

# Algorithmic mechanism design

Vincent Conitzer  
conitzer@cs.duke.edu

# Algorithmic mechanism design

- Mechanisms should be accompanied by an efficient **algorithm** for computing the outcome
- May not be easy
  - E.g., using the Clarke (VCG) mechanism in combinatorial auctions requires solving the winner determination problem optimally
- If the mechanism's outcomes are too hard to compute, we may need a different mechanism
- **Algorithmic mechanism design** [Nisan & Ronen STOC 99] = simultaneous design of mechanism and algorithm for computing its outcomes
  - Given a mechanism, is there an efficient algorithm for computing its outcomes?
  - Given an algorithm for choosing outcomes, can we make it incentive compatible (e.g., using payments)?

# Combinatorial auctions: mechanisms that solve the winner determination problem approximately

- Running Clarke mechanism using approximation algorithms for WDP is generally not strategy-proof
- Assume bidders are single-minded (only want a single bundle)
- A greedy strategy-proof mechanism [Lehmann, O'Callaghan, Shoham JACM 03]:

1. Sort bids by (value/bundle size)
2. Accept greedily starting from top

✓	{a}, 11	}	$1 \cdot (18/2) = 9$
✓	{b, c}, 20		
✗	{a, d}, 18	}	$2 \cdot (7/1) = 14$
✗	{a, c}, 16		
✗	{c}, 7		
✓	{d}, 6		0

3. Winning bid pays bundle size times (value/bundle size) of first bid forced out by the winning bid

Worst-case approximation ratio = (#items)

Can get a better approximation ratio,  $\sqrt{\text{\#items}}$ , by sorting by  $\text{value}/\sqrt{\text{bundle size}}$

# Clarke-type payments with same approximation algorithm do not work

✓ {a}, 11

✓ {b, c}, 20

✗ {a, d}, 18

✗ {a, c}, 16

✗ {c}, 7

✓ {d}, 6

Total value to  
bidders other  
than the {a}  
bidder: 26

✓ {b, c}, 20

✓ {a, d}, 18

✗ {a, c}, 16

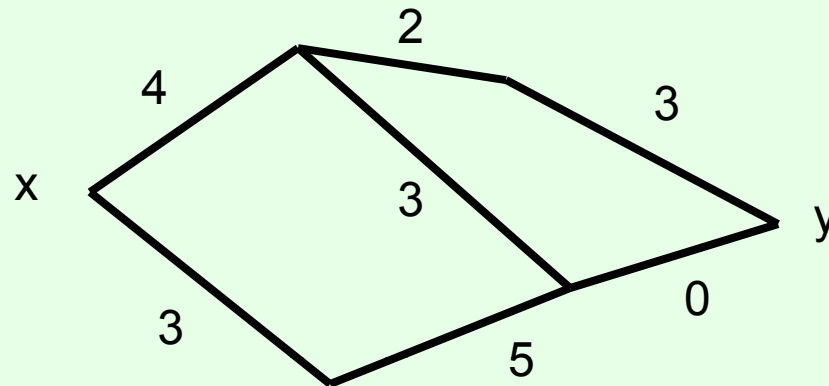
✗ {c}, 7

✗ {d}, 6

Total value: 38

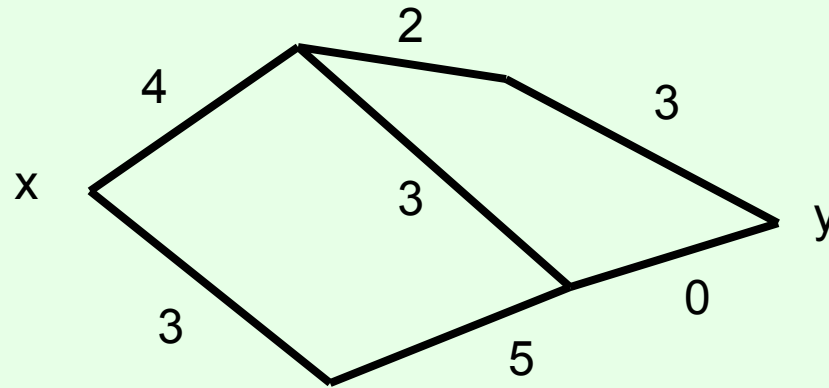
{a} bidder should  
pay  $38 - 26 = 12$ ,  
more than her  
valuation!

# A shortest path/combinatorial reverse auction problem [Nisan & Ronen STOC 99]

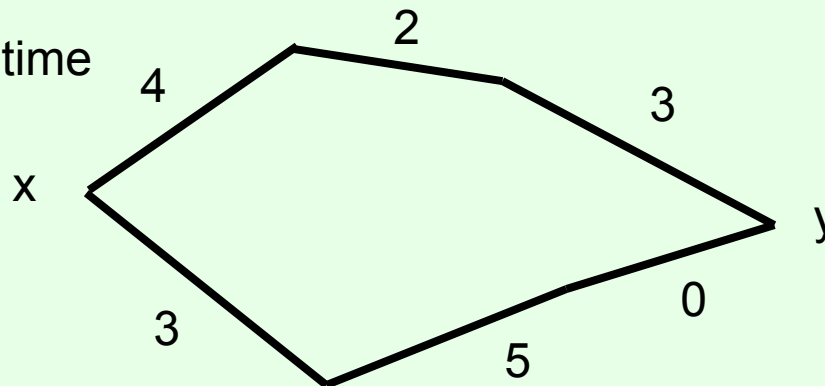


- Someone wants to buy edges that constitute a path from x to y
- Each edge e has a separate owner, and that owner submits (bids) a cost  $c_e$  for it
- Goal:
  - buy the shortest path (= path with minimum total weight),
  - pay every edge according to Clarke mechanism
    - no incentive to misreport costs
- That is, an edge e on the shortest path is paid  $(\text{cost of shortest path without } e) - (\text{cost of shortest path with } e) + c_e$

# Computing Clarke payments



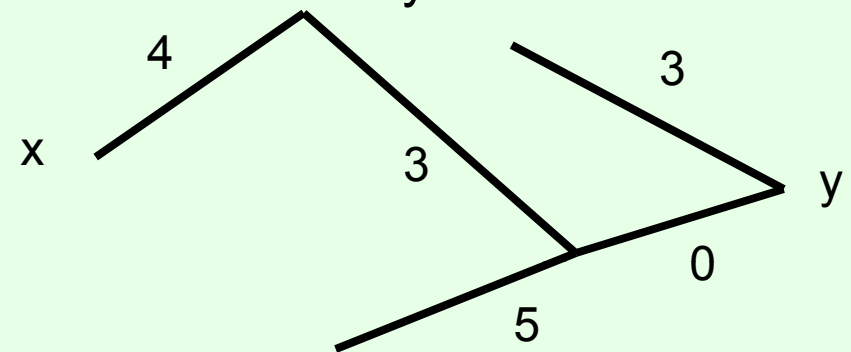
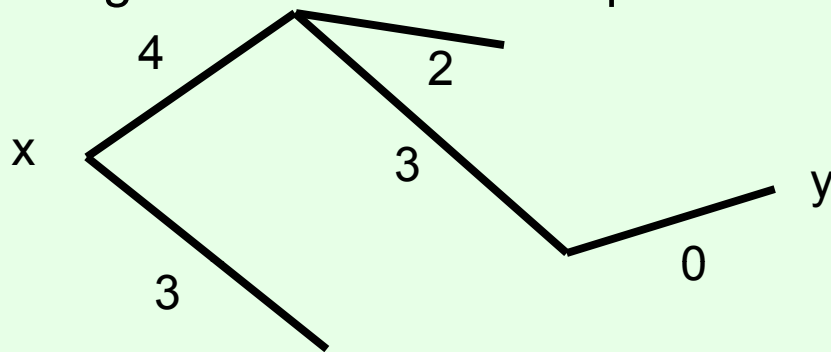
- One strategy:
  - Compute shortest path (e.g., using Dijkstra's algorithm)
  - For each edge on the shortest path, remove that edge, solve the problem again
    - $O(nm + n^2 \log n)$  total time



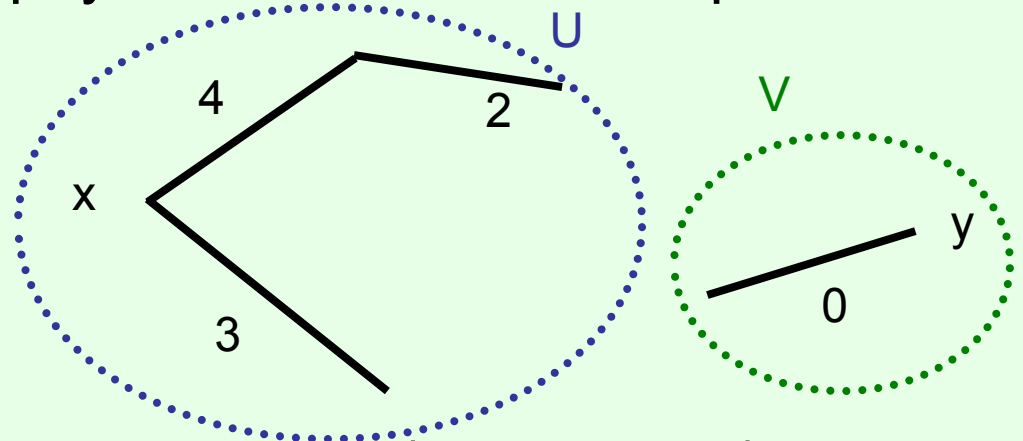
- Is there a more efficient algorithm?

# Hershberger-Suri [FOCS 01] algorithm

- Compute the shortest path trees from  $x$  and from  $y$ 
  - using Dijkstra
  - gives us the shortest path from any vertex to  $x$  and to  $y$



- Remove the edge  $e$  whose payment we wish to compute from the first ( $x$ ) tree
  - Cuts the graph into  $U$  and  $V$



- Over all edges  $(u, v)$  across components (excluding  $e$ ), minimize  $d(x, u) + c(u, v) + d(v, y)$ 
  - Using data structures, can be done for all edges in  $O(n \log n + m)$

# A make-span/reverse auction problem

[Nisan & Ronen STOC 99]

- There are  $m$  jobs that need to be scheduled on (say) 2 machines
- Each machine is owned by a separate agent
- $c_{ij}$  is the time that machine  $i$  would take on job  $j$ 
  - also the cost that machine  $i$  has for doing  $j$
  - private information
- The objective is to minimize the **make-span**
  - = highest total cost for an agent, = completion time of last job
- One possibility: just use Clarke mechanism
  - Award job  $j$  to the machine that can do it faster (minimize total work),
  - Pay that machine the cost of the other machine for  $j$
- Gives a 2-approximation to the make-span
  - **Theorem:** No deterministic mechanism does better



# A bad instance for the Clarke mechanism

- Two jobs
- Machine 1:  $c_{11} = 1, c_{12} = 1$
- Machine 2:  $c_{21} = 1+\varepsilon, c_{22} = 1+\varepsilon$
  
- Clarke mechanism will give both jobs to 1
- Make-span: 2
- Can get  $1+\varepsilon$  by giving one job to each (ignoring mechanism design considerations)

# Weighted Groves mechanisms

- Recall a Groves mechanism
  - chooses an allocation  $o$  that maximizes the sum of reported utilities,
  - pays agent  $i$ :  $\sum_{j \neq i} u_j(\theta_j', o) + h(\theta_{-i}')$  for some function  $h$
- A **weighted Groves mechanism**
  - has a weight  $w_i$  for each agent,
  - chooses an allocation  $o$  that maximizes  $\sum w_i u_i(\theta_i', o)$ ,
  - pays agent  $i$ :  $(1/w_i) \sum_{j \neq i} w_j u_j(\theta_j', o) + h(\theta_{-i}')$  for some function  $h$
- Weighted Groves mechanisms are strategy-proof  
[Roberts 1979]

# A biased mechanism based on a weighted Groves mechanism

[Nisan & Ronen STOC 99]

- For each job  $j$ , **bias** the mechanism towards accepting one of the two agents  $i$ 
  - For some  $b > 1$ , award job  $j$  to  $i$  if and only if  $c_{ij} < bc_{(-i)j}$
  - If so,  $i$  gets payment  $bc_{(-i)j}$
  - Otherwise,  $-i$  gets payment  $c_{ij}/b$
- Weighted Groves mechanism, so strategy-proof
- A randomized mechanism:
  - set  $b = 4/3$ ,
  - for each job independently, randomly choose the agent to which the mechanism is biased
- Gives a  $7/4$  approximation

# Characterizing allocation rules that can be made incentive compatible

- We saw that we may be interested in choosing allocations that **do not maximize social welfare** (sum of utilities)
  - Different objectives (e.g., make-span)
  - Social welfare maximizing allocation may be computationally too hard to find
- Some (not all) allocation rules can be made incentive compatible with the right payment rule
- What are **necessary and sufficient** conditions on an allocation rule for this to be possible?

# Weak monotonicity

[Bikhchandani et al. Econometrica 06]

- Consider the case of a single type reporting agent
  - Equivalently, fix the types of the other players
- $o(\theta)$  is the allocation chosen when the agent reports  $\theta$
- $u(\theta, o)$  is the agent's utility for allocation  $o$  given true type  $\theta$
- Rule  $o(\cdot)$  is said to be **weakly monotone** if the following condition holds for every  $\theta, \theta'$ :  
$$u(\theta, o(\theta)) - u(\theta, o(\theta')) \geq u(\theta', o(\theta)) - u(\theta', o(\theta'))$$
- In words: if there are no payments, then
  - the utility **loss** from misreporting  $\theta'$  when the true type is  $\theta$
  - is at least as great as
  - the utility **gain** from misreporting  $\theta$  when the true type is  $\theta'$

# Necessity of weak monotonicity

- Suppose an allocation rule  $o(\cdot)$ , together with a payment rule  $\pi(\cdot)$ , incentivizes the agent to tell the truth
- Then, for any  $\theta, \theta'$ , the following must hold:
- $u(\theta, o(\theta)) + \pi(\theta) - u(\theta, o(\theta')) - \pi(\theta') \geq 0$
- $u(\theta', o(\theta')) + \pi(\theta') - u(\theta', o(\theta)) - \pi(\theta) \geq 0$
- Adding these two together gives
- $u(\theta, o(\theta)) - u(\theta, o(\theta')) + u(\theta', o(\theta')) - u(\theta', o(\theta)) \geq 0$
- Equivalently
- $u(\theta, o(\theta)) - u(\theta, o(\theta')) \geq u(\theta', o(\theta)) - u(\theta', o(\theta'))$
- But this is the weak monotonicity condition!

# Sufficiency of weak monotonicity

- Suppose the agent has a partial order  $\geq$  over the allocations
- Here  $o \geq o'$  indicates that the agent prefers  $o$  to  $o'$  for **every** type that she may have
  - E.g., in  $o$ , she is allocated a superset of what she is allocated in  $o'$ , and free disposal holds
- The set of types is said to be **rich** if every utility function consistent with  $\geq$  corresponds to some type
- **Theorem.** If preferences are rich, weak monotonicity is sufficient for incentive compatibility
  - I.e., for any weakly monotone allocation rule, a payment function making this rule incentive compatible exists
- With more restricted type spaces, weak monotonicity is not always sufficient