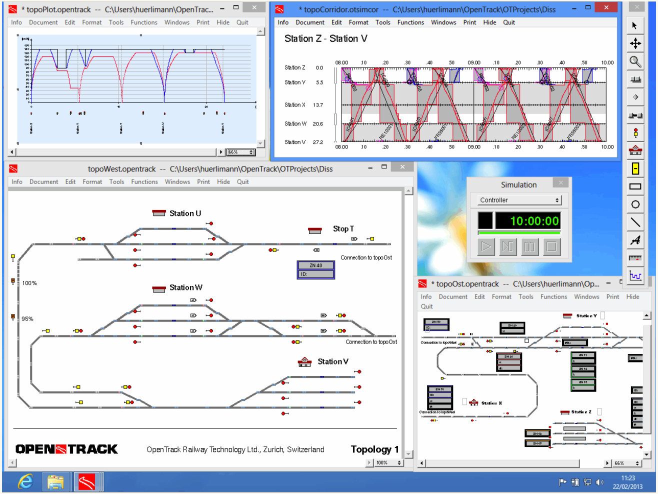
Algorithmic Game Theory

Bjørnar Luteberget

PhD Trial Lecture
18 Oct 2019

Algorithms vs. games

Controllable system $\rightarrow$ design algorithms.



Algorithms vs. games

Individual choices $\rightarrow$
design incentives, constraints.

Algorithms vs. games

Individual choices $\rightarrow$
design incentives, constraints.

What is a game?

- Players
- Actions
- Scores

Rock, paper, scissors

Player 2 / Player 1	✊	🖐	✌
✊	(0,0)	(-1,1)	(1,-1)
🖐	(1,-1)	(0,0)	(-1,1)
✌	(-1,1)	(1,-1)	(0,0)

This game is

- two-player,
- one-shot / simultaneous,
- zero-sum.

Game theory

- Game theory: the study of strategic interactions between rational decision-makers.

- Each player selects an action hoping to maximize their score.
- Can we predict which choices they make?

Game theory

A main concept in game theory:

- Nash equilibrium is a set of strategies, one for each player, such that no player has incentive to deviate given what the others are doing.
- Pure strategy: player selects one of the actions.
- Mixed strategy: player selects a probability distribution over actions.
- Nash's Existence Theorem: every game with a finite number of players and a finite number of actions has at least one mixed strategy Nash equilibrium (Nash, 1951).

Battle of the sexes

You \ Friend	Cinema	Theatre
Cinema	(5,6)	(1,1)
Theatre	(2,2)	(6,5)

- You like theatre, your friend likes cinema.
- But you both prefer going together over getting your will.

This game is

- two-player,
- one-shot / simultaneous,
- non-zero-sum.

It has two pure equilibria.

Prisoner's dilemma

You \ Other	Silent	Betray
Silent	(-1,-1)	(-3,0)
Betray	(0,-3)	(-2,-2)

- Two players face prison time convictions on weak evidence.
- If you betray the other player, you go free while they get a harsher sentence.
- If you both betray each other, both get a harsher sentence than staying silent.

This game is

- two-player,
- one-shot / simultaneous,
- non-zero-sum.

It has one pure equilibrium.

Algorithmic game theory

Algorithmic game theory: the intersection of game theory and computer science: understanding and designing algorithms in strategic environments.

Overview:

- Part I: Computing equilibria
- Part II: Selfish behaviour and optimal systems
- Part III: Mechanism design

Based on material from:

- Tim Roughgarden: *Algorithmic Game Theory lecture notes*, Stanford University.
- Aaron Roth: *Algorithmic Game Theory lecture notes*, University of Pennsylvania.
- Nisan, Roughgarden, Tardos, Vazirani: *Algorithmic Game Theory*, Cambridge, 2007.

Part I:
Computing equilibria

Computing equilibria

How do strategic players reach an equilibrium? Do they at all?

- Nash's Existence Theorem: every game with a finite number of players and a finite number of actions has at least one mixed strategy Nash equilibrium (Nash, 1951).
- For two-player zero-sum games we can use linear programming.

Solving two-player zero-sum games

- Two-player zero-sum games: the sum of payoffs is zero for any choice of strategies.
- So, we need the score for only one of the players.
- Rock, paper, scissors:

$$A = \begin{bmatrix} 0 & -1 & 1 \\ 1 & 0 & -1 \\ -1 & 1 & 0 \end{bmatrix}$$

- Player 1 (row player) chooses a strategy p:

$$p = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}, \quad pA = \begin{bmatrix} 0 & -1 & 1 \end{bmatrix}$$

- Player 2 (column player) chooses strategy q:

$$q = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix}^T, \quad pAq = -1$$

Linear programming

Linear programming standard form:

- A linear function to be optimized:

$$f(x_1, x_2, \dots) = c_1 x_1 + c_2 x_2 + \dots$$

- Linear problem constraints:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \dots &\leq b_1 \\ a_{21}x_1 + a_{22}x_2 + \dots &\leq b_2 \\ a_{31}x_1 + a_{32}x_2 + \dots &\leq b_3 \\ &\dots \end{aligned}$$

- Non-negative variables: $x_1 \geq 0$, $x_2 \geq 0$, $\dots$
- Is solvable in polynomial time (Khachiyan 1979).
- In practice, often solved using the Simplex algorithm (Dantzig 1947) worst-case exponential but usually efficient.

Solving two-player zero-sum games

- Consider p^*, q^*, a distribution for the row and column players that is a mixed-strategy Nash equilibrium with value v^*.
- Consider an unknown strategy p for the row player.
- If we know p, we would like to find q that minimizes the column's players loss.
- The best strategy for choosing this publicly known p is then to maximize this minimum value.
- This corresponds to the following linear program:

$$\text{maximize } v_r$$

$$\forall i : p_i \geq 0, \quad \sum_i p_i = 1, \quad \forall j : (pA)_j \geq v_r$$

- Now, choosing p, the row player is guaranteed the score v_r.

Solving two-player zero-sum games

- We see that $v_r \leq v^*$, since the row player can guarantee to win v_r, so this is a minimum value for any equilibrium.
- Also, an equilibrium is stable even if known by the opponent, so the column player must be selecting the columns with minimum value $p^* A$. Therefore $v^* \leq v_r$, and we have $v_r = v^*$.

Correspondingly, for the column player:

$$\text{minimize } v_c$$

$$\forall i : q_i \geq 0, \quad \sum_j q_j = 1, \quad \forall j : (Aq)_j \leq v_c$$

Comparing: the row player's LP:

$$\text{maximize } v_r$$

$$\forall i : p_i \geq 0, \quad \sum_i p_i = 1, \quad \forall j : (pA)_j \geq v_r$$

... and the column player's LP:

$$\text{minimize } v_c$$

$$\forall i : q_i \geq 0, \quad \sum_j q_j = 1, \quad \forall j : (Aq)_j \leq v_c$$

These two linear programs are duals of each other. Linear programming theory already gives us $v_r = v_c$ in the special case of zero-sum two-player games.

Linear programming solution

```python
from pulp import *
problem = LpProblem("rock-paper-scissors", LpMaximize)

# Variables: Strategy vector and the row player's valuation.
rock = LpVariable("rock", 0.0, 1.0)
paper = LpVariable("paper", 0.0, 1.0)
scissors = LpVariable("scissors", 0.0, 1.0)
value = LpVariable("value")

# Objective: Value of solution for row player
problem += value

# Constraint: Strategy vector is a distribution over actions
problem += rock + paper + scissors == 1.0

# Constraint: Each column of pA is greater than the player's value
problem +=    0.0*rock +    1.0*paper + (-1.0)*scissors >= value
problem += (-1.0)*rock +    0.0*paper +    1.0*scissors >= value
problem +=    1.0*rock + (-1.0)*paper +    0.0*scissors >= value

problem.solve()

print("value:",value.varValue)
print("strategy:",(rock.varValue, paper.varValue, scissors.varValue))
```

Linear programming solution

```
OPTIMAL LP SOLUTION FOUND
Time used:   0.0 secs
Memory used: 0.0 Mb (39701 bytes)
('value:', 0.0)
('strategy:', (0.333333, 0.333333, 0.333333))
```

Non-zero-sum games

- Solving non-zero-sum games can be done with the Lemke-Howson algorithm, which takes worst-case exponential time.
- Like the Simplex algorithm, it *pivots* between vertices in polytopes. Uses two polytopes instead of one.

Non-zero-sum games

- There is no known polynomial-time algorithm for computing a Nash equilibrium in general.
- If all parties act rationally, equilibria could predict their behaviour, but only if these are reasonably easy to calculate.
- A complexity class named Polynomial Parity Arguments on Directed Graphs (PPAD) (Papadimitriou 1994) was introduced to characterize finding mixed-strategy Nash equilibria.
- Finding Nash equilibria is PPAD-complete, and cannot be NP-complete because the solution is known to exist.
- Many related problems, for example deciding whether there are two or more equilibria, are NP-complete.

Further topics in computing equlibria

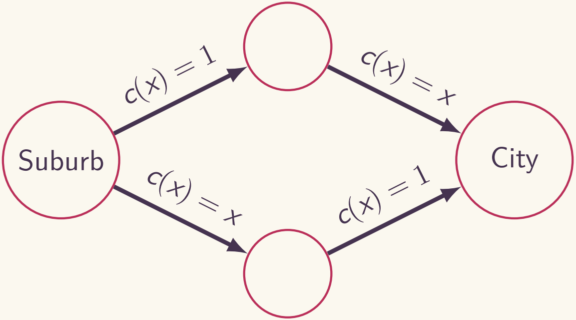
- Learning, regret minimization and equilibria
- Combinatorial algorithms for market equilibria
- Computation of market equilibria by convex programming
- Graphical games
- Cryptography

Part II: Selfish behaviour and optimal systems

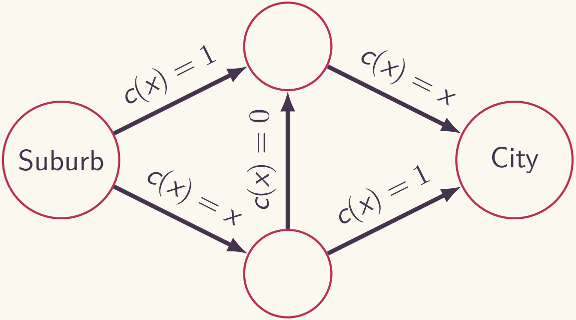
Selfish behaviour and optimal systems

Example:

- A number of cars travel simultaneously from suburb to city using one of two roads. Each road takes time $1 + x$, depending on the fraction of cars x that choose that road.
- If a fraction u of cars take the upper path, the total travel time is: $u(1 + u) + (1 - u)(1 + (1 - u)) = 2u^2 - 2u + 2$.
- Minimum total travel time at $u = \frac{1}{2}$.



Selfish behaviour and optimal systems



Braess' paradox (1968).

- Optimal traffic results in $t = 1.5$ for all cars.
- Selfish traffic results in $t = 2.0$ for all cars.
- "Price of anarchy" is $2.0/1.5$.

Selfish behaviour and optimal systems

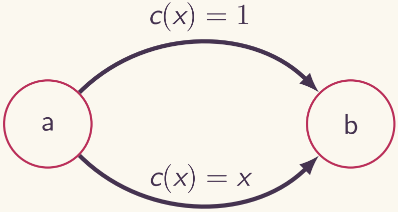
- Game equilibria are inefficient in general.
- But when is the price of anarchy low (ratio ≈ 1)?
- Applications in
  - network routing
  - scheduling
  - resource allocation
  - auction design

Plan:

- An upper bound on the price of anarchy can be found by considering just one example network!
- Use this upper bound in an analysis of communication networks.

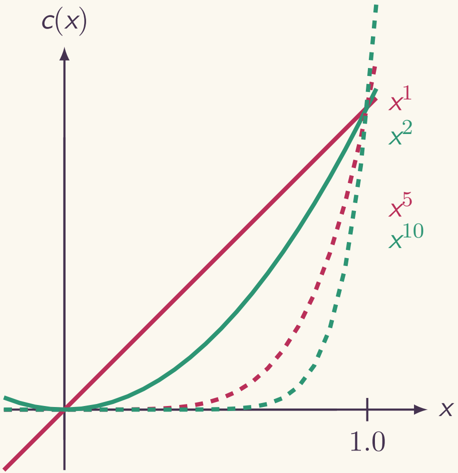
Pigou's example

- An even simpler network (Pigou, 1920) shows the same phenomenon as Braess' paradox.
- Every driver's dominant strategy is to take the lower link even when fully congested.
- (Dominant: the strategy is better no matter what the opponents do)
- Any other solution is better overall!



Non-linear Pigou's example

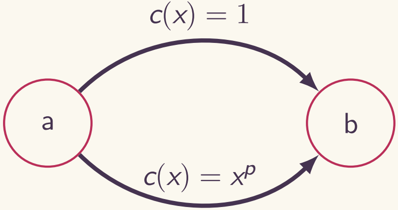
- The price of anarchy is $\frac{4}{3}$ in both Braess' and Pigou's examples.
- Now, change the cost function to $c(x) = x^p$.



Non-linear Pigou's example

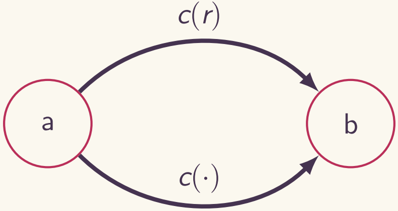
With $c(x) = x^p$:

- Same dominant strategy, same equilibrium travel time.
- Let $(1 - \epsilon)$ traffic on the bottom link.
- As $p \to \infty$, the travel time tends to 0 on average.
- Price of anarchy is unbounded.



Pigou's example is the worst case for the price of anarchy

Generalize Pigou's example to other non-negative, continuous, non-decreasing functions.



- Assume we know the set of possible cost functions $c \in \mathcal{C}$.
- Define the Pigou bound:

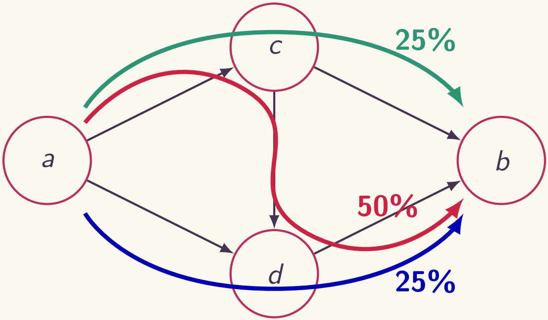
$$\alpha(\mathcal{C}) := \sup_{c \in \mathcal{C}} \sup_{r \geq 0} \sup_{x \geq 0} \left\{ \frac{r \cdot c(r)}{x \cdot c(x) + (r - x) \cdot c(r)} \right\}$$

- Theorem: of all networks with cost functions from $\mathcal{C}$, Pigou's example has the highest price of anarchy. (Roughgarden, 2003)

Pigou is worst-case – proof sketch

- A flow $\{f_p\}_{p \in P}$ is the distribution of traffic over all paths $p \in P$ from a to b.

$$\sum_{p \in P} f_p = r$$



Pigou is worst-case – proof sketch

- A flow is an equilibrium iff traffic travels only on the shortest paths from a to b.
- Note that shortest is defined with respect to the c induced by the flow.
- The cost of the flow (by paths) is $C(f) = \sum_{p \in P} f_p c_p(f)$.

- First part: if edge costs are frozen at equilibrium costs $c_e(f_e)$, then f is optimal.
- Intuition: an equilibrium flow f routes all traffic on shortest paths, so no other flow f^* can be better if we keep the same edge costs.

$$\sum_{e \in E} (f_e^* - f_e) c_e(f_e) \geq 0$$

Pigou is worst-case – proof sketch

- Second part: how can the optimal flow f^* be better than f?
- Intuition: edge by edge, the gap in costs between f and f^* is no worse that the Pigou bound.
- Use the Pigou bound:

$$\alpha(\mathcal{C}) := \sup_{c \in \mathcal{C}} \sup_{r \geq 0} \sup_{x \geq 0} \left\{ \frac{r \cdot c(r)}{x \cdot c(x) + (r - x) \cdot c(r)} \right\}$$

- Insert $c \to c_e$, $r \to f_e$, $x \to f_e^*$.

$$\alpha(\mathcal{C}) \geq \frac{f_e \cdot c_e(f_e)}{f_e^* \cdot c_e(f_e^*) + (f_e - f_e^*) \cdot c_e(f_e)}$$

- Rearrange to:

$$f_e^* \cdot c_e(f_e^*) \geq \frac{1}{\alpha(\mathcal{C})} \cdot f_e \cdot c_e(f_e) + (f_e^c - f_e)c_e(f_e)$$

- Take the sum over all edges $e \in E$:

$$C(f^*) \geq \frac{1}{\alpha(\mathcal{C})} \cdot C(f) + \sum_{e \in E}(f_e^* - f_e)c_e(f_e)$$

- Finally,

$$\frac{C(f)}{C(f^*)} \leq \alpha(\mathcal{C})$$

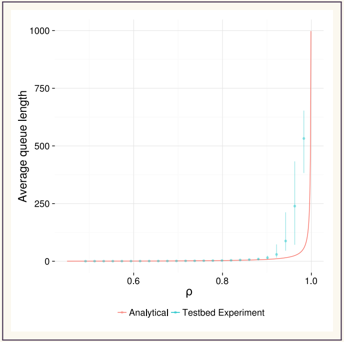
An application in network provisioning

- The selfish routing model can give insight into network for transportation, communication, and electrical networks.
- We will see how Pigou's bound explains internet service provider's strategy for over-provisioning.

- In communication networks, it is often relatively cheap to add additional capacity to the network.
- Communication networks are over-provisioned to anticipate future increase in demand, and because networks perform better when capacity is not saturated.

An application in network provisioning

- Consider a queue where jobs arrive according to a Poisson process with rate x.
- Jobs are processed with independent exponential distributed time with mean $\frac{1}{u}$.
- Known as an M/M/1 queue.



M/M/1 queue length and computer network simulation (Fund 2016)

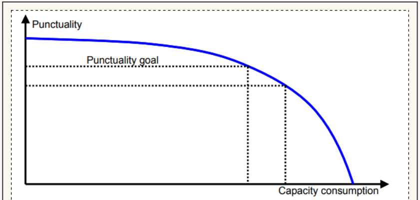


Figure 3.36: The coherence between punctuality and capacity consumption. Based on (Kaas 1998b, Rasch 1998).

Empirical railway capacity vs. punctuality (Landex 2008)

An application in network provisioning

- A selfish routing network of M/M/1 queues is β-*over-provisioned* if $f_e \leq (1-\beta)u_e$ for every edge e, where f_e is an equilibrium flow.
- At equilibrium, the maximum link utilization in the network is $(1-\beta)$.
- With cost function $c(x) = 1/(u-x)$, the worst-case price of anarchy is

$$\frac{c_{\text{selfish}}}{c_{\text{optimal}}} = \frac{1}{2}\left(1+\sqrt{\frac{1}{\beta}}\right)$$

An application in network provisioning

$$\frac{c_{\text{selfish}}}{c_{\text{optimal}}} = \frac{1}{2}\left(1 + \sqrt{\frac{1}{\beta}}\right)$$

- $\beta \to 1$: infinite capacity means there is no price of selfishness.
- $\beta \to 0$: no spare capacity means that the price of selfishness can be arbitrarily high (*in the worst case*, Pigou's example).
- $\beta = 0.1$ gives
$$\frac{c_{\text{selfish}}}{c_{\text{optimal}}} \approx 2.1$$
- A little over-provisioning allows selfish routing to be close enough to optimal.
- Explains empirical knowledge from internet service providers.

Further topics in quatifying the inefficiency of equilibria

- Routing games (selfish routing)
- Network formation games
- Selfish load balancing
- Design of scalable resource allocation mechanisms

Part III:
Mechanism design

Mechanism design

- Mechanism design: sub-field of economic theory with an engineering perspective.
- "Reverse game theory".

Application areas:

- Elections
- Markets
- Auctions
- Government policy

Especially with computerized and internet-scale choice systems (e.g. high-frequency trading), the pure mathematical properties become more directly relevant.

Sealed-bid auction

- A single item for sale, each player i values the item v_i.
- One player is awarded the item and pays a price p.
- The player's utility is $v_i - p$.

First-price auction:

- Award the item to the player i with the highest bid b_i, and set the price $p = b_i$.

... this makes it hard to predict the player's actions, but we can under some assumptions:

- Use assumed distribution of other player's valuations.
- If two bidders a and b have valuations uniformly distributed in $[0, 1]$, they bid $b_a = v_a/2$ and $b_b = v_b/2$.

Second-price auction (Vickrey)

Second-price auction (Vickrey):

- Award the item to the highest bidder, but set the price at the next-highest bid.
- Every bidder's dominant strategy is to bid their valuation $b_i = v_i$.
- (Dominant: the strategy is better no matter what the opponents do)

Proof:

- Fix an arbitrary player i, their valuation v_i, and the bids of other players $\vec{b}_{-i}$. Let $B = \max_{j \neq i} b_j$ be the highest bid that any of the other players gave.
- May select $b_i < B$: gives utility 0.
- May select $b_i >= B$: gives utility $v_i - B$.
- If $v_i < B$, the maximum utility is $\max\{0, v_i - B\} = 0$.
- If $v_i >= B$, the maximum utility is $\max\{0, v_i - B\} = v_i - B$.
- Both cases are optimal when choosing $b_i = v_i$.

Second-price auction (Vickrey)

- Also, $b_i = v_i$ guarantees non-negative utility.

Vickrey auctions have the following good properties:

- Each player has an optimal strategy which does not depend on the other players' strategies, and which reveals their true valuation.
- The item is awarded to the player who values it the most (social welfare maximization).
- Computationally trivial.

More generally: mechanisms with pricing

We want to collectively decide on an action $a \in A$.

The preference of each player i is a function $v_i : A \to (V_i \subseteq \mathbb{R})$.

A mechanism (direct revelation mechanism) is:

- a social choice function

$$f \colon V_1 \times \cdots \times V_n \to A$$

- a vector of payment functions $p_1, \ldots, p_n$ where

$$p_i : V_1 \times \cdots \times V_n \to \mathbb{R}$$

  is the price that player i pays.

Incentive compatible mechanisms

A mechanism $(f, p_1, \ldots, p_n)$ is incentive compatible if

- for every player i,
- for every actual valuation $v_1 \in V_1$, …, $v_n \in V_n$
- for every $v'_i \in V_i$,

let $a = f(v_i, v_{-i})$ and $a' = f(v'_i, v_{-i})$, then

- $v_i(a) - p_i(v_i, v_{-i}) \geq v_i(a') - p_i(v'_i, v_{-i})$.

Intuitively, this means that player i whose valuation is v_i would prefer telling the truth v_i to the mechanism rather than any possible lie v'_i since this gives them higher utility.

Vickrey-Clarke-Groves mechanism

A mechanism $(f, p_1, \ldots, p_n)$ is a Vickrey-Clarke-Groves (VCG) mechanism if:

- f maximizes the social welfare:

$$f(v_1, \ldots, v_n) \in \operatorname*{argmax}_{a \in A} \sum_i v_i(a)$$

- for some $h_1, \ldots, h_n$ where $h_i : V_{-i} \to \mathbb{R}$,
- for all $v_1 \in V_1, \ldots, v_n \in V_n$:
- $p_i(v_1, \ldots, v_n) = h_i(v_{-i}) - \sum_{j \neq i} v_j(f(v_1, \ldots, v_n))$.

The price function adds some amount depending on the choice of the other players, and subtracts some amount corresponding to the valuations of other players.

VCG mechanisms are incentive-compatible.

Vickrey-Clarke-Groves mechanisms are incentive-compatible.

Proof:

- Fix i, v_{-i}, v_i and v'_i.
- Let $a = f(v_i, v_{-i})$ and $a' = f(v'_i, v_{-i})$.
- The utility of i when declaring v_i is

$$v_i(a) - (h_i(v_{-i}) - \sum_{j \neq i} v_j(a))$$

- when declaring v'_i:

$$v_i(a') - (h_i(v_{-i}) - \sum_{j \neq i} v_j(a'))$$

- Since a maximizes social welfare:

$$v_i(a) + \sum_{j \neq i} v_j(a) \geq v_i(a') + \sum_{j \neq i} v_j(a')$$

Clarke pivot rule

Which h_i functions do we want?

- A mechanism is ex-post individually rational if players always get non-negative utility.

$$\forall v_1, \ldots, v_n : v_i(f(v_1, \ldots, v_n)) - p_i(v_1, \ldots, v_n) \geq 0$$

- Has no positive transfers if no player is ever paid money.

$$\forall v_1, \ldots, v_n : \forall i : p_i(v_1, \ldots, v_n) \geq 0$$

- The Clarke pivot payment has these properties:

$$h_i(v_{-i}) = \max_{b \in A} \sum_{j \neq i} v_j(b)$$

- Under this rule the payment of player i is:

$$p_i(v_1, \ldots, v_n) = \max_{b} \sum_{j \neq i} v_j(b) - \sum_{j \neq i} v_j(a)$$

Intuitively, player i pays for the damage done to the other players. What value would they have gained with vs. without me?

Example: buying a path in a network.

Example: buying a path in a network.

- In a directed graph $G = (V, E)$, consider each edge $e \in E$ as a player who has a cost $c_e \geq 0$ if the edge is used.
- We want to buy a path $s \to t$, and the players (edges) have (actual) valuation of 0 if they are not part of the path and $-c_e$ if they are.
- Maximizing social welfare corresponds to finding the shortest path p, $\sum_{e \in p} c_e$.
- VCG: for each edge e_0 in p, pay

$$\sum_{e \in p'} c_e - \sum_{e \in p \setminus \{e_0\}} c_e$$

where p is the shortest path and p' is the shortest path not containing e_0.

More on mechanism design

Other topics in mechanism design:

- Mechanisms with or without money
- Combinatorial auctions
- Computationally efficient approximation mechanisms
- Profit maximization in mechanism design
- Distributed algorithmic mechanism design
- Cost sharing
- Online mechanisms

Further reading

- Nisan, Roughgarden, Tardos, Vazirani: *Algorithmic Game Theory*, Cambridge, 2007.
- Tim Roughgarden: *Twenty Lectures on Algorithmic Game Theory*, Cambridge University Press, 2016.