W is a partition of M

Integer programming formulation

$\mathbf{b}$ is bid vector $(\$(S_1), \ldots, \$(S_{2^m}))$, where S_i is the "ith subset" of M and $m = |M|$

$\mathbf{x}$ is 0-1 vector $(x_1, \ldots, x_{2^m})$

$$x_i = \begin{cases} 1 & \text{if (highest) bid on } S_i \text{ wins} \\ 0 & \text{otherwise} \end{cases}$$

Maximize $\mathbf{b} \bullet \mathbf{x}$ under the constraints

$$\forall\ item \in M, \sum_{j\ :\ item \in S_j} x_j \leq 1$$

Maximum revenue determination

It is NP-hard in general:

- Exhaustive enumeration: running time $\omega\left(m^{m/2}\right)$
- Dynamic programming: $\mathcal{O}(3^m)$, $\Omega(2^m)$
- Approximation? In polynomial time, approximation guarantees are not good enough (remember, **money** is involved)
- Special cases
- Heuristic search

Outline

- Motivation and problem definition $\surd$
- Dynamic programming solution
- Solution using a search tree
- Making the algorithm practical using heuristic search
- Extensions

Dynamic programming

Determine for each nonempty subset S of M the highest possible revenue using only items from S ($2^m - 1$ subsets)

For each set S, the maximum revenue $r^*(S)$ comes from either:

- Single bid (let $\mathcal{C}(S) = S$), or
- Sum of maximum revenues of two disjoint proper subsets of S (let $\mathcal{C}(S)$ be the smaller of the two subsets)

Computing maximum revenues r^*

For each item $i \in M$, $r^*(\{i\}) \leftarrow \$(\{i\})$ and $\mathcal{C}(\{i\}) \leftarrow \{i\}$

For each k from 2 to m and for each $S \subseteq M$ s.t. $|S| = k$ do

- $r^*(S) \leftarrow \max\{r^*(S \setminus T) + r^*(T)\}$ over all $T \subseteq S$ such that $1 \leq |T| \leq |S|/2$
- If $r^*(S) < \$(S)$ then $r^*(S) \leftarrow \$(S)$ and $\mathcal{C}(S) \leftarrow S$
- Else $\mathcal{C}(S) \leftarrow$ set T that maximizes the right-hand side of the recurrence

Recovering an optimal solution

- $W_{\mathrm{opt}} \leftarrow \{M\}$ (initialize an optimal partition of M)
- For each $S \in W_{\mathrm{opt}}$ until W_{opt} does not change do
- If $\mathcal{C}(S) \neq S$ then $W_{\mathrm{opt}} \leftarrow (W_{\mathrm{opt}} - \{S\}) \bigcup \{\mathcal{C}(S), S \setminus \mathcal{C}(S)\}$

Is this any good?

$\mathcal{O}(3^m)$, $\Omega(2^m)$ running time

Good news: running time is independent of the number of bids

Bad news: useful only for small values of m; the algorithm examines subsets for which bids have not been submitted

The algorithm is actually polynomial in the number of bids n if $n \in \Omega(2^m)$ (unlikely)

In general, if $n \in \Omega\left(2^{m/\rho}\right)$ then running time is $O\left(n^{\rho \log_2 3}\right)$

Outline

- Motivation and problem definition $\surd$
- Dynamic programming solution $\surd$
- Solution using a search tree
- Making the algorithm practical using heuristic search
- Extensions

Search tree approach (Sandholm 2002)

Each node corresponds to a bid (except the root)

The bids on the path from the root to node α have been accepted and thus must be disjoint

That is, α is a leaf iff no more bids can be accepted

Wait!

What if we have bids: \$5 on $\{$apple$\}$ and \$3 on $\{$apple, orange$\}$. Then it is preferable for the auctioneer to keep orange and sell apple $\implies$ introduce *dummy bids* (one-item bids with value \$0)

Then every path from the root to a leaf induces a partition of M

Branching strategy

Bid B is a child of node α iff:

1. B includes the smallest-index item i among the items not already allocated on the path to α (all siblings of B contain i)
2. B does not include items already allocated

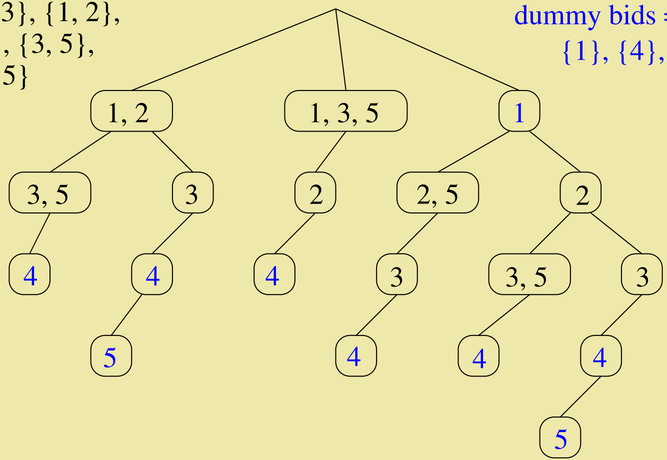
\# of nodes in the search tree $\leq \left(\frac{n}{m}\right)^m < 2^n$
n is # of bids, m is # of items

This is **polynomial** in the number of bids

An example search tree

bids = {2}, {3}, {1, 2},
{2, 5}, {3, 5},
{1, 3, 5}

dummy bids =
{1}, {4}, {5}



How to determine the children of a node?

Naïve approach: for each bid, determine if it includes the lowest-index unallocated item $\implies \Theta(nm)$ running time

Can this be improved?

Bidtree - binary tree in which each level except the last corresponds to an item and each leaf corresponds to a bid

A path from the root to a bid (leaf) determines what items are in the bid (follow left edge $\iff$ include item, follow right edge $\iff$ don't include item)

Additional information

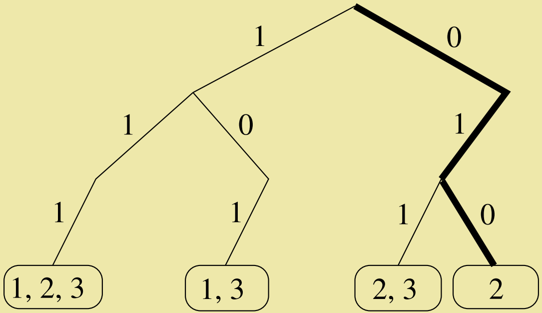
For each item $i \in M$, *stopmask*$[i] =$

- *blocked* iff i has already been allocated
- *must* iff i is the lowest-index unallocated item
- *any* for all other items

blocked $\implies$ may not follow left ("include") edge in the bidtree
must $\implies$ may not follow right ("don't-include") edge
any $\implies$ may follow either edge

Children of node α in the search tree are those bids that are reachable in the bidtree under these constraints

Bidtree example (determining children of $\{1,3\}$*)*



stopmask[1] = blocked
stopmask[2] = must
stopmask[3] = blocked

Using the bidtree

When search begins: $stopmask[1] = must$ and
$stopmask[i] = any$ for $2 \leq i \leq m$

The children of a node in the search tree are determined via DFS on the bidtree

When a child with bid B is explored in the search tree:

- $stopmask[i] \leftarrow blocked$ for all $i \in B$ and
- $stopmask[i^*] \leftarrow must$, where i^* is the next smallest-index unallocated item

After the DFS on the search tree has explored the subtree rooted at B, it backtracks, resetting *stopmask* values

Complexity of bidtree search

Each edge is traversed twice: once forward, once backward $\implies$ time complexity of bidtree search $\in \mathcal{O}(\#$ of edges$)$

Tight bound on $\#$ of edges $= nm - n\lfloor \log n \rfloor + 2 \cdot 2^{\lfloor \log n \rfloor} - 2$

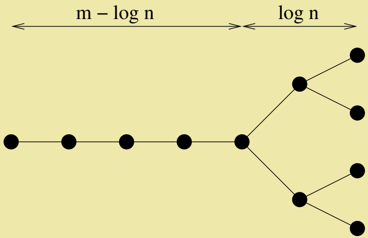
$m - \log n \geq c \implies \mathcal{O}(n(m - \log n))$

$m - \log n < c \implies \mathcal{O}(n)$

Thus, the worst-case running time reduction (from $\Theta(mn)$) is only slight

But!

An example where the running time is linear



$\mathcal{O}\left(2^{\log n}\right) + \mathcal{O}(m - \log n) = \mathcal{O}(n + m - \log n)$ edges—this is **linear**

We have analyzed the worst case for a single node

What happens if we amortize over all nodes with *stopmask* pruning? **Open problem!**

Outline

- Motivation and problem definition $\surd$
- Dynamic programming solution $\surd$
- Solution using a search tree $\surd$
- Making the algorithm practical using heuristic search
- Extensions

Improving the (average) running time with heuristics

As given, the algorithm is an uninformed search; that is, it blindly explores the search tree

In order to be sure of an optimal solution, the algorithm needs to explore the entire tree

Anytime feature of the algorithm:

- Keep the best solution found so far
- Terminate if computation takes too long $\Longrightarrow$ have a feasible solution

Experiments showed the solution is usually close to optimum

Preprocessing heuristic I

Remove noncompetitive bids; a bid B is *noncompetitive* if

$$\$(B) \leq \sum_{\text{disjoint } B' \subseteq B} \$\left(B'\right)$$

This can take time exponential in the size of $B \implies$

- Apply the heuristic for "small" bids only
- Restrict the number of subsets B'

Preprocessing heuristic II

Decompose bids into connected components:

- Bid $=$ vertex
- Two bids are connected by an edge if they share an item

The subsets of bids corresponding to connected components of the graph can be solved **independently**

Improved search strategy using IDA^*

$g(\alpha)$ = total value of the bids on the path from the root to node α

$h(\alpha)$ = *admissible heuristic function*—the maximum possible revenue that could be obtained by allocating unallocated items

$f(\alpha) = g(\alpha) + h(\alpha)$ = the maximum possible total revenue that could be obtained by accepting bids on the path from root to α

A^* *search* - explore the children of a node in the order of nonincreasing value of f

Crucial observation: if $f(\alpha) \leq$ best revenue found so far, the current best solution cannot be improved by searching the subtree rooted at $\alpha \implies$ prune the subtree

Iterative deepening A^* *search* (IDA^*)

Don't explore any nodes α that have $f(\alpha) < \textit{f-limit}$

After the search returns, decrease the value of *f-limit* and repeat

This forces exploration of promising paths in the tree first

Once a leaf is reached during an iteration of IDA^*, *f-limit* is set to the revenue at that leaf

For the next iteration, *f-limit* is set to $\min\{\textit{new-f}, 0.95 \cdot \textit{f-limit}\}$, where *new-f* is the maximum value of f in the previous iteration

Constant 0.95 was determined empirically

Heuristic functions

$$h_1 = \sum_{i \in F} c(i)$$

- F is the set of unallocated items
- $c(i)$ = item's maximum possible contribution to a bid =

$$\max_{S \mid i \in S} \left\{ \frac{\$(S)}{|S|} \right\}$$

h_2 = same as h_1, but compute $c(i)$ only using bids not containing any allocated items (more expensive to compute)

Outline

- Motivation and problem definition $\surd$
- Dynamic programming solution $\surd$
- Solution using a search tree $\surd$
- Making the algorithm practical using heuristic search $\surd$
- Extensions

Extensions of the algorithm

- What if the auctioneer cannot keep any items?
    1. Don't introduce dummy bids
    2. A solution is feasible iff all items are allocated
- Incremental winner determination
- Incremental quote computation

Incremental winner determination

Having computed an optimal solution for the previous set of bids, can we update it quickly if a new bid B_{new} arrives?

- If B_{new} gets pruned by Preprocessing heuristic I, ignore it, and new solution = old solution
- Otherwise:
  - Recompute an optimal solution sol^* on set of items $M \setminus B_{\text{new}}$; use one iter. of IDA^* with *f-limit* = old revenue − $\$(B_{\text{new}})$
  - If revenue of sol^* is greater than old revenue then the new solution is $sol^* \bigcup \{B_{\text{new}}\}$; otherwise, the old solution remains optimal

Incremental quote computation (exact)

"How much do I need to bid on a set of items S to be a winner of those items?"

- Remove the items S and all bids containing those items
- Recompute the maximum revenue r^*_{reduced}
- Must bid at least \$ $(r^* - r^*_{\mathrm{reduced}})$, where r^* is the current optimal revenue

Incremental quote computation (approximate)

"If I bid \$$x$ on S, will I win S?"

"Yes" if $x >$ (upper bound on r^*- lower bound on r^*_{reduced})

"If a bid \$$x$ on S, will I NOT get S?"

"Yes" if $x <$ (lower bound on r^*- upper bound r^*_{reduced})

The bounds can be computed using approximation algorithms

FCC auction of radio frequencies (FCC = Federal Communications Commission (US))

Did not actually use a combinatorial auction, even though the auction is often mentioned as the archetype where a combinatorial approach would allocate resources much more efficiently. Why?

- Fear of trying something new on such large scale
- Computation cost could be too high
- Bidders may find the auction confusing
- Lack of transparency—will the bidders understand and trust the winner determination process?
- Hidden agendas and prejudices of the economists advising the FCC committee

Summary

Combinatorial auctions are useful when bidder's valuations of items are not additive

Finding an optimal winner collection of bids is NP-hard; a dynamic programming algorithm is exponential in the number of items

Exact solutions are very much preferred $\Longrightarrow$ heuristic search (IDA^*) is applied over possible bid combinations

Extensions of the heuristic search to incremental winner determination and incremental quote computations can dramatically reduce computation time