Combinatorial Auction Winner Determination with Branch-and-Price

Marta Eso (IBM T.J. Watson)
Soumyadip Ghosh (Cornell)
Jayant Kalagnanam (IBM T.J. Watson)
Laszlo Ladanyi (IBM T.J. Watson)

www.research.ibm.com/auctions/
Outline

- Role of winner determination in iterative auctions
- Scenario: procurement auction with supply curves
- Modeling the problem as a combinatorial auction
- Solution method of choice: Branch-and-Price
  - Column generation
  - Feasible solution heuristics
  - Branching
- Comparison with naïve models
- Future directions
Motivation

- Procurement auctions are still going strong since powerful buyer can set the rules for its suppliers
- Multiple items, multiple attributes and business requirements typical for particular industry must be considered
- Need a flexible formulation and efficient solution method
  - Can capture a variety of business requirements
  - Strong relaxations, high scalability
- Branch-and-Price as solution technology
  - Successful in other application areas (transportation, assignment, inventory logistics)
  - BCP framework readily available (www.coin-or.org session TB42)
    - Need to implement only the problem specific components
Iterative auctions

- Auction: negotiation through bidding (forward, reverse, double)
  - Participants: market maker (MM) and agents
- Bids: what and for how much agents want to trade
- Winner Determination: MM selects best allocation of goods
  - May be a subroutine of pricing and feedback
Multi-dimensional auctions in B2B

- Multiple items
  - Necessary (and profitable) if there is correlation between goods

- Multiple units
  - Decomposable goods and possibility of aggregation (buy-side, sell-side or both) warrants multi-unit auctions
  - Single unit auctions when items are non-decomposable for technical or marketing reasons (e.g., FCC licence)

- Multiple attributes
  - Goods are very rarely described by price and quantity only
    - Quality, geography, delivery time, etc.
  - Complex business requirements govern how goods can be traded

Combinatorial auctions: multiple items, indivisible bids
Mechanism design and its impact on WD

- How often is the market cleared?
  - Continuous (after each bid) or periodic (e.g., every hour)
  - Very fast response time needed if continuous, but bids change little from round to round

- How are the settlement prices computed?
  - Incentive compatibility can be computationally expensive

- Are approximate solutions acceptable?
  - Value of approximate solution might be close to optimal but the identity of the winners can be very different
  - How quickly is incentive compatibility lost if solutions are approximated?

- How do business requirements influence performance?
  - Even feasible allocations might be much more difficult to find
Winner determination

- Given a set of bids …
  - Winning bids from previous round(s) (could also include rejected bids)
  - Bids submitted since last round
- … compute an allocation of goods to bidders …
  - Determine which agents trade what
- … so that market maker’s objective is optimized
  - Maximize profit or maximize social welfare

- Usually computationally difficult
  - NP hard and no good (ex ante) bounds on approximability
Scenario: procurement auction with supply curves

- Buyer wishes to purchase multiple items in large quantities to meet long-term need
- Must follow business requirements on allowable trades
- Food manufacturer

- Determine how much of each item to buy from the suppliers
  - Demand and business constraints are met
  - Cost is minimized

- Sellers provide price-quantity curves
  - Additive separable
  - Piece-wise linear
Examples of supply curves

- “volume discount”: unit price curve is decreasing step function
- continuous
- concave

- may be discontinuous
- may have decreasing slopes
- any curve can be approximated by piece-wise linear curves
A small example for one good

- Procurer needs 60 units
- A greedy algorithm results in allocation 40, 20 (green points)
- Optimal solution is 30, 30 (blue points)
Examples of business requirements

- Lower and upper limits on number of winning suppliers
  - Relying on too few suppliers is risky
  - Too many winners increase overhead
- Lower and upper limits on the total quantity allocated to a winning supplier
- Lower and upper limits on the quantity per item allocated to a winning supplier
  - Too small allocated quantity discourages suppliers
  - Too large allocated quantity makes the buyer dependent on particular suppliers

- Business requirements result in interdependencies between items => need to trade items simultaneously
Why is this a combinatorial auction?

- Define supply pattern as an array of quantities \( s = (a^s_1, \ldots, a^s_K) \).
- Pattern is feasible for supplier if it satisfies all supplier-specific business requirements:
  - bounds on quantity for item \( k \) supplied by \( j \): \( l^j_k, u^j_k \)
    - Can be handled by “trimming” the supply curve
  - bounds on total amount supplied by \( j \):
  
    \[
    l^j \leq \sum_k a^s_k \leq u^j
    \]
  - denote set for supplier \( j \) by \( S^j \)
- Cost of supply pattern for supplier \( j \) is
  
    \[
    p^j(s) = \sum_k p^j_k(s)
    \]

Number of feasible patterns for a supplier might be exponential!
Why is this a combinatorial auction?

- Supplier’s bid is represented by the XOR of patterns
- Business requirements that apply across agents are added as side constraints
- Multi-unit reverse combinatorial auction with patterns as bundles

\[
\begin{align*}
\min & \sum_j \sum_{s \in S_j} p_j(s) y^s \\
\text{s.t.} & \sum_j \sum_{s \in S_j} a_k^s y^s \geq Q_k \quad \forall k \\
& \sum_{s \in S_j} y^s \leq 1 \quad \forall j \\
& L \leq \sum_j \sum_{s \in S_j} y^s \leq U \\
& y^s \in \{0,1\} \quad s \in \bigcup_j S^j
\end{align*}
\]

- \( \min \) minimize total cost
- \( \text{satisfy demand} \)
- \( \text{at most one pattern per supplier} \)
- \( \text{number of winning suppliers is limited} \)
- \( \text{decision variables indicate which patterns are chosen} \)
How to solve: Branch-and-Price

Two difficulties:
- too many variables to enumerate all before optimization
- integrality requirement on variables

Method of choice: Branch-and-Price
- Branch-and-Bound backbone…
  - Solve model with variables *relaxed from integral to continuous*
  - If solution is not integral subdivide the feasible region (cut off frac opt)
  - Maintain best integral soln found
  - Branches provably w/o improving soln are pruned
- …with generating patterns in each search tree node as needed
  - Start with an initial set of patterns
  - Generate patterns that improve objective
  - Branch if no patterns can be generated
Processing one node of the search tree

- Search tree nodes in a queue (initially populated with root only)
- Best integral solution found is maintained globally
- Main tasks: int soln heuristics, pattern generation, branching
- Rest of the tasks are taken care of BCP framework (coin-or.org)
I: Integer solution heuristics

- Relaxed problem results in solution with fractional-valued patterns (primal solution to the LP)
- Goal is to find a solution with integer-valued patterns whose value is close to the value of the fractional solution
- Accomplished in two steps:
  - Rounding heuristics: construct solution with integer-valued patterns
    - Idea: “weight” patterns with their respective solution value; combine patterns with these weights for each supplier
  - Local improvement heuristics: improve an integer-valued solution
    - Idea: look for opportunities where some suppliers could form a “circle” and swap around a small quantity of some items while maintaining the feasibility of the patterns and the solution itself
I: Rounding heuristics

- Weight of pattern: corresponding solution value $y^s$
- Make sure total weight for each supplier is either one or zero
  - Interpreted as supplier is a winner or not
  - Can be obtained by solving an easier Mixed Integer Program
- Create a single pattern for each winner as weighted combination of his patterns
  - Will be a feasible pattern if lower bounds on supplied quantities are zeros and the price curve has no discontinuities
  - Solution consisting of these patterns will be feasible
- Problem: value of solution will be far from value of fractional soln $\rightarrow$ local improvement heuristics
I: Local improvement heuristics

- Given a set of winners and their patterns, modify the entries in the patterns to obtain a better (less costly) solution.

If total supplied amounts are fixed: transportation problem with
- Non-linear edge costs
- Capacity limits on the edges
- NP-hard (unless cost fn is convex)
  - Here: concave fns (volume discount)
- Given patterns correspond to a feasible solution of this transportation problem
- Find improving solution by looking for negative cost cycles (circulation) in the residual graph and pushing flow around them
  (disclaimer: need some additional tricks b/c of non-linearity of cost functions – see tech report for details)
II: Pattern generation for a supplier

- Relaxed problem also yields price information (dual solution)
- For each supplier find pattern(s) that would give a positive surplus or prove that none exists; add patterns to relaxed problem
  - i.e., find column with smallest reduced cost
- Pattern generator separately for each supplier
  - Don’t need to be identical (different feasibility requirements)

Constraints added to relaxed model affect pattern generators through constants

Constraints added to generators affect relaxed model indirectly through the patterns
II: Pattern generation for a supplier

Any value in a piece-wise linear function’s domain can be represented as a convex combination of two neighboring breakpoints:

- That is, weights in convex combination form an SOS Type 2 set
- Include dummy breakpoints at discontinuities

\[ x = \sum_i \lambda_i b_i \]
\[ \sum_i \lambda_i = 1 \]
\[ \lambda_i \text{- s form an SOS Type 2 set} \]

The value of the fn can be expressed as the same convex combination:

\[ p(x) = \sum_i \lambda_i p(b_i) \]
II: Pattern generation for a supplier

- The unknowns are the entries in the pattern we seek
- Consider the breakpoint representation for each item this supplier bids on
- Represent each entry in the pattern with the set of weights corresponding to the breakpoints
- Minimize reduced cost so that pattern is feasible:

\[
\min \sum_k \sum_i \left( p^j_k (b^j_{ki}) - \pi^j_k b^j_{ki} \right) \lambda^j_{ki} - \rho^j_j - \tau
\]

\[
l^j_j \leq \sum_k \sum_i b^i_{ki} \lambda^j_{ki} \leq u^j_j \quad \leftarrow \text{supplied amount between bounds}
\]

\[
\sum_i \lambda^j_{ki} = 1, \quad \forall k
\]

\[
\lambda^j_{ki} - \text{s form an SOS Type 2 set, } \forall k
\]

\[\pi^j_k, \rho^j_j, \tau\]

Dual prices from relaxed problem
III: Branching

- When no more patterns can be found at a search tree node the feasible region is subdivided
- Traditional (variable) branching:
  - For a variable at non-integral value: 2 branches, set variable to 0 or 1
  - 1-branch sets this supplier to be a winner but 0-branch carries little additional information (feasible region is split into uneven chunks)
  - Pattern generation must avoid a set of “forbidden” patterns – this is difficult to achieve
- Pattern generation and branching must be consistent
- Branching should split feasible region more-or-less equally between children
- So what should we branch on?
III: Branching

If there are suppliers for whom total weight of patterns is not one or zero then branch on whether supplier is a winner or not
  - Set supplier’s XOR constraint to =1 or to =0
  - Effect on pattern generation: do not generate patterns for supplier in 0-branch

Otherwise find a supplier and an item so that in two of the supplier’s patterns the item is sold in quantities with different unit price (i.e., different intervals)
  - Branch on what unit price the item should have
  - Branches are specified by new bounds on supplied quantity – pattern generation is the same problem

If neither of above: weighted combination of patterns gives optimal solution
III: Branching

… find a supplier and an item so that in two of the supplier’s patterns the item is sold in quantities for different unit price (i.e., different intervals)

- Branch on what unit price the item should have
- Branches are specified by new bounds on supplied quantity – pattern generation is the same problem

If neither of above: weighted combination of patterns gives optimal solution
Comparison with naïve model

- Naïve model (multiple choice knapsack)
  - Variables specify the amount purchased from each supplier of each item, decision variables specify which suppliers are winners
  - Constraints for meeting demand, sets of constraints for each business requirement, constraints to define variables
  - New business constraints may require introduction of new variables
  - Having different requirements for suppliers is complicated to model
  - Solve to optimality with a commercial solver (CPLEX)

- We achieve:
  - Stronger lower bounds and about the same integrality gap in a few seconds (even before branching) than naïve model in 10 minutes
  - Our model scales much better, most large problems take <2 mins, while naïve formulation almost always exhausts available
Test data generation

- Implemented problem generator
  - Data sets are assured to be feasible
  - Tightness can be adjusted via parameters
- Number of suppliers 15-75, number of items 10-60
- 4 different tightness settings
- Solve problems without limit on the number of winning suppliers
- Then make problems more difficult by setting upper limit on winning suppliers to 2 less than one obtained above
- Run commercial solver on naïve formulation and evaluating the root node for the new formulation (10 mins limit)
Experimental results

![Graph showing experimental results](image_url)

- **Heuristics (present work)**
- **Optimal (Davenport et al, 2001)**
- **Average (heuristics)**
Experimental results

Upper Bound on Optimality Gap Vs Number of Suppliers
Future directions

- Warmstart next round with solution from previous round
- Technique can be applied to a variety of problems
  - Map out other industries and other business requirements
- Now that a robust solution method is available construct experiments to test impact of solving approximately on incentive compatibility
Technical report available from