An Algorithm for Multi-Unit Combinatorial Auctions

Kevin Leyton-Brown

Yoav Shoham

Moshe Tennenholtz

Computer Science Dept.

Stanford University

thanks also to Shobha Venkataraman

Combinatorial Auctions

- Mechanisms that allow bidders to explicitly indicate complementarities and substitutabilities
  - many goods are auctioned simultaneously
  - bids name an arbitrary bundle and a price offer
  - bidders may submit multiple bids
    - if desired, some bids may be mutually exclusive
    - otherwise, more than one of a bidder's bids may win
- Benefit: less risk for bidders
  - won't win a subset of a bundle for more than it is worth to them
  - can request multiple mutually-exclusive bundles
  - More efficient / higher revenue
    - no need to hedge bids or restrict bidding to a single bundle

Multi-Unit CA's

- Sometimes a set of goods are identical
  - traditionally, bidders have no way to compactly represent indifference between members of the set
    - instead, they must enumerate bundles between which they are indifferent
    - this can require a huge number of bids
- Multi-Unit CA
  - set of identical goods: a single multi-unit good
    - in general, consider all goods to have a fixed number of units
  - bids specify goods, number of units for each good, a price offer for the whole package

Winner Determination

- Auctioneer's task:
  - given a set of bids, find the revenue-maximizing subset of these bids allocating no more than the maximum number of units for each good
- We can handle XOR with "dummy goods"
  - unique virtual goods with one unit
  - add a dummy good to every bid in an XOR set
  - now at most one bid from each set can be satisfied
- Same winner-determination procedure used by:
  - first-price combinatorial auction
  - generalized Vickrey auction
  - various ascending auction mechanisms

Computational Problem

- Unfortunately, winner determination is NP-Hard, even with only one unit per good
  - Responses to intractability
    - approximation
    - restrict bids (tractable subcase)
    - find optimal solution anyway
- Benefits of finding optimal solution
  - constant-bounded approximation is still intractable
  - bidders' strategies affected by approximation
  - restriction can prevent bidders from expressing full preferences

Finding Optimal Solution

- All previously-published work on CA's has concerned single-unit case
- A natural solution: mixed-integer programming
  - rich history
  - commercial packages (CPLEX)

CAMUS

- **C**ombinatorial **A**uction **M**ulti-**U**nit **S**earch
  - branch and bound search
  - structure the search space
    - avoid considering impossible allocations
    - efficient upper-bound function for pruning
  - enhancements
    - preprocessing dominated bids
    - dynamic programming
    - caching to improve tightness of upper-bound
  - heuristics
    - maximize effectiveness of pruning: upper bound
    - find good allocations quickly: lower bound
- A generalization of our CASS algorithm (1999)

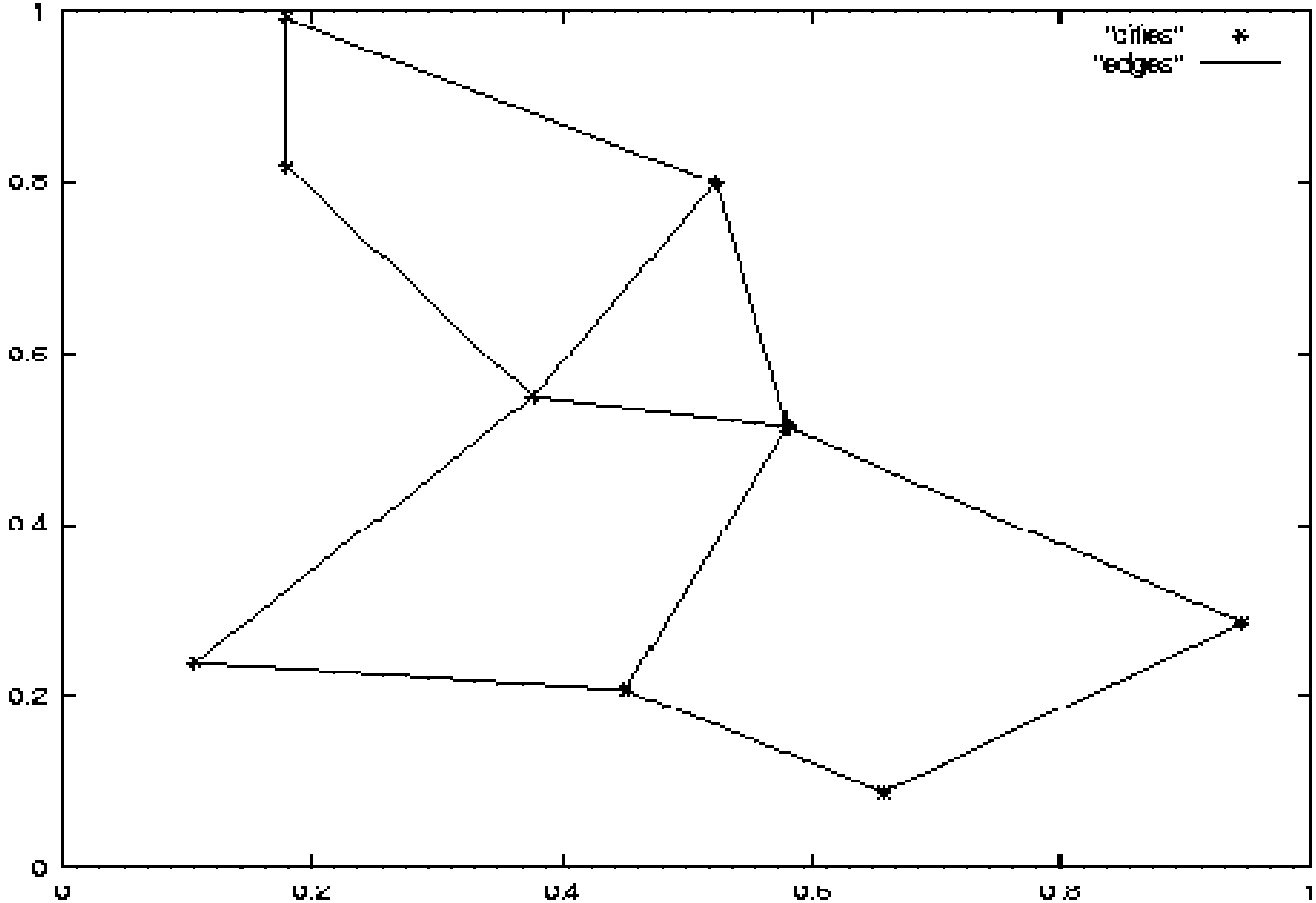
First: CAMUS/CPLEX comparison

- Necessary to use artificial data for testing
  - used a distribution from our new paper (to appear at EC-00)
  - aims to model bidding in real-world domains
- Railroad Shipping Domain: Railroad Graph
  - nodes: cities
  - edges: railroad link between cities
  - edge weights: link capacity

Railroad Distribution

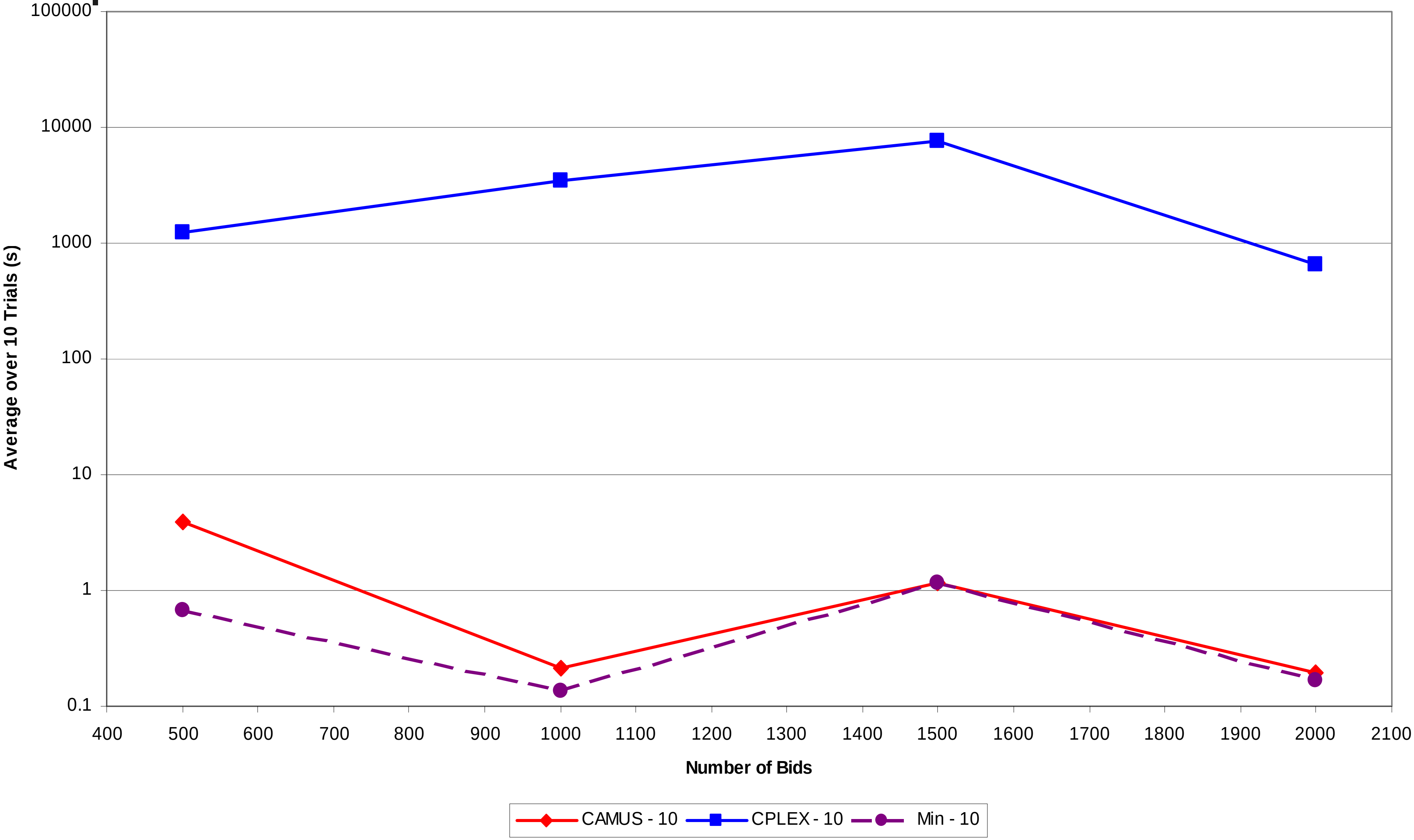
- Randomly generate a graph
  - random num units per edge: $[1, max\_units\_per\_good]$
- Create a new bidder
  - randomly choose start and end cities, number of units to ship
  - valuation for route: random proportional to the distance, superadditive in number of units
  - generate substitutable bids for all bundles of edges where valuation > cost of shipping (c * distance)
  - price offer: valuation – cost, rounded to integer

Railroad Distribution: Example

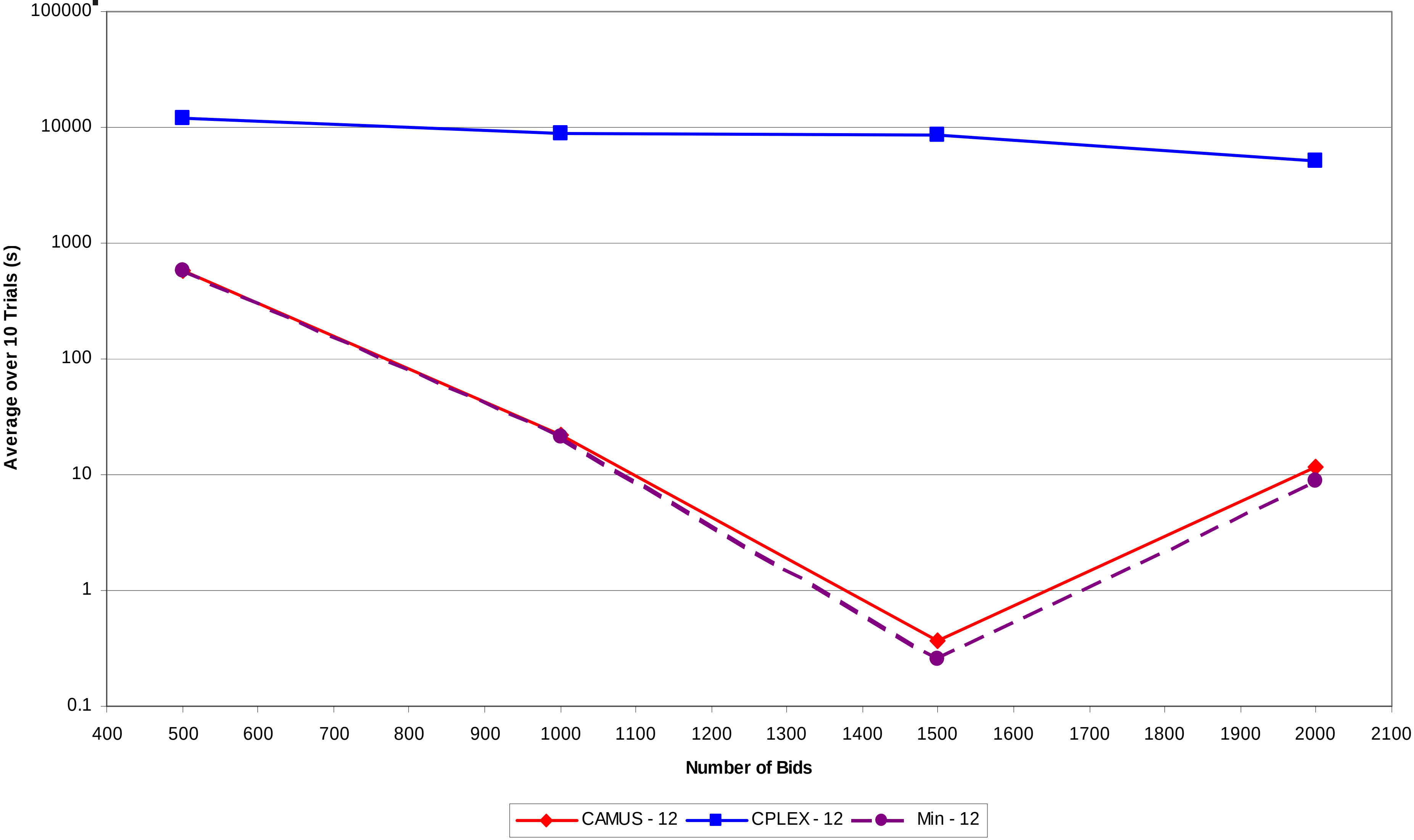


Parameters: num_cities = 5.3 * goods + 3.5, initial_connections = 2, building_penalty = 2.7, num_building_paths = $(num\_cities)^2/4$, shipping_cost_factor = 1.1, max_bid_set_size = 8, max_cap = 20, additivity = 0.2.

10 goods:
CAMUS, CPLEX, Min Performance



12 goods: CAMUS, CPLEX, Min Performance



CAMUS Implementation: Search

- Depth-First Search on allocations
  - begin with empty allocation
  - add bids to current partial allocation until complete; backtrack
- Branch and Bound Search
  - lower bound: best allocation observed so far
  - upper bound: revenue of current partial allocation + overestimate of revenue from unallocated units
  - when upper bound $\leq$ lower bound, backtrack

Structure the Search Space

- Partition the bids into bins
  - one bin for each good
  - each bid belongs to the bin corresponding to its lowest-order good
- After adding a bid, move to the bin for the lowest-order good with unallocated units
  - this may be the bin we just left (multi-unit!)
    - create a *subbin* of the current bin and keep searching
    - subbin: include only higher-order bids than the last bid chosen from this bin
  - any bids that we skip are guaranteed to conflict with the current partial allocation

Upper Bound Function $h(g,i,\pi)$

- An overestimate of the revenue that can be achieved from the remaining units of good g
  - given that the search is in bin i and has partial allocation π
  - precompute lists for all g, i:
    - each list: all bids for units of good g in bin i or beyond
    - sorted in descending order of average price per unit (APPU)
- Let b be first bid in list i that doesn't conflict with π
  - b's contribution to the overestimate: APPU(b) * $\min(units_i(b), units\_needed_i)$
  - if more units are still needed, keep moving down the list and find another non-conflicting bid; repeat
- Why does this work? Please see our paper...

Dominated Bids

- For each pair of bids (b_1, b_2), where:
  - $price(b_1) \geq price(b_2)$
  - for all goods j, $units_j(b_1) \leq units_j(b_2)$
- b_2 will not win unless b_1 also wins
  - store b_2 as a “child” of b_1
    - only consider adding b_2 after adding b_1
  - if $units_j(b_1) + units_j(b_2) \geq maxunits_j$ for any j
    - we will never add b_2: delete it

Dynamic Programming

- In some auctions, singleton bids will be relatively common
  - Additionally, singleton bids can be computationally expensive to consider: can lead to deep searches
- Dynamic programming preprocessing:
  - find the optimal set of singleton bids requesting from 1 to $maxunits_j$, for each good j
  - in search, only ever consider the optimal singleton set that consumes all remaining units of a good

Caching

- It is possible to allocate the same number of units of the same goods in more than one way
  - the search beyond this point is always the same
  - store the results of search in a hash table, then reuse them if we get to the same point again
    - most searches are pruned before they reach a full allocation, so we can't store the best allocation in the cache
  - use the cache to store upper bounds
    - only store the results that involved non-negligible cost to compute
    - cache upper bounds often tighter than $h(\,)$
  - cache can be seen as learning a better $h(\,)$
    - a tighter upper bound

Good-Ordering Heuristic

- designate as good #1 the good i that minimizes $(numbids_i \cdot maxunits_i) / (avgunits_i)$
  - minimize number of bids in low-order bins
    - reduce branching
  - minimize number of units of goods in low-order bins
    - move quickly past the first bins, where the pruning function is least informative
  - maximize total number of units requested by bids in low-order bins
    - move quickly to high-order bins
- remove bids involving good #1 and repeat for good #2, etc.

Bid-Ordering Heuristic

- Order bids within bin so we encounter most promising bids first
  - improve lower bound
- Sort bids b in descending order of $\mathrm{APPU}(b) + h(\pi \cup b)$
  - $\mathrm{APPU}(b)$ is a measure of b's promise
  - $h(\ )$ is a measure of how promising the unallocated units are, given partial allocation
    - This ordering is dynamic, because $h(\pi \cup b)$ depends on the past search

CAMUS vs. CPLEX

- The jury's still out
  - CAMUS outperforms CPLEX on the railroad distribution
  - we've seen other cases where CPLEX is better
  - what are the strengths of each approach?
- Choice of distribution is fundamental to testing
  - can we agree on distributions that capture the patterns we expect from real-world bidding?
  - *Towards a Universal Test Suite for Combinatorial Auctions*, http://robotics.stanford.edu/CATS
  - we'd love to get your feedback on this!

Conclusion

- CAMUS is a general-purpose algorithm for finding the winners of multi-unit combinatorial auctions
- A branch and bound search:
  - structuring the search space
  - preprocessing
  - dynamic programming
  - caching
  - heuristics for ordering goods and bids
- Promising performance when compared to CPLEX on our railroad distribution
  - more work needed to understand strengths and weaknesses of each approach on other real-world CA distributions