Fast and Distributed Mechanisms

Juho Hirvonen, Aalto University and HIIT
Helsinki Algorithms & Theory Days

Joint work with Sara Ranjbaran (Aalto)

Today's talk

- ***What is a mechanism?*** Why and when do we want a mechanism?
- ***The trouble with mechanisms:*** a constrained design space
- ***Designing local mechanisms***: how to implement mechanisms as fast distributed algorithms
- ***Open problems and future directions***

What is a mechanism?

- Mechanism is a ***game*** and an ***algorithm (ex: an auction)***
    - There is a set of *agents* that are strategic participants with ***private information (ex: valuation for the item to be assigned)***
    - Agents participate by strategically ***revealing some of their private information (ex: bid for the item)***
    - Based on the information revealed, the mechanism ***assigns an outcome (ex: give the item to the highest bidder)***
    - Each agent gains some ***utility*** based on the outcome ***(ex: winner gets item, others get nothing)***

Designing mechanisms

- Primary goal (default): ***Maximise total utility*** of the agents (*total welfare*)
    - To do this, the algorithm ***should somehow know the private inputs*** of the agents?
    - Solution concept: design ***truthful (incentive-compatible)*** mechanisms, where ***revealing the truth is a dominant strategy*** for the agents
- Sidenote: ***Revelation principle*** states that we can always consider mechanisms where only interaction is agents revealing their information

Example: Sealed bid auction

- Goal: assign a single indivisible item to the ***agent that values it the most*** (to maximise welfare)
- If we just ask the agents, ***each will bid infinity!*** There must be a cost
- The classic solution: assign to the ***highest bidder***, winner pays its own bid (***first-price auction***)
- Problem: to optimise its own utility, ***winner must guess the second-highest bid*** and bid just above it (instead of its true valuation)
  - Not truthful, difficult to participate in

Example: Sealed bid auction

- To fix this, we design a new auction that ”bids optimally” for the winner: the highest bidder gets the item and ***pays the second-highest bid*** (*second-price auction*)

- ***Truthful:***

    - If agent’s true valuation is the highest bid, it gets ***the same utility for every winning bid: true value - second price***

    - If agent’s true valuation is not the highest bid, ***overbidding results in larger payment than gained utility***

”The” truthful mechanism: Vickrey-Clarke-Groves (VCG)

- Second-price auction is a special case of the (only) ***general truthful mechanism*** (*with non-trivial guarantees)
- Each agent submits their private information, mechanism computes the assignment that ***maximises total utility (!)***
- The VCG-framework can be used for essentially any optimisation problem

The VCG mechanism

- Each agent gets their utility + a ***payment function p*** consisting of two parts:
  + *The total utility of all other agents* (the incentive)
  - *The total utility of all other agents* in the assignment that maximises this sum (normalisation)

- Normalisation term independent of agent's bid

- ***Utility + incentive = total utility***: should report the truth so that the algorithm can maximise with respect to it!

The problem with the VCG

- The VCG-mechanism is computationally infeasible for many interesting problems

- What if we just switch the optimal solution to the ***best efficiently computable solution?***

    - Unfortunately this does not work! Each agent wants to find the report that maximises total utility, and this ***might not be the truth!***

    - When all agents do this in parallel, behaviour is highly unpredictable (just like first-price auction)

Greedy mechanisms

- It is known that in certain settings, greedy algorithms can be turned into mechanisms
- Example (from our work): ***maximum weight independent set (MWIS)***
  - Each agent ***v*** is a node in a graph and has a (private) weight ***w(v)***
  - Utility: ***w(v)*** if chosen and no neighbour chosen, ***0*** otherwise
  - Goal is to ***find an independent set of large weight***

Simple greedy algorithm

- The following simple algorithm is known to compute a ***Δ-approximation*** (where Δ = maximum degree) [Sakai et al., 2003]

  Repeatedly pick the node with the largest weight into the set and remove it along with its neighbours

- ***Mechanism:*** each agent reports its weight, find an independent set using the greedy algorithm, replacing weights with what agents reported

- ***Payments:*** each selected agent pays the ***critical price*** = *the smallest bid that would have led to it being selected*

Truthfulness

- This is again the format of the second-price auction: the ***mechanism "bids optimally"*** for the winners
- The algorithm has to be ***monotone***: selected nodes are still selected with higher bids, and vice versa
- ***Pf.*** *If truth is above the critical price, any bid above it will produce the same (non-negative) utility. If truth is below the critical price, overbidding will lead to non-positive utility.*

Distributed setting

- Now each agent is an entity in a communication network
- Computation proceeds in synchronous rounds, and in each round each agent can exchange messages with its neighbours and update its state
- At some point each agent must announce its own output
- We aim for local algorithms: number of communication rounds much lower than the size of the network

A distributed mechanism

- To make the greedy algorithm distributed, instead of choosing the global maximum, ***choose all local maxima*** (approximation retained)

- The main loop of the greedy algorithm can easily be implemented in a distributed setting: agents keep checking if they are the local maximum, join if yes, and stop if a neighbour joins

- The issue is the running time: there might be a long chain of increasing values that takes ***O(n)*** rounds to resolve (fast sequential, slow distributed)

Dealing with long chains

- To deal with this, we ***discretise the values*** (i.e. round reported values to K allowed values)
- If done right, the mechanism is still truthful
- Loss in the quality of the solution depending on ***K***

Computing the payments

- If message sizes are not bounded, easy without overhead!
    - ***T***-round mechanism → Gather full ***T***-neighbourhood to each node, simulate mechanism with each possible bid to determine the critical price
- Non-trivial for bounded messages!
    - MWIS-mechanism: selected nodes determine the largest neighbour that is not already blocked by some other neighbour

Our work

- We present greedy distributed mechanisms for (weighted) independent set, dominating set, vertex cover, and coloring (their ”natural” interpretations as mechanism design tasks) + a general framework when such mechanisms exist (to be submitted soon)

- We also study stable matching: characterise the most general special case that has a fast distributed mechanism, show how to break ties fairly by sampling colourings

  - https://arxiv.org/abs/2402.16532

- First examples of local mechanisms (***Distributed Algorithmic Mechanism Design*** has focused on global problems such as routing or leader election)

Implementing distributed mechanisms

- One of the advantages of distributed mechanisms would be eliminating the need for a centralised entity that runs the whole mechanism
- In our analysis the ***algorithm itself is not game-theoretic***, only the reporting of the private information
- Can we make the execution also resilient against strategic behaviour?
  - E.g. agent might not pass on message correctly or delete them altogether

Greedy mechanisms and beyond

- Proving that a ”greedy” algorithm with a non-static score is monotone can be tricky
  - Lot of work in turning more convoluted greedy algorithms into mechanisms
- In the centralised setting there are other approaches such as linear programming
- Arguments from computational hardness of lying profitably: different arguments for the distributed setting

Distributed tie-breaking

- In sequential mechanisms tie-breaking is essentially trivial
- In distributed mechanisms, if not done carefully, it might blow up the running time (long chains of dependencies created by tie-breaking)
- We resolve this issue by computing a colouring: tie-breaking is consistent and length of chains is bounded by the number of colours
  - ***Tie-breaking problem:*** orient the conflict graph such that it is a DAG and the distance reachable along the orientation from any node is bounded