Mechanism Design for Scheduling Unrelated Machines

Elias Koutsoupias

University of Athens
http://www.di.uoa.gr/~elias

Paderborn 2008/04/29

Outline

Mechanism design

Mechanisms as Algorithms

- Mechanism Design = Algorithms with payments
- Given an objective, design **a game with payments** whose equilibrium is the objective.
- Here we consider dominant equilibria (i.e., a player has an optimal strategy, no matter what the other players do).

Typical Example: Single-item Auction

Problem

- *We want to sell an object to n players (buyers).*
- *Each player has a value v_i for the object, which is known only to him/her .*
- *Objective: Give the item to the player with the highest value.*

Features

- Incomplete information: only the players know their values
- Money is used as an incentive. But: money is not part of the objective.
- Direct revelation: The players declare all their values at the beginning.

Example: Single-item Auction (cont.)

The VCG mechanism

- Each player declares a value $\hat{v}_i$, not necessarily equal to the true value v_i.
- The mechanism allocates the object to the player with the highest bid, $\max_i \hat{v}_i$. **This is the objective when the players are truthful.**
- The player pays only the second highest bid.

Proposition

The VCG mechanism is truthful.

The Mechanism Design Framework

The general mechanism design (social choice) setting

- There are n players and m outcomes. Let v_{ij} be the gain of player i when the outcome of the game is j.

$$\begin{bmatrix} v_{11} & v_{12} & v_{13} \\ v_{21} & v_{22} & v_{23} \end{bmatrix}$$

- The domain D of the problem is a set of $n \times m$ matrices.
- The objective of the mechanism designer is to select the outcome (i.e., column) which optimizes his/her objective.
- The objective of each player is to maximize his/her gain.
- Only the players know the values v_{ij}.

The Mechanism Design Framework (cont.)

Problem (The single-item auction)

There are n players and $m = n$ outcomes. The i-th outcome is for player i to get the item.
The domain of the problem is all $n \times n$ matrices of the form

$$\begin{bmatrix} v_1 & 0 & 0 \\ 0 & v_2 & 0 \\ 0 & 0 & v_3 \end{bmatrix}$$

Each row corresponds to a player, and each column to an outcome.

Combinatorial Auction

Problem (Combinatorial auction)

- *There are n players (bidders) and m objects (items)*
- *Each player i has a value $u_{i,S}$ for each subset (bundle) S of the objects. These are private values.*
- *Objective: Allocate the objects to the players to maximize the sum of the values of their bundles.*

Example (3 players, 2 items)

$$\begin{bmatrix} u_{1,12} & u_{1,1} & u_{1,1} & u_{1,2} & u_{1,2} & 0 & 0 & 0 & 0 \\ 0 & u_{2,2} & 0 & u_{2,1} & 0 & u_{2,12} & u_{2,1} & u_{2,2} & 0 \\ 0 & 0 & u_{3,2} & 0 & u_{3,1} & 0 & u_{3,2} & u_{3,1} & u_{3,12} \end{bmatrix}$$

Scheduling unrelated machines

Problem (Scheduling)

- *There are n players (machines) and m objects (tasks)*
- *Each player i has a (private) value t_{ij} for each task j*
- *Objective: Allocate the tasks to the players to minimize the maximum value among the players (i.e., the makespan)*

Example (2 players, 2 tasks)

$$\begin{bmatrix} t_{11}+t_{12} & t_{11} & t_{12} & 0 \\ 0 & t_{22} & t_{21} & t_{21}+t_{22} \end{bmatrix}$$

Direct Revelation Mechanisms

The protocol of the mechanism

Declare Each player i declares his/her values $\hat{v}_{ij}$.

Allocate An allocation algorithm A computes the outcome $j^* = A(\hat{v})$.

Pay A payment algorithm p computes for each player i a payment $p_i(\hat{v}, j^*)$.

The objectives

Player Player i gains $v_{ij^*} - p_i(\hat{v}, j^*)$.

Social The objective of the mechanism is to select the outcome j^* which optimizes some global objective $f(v)$. (For example to select the column with maximum total value).

Truthful mechanisms

Definition (Truthful mechanisms)

A mechanism is truthful when revealing the true values ($\hat{v}_{ij} = v_{ij}$) is a dominant strategy of every player.

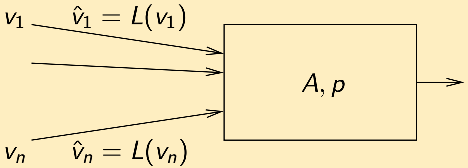
Theorem (The revelation principle)

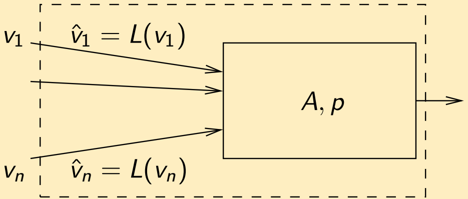
For every mechanism there is an equivalent truthful mechanism (with the same payments and outcome) .

Why?

Given a non-truthful mechanism, we can design a new truthful mechanism which first simulates the lying strategies of the players and then applies the original mechanism. The players would tell the truth to this mechanism.

The revelation principle





Truthful mechanisms for single-item auction

First Price

- The mechanism in which the highest bidder gets the item and pays his declared price is **not truthful**.
- Counterexample: $v_1 = 2$, $v_2 = 1$. Player 1 gains by bidding $\hat{v}_1 = 1 + \epsilon$.

Second Price

- The mechanism in which the highest bidder gets the item and pays the second highest price is **truthful**.

Central question

Which mechanisms are truthful?

Which mechanisms are truthful?

Focus on allocations

- The objective (social choice) does not involve the payments.
- Which allocation algorithms admit a payment policy that makes the mechanism truthful?

Example (Single-item auction)

- The algorithm which allocates the object to the **highest value** is truthful. (The second price payment policy makes it **truthful**).
- The algorithm which allocates the object to the **second highest value** is **not truthful**. (There is no payment policy to make it truthful). Why?

The VCG and the affine maximizer

Definition (VCG)

The Vickrey-Clarke-Groves (VCG) mechanism selects the outcome which maximizes the **sum of the values** of the players.

Definition (Affine maximizer)

In an affine maximizer (or generalized VCG) there are constants λ_i (one for each player) and γ_j (one for each outcome) and the mechanism selects the outcome j which maximizes $\sum_i \lambda_i v_{ij} + \gamma_j$.

Example (Affine maximizer for 2 players, 3 outcomes)

$$
\begin{array}{ccccc}
v_{11} & v_{12} & v_{13} & \leftarrow & \lambda_1 \\
v_{21} & v_{22} & v_{23} & \leftarrow & \lambda_2 \\
\uparrow & \uparrow & \uparrow & & \\
\gamma_1 & \gamma_2 & \gamma_3 & &
\end{array}
$$

The VCG Mechanism

Theorem

The generalized VCG mechanism is truthful.

- The payment of each player i is equal to the (weighted) sum of the remaining players plus an arbitrary value that depends on the values of the other players:

$$\lambda_i p_i(v, j) = -\sum_{i' \neq i} \lambda_{i'} v_{i'j} + h_i(v_{-i})$$

- The objective (value + payment) of each player i becomes (almost) identical to the global objective!
- We can think of it, as giving a discount to a player equal to the increase of the global objective because of his/her participation (by carefully selecting the function h).

The VCG Mechanism for the combinatorial auction

Is VCG good?

- For the combinatorial auction problem, where the global objective is to maximize the total value, the VCG achieves the global objective.
- There is however a problem: Computing the optimal solution may be computationally hard.
  - If the input is the whole $n \times k^n$ array, then the problem is computationally trivial (linear-time).
  - If the input is given implicitly, then the problem can be NP-hard.

The VCG Mechanism for the scheduling problem

VCG does not match the social optimum

- The VCG mechanism is not appropriate for the scheduling problem. It maximizes the sum, while the objective is the makespan!

Comparison of combinatorial auctions and scheduling

- The domain of scheduling is a **restriction** of the domain of combinatorial auction in which the valuations of bundles are additive.
- Auction is a maximization problem, scheduling is a minimization problem. (Not a significant difference.)
- They differ in the objective. One aims at the sum the other at the max. In this respect, scheduling is more difficult.

Characterization of truthful mechanisms

Problem

Given a domain—a mechanism design problem—characterize the truthful mechanisms.

- Let $x_j = x_j(v)$ be a 0-1 value that indicates the selected outcome.

$$x_j = \begin{cases} 1 & \text{if the allocation algorithm selects outcome } j \\ 0 & \text{otherwise} \end{cases}$$

Definition (Monotonicity)

An allocation algorithm is called monotone if for every two inputs v and v' that differ only on the i-th player, the allocations x and x' satisfy

$$\sum_j (x_j - x_j')(v_{ij} - v_{ij}') \geq 0$$

Characterization of truthful mechanisms

Theorem (Saks-Yu, 2005)

Monotonicity is necessary and sufficient condition for truthfulness for convex domains.

- The proof of necessity is easy. The proof for sufficiency is deeper.
- The characterization applies to almost all interesting problems with continuous domains.
- It does not apply to discrete domains. For example, when there are two possible values for each item, low and high.
- This characterization is complete but not necessarily useful.

Roberts' Theorem

Theorem (Roberts, 1979)

For the unrestricted domain with at least 3 outcomes, the only truthful mechanisms are the generalized VCG mechanisms.

Desired characterization

- This characterization is much more useful than the monotonicity property.
- Can we get similar characterizations for the problems of combinatorial auctions and scheduling?

Open problems

- Characterize the truthful mechanisms for the combinatorial auction problem.
  Ron Lavi, Ahuva Mualem, and Noam Nisan [2003] gave an almost complete answer: The generalized VCG is essentially the only truthful mechanism, under some mild (?) assumptions.
- Characterize the truthful mechanisms for the scheduling problem.

Algorithms for auctions and scheduling

Open problems for combinatorial auctions

- Design a mechanism that achieves allocations with good approximation ratio and has low computational and communication complexity
- Characterize the allocation algorithms of the truthful mechanisms.

Open problems for scheduling

- Design a mechanism with good approximation ratio.
- Characterize the allocation algorithms of the truthful mechanisms.

- In both problems we seek good approximations algorithms.
- In the combinatorial auction problem, the issue is only computational. After all, we have a perfect (exponential-time) algorithm, the VCG.
- In the scheduling problem, the issue is truthfulness. The VCG does not apply.

Definition

- There are n players (machines) and m objects (tasks)
- Each player i has a (private) value t_{ij} for each task j
- Objective: Allocate the tasks to the players to minimize the maximum value among the players (i.e., the makespan)

The setting

Input

$$t = \begin{pmatrix} t_{11} & t_{12} & \cdots & t_{1m} \\ t_{21} & t_{22} & \cdots & t_{2m} \\ \cdots \\ t_{n1} & t_{n2} & \cdots & t_{nm} \end{pmatrix}$$

$\longrightarrow$

Output

$$x = \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1m} \\ x_{21} & x_{22} & \cdots & x_{2m} \\ \cdots \\ x_{n1} & x_{n2} & \cdots & x_{nm} \end{pmatrix}$$

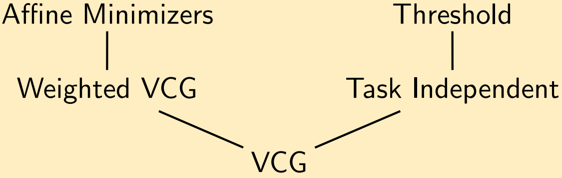
Input – Output

- n players/machines (rows).
- m tasks (columns).
- The input consists of nonnegative values t_{ij}.
- The output is an allocation:

$$x_{ij} = \begin{cases} 1 & \text{when task } j \text{ is allocated to machine } i \\ 0 & \text{otherwise} \end{cases}$$

Simple algorithms

VCG and its generalizations



VCG: Allocate each task to the machine of minimum value

Weighted VCG: VCG but first speedup some machines

Affine Minimizer: Weigthed VCG with additional constants for each allocation

Task Independent: Allocate every task independently of the others

Threshold: Allocate a task j to machine i independently of the other values of machine i

Truthful ≡ Monotone

Definition (Monotonicity Property)

An allocation algorithm is called monotone if it satisfies the following property: for every two sets of tasks t and t' which differ only on machine i (i.e., on the i-the row) the associated allocations x and x' satisfy

$$(x_i - x'_i) \cdot (t_i - t'_i) \leq 0,$$

where $\cdot$ denotes the dot product of the vectors, that is, $\sum_{j=1}^{m}(x_{ij} - x'_{ij})(t_{ij} - t'_{ij}) \leq 0$.

Theorem (Nisan, Ronen 1998, Saks, Lan Yu 2005)

Truthful $\equiv$ *Monotone*

The Monotonicity Property

First Input

$$\begin{pmatrix} t_{11} & t_{12} & \cdots & t_{1m} \\ \cdots & & & \\ t_{i1} & t_{i2} & \cdots & t_{im} \\ \cdots & & & \\ t_{n1} & t_{n2} & \cdots & t_{nm} \end{pmatrix} \Rightarrow \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1m} \\ \cdots & & & \\ x_{i1} & x_{i2} & \cdots & x_{im} \\ \cdots & & & \\ x_{n1} & x_{n2} & \cdots & x_{nm} \end{pmatrix}$$

Second Input

$$\begin{pmatrix} t_{11} & t_{12} & \cdots & t_{1m} \\ \cdots & & & \\ t'_{i1} & t'_{i2} & \cdots & t'_{im} \\ \cdots & & & \\ t_{n1} & t_{n2} & \cdots & t_{nm} \end{pmatrix} \Rightarrow \begin{pmatrix} x'_{11} & x'_{12} & \cdots & x'_{1m} \\ \cdots & & & \\ x'_{i1} & x'_{i2} & \cdots & x'_{im} \\ \cdots & & & \\ x'_{n1} & x'_{n2} & \cdots & x'_{nm} \end{pmatrix}$$

Monotonicity

$$\sum^{m} (x_{ij} - x'_{ij})(t_{ij} - t'_{ij}) \leq 0$$

A geometric approach to truthfulness

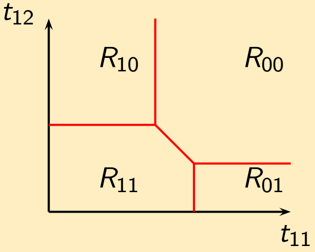
2 tasks

- Fix all values except of t_{11} and t_{12}. Consider how the space of t_{11} and t_{12} is partitioned by a truthful mechanism.
- R_{ab}: the region for which the allocation of the first machine is $x_{11} = a$ and $x_{12} = b$.
- The Monotonicity Property implies that a mechanism is truthful iff the regions R_{ab} and $R_{a'b'}$ are separated by a line of the form

$$(a - a')t_{11} + (b - b')t_{12} = const.$$

A geometric approach to truthfulness

The geometry of truthful mechanisms



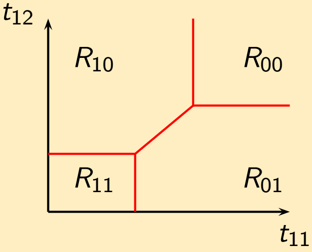


Figure: The two possible ways to partition the positive orthant.

Boundaries for the special cases

- For affine maximizers, the boundaries depend linearly on the values of the other players, and the diagonal part has constant length
- For threshold mechanisms, the diagonal part does not exist but the boundaries can be arbitrary (monotone) functions.

History of the scheduling problem

Computational issues

- It is a well-studied NP-hard problem
- Lenstra, Shmoys, and Tardos showed that its approximation ratio is in $[1.5, 2]$.

Mechanisms for scheduling

- Nisan and Ronen in 1998 initiated the study of its mechanism-design version.
- They gave an **upper bound** (VCG) with approximation ratio n.
- They gave a **lower bound** of 2.
- They conjectured that the right answer is the upper bound.
- They also gave a randomized mechanism with approximation ratio $7/4$ for 2 players.

Recent results

Deterministic

- The lower bound was improved to 2.41 (Christodoulou – K – Vidali) and to 2.61 (K – Vidali).
- For 2 machines the only truthful mechanisms with bounded approximation ratio are task-independent (Dobzinski – Sundararajan).
- For 2 machines, (with some mild asumptions) the only truthful mechanims are either affine minimizers or task-independent (Christodoulou – K – Vidali, submitted).

Recent results

Randomized

- The lower bound was improved to $2 - 1/n$ (Mu'alem – Schapira).
- The upper bound was improved to $7n/8$ (Mu'alem – Shapira).

Fractional

- The lower bound was improved to $2 - 1/n$ (Christodoulou – K – Kovács).
- The upper bound for task-independent mechanisms was pinned to $(n+1)/2$ (Christodoulou – K – Kovács).

Discrete (high and low value)

- Mechanism with approximation ratio 2 (Lavi – Swamy).

Results

- Archer and Tardos considered the related machines problem
- In this case, for each machine there is a single value (instead of a vector), its speed.
- They gave a variant of the (exponential-time) optimal algorithm which is truthful
- They also gave a polynomial-time randomized 3-approximation mechanism, which was later improved by Archer to 2-approximation.
- Andelman, Azar, and Sorani gave a 5-approximation deterministic truthful mechanism.
- Kovács improved it to 3-approximation and later to 2.8.

How to use the Monotonicity Property

We manipulate the values of one player in a particular way which guarantees that his allocation remains the same.

Example (Change the values, keep the allocation)

$$t = \begin{pmatrix} \mathbf{1} & 2 & \mathbf{2} \\ 2 & \mathbf{3} & 1 \\ 1 & 2 & 2 \end{pmatrix}$$

How to use the Monotonicity Property

We manipulate the values of one player in a particular way which guarantees that his allocation remains the same.

Example (Change the values, keep the allocation)

$$t = \begin{pmatrix} \mathbf{1} & 2 & \mathbf{2} \\ 2 & \mathbf{3} & 1 \\ 1 & 2 & 2 \end{pmatrix} \quad \rightarrow \quad t' = \begin{pmatrix} \mathbf{1} - \epsilon_1 & 2 + \epsilon_2 & \mathbf{2} - \epsilon_3 \\ 2 & 3 & 1 \\ 1 & \mathbf{2} & 2 \end{pmatrix}$$

How to use the Monotonicity Property

We manipulate the values of one player in a particular way which guarantees that his allocation remains the same.

Example (Change the values, keep the allocation)

$$t = \begin{pmatrix} \mathbf{1} & 2 & \mathbf{2} \\ 2 & \mathbf{3} & 1 \\ 1 & 2 & 2 \end{pmatrix} \quad \rightarrow \quad t' = \begin{pmatrix} \mathbf{1} - \epsilon_1 & 2 + \epsilon_2 & \mathbf{2} - \epsilon_3 \\ 2 & 3 & 1 \\ 1 & \mathbf{2} & 2 \end{pmatrix}$$

Example (Increase a value, keep the allocation)

$$t = \begin{pmatrix} \mathbf{0} & \cdots \\ \infty & \cdots \\ \infty & \cdots \end{pmatrix}$$

How to use the Monotonicity Property

We manipulate the values of one player in a particular way which guarantees that his allocation remains the same.

Example (Change the values, keep the allocation)

$$t = \begin{pmatrix} \mathbf{1} & 2 & \mathbf{2} \\ 2 & \mathbf{3} & 1 \\ 1 & 2 & 2 \end{pmatrix} \quad \rightarrow \quad t' = \begin{pmatrix} \mathbf{1} - \epsilon_1 & 2 + \epsilon_2 & \mathbf{2} - \epsilon_3 \\ 2 & 3 & 1 \\ 1 & \mathbf{2} & 2 \end{pmatrix}$$

Example (Increase a value, keep the allocation)

$$t = \begin{pmatrix} \mathbf{0} & \cdots \\ \infty & \cdots \\ \infty & \cdots \end{pmatrix} \quad \rightarrow \quad t' = \begin{pmatrix} \mathbf{1} & \cdots \\ \infty & \cdots \\ \infty & \cdots \end{pmatrix}$$

Easy proof of lower bound 2

2 players, 3 tasks

Either the mechanism partitions the tasks to the two machines

$$\begin{pmatrix} \color{red}{\mathbf{1}} & 1 & 1 \\ 1 & \color{red}{\mathbf{1}} & \color{red}{\mathbf{1}} \end{pmatrix}$$

or gives all tasks to the same machine

$$\begin{pmatrix} 1 & 1 & 1 \\ \color{red}{\mathbf{1}} & \color{red}{\mathbf{1}} & \color{red}{\mathbf{1}} \end{pmatrix}$$

Easy proof of lower bound 2

2 players, 3 tasks

Either the mechanism partitions the tasks to the two machines

$$\begin{pmatrix} \color{red}{\mathbf{1}} & 1 & 1 \\ 1 & \color{red}{\mathbf{1}} & \color{red}{\mathbf{1}} \end{pmatrix} \quad \rightarrow \quad \begin{pmatrix} \color{red}{\mathbf{0}} & 1 & 1 \\ 1 & \color{red}{\mathbf{1}} & \color{red}{\mathbf{1}} \end{pmatrix}$$

or gives all tasks to the same machine

$$\begin{pmatrix} 1 & 1 & 1 \\ \color{red}{\mathbf{1}} & \color{red}{\mathbf{1}} & \color{red}{\mathbf{1}} \end{pmatrix} \quad \rightarrow \quad \begin{pmatrix} 1 & 1 & 1 \\ \color{red}{\mathbf{0}} & \color{red}{\mathbf{1}} & \color{red}{\mathbf{1}} \end{pmatrix}$$

The instances of the 2.61 lower bound

$$\begin{pmatrix} 0 & \cdots & \infty & a & a^2 & \cdots & a^{n-1} \\ \infty & \cdots & \infty & a^2 & a^3 & \cdots & a^n \\ \cdots \\ \infty & \cdots & 0 & a^n & a^{n+1} & \cdots & a^{2n-1} \end{pmatrix}$$

Claim

If the first player does not get all the non-dummy tasks (the a^j tasks), then the approximation ratio is at least $1 + a$.

Therefore the approximation ratio is

$$\min\{1 + a, \frac{a + a^2 + \cdots + a^{n-1}}{a^{n-1}}\}.$$

For $n \to \infty$ and $a = \phi$, the ratio is $2.618\ldots$.

The Proof of the Claim

- We prove the claim by induction. For this we need to strengthen the induction hypothesis. The claim holds for all instances of the form

$$\begin{pmatrix} 0 & \cdots & \infty & a^{i_1} & a^{i_2} & \cdots & a^{i_k} \\ \infty & \cdots & \infty & a^{i_1+1} & a^{i_2+1} & \cdots & a^{i_k+1} \\ \cdots \\ \infty & \cdots & 0 & a^{i_1+n-1} & a^{i_2+n} & \cdots & a^{i_k+n-1} \end{pmatrix}$$

- $k \in \{1, \ldots, n-1\}$ and $i_1 < i_2 < \cdots < i_k$.

The Proof of the Claim (cont.)

Manipulating the values

- Assume that the first player does not get all the non-dummy tasks.
- We first manipulate the values so that
  - the **first player** gets **no** non-zero task and
  - **every other player** gets **at most one** non-zero task.

The Proof of the Claim (cont.)

Example (No task for the first player)

$$\begin{pmatrix} 0 & \cdots & \infty & a^{i_1} & a^{i_2} & \cdots & \mathbf{a^{i_k}} \\ \infty & \cdots & \infty & a^{i_1+1} & a^{i_2+1} & \cdots & a^{i_k+1} \\ \cdots \\ \infty & \cdots & 0 & a^{i_1+n-1} & a^{i_2+n-1} & \cdots & a^{i_k+n-1} \end{pmatrix}$$

If the first player gets some non-zero task, reduce the value to 0. The first player keeps the same tasks (by Monotonicity).

$$\begin{pmatrix} 0 & \cdots & \infty & a^{i_1} & a^{i_2} & \cdots & \mathbf{0} \\ \infty & \cdots & \infty & a^{i_1+1} & a^{i_2+1} & \cdots & a^{i_k+1} \\ \cdots \\ \infty & \cdots & 0 & a^{i_1+n-1} & a^{i_2+n-1} & \cdots & a^{i_k+n-1} \end{pmatrix}$$

The Proof of the Claim (cont.)

Example (Zero or one task for other players)

$$\begin{pmatrix} 0 & \cdots & \infty & a^{i_1} & a^{i_2} & \cdots & a^{i_k} \\ \infty & \cdots & \infty & \mathbf{a^{i_1+1}} & \mathbf{a^{i_2+1}} & \cdots & a^{i_k+1} \\ \cdots & & & & & & \\ \infty & \cdots & 0 & a^{i_1+n-1} & a^{i_2+n-1} & \cdots & a^{i_k+n-1} \end{pmatrix}$$

If some other player gets at least two non-zero tasks, reduce one value to 0. The player still gets at least one non-zero task.

$$\begin{pmatrix} 0 & \cdots & \infty & a^{i_1} & a^{i_2} & \cdots & a^{i_k} \\ \infty & \cdots & \infty & \mathbf{0} & \mathbf{a^{i_2+1}} & \cdots & a^{i_k+1} \\ \cdots & & & & & & \\ \infty & \cdots & 0 & a^{i_1+n-1} & a^{i_2+n-1} & \cdots & a^{i_k+n-1} \end{pmatrix}$$

The Proof of the Claim (cont.)

The result

At the end of the process,

- the first player has no non-zero tasks,
- every other player has at most one non-zero task,
- some other player has exactly one non-zero task.

The Proof of the Claim (cont.)

Estimating

- The optimum value is a^{i_k} (the diagonal right-to-left).
- We find a task with cost at least a^{i_k+1} and we raise its dummy (diagonal) value to a^{i_k}.
- The heart of the proof is that there always exists such a task which will not affect the optimum value.
- The cost of the mechanism is at least $a^{i_k} + a^{i_k+1}$ while the optimum is a^{i_k}. The approximation ratio is at least $1 + a$.

Example

$$\begin{pmatrix} 0 & \infty & \infty & \cdots & a^{i_k-3} & a^{i_k-1} & a^{i_k} \\ \infty & 0 & \infty & \cdots & a^{i_k-2} & \mathbf{a^{i_k}} & a^{i_k+1} \\ \infty & \infty & 0 & \cdots & a^{i_k-1} & a^{i_k+1} & \mathbf{a^{i_k+2}} \\ \cdots & & & & & & \end{pmatrix}$$

The Proof of the Claim (cont.)

Estimating

- The optimum value is a^{i_k} (the diagonal right-to-left).
- We find a task with cost at least a^{i_k+1} and we raise its dummy (diagonal) value to a^{i_k}.
- The heart of the proof is that there always exists such a task which will not affect the optimum value.
- The cost of the mechanism is at least $a^{i_k} + a^{i_k+1}$ while the optimum is a^{i_k}. The approximation ratio is at least $1 + a$.

Example

$$\begin{pmatrix} 0 & \infty & \infty & \cdots & a^{i_k-3} & a^{i_k-1} & a^{i_k} \\ \infty & 0 & \infty & \cdots & a^{i_k-2} & a^{i_k} & a^{i_k+1} \\ \infty & \infty & \mathbf{a^{i_k}} & \cdots & a^{i_k-1} & a^{i_k+1} & \mathbf{a^{i_k+2}} \\ \cdots & & & & & & \end{pmatrix}$$

Fractional allocations

- With fractional allocations each task can be split across the machines.
- The classical version of the problem is solvable in polynomial time (by linear programming).
- fractional approximation ratio $\leq$ randomized approximation ratio

Fractional Version: Lower Bound

A bad input

$$\begin{pmatrix} 0 & \infty & \cdots & \infty & \cdots & \infty & n-1 \\ \infty & 0 & \cdots & \infty & \cdots & \infty & n-1 \\ \cdots & & & & & & \\ \infty & \infty & \cdots & 0 & \cdots & \infty & n-1 \\ \cdots & & & & & & \\ \infty & \infty & \cdots & \infty & \cdots & 0 & n-1 \end{pmatrix}$$

Proving a lower bound of $2 - 1/n$

- Find the player who gets the largest fraction of the last task and raise its diagonal 0 value to 1.

Fractional Version: Lower Bound

A bad input

$$\begin{pmatrix} 0 & \infty & \cdots & \infty & \cdots & \infty & n-1 \\ \infty & 0 & \cdots & \infty & \cdots & \infty & n-1 \\ \cdots \\ \infty & \infty & \cdots & \mathbf{1} & \cdots & \infty & \mathbf{n-1} \\ \cdots \\ \infty & \infty & \cdots & \infty & \cdots & 0 & n-1 \end{pmatrix}$$

Proving a lower bound of $2 - 1/n$

- Find the player who gets the largest fraction of the last task and raise its diagonal 0 value to 1.
- When we change the values, the allocation remains almost the same.
- The optimal cost for the new input is 1.
- The cost of the changed player is at least $1 + \frac{n-1}{n} - \epsilon$.
- The approximation ratio is at least $2 - \frac{1}{n} - \epsilon$.

Fractional Version: Upper Bound

The SQUARE Algorithm

The mechanism SQUARE is a task independent algorithm which allocates to every player i a fraction inversely proportional to t_{ij}^2 of task j.

Theorem

The mechanism SQUARE is truthful with approximation ratio $\frac{n+1}{2}$.

Fractional Version: Upper Bound (cont.)

Ingredients of the proof

- The approximation ratio remains unaffected when we concentrate on instances in which the optimum allocation is integral.
- Let $S_1, \dots, S_n$ be an optimal allocation.
- We consider the execution time of SQUARE for machine i:

$$cost_i = \sum_j x_{ij} t_{ij} = \sum_r \sum_{j \in S_r} x_{ij} t_{ij}$$

- We will show that

$$\sum_{j \in S_r} x_{ij} t_{ij} \leq opt_r \qquad \text{for } r = i$$

$$\sum_{j \in S_r} x_{ij} t_{ij} \leq \frac{1}{2} opt_r \qquad \text{for } r \neq i$$

Fractional Version: Upper Bound (cont.)

- For $r = i$, we have $\sum_{j \in S_r} x_{ij} t_{ij} \le \sum_{j \in S_r} t_{ij} = opt_r$.
- For $r \ne i$, we have

$$
\begin{aligned}
\sum_{j \in S_r} x_{ij} t_{ij} &= \sum_{j \in S_r} \frac{\frac{1}{t_{ij}^2}}{\sum_k \frac{1}{t_{kj}^2}} t_{ij} \\
&\le \sum_{j \in S_r} \frac{\frac{1}{t_{ij}^2}}{\frac{1}{t_{rj}^2} + \frac{1}{t_{ij}^2}} t_{ij} \\
&= \sum_{j \in S_r} \frac{t_{ij} t_{rj}}{t_{ij}^2 + t_{rj}^2} t_{rj} \\
&\le \sum_{j \in S_r} \frac{1}{2} t_{rj} \\
&= \frac{1}{2} opt_r
\end{aligned}
$$

Monotone algorithms

Algorithms and Monotonicity

- Monotonicity, which is not specific to the scheduling task problem but it has much wider applicability, poses a new challenging framework for designing algorithms.
- In the traditional theory of algorithms, the algorithm designer could concentrate on how to solve every instance of the problem by itself.
- With monotone algorithms, this is no longer the case. The solutions for one instance must be consistent with the solutions of the remaining instances—they must satisfy the Monotonicity Property.
- Monotone algorithms are holistic algorithms: they must consider the whole space of inputs together.

My favorite problems in mechanism design

Scheduling unrelated machines

- Characterize the truthful mechanisms for scheduling unrelated machines.
- Close the gap between 2.41 and n for this problem.
- Improve the bounds 2 and $\Theta(n)$ for randomized mechanisms.
- Study the fractional allocation version for the makespan as well as the max-min objective (fairness).

Other problems

- Characterize the truthful mechanisms for more general settings such as the combinatorial auction problem
- Online mechanism design
  - Secretary problems
  - Competitive auctions