Taming the Computational Complexity of Combinatorial Auctions

Kevin Leyton-Brown
Yoav Shoham

Overview

1. **Problem Statement**
2. CASS
3. Experimental Results
4. Conclusions

Combinatorial Auctions

- Agents often desire goods more in combination with other goods than separately
  - Example: two pieces of adjacent property
- Combinatorial Auctions: mechanisms that allow agents to explicitly indicate complementarities
  - Multiple goods are auctioned simultaneously
  - Bidders place as many bids as they want
  - Each bid may claim any number of goods
- Agents assume less risk than in sequential auctions
  - The auctioneer can hope to achieve higher revenues and/or greater social welfare

Problem Statement

- Determine the winners of a combinatorial auction
  - Given a set of bids on bundles of goods, find a subset containing non-conflicting bids that maximizes revenue
  - This procedure can be used as a building block for more complex combinatorial auction mechanisms
    - e.g., the Generalized Vickrey Auction mechanism
- Unfortunately, even this building block is an NP-complete problem
- Finding optimal allocations remains desirable
  - properties like truth revelation may not hold with approximation
  - problems up to a certain size will be tractable

Substitutability

- Sometimes bidders will pay *less* for combinations of goods than the sum of what they would pay for each good individually
  - e.g., copies of the same book
- A bidder submits: (\$20, $\{g\}$); (\$20, $\{h\}$); (\$30, $\{g,h\}$)
  - $\{g\}$ and $\{h\}$ would be the winning bids: the bidder would be charged \$40 instead of \$30
- Dummy goods:
  - The bidder submits: (\$20, $\{g,d\}$), (\$20, $\{h,d\}$), and (\$30, $\{g,h\}$) where d is a new, unique dummy good
  - The first two bids now name the same good and so will never be allocated together

Overview

1. Problem Statement
2. **CASS**
3. Experimental Results
4. Conclusions

CASS: Introduction

- CASS – **C**ombinatorial **A**uction **S**tructured **S**earch
- CASS considers fewer partial allocations than a naïve DFS:
  - **structure the search space:** consider fewer conflicting bids
  - **pruning:** use context from the search structure to generate close overestimates of total revenue
  - **ordering heuristics:** capitalize on this structure to speed searching and improve anytime performance
- CASS has low memory demands
  - only stores nodes that are part of current allocation (# goods)
  - most memory is used for pruning tables
  - average 10-20 MB used for problems discussed today
- Originally we proposed two algorithms, now CASS is always faster

Naïve Depth-First Search

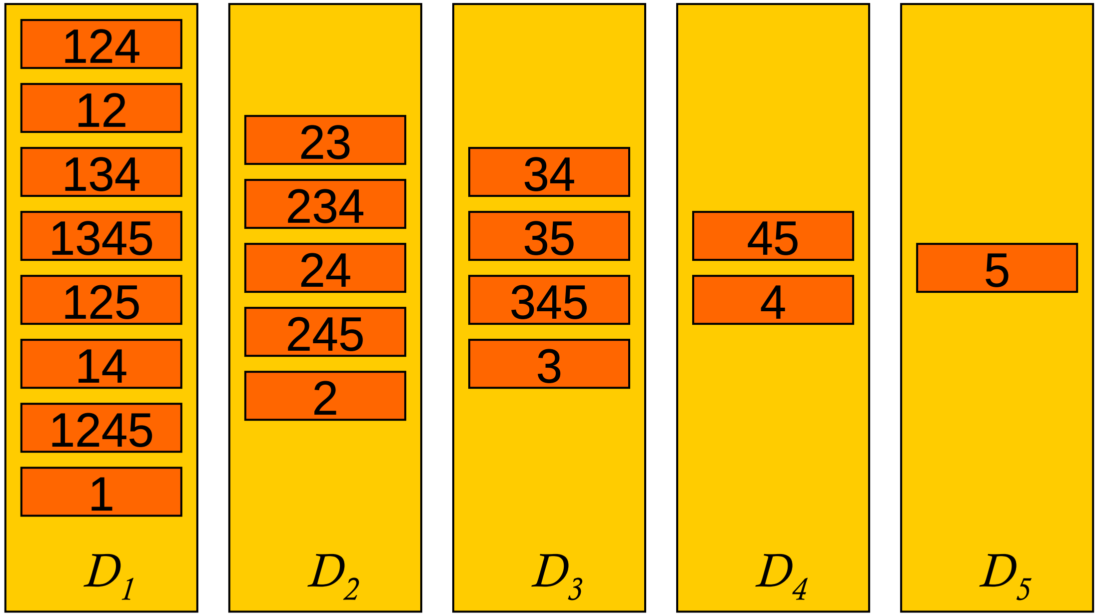
- bids are tuples: (a binary set of goods, a price)
- nodes are partial allocations (sums of bids)
- start node: empty set (no goods, $0)
- transitions between nodes: add one bid to the partial allocation
  - only add non-conflicting bids (bids whose intersection with the current partial allocation is empty)
- terminal node: no non-conflicting bids exist
  - the terminal node with the highest revenue is the optimal allocation

CASS Improvement #1: Preprocessing

1. Remove dominated bids
   - If there exist bids $b_k = (p_k, G_k)$ and $b_l = (p_l, G_l)$ such that $p_l \geq p_k$ and $G_l \subseteq G_k$, then remove b_k
     - Two bids for the same bundle of goods with different prices
     - One bundle is a a strict subset of another and has a higher price
2. For each good g, if there is no bid $b=(x, \{g\})$, add a dummy bid $b=(0, \{g\})$
   - This ensures that the optimal set of bids will name every good, even if some goods are not actually allocated

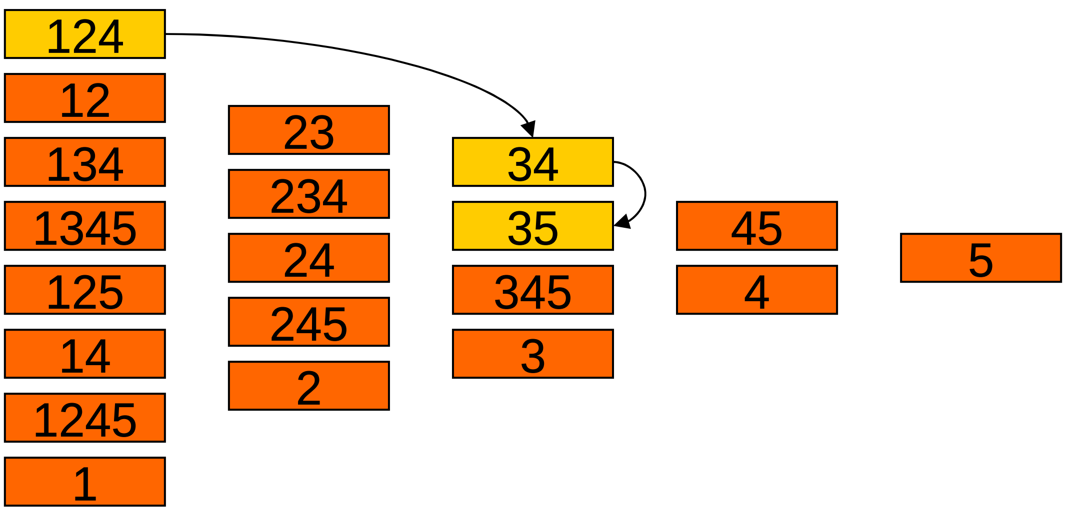
CASS Improvement #2: Bins

- Structure the search space to reduce the number of infeasible allocations that are considered
  - Partition bids into bins, D_i, containing all bids b where good $i \in G_b$ and for all $j < i, j \notin G_b$
  - Add only one bid from each bin



CASS Improvement #3: Skipping Bins

- When considering bin D_i, if good $j > i$ is already part of the allocation then do not consider any of the bids in D_j
  - All the bids in D_j are guaranteed to conflict with our allocation
- In general, instead of considering each bin in turn, skip to D_k where $k \notin G(F)$ and $\forall i<k, I \in G(F)$



CASS Improvement #4: Pruning

- Backtrack when it is impossible to add bids to the current allocation to achieve more revenue than the current best allocation
- Revenue overestimate function $o(g,i,F)$
  - an overestimate of the revenue that can be achieved with good g, searching from bin i with current partial allocation F
    - an admissible heuristic
  - precompute lists for all g, i:
    - all bids that contain good g and appear in bin i or beyond
    - sorted in descending order of average price per bid (APPB)
  - return APPB of the first bid in the list that doesn't conflict with F
- Backtrack at any point during the search if $revenue(F) + \sum_{g \notin F} o(g,i,F) \leq revenue(best\_allocation)$

CASS Improvement #5: Good Ordering Heuristic

- Good ordering: what good will be numbered #1, #2...
- Goal: reduce branching factor at the top of the tree
  - pruning will often occur before bins with a higher branching factor are reached
- Ordering of goods:
  - Sort goods in ascending order of score,

$$score(g) := \frac{number\ of\ bids\ containing\ g}{average\ length\ of\ bids\ containing\ g}$$

  - more bids → more branching
  - longer bids → shallower search

CASS Improvement #6: Bid Ordering Heuristic

- Finding good allocations quickly:
  1. Makes pruning more effective
  2. Is useful if anytime performance is important

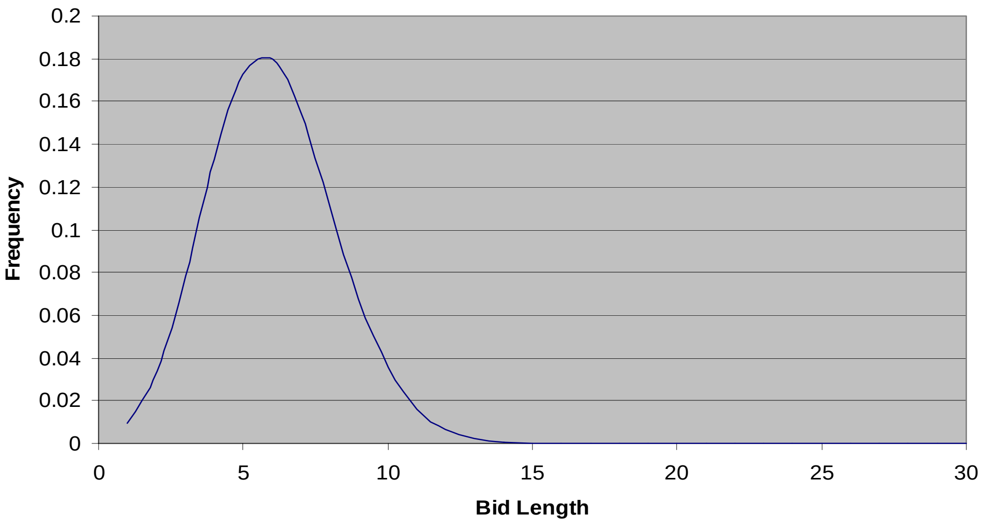
- Ordering of bids in each bin:
  - Sort bids in descending order of average price per good
  - More promising bids will be encountered earlier in the search

Overview

1. Problem Statement
2. CASS
3. **Experimental Results**
4. Conclusions

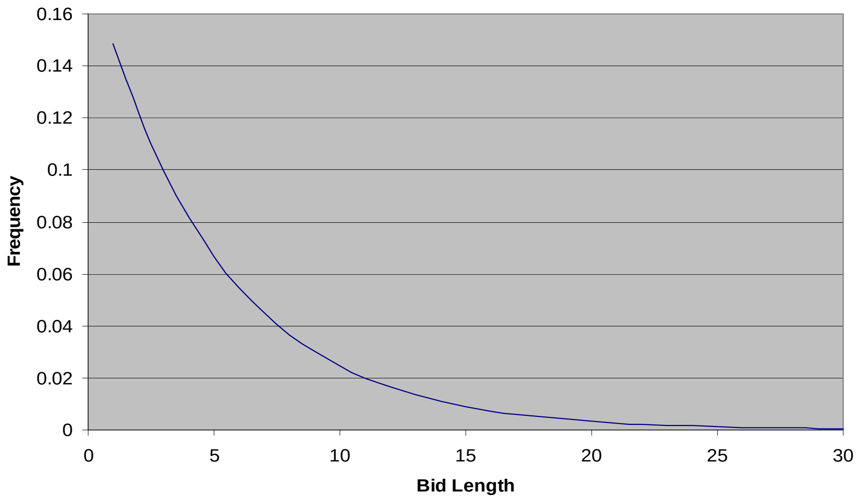
Experimental Results: Data Distributions

- There is little or no real data available, so we drew bids randomly from specific distributions
- Binomial: $f_b(n) = \dfrac{p^n (1-p)^{N-n} N!}{n!(N-n)!}, p = 0.2$
  - The probability of each good being included in a given bid is independent of which other goods are included



Experimental Results: Data Distributions

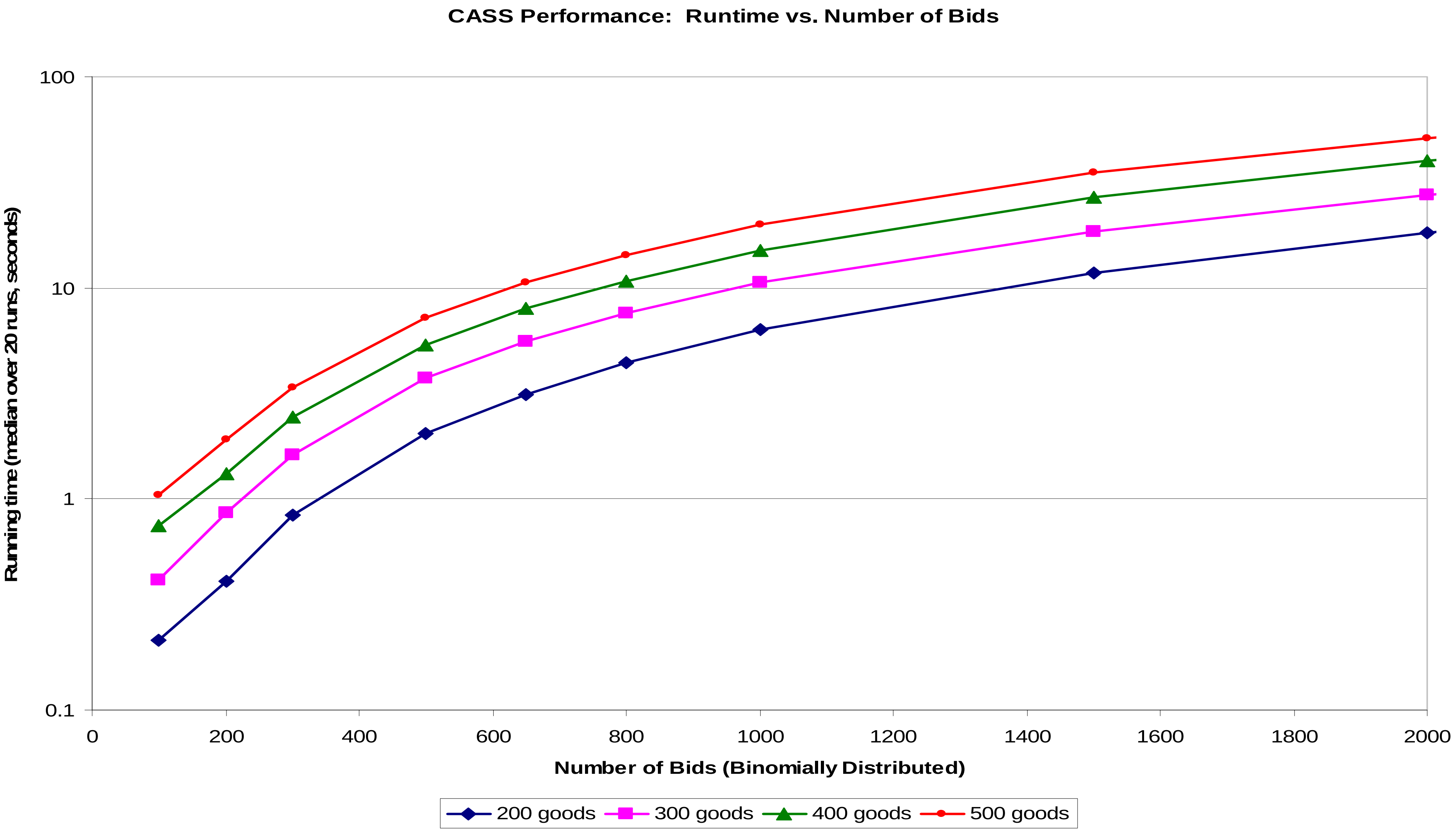
- Binomial is fairly easy to analyze, but not very realistic
  - in a real auction, we expect mostly short bids
  - harder → more bids must be combined in an allocation
- Exponential: $f_e(n) = Ce^{-x/p}$, $p = 5$
  - a bid for $n+1$ goods appears $e^{-1/p}$ times less often than a bid for n goods.



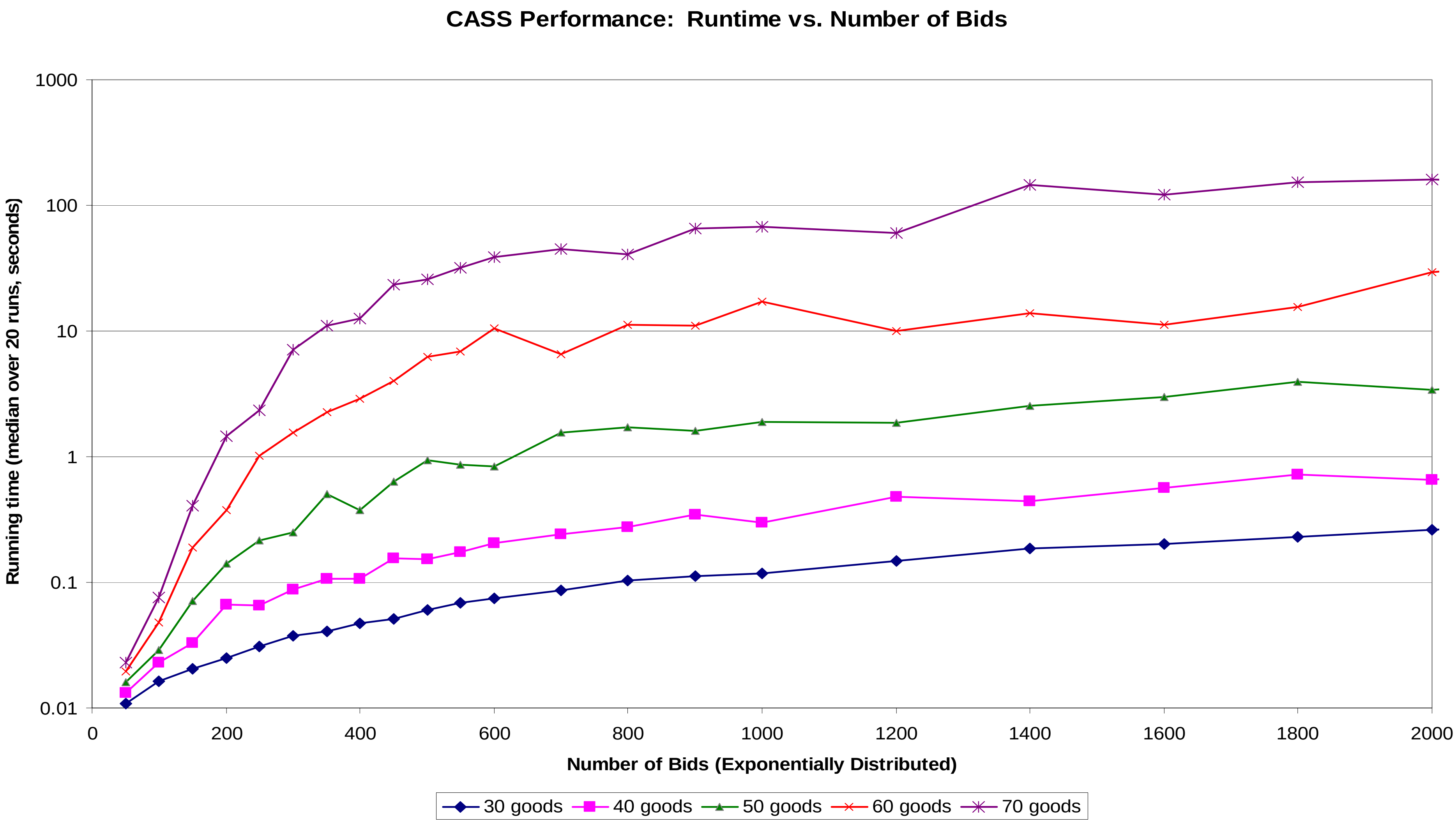
Experimental Results: Data Distributions

- Distribution of prices is also very important
  - pruning is done on the basis of price
- Prices of bids for n goods is uniformly distributed between $[n(1-d), n(1+d)]$, $d = 0.5$
  - prices cluster around a "natural" average price per bid, and deviate by a random amount
  - if prices were completely random, the pruning algorithm would have more of an advantage

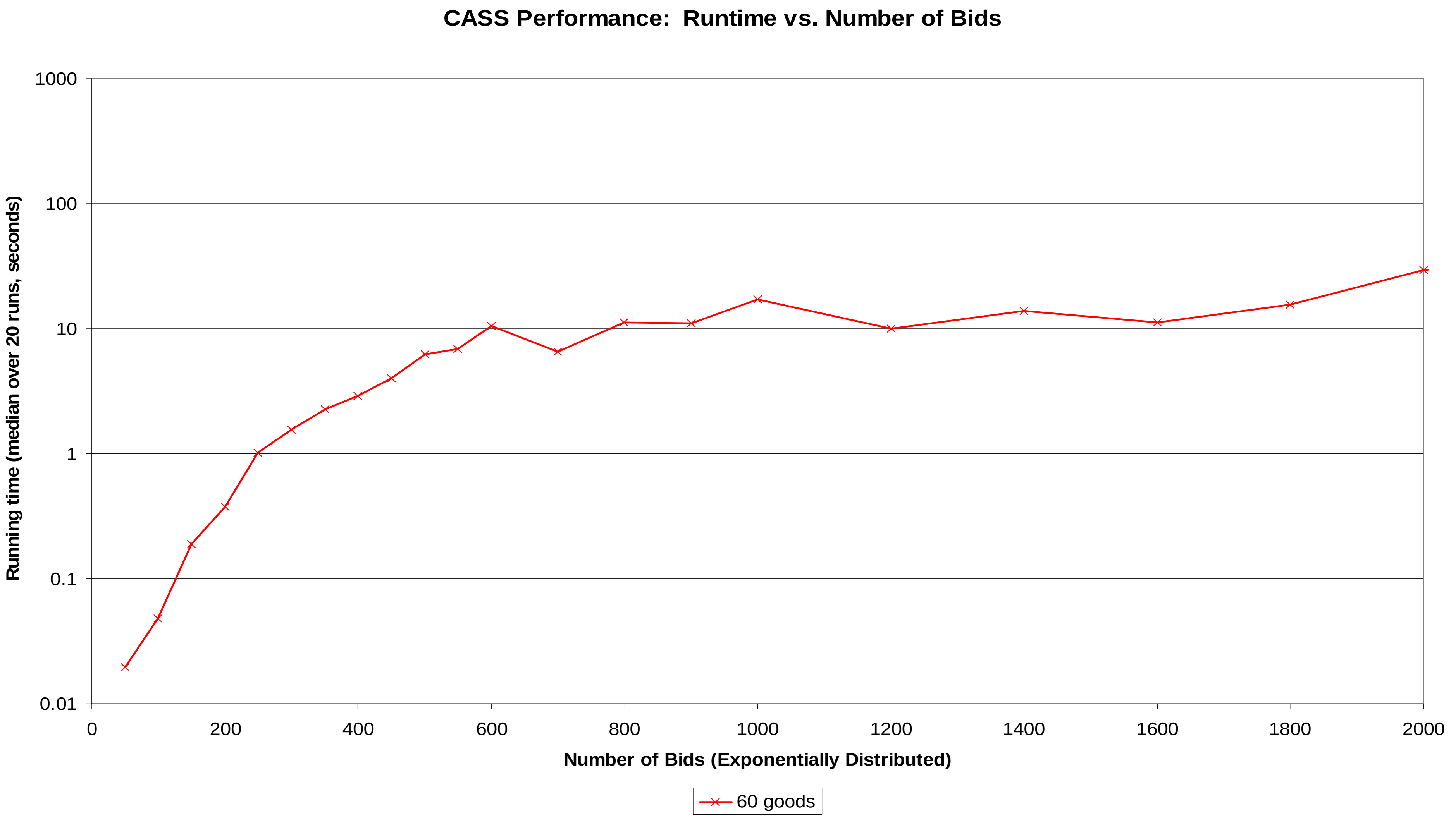
Experimental Results: Running Time (Binomial)



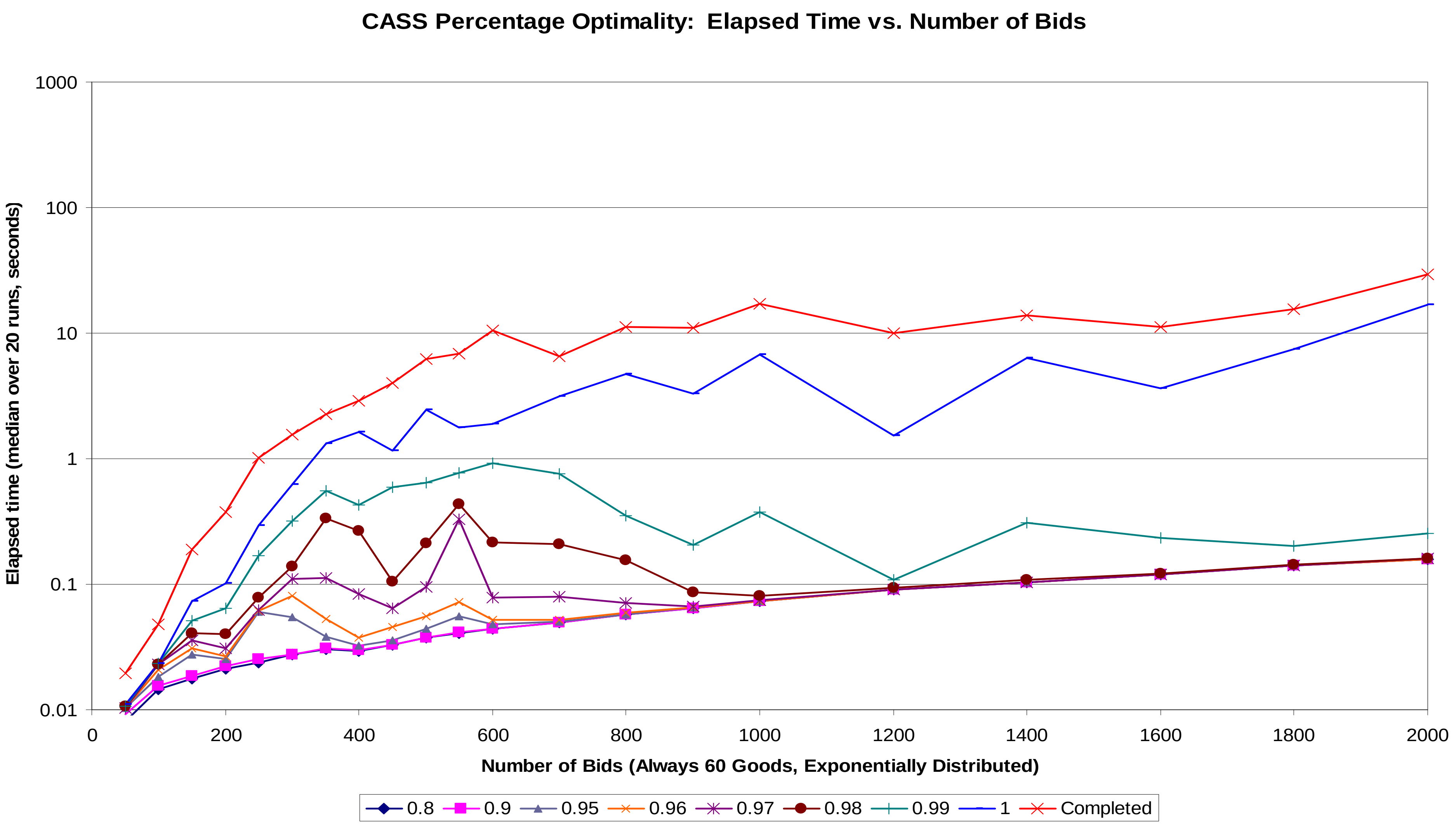
Experimental Results: Running Time (Exp.)



Experimental Results: Running Time (Exp.)



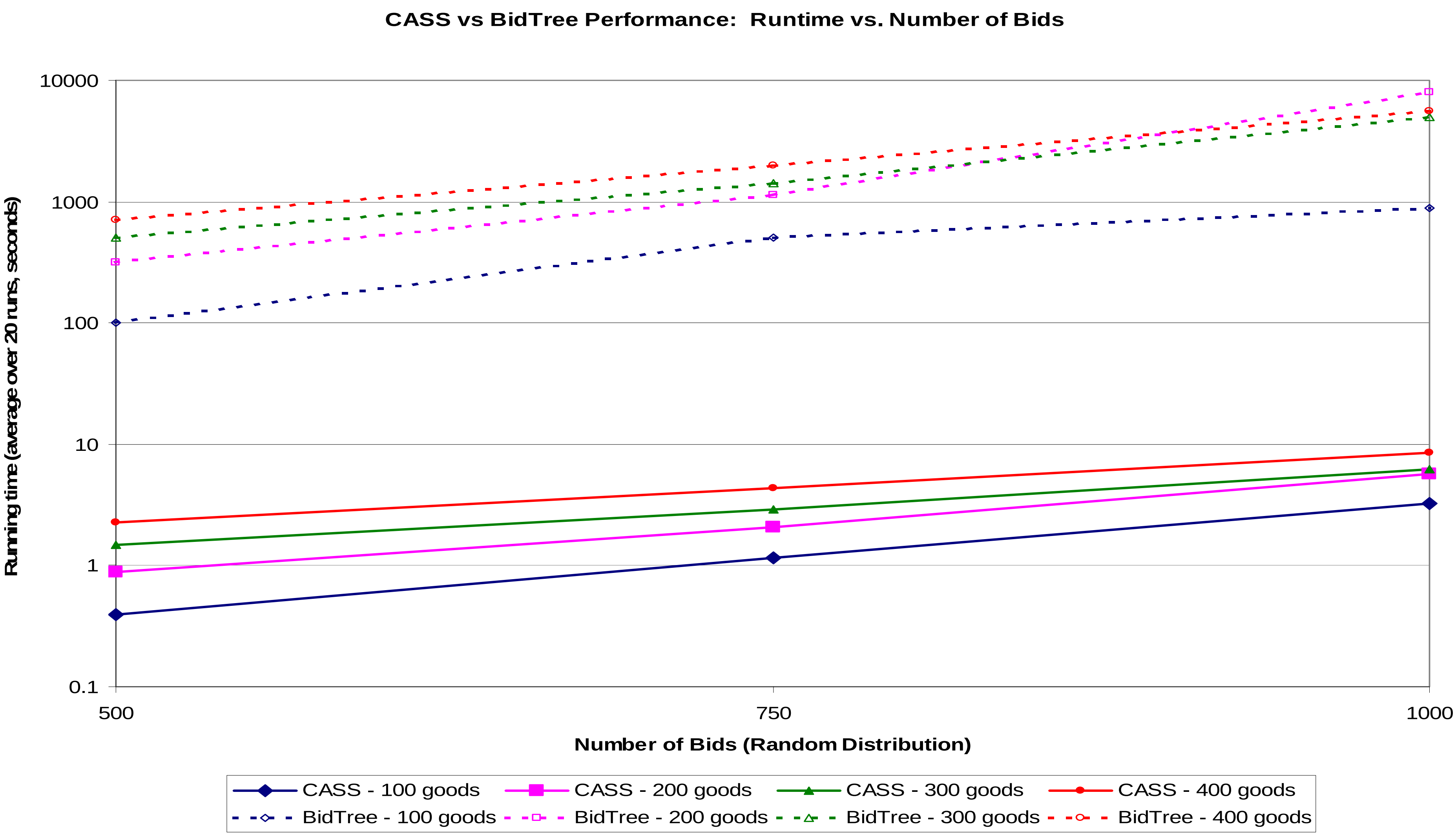
Experimental Results: Anytime Performance (Exp)



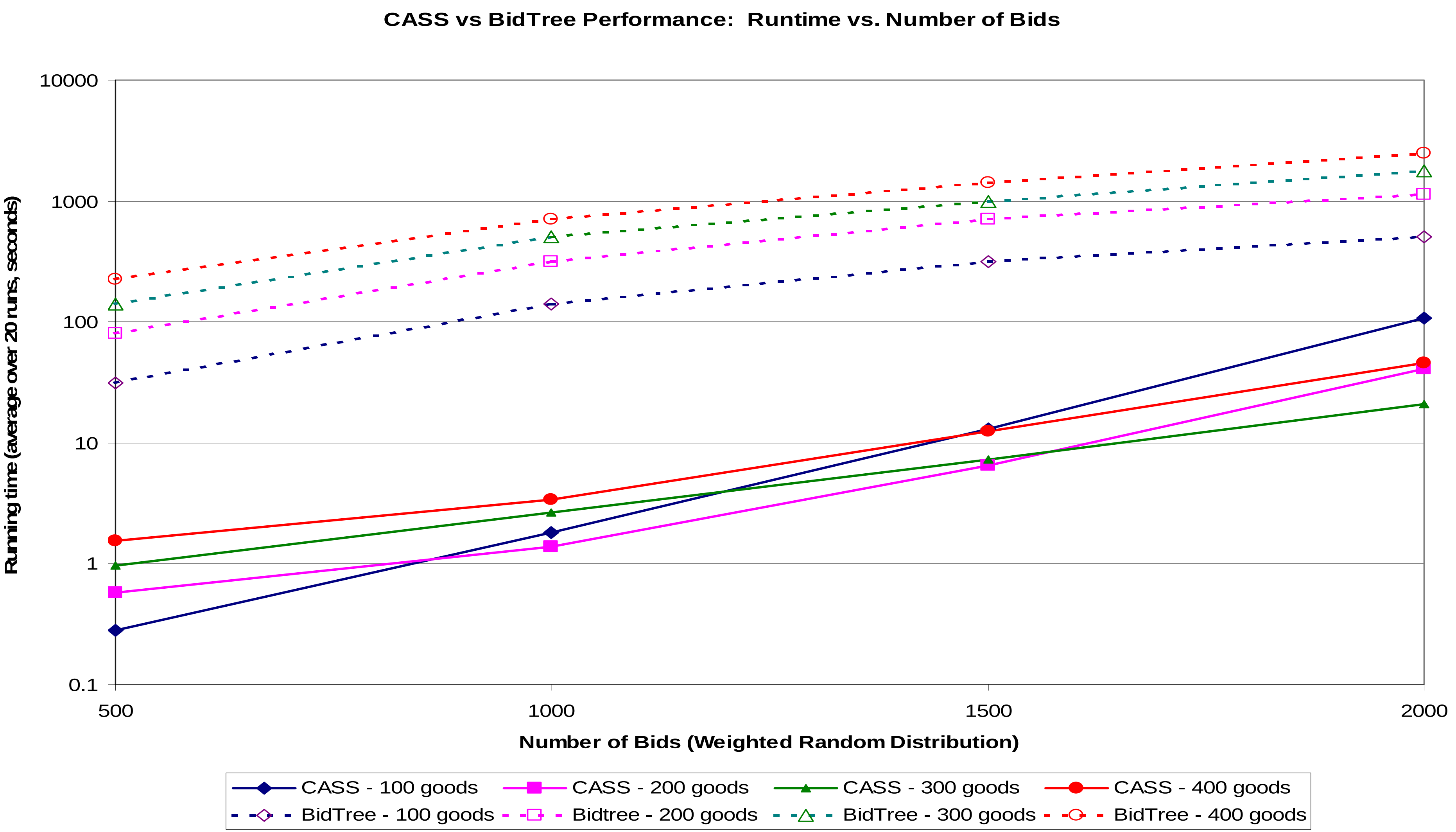
Sandholm's BidTree Algorithm

- Presents results for four different distributions:
  - Random Distribution:
    - Select the number of goods, N, in a given bid (uniform random)
    - Uniquely choose the goods
    - Price: uniform random between [0, 1]
  - Weighted Random Distribution:
    - Same as above, but price is [0, N]
  - Uniform Distribution
    - All bids have same length (3 goods in this case)
    - Price: uniform random between [0, 1]
  - Decay Distribution
    - A given bid starts with one random good
    - Keep adding random unique goods with probability α
    - Price: uniform random between [0, N]

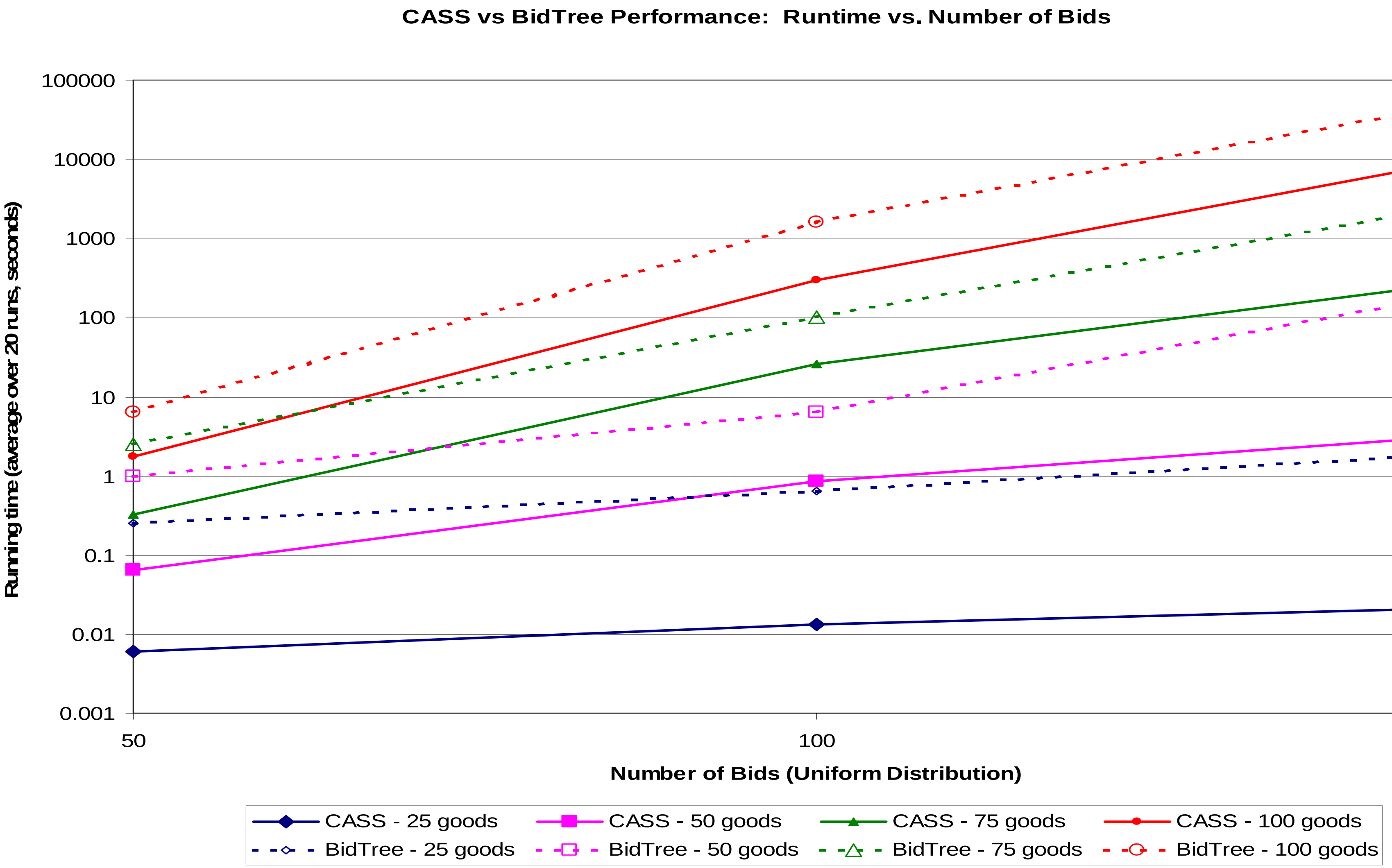
Experimental Results: Random Distribution



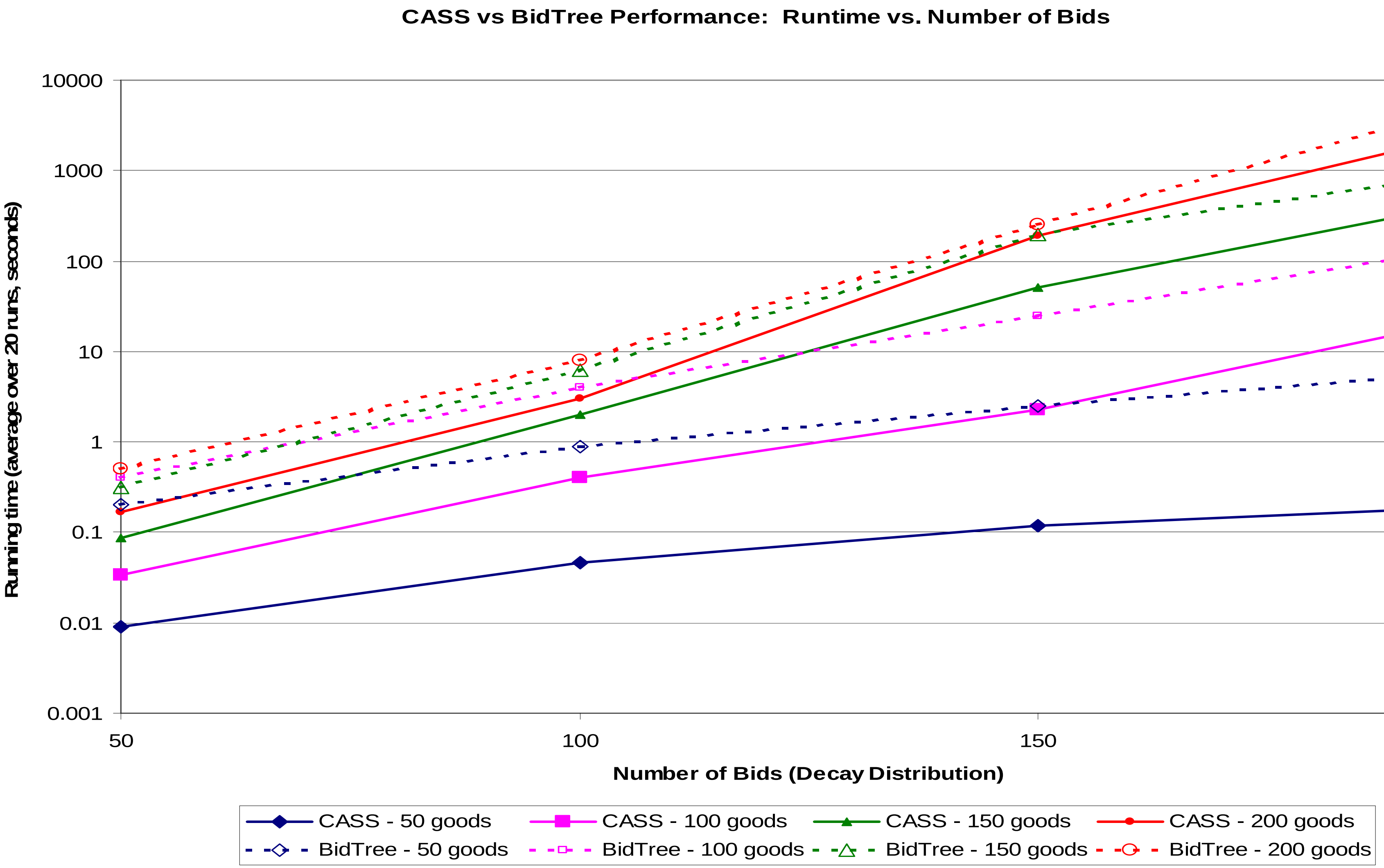
Experimental Results: Weighted Random Distribution



Experimental Results: Uniform Distribution



Experimental Results: Decay Distribution



Overview

1. Problem Statement
2. CASS
3. Experimental Results
4. **Conclusions**

Conclusions

- We have proposed an algorithm to mitigate the computational complexity of combinatorial auctions, which works surprisingly well on simulated data
  - determines optimal allocations in a small fraction of the time taken by a naïve DFS approach to solve the same problem
  - can find good approximate solutions quickly

Future Work

- Investigate the effects of different bin orderings and orderings of bids within bins
- Compare to other search techniques
  - integer programming
  - other combinatorial auction search techniques
- Experiments with real data (FCC auctions?)
- Caching: referenced in our paper, but currently disabled
- Divisible/identical goods
  - some of our work on CASS is relevant to the new problem; much is not