The Winner-Determination Problem

Based on
“A Branch-and-Price Algorithm and New Test Problems for Spectrum Auctions” (A3)

Hassan Sultan, Nick Elprin, Matthew Perone
Problem formulation

- Combinatorial Auction
- Bidding language: XOR-of-OR
- 2 ILP formulations
- Solved using custom cut-and-price algorithm and CPLEX
- Underlying problem reduces to set packing
Background

- FCC auctions progression:
  - SAA (Simultaneous Ascending Auctions):
    * Multiple rounds
    * Bid on individual goods
    * Highest bid announced
    * New bids
    * Close when no more bids
  - Domain restrictions:
    * Nationwide licenses (differ in quality of coverage, not geography). At most 3 licenses
    * Geographical regions introduced (differ in bands and geography). Only 5 regions.
    * More regions introduced (MTAs and BTAs).
    * SAA-PB (SAA with package bidding). Predetermined packages.
    * Attempts to move to combinatorial auctions.

- Tradeoff: Restrictions $\implies$ Easier problems but worse solutions thus move to finer grained domains and combinatorial bidding.
Bidding language: XOR-of-OR

• In round $r$ bidder $j$ submits bid:
\[
(S_{j,1}^r, b_{j,1}^r) \text{ OR } (S_{j,2}^r, b_{j,2}^r) \text{ OR } \ldots \text{ OR } (S_{j,l(j,r)}^r, b_{j,l(j,r)}^r)
\]

• $l(j,r)$: number of bids in round $r$

• $(S_{j,k}^r, b_{j,k}^r)$: package $S_{j,k}^r$ for which bidder is bidding $b_{j,k}^r$

• $\bigotimes_{r=1}^t \left[ \bigvee_{k=1}^{l(j,r)} (S_{j,k}^r, b_{j,k}^r) \right]$

  \[
  = XOR_{r=1}^t \left[ (S_{j,1}^r, b_{j,1}^r) \text{ OR } (S_{j,2}^r, b_{j,2}^r) \text{ OR } \ldots \text{ OR } (S_{j,l(j,r)}^r, b_{j,l(j,r)}^r) \right]
  \]
WD formulation 1

$$\text{max} \sum_{j \in N} \sum_{r=1}^{t} \sum_{(S,b) \in B_{j,r}} bx_{j,r,S}$$  \hspace{1cm} (1)

s.t. \hspace{1cm} \sum_{j \in N} \sum_{r=1}^{t} \sum_{(S,b) \in B_{j,r}: S \ni i} x_{j,r,S} \leq u_i \quad \forall i \in M \hspace{1cm} (2)

$$x_{j,r,S} \leq z_{j,r} \quad \forall j \in N, (S,b) \in B_{j,r}, r = 1, \ldots, t \hspace{1cm} (3)$$

$$\sum_{r=1}^{t} z_{j,r} \leq 1 \quad \forall j \in N \hspace{1cm} (4)$$

$$x_{j,r,S} \in \{0,1\} \quad \forall S \subseteq M, j \in N, r = 1, \ldots, t \hspace{1cm} (5)$$

$$z_{j,r} \in \{0,1\} \quad \forall j \in N, r = 1, \ldots, t \hspace{1cm} (6)$$

1. Maximize revenues

2. Supply constraints

3. \(z_{j,r}\) indicates whether bidder \(j\) gets something from \(B_{j,r}\). \(x_{j,r,S}\) indicates if bid \((S,b)\) in round \(r\) is a winning bid.

4. Each bidder only gets bids from a unique round satisfied (i.e. XOR)
WD formulation 2

\[
\text{max} \sum_{j \in N} \sum_{(S,b) \in P_j} b_{y_j,S} \\
\text{s.t.} \sum_{j \in N} \sum_{(S,b) \in P_j : S \ni i} y_{j,S} \leq u_i \quad \forall i \in M \\
\sum_{(S,b) \in P_j} y_{j,S} \leq 1 \quad \forall j \in N \\
y_{j,S} \in \{0,1\} \quad \forall S \in P_k, j \in N
\]  

Where \( P_{j,r} \) is the set of all bid combinations that bidder \( j \) could win in round \( r \) (i.e. bundles). \( P_j = \bigcup_{r=1}^{t} P_{j,r} \)

This formulation is exponential in size and thus cannot be solved using an general purpose IP solver. It is useful however for the custom branch-and-price algorithm used in this paper.
Why two formulations?

• Formulation 2 is exponential in size

• Formulation 2 cannot be solved by ILP, must use the custom package

• However, Formulation 2 provides a strong LP-relaxation and thus results in a smaller enumeration tree for branch-and-bound.

• Lemma 1: For every solution of the LP-relaxation of 2 with a certain objective value, there exists a solution of the LP-relaxation of 1 that has the same objective value. However, there are many solutions of 1 that don’t have corresponding solutions (i.e. with the same o.v.) in 2.

• Thus, the LP-relaxation of 2 can give better bounds than 1.
Why two formulations?

- In general, CPLEX was used with formulation 1 and the custom branch-and-price algorithm was used with formulation 2. At first, the latter performed better (presumably because it exploits inherent structure in the XOR-of-OR bidding language).

- However, CPLEX adds additional constraints that might take away from this advantage that the cut-and-price method has. Thus, as we will see later, certain tweaks to parameters of CPLEX made it perform better.

- Note however that in the extreme case of single OR clauses (i.e. XOR-of-OR reduces to XOR), the structure of XOR-of-OR that formulation 2 exploits is does not exist (i.e. both formulations are equivalent).
Caveats

• The paper considers an auction, not an exchange. We could get around this by adding indicator variables for which sellers to buy from and add their valuations etc into constraints (their bids would be considered as asks).

• There are no upper and lower bounds, just bids. We could just run the optimistic and pessimistic version of the ILPs as discussed in lecture.

• The second formulation is theoretically useful, but requires implementing custom (branch-and-price) algorithm. Thus using this is a possible tradeoff between efficiency and implementation time. Considering this a semester long course (and the fact that a well tweaked CPLEX outperforms the branch-and-price), we probably want to stick to CPLEX.
Branching rules

• (B1) Branch on a bid
  • select a bid
  • divide search space into two parts, one that accepts the bid, one that doesn’t

• (B2) Branch on an item
  • Select agent and item:  \( j \in N, i \in M \)
  • divide search space in two parts: one where \( i \) is given to \( j \), one where it isn’t.
  • I.e., set \( \sum_{r=1}^{t} \sum_{(S,b) \in B_{j,r}:S \ni i} y_{j,r,S} \) to be 1 or 0
Branching rules (continued)

• (B3) multi-way branch on an item
  - select an item \( i \)
  - up to \(|N| + 1\) branches: in each branch, a different agent gets \( i \) (and in one branch nobody gets \( i \))

• (B4) Restricted multi-way branch on an item
  - like B3, but select a subset of bidders interested in \( i \) (e.g., up to five bidders) and branch only on those
Dependencies

Why other groups’ decisions are relevant to us

- Constraints on the bidding language can simplify the task of solving
- Features about the domain affect how we should test our solver
Bidding language constraints

- Authors use an XOR-of-OR language
- Easy to add features to allow for:
  - fixed offset signal at a new round
  - budget or capacity constraints for certain agents
- Restrictions: limit size or structure of bid set
  - E.g., Non-crossing sets (every two bundles are either disjoint, or one contains the other)
- Restrictions reduce computation costs and limit opportunities for bidders to stall; but at a cost of expressiveness.
How do we evaluate our solver?

• Generating test problems is easier said than done

• Use some distribution to select bundles for each agent (e.g., random, uniform, decay, binomial)
  - But these are just an anonymous list of bids: they don’t capture auction-specific features.
  - Usually these are easy problems, CPLEX just tears them apart.

• We need domain-specific test problems
Test problems for the FCC auction

- Use data from older FCC auctions to generate test problems for the combinatorial version
- Build adjacency graph of “major trading areas” (MTAs)
- Let an agent’s (location, frequency) pairs induce a subgraph $G$ on the MTA graph
- bid on connected components of $G$ and some of their subgraphs
Adjacency graph from FCC auction #4
Test problems for the FCC auction

- Use data from older FCC auctions to generate bids

\[ b_C = \left( \sum_{u \in C} b_u \right) \cdot \left( 1 + \frac{\varepsilon}{5} \cdot \begin{cases} 
|C| - 1 & \text{if } |C| \leq 5 \\
4 + 0.02(|C| - 5) & \text{if } 6 \leq |C| \leq 10 \\
5 & \text{if } 11 \leq |C| 
\end{cases} \right) \]

- non-decreasing, concave, threshold, simple

- More details in the paper (group D should read this!)
Performance Results

• Want to compare computation performance of CPLEX implementation to Branch and Price implementation

• 5278 test problems generated with the algorithm
  – Epsilon $\epsilon$ : (0.00, 0.02, 0.05, 0.10, 0.25, 0.50, 1.00)
  – Rounds $r$: (1, 10, 20, 40)

<table>
<thead>
<tr>
<th></th>
<th>Consecutive Rounds</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Bidders</td>
<td>17/23/30</td>
</tr>
<tr>
<td>Bids</td>
<td>183/274/346</td>
</tr>
<tr>
<td>Bundles</td>
<td>$2^{15}$/-$/2^{41}$</td>
</tr>
</tbody>
</table>

minimum / average / maximum values
Performance Results

- For BaP, rule B2 (branching on an item) is the best
  - Independent of number of rounds, value of epsilon

<table>
<thead>
<tr>
<th>Consecutive Rounds</th>
<th>1</th>
<th>10</th>
<th>20</th>
<th>40</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\epsilon=0.00$</td>
<td>0,0,0</td>
<td>5,5,5</td>
<td>11,11,11</td>
<td>20,20,20</td>
<td>8,8,8</td>
</tr>
<tr>
<td>0.02</td>
<td>0,0,0</td>
<td>6,5,5</td>
<td>15,13,14</td>
<td>29,25,27</td>
<td>11,9,10</td>
</tr>
<tr>
<td>0.05</td>
<td>1,0,0</td>
<td>15,6,6</td>
<td>25,15,16</td>
<td>49,23,28</td>
<td>20,10,11</td>
</tr>
<tr>
<td>0.10</td>
<td>1,0,0,</td>
<td>36,7,8</td>
<td>36,13,16</td>
<td>75,26,32</td>
<td>33,10,12</td>
</tr>
<tr>
<td>0.25</td>
<td>2,0,0</td>
<td>87,7,9</td>
<td>106,13,17</td>
<td>147,23,26</td>
<td>78,10,12</td>
</tr>
<tr>
<td>0.50</td>
<td>2,0,0</td>
<td>142,11,15</td>
<td>173,24,30</td>
<td>199,45,56</td>
<td>119,18,22</td>
</tr>
<tr>
<td>1.00</td>
<td>3,0,0</td>
<td>148,12,23</td>
<td>204,31,56</td>
<td>250,69,114</td>
<td>138,24,42</td>
</tr>
<tr>
<td>Total</td>
<td>1,0,0</td>
<td>63,8,10</td>
<td>81,17,23</td>
<td>110,33,43</td>
<td>58,13,17</td>
</tr>
</tbody>
</table>

Average Running Time for B1,B2,B4
Performance Results

• Compare BaP to CPLEX with cut generation options enabled
  – BaP solves more problems within 10 minute time limit
  – BaP appears to scale better as synergies between items, number of rounds grow
  – Reason: tighter bounds on LP relaxation of BaP formulation

<table>
<thead>
<tr>
<th></th>
<th>$\epsilon$</th>
<th>Consecutive Rounds</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.00</td>
<td>0.02</td>
<td>0.05</td>
</tr>
<tr>
<td>BaP</td>
<td>Avg</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>Max</td>
<td>103</td>
<td>119</td>
</tr>
<tr>
<td>CPLEX</td>
<td>Avg</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>Max</td>
<td>857</td>
<td>119</td>
</tr>
</tbody>
</table>

Avg and Max runtimes compared with no time limit
Performance Results

• Improvements to CPLEX by tuning:
  – Explore “up” branch first in enumeration tree
  – Set mip clique cut generation strategy to
  – Set mip strategy to emphasize feasibility

• With these 3 settings, the CPLEX runtime is cut by a factor of 6.5, now only 2.7x slower than BaP

• For average problem, CPLEX is even faster than BaP
  – But, again, BaP scales exponentially better
Performance Results

• CPLEX seems to be the right choice for our cases

• We think that we are in a reasonably good range for CPLEX to come close to BaP performance

• The tradeoff in speed is more than compensated for by an immensely easier implementation