CS599: Algorithm Design in Strategic Settings
Fall 2012
Lecture 5: Prior-Free Single-Parameter Mechanism Design (Continued)

Instructor: Shaddin Dughmi

Administrivia

- Recall: HW1 due Friday 10/5
- Office hours next Tuesday rescheduled to noon-2pm.

Outline

Outline

Informally

- There is a single homogenous resource (items, bandwidth, clicks, spots in a knapsack, etc).
- There are constraints on how the resource may be divided up.
- Each player's private data is his "value (or cost) per unit resource."

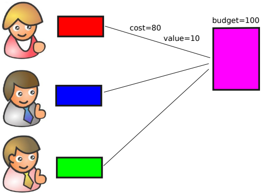
Informally

- There is a single homogenous resource (items, bandwidth, clicks, spots in a knapsack, etc).
- There are constraints on how the resource may be divided up.
- Each player's private data is his “value (or cost) per unit resource.”

Formally

- Set Ω of allocations is common knowledge.
- Each player i's type is a single real number t_i. Player i's type-space T_i is an interval in $\mathbb{R}$.
- Each allocation $x \in \Omega$ is a vector in $\mathbb{R}^n$.
- A player's utility for allocation x and payment p_i is $t_i x_i - p_i$.

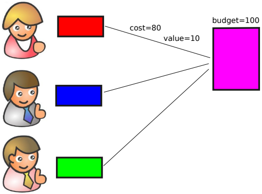
Example: Knapsack Allocation



- n players, player i with task requiring s_i time (the task's size)
- Machine has total processing time (capacity) B (public)
- Allocation: set of tasks with total size at most the capacity
- Player i has (private) value v_i for his task being included.

Objective: maximize welfare (sum of values of tasks included in knapsack).

Example: Knapsack Allocation



- n players, player i with task requiring s_i time (the task's size)
- Machine has total processing time (capacity) B (public)
- Allocation: set of tasks with total size at most the capacity
- Player i has (private) value v_i for his task being included.

Objective: maximize welfare (sum of values of tasks included in knapsack).

Modeling

Ω is the set of indicator vectors of players who fit in knapsack, $T_i = \mathbb{R}_+$, and $v_i \in T_i$ is player i's value for being included in the knapsack.

Example: Single-minded Combinatorial Allocation

- n players, m non-identical items
- For each player, publicly known subset A_i of items the player desires
- Allocations: partitions of items among players
- Each player has private value $v_i \in \mathbb{R}_+$, indicating his value for receiving a bundle including A_i (0 otherwise)

Objective: Maximize welfare (sum of values of players who receive their desired bundles)

Example: Single-minded Combinatorial Allocation

- n players, m non-identical items
- For each player, publicly known subset A_i of items the player desires
- Allocations: partitions of items among players
- Each player has private value $v_i \in \mathbb{R}_+$, indicating his value for receiving a bundle including A_i (0 otherwise)

Objective: Maximize welfare (sum of values of players who receive their desired bundles)

Modeling

Ω is the set of indicator vectors of players who can be jointly satisfied, $T_i = \mathbb{R}_+$, and $v_i \in T_i$ is player i's value for receiving his desired set.

Scheduling

- Designer has m jobs, with publicly known sizes $p_1, \ldots, p_m$
- n players, each own a machine
- Allocation: schedule mapping jobs onto machines
- Player i's private data t_i is his time (cost) per unit job scheduled on his machine.

Objective: Minimize makespan (the maximum, over machines, of time spent processing)

Scheduling

- Designer has m jobs, with publicly known sizes $p_1, \dots, p_m$
- n players, each own a machine
- Allocation: schedule mapping jobs onto machines
- Player i's private data t_i is his time (cost) per unit job scheduled on his machine.

Objective: Minimize makespan (the maximum, over machines, of time spent processing)

Modeling

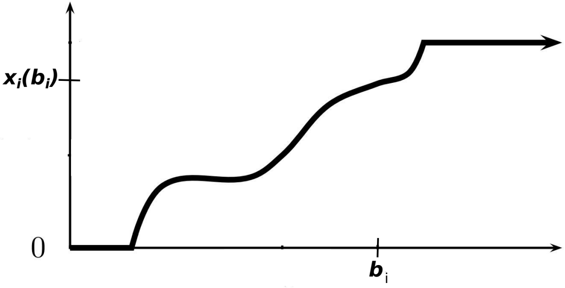
$\Omega \subseteq \mathbb{R}_+^n$ is the family of work vectors that can be induced by scheduling jobs with sizes $p_1, \dots, p_m$. Player's type t_i is his cost per unit job, and $T_i \in \mathbb{R}_+$. Utility of player i for load vector x is $p_i - t_i x_i$.
(Note we flipped signs)

Myerson's Lemma (Dominant Strategy)

A mechanism (x, p) for a single-parameter problem is dominant-strategy truthful if and only if for every player i and fixed reports b_{-i} of other players,

- $x_i(b_i)$ is a monotone non-decreasing function of b_i
- $p_i(b_i)$ is an integral of $b_i \; dx_i$. Specifically, when $p_i(0) = 0$ then

$$p_i(b_i) = b_i \cdot x_i(b_i) - \int_{b=0}^{b_i} x_i(b)db$$

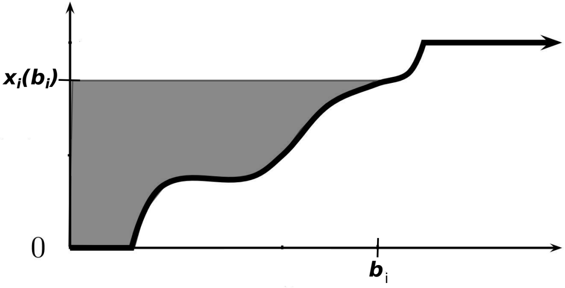


Myerson's Lemma (Dominant Strategy)

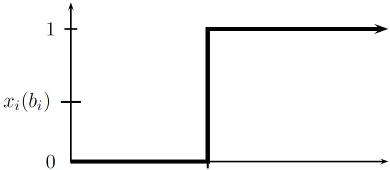
A mechanism (x, p) for a single-parameter problem is dominant-strategy truthful if and only if for every player i and fixed reports b_{-i} of other players,

- $x_i(b_i)$ is a monotone non-decreasing function of b_i
- $p_i(b_i)$ is an integral of $b_i \; dx_i$. Specifically, when $p_i(0) = 0$ then

$$p_i(b_i) = b_i \cdot x_i(b_i) - \int_{b=0}^{b_i} x_i(b) db$$



Interpretation of Myerson’s Lemma



Single-item Allocation

In the case of a deterministic mechanism.

- Monotonicity: If a player wins on certain bids, he remains a winner by increasing his bid (assuming other bids held fixed)
- The player must pay his critical value if he wins: the minimum bid he needs to win.

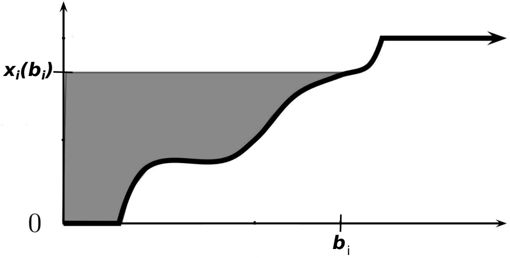
Therefore, Vickrey is the unique welfare-maximizing, individually rational, single-item auction.

Holds for every problem with a binary (win/lose) outcome per player.

Interpretation of Myerson's Lemma

General Interpretation

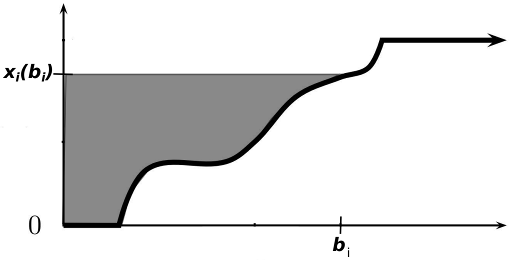
As player increases his reported value per unit of resource, he pays for each additional chunk of resource at a rate equal to the minimum report needed to win that chunk.



Interpretation of Myerson's Lemma

General Interpretation

As player increases his reported value per unit of resource, he pays for each additional chunk of resource at a rate equal to the minimum report needed to win that chunk.



Equivalently. . .

As player decreases his reported cost per unit of work, he is paid for each additional chunk of work at a rate equal to the maximum report at which he gets that chunk.

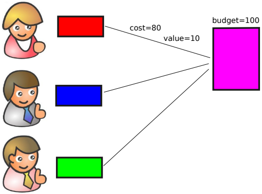
Next Up

We will embark on designing truthul mechanisms that run in polynomial time, for less trivial problems who'se non-strategic variant is NP-hard.

- Knapsack allocation
- Single-minded combinatorial allocation
- Scheduling

Outline

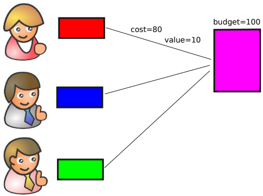
Knapsack Allocation



- n players, player i with task requiring s_i time (the task's size)
- Machine has total processing time (capacity) B (public)
- Allocation: set of tasks with total size at most the capacity
- Player i has (private) value v_i for his task being included.

Objective: maximize welfare (sum of values of tasks included in knapsack).

Knapsack Allocation



- n players, player i with task requiring s_i time (the task's size)
- Machine has total processing time (capacity) B (public)
- Allocation: set of tasks with total size at most the capacity
- Player i has (private) value v_i for his task being included.

Objective: maximize welfare (sum of values of tasks included in knapsack).

Modeling

Ω is the set of indicator vectors of players who fit in knapsack, $T_i = \mathbb{R}_+$, and $v_i \in T_i$ is player i's value for being included in the knapsack.

Design Goals

Want a mechanism (allocation rule and payment rule) satisfying the following properties:

1. Dominant strategy Truthfulness
2. Individual rationality: payment should be less than (reported) value for allocation

By Myerson's Lemma, these are satisfied if and only if the allocation rule is monotone, and the payment rule is the (unique) one indicated by Myerson's Lemma.

Design Goals

Want a mechanism (allocation rule and payment rule) satisfying the following properties:

1. Dominant strategy Truthfulness
2. Individual rationality: payment should be less than (reported) value for allocation

By Myerson's Lemma, these are satisfied if and only if the allocation rule is monotone, and the payment rule is the (unique) one indicated by Myerson's Lemma.

3. Polynomial time: The allocation algorithm must run in time polynomial in n, and the maximum number of bits in any of the real number inputs.

Design Goals

Want a mechanism (allocation rule and payment rule) satisfying the following properties:

1. Dominant strategy Truthfulness
2. Individual rationality: payment should be less than (reported) value for allocation

By Myerson's Lemma, these are satisfied if and only if the allocation rule is monotone, and the payment rule is the (unique) one indicated by Myerson's Lemma.

3. Polynomial time: The allocation algorithm must run in time polynomial in n, and the maximum number of bits in any of the real number inputs.
4. Worst-case approximation ratio: close to 1.

Recall: the approximation ratio of an allocation algorithm is the maximum, over all instances, of the ratio of the optimum welfare to that gotten by the algorithm.

Dropping Polynomial Time

Claim

A welfare-maximizing allocation rule, i.e. one computing $S^* \in \Omega$ maximizing $\sum_{i \in S} v_i$ is monotone.

- Computable in time exponential in n (brute force: try all subsets $S \subseteq [n]$ of players)
- The Myerson payment rule can also be computed using brute force in time exponential in n.

Proof of Monotonicity

- Assume player i wins when players report (v_{-i}, v_i).
- Consider bid vector $(v_{-i}, \widehat{v}_i)$ with $\widehat{v}_i > v_i$. Does i still win?

Proof of Monotonicity

- Assume player i wins when players report (v_{-i}, v_i).
- Consider bid vector $(v_{-i}, \widehat{v}_i)$ with $\widehat{v}_i > v_i$. Does i still win?
- Divide Ω into
  - Ω_i: sets including i
  - $\overline{\Omega_i}$: feasible sets excluding i

Proof of Monotonicity

- Assume player i wins when players report (v_{-i}, v_i).
- Consider bid vector $(v_{-i}, \widehat{v}_i)$ with $\widehat{v}_i > v_i$. Does i still win?
- Divide Ω into
  - Ω_i: sets including i
  - $\overline{\Omega_i}$: feasible sets excluding i
- S^* lies in Ω_i when player i reports v_i

Proof of Monotonicity

- Assume player i wins when players report (v_{-i}, v_i).
- Consider bid vector $(v_{-i}, \widehat{v}_i)$ with $\widehat{v}_i > v_i$. Does i still win?
- Divide Ω into
  - Ω_i: sets including i
  - $\overline{\Omega_i}$: feasible sets excluding i
- S^* lies in Ω_i when player i reports v_i
- Increasing to $\widehat{v}_i$ (holding v_{-i} fixed)
  - Welfare of each $S \in \Omega_i$ strictly increases.
  - Welfare of each $S \in \overline{\Omega_i}$ unchanged.

Proof of Monotonicity

- Assume player i wins when players report (v_{-i}, v_i).
- Consider bid vector $(v_{-i}, \widehat{v}_i)$ with $\widehat{v}_i > v_i$. Does i still win?
- Divide Ω into
  - Ω_i: sets including i
  - $\overline{\Omega_i}$: feasible sets excluding i
- S^* lies in Ω_i when player i reports v_i
- Increasing to $\widehat{v}_i$ (holding v_{-i} fixed)
  - Welfare of each $S \in \Omega_i$ strictly increases.
  - Welfare of each $S \in \overline{\Omega_i}$ unchanged.
- Since optimum was in Ω_i before increase, it remains in Ω_i.

Payments

Exercise

Write an expression for the critical payment of player i, as a function of the reports v_{-i} of other players. Your expression should be computable in time exponential in number of players.

Computational Complexity Facts

Assuming the sizes and values are written in binary,

Fact

The Knapsack problem is NP-hard.

i.e. unless $P = NP$, there is no optimal algorithm that runs in time polynomial in the length of the description of the input.

Our previous monotone algorithm can not be implemented in polynomial time, unless $P = NP$.

Computational Complexity Facts

Fact

The Knapsack problem admits a fully polynomial-time approximation scheme.

i.e. A $(1+\epsilon)$-approximation algorithm running in time polynomial in length of the description of the input and $1/\epsilon$.

Computational Complexity Facts

Fact

The Knapsack problem admits a fully polynomial-time approximation scheme.

i.e. A $(1+\epsilon)$-approximation algorithm running in time polynomial in length of the description of the input and $1/\epsilon$.

Knapsack FPTAS

1. Input: sizes $\vec{s}$, budget B, and values $\vec{v}$.

Computational Complexity Facts

Fact

The Knapsack problem admits a fully polynomial-time approximation scheme.

i.e. A $(1+\epsilon)$-approximation algorithm running in time polynomial in length of the description of the input and $1/\epsilon$.

Knapsack FPTAS

1. Input: sizes $\vec{s}$, budget B, and values $\vec{v}$.
2. Round down value of each job to nearst multiple of $\epsilon v_{\max}$

Computational Complexity Facts

Fact

The Knapsack problem admits a fully polynomial-time approximation scheme.

i.e. A $(1 + \epsilon)$-approximation algorithm running in time polynomial in length of the description of the input and $1/\epsilon$.

Knapsack FPTAS

1. Input: sizes $\vec{s}$, budget B, and values $\vec{v}$.
2. Round down value of each job to nearst multiple of $\epsilon v_{\max}$
3. Dynamic Programming maximizes rounded value:
    - Subproblems indexed by $i \in [n]$ and $k \in \{0, \ldots, \lceil n/\epsilon \rceil\}$
    - Subproblem(i,k): Find the minimum size set $S \subseteq \{i, \ldots, n\}$ with rounded value at least $k \epsilon v_{\max}$, if any.

Computational Complexity Facts

Fact

The Knapsack problem admits a fully polynomial-time approximation scheme.

i.e. A $(1+\epsilon)$-approximation algorithm running in time polynomial in length of the description of the input and $1/\epsilon$.

Knapsack FPTAS

1. Input: sizes $\vec{s}$, budget B, and values $\vec{v}$.
2. Round down value of each job to nearst multiple of $\epsilon v_{\max}$
3. Dynamic Programming maximizes rounded value:
   - Subproblems indexed by $i \in [n]$ and $k \in \{0, \ldots, \lceil n/\epsilon \rceil\}$
   - Subproblem(i,k): Find the minimum size set $S \subseteq \{i, \ldots, n\}$ with rounded value at least $k\epsilon v_{\max}$, if any.
4. Polynomial number of subproblems. Of the ones with total size $\leq B$, find the subproblem with maximum k.

Standard FPTAS Non-monotone

Unfortunately

The DP FPTAS for knapsack is non-monotone…

Non-monotonicity

- Capacity $B = 4$
- Task 1 has value $v_1 = 30$ and size $1 + 3\delta$
- Many tasks have value 20 and size $1 - \delta$
- Many tasks have value 22 and size 1.
- $\epsilon = 1/3$

Check: Task 1 is in solution of DP when $v_1 = 30$, but not when $v_1 = 33$.

Combining Polynomial-time, Truthfulness, and a Good Approximation

This is part of a general trend…

Trend

In many cases we will see in this course, an optimal (and exponential-time) mechanism is monotone (i.e. amenable to truthfulness), but traditional polynomial-time approximation algorithms are not.

This raises the following philosophical question, which has received much research attention

Question

Are computational tractability and incentive-compatibility in conflict? In particular can we do nearly as well, in terms of approximation ratio, with a truthful polynomial-time mechanism as with a non-truthful polynomial-time algorithm?

Combining Polynomial-time, Truthfulness, and a Good Approximation

Myerson's monotonicity lemma helps!

Observation

Polynomial-time truthful mechanism design reduces to monotone polynomial-time approximation algorithm design.

Computing payments for a monotone algorithm is usually the easy part, due to a bunch of fairly general "tricks"... (Next homework)

Upshot

Forget about truthfulness, incentives, etc. Just design a monotone algorithm for the non-strategic problem, with a good approximation ratio.

A Monotone, Polynomial-time 2-Approximation Algorithm

Algorithm

1. Input: capacity B, sizes $s_1, \ldots, s_n$, values $v_1, \ldots, v_n$.
2. Sort jobs by density $d_i = v_i / s_i$.
3. Greedily pack jobs in the knapsack in decreasing order of density, allowing overflow of one job.
   - Call these the dense jobs, and the least dense of them the overflow job.
4. Remove either the overflow job or everything else, whichever leaves the most total value in the knapsack.

Proof of Approximation

Let OPT denote the maximum value of any feasible set of jobs (i.e. set of jobs that fits in the knapsack).

Claim

After step 3 (with overflow), the total value of jobs in the knapsack (the dense jobs) is at least OPT.

Should be obvious, (most bang-per-buck) ...

Proof of Approximation

Let OPT denote the maximum value of any feasible set of jobs (i.e. set of jobs that fits in the knapsack).

Claim

After step 3 (with overflow), the total value of jobs in the knapsack (the dense jobs) is at least OPT.

Should be obvious, (most bang-per-buck) . . .

Therefore. . .

Either the overflow job, or the other dense jobs have value at least $OPT/2$, so picking the better of the two gives a 2-approximation.

Proof of Monotonicity

- Fix sizes $s_1, \ldots, s_n$, capacity B, and reports v_{-i} of players other than i.
- Assume i wins when reporting v_i. Consider what happens when reporting $\widehat{v}_i > v_i$. Two cases:

Proof of Monotonicity

- Fix sizes $s_1, \ldots, s_n$, capacity B, and reports v_{-i} of players other than i.
- Assume i wins when reporting v_i. Consider what happens when reporting $\widehat{v}_i > v_i$. Two cases:

1. On report v_i, i was a dense job but not the overflow job, and the non-overflow dense jobs had greater total value than the overflow job.
   - When increasing report to $\widehat{v}_i$, job i remains a non-overflow dense job.
   - Moreover the non-overflow dense jobs remain better than the overflow job.
   - Therefore, i still wins.

Proof of Monotonicity

- Fix sizes $s_1, \ldots, s_n$, capacity B, and reports v_{-i} of players other than i.
- Assume i wins when reporting v_i. Consider what happens when reporting $\widehat{v}_i > v_i$. Two cases:

2. On report v_i, i was the overflow job, and v_i was greater the total value of other dense jobs combined. When increasing to $\widehat{v}_i$, two cases:
   (a) i remains the overflow job, and its value remains greater than the total value of other jobs combined.
   (b) i moves up in the density order, and becomes a non-overflow dense job.
      - Need to show that the non-overflow dense jobs are chosen.
      - New overflow job j was one of the old dense jobs other than i.
      - Back then, $v_i > v_j$.
      - Value of new dense jobs is at least $\widehat{v}_i > v_i > v_j$, so new overflow job j is tossed.

Computing Payments

How do we compute payments for each player i?

- Let (v_{-i}, v_i) be the reported values.
- If i does not win on input (v_{-i}, v_i), then we know his payment is zero.
- Otherwise, need to charge him the minimum b_i such that i would have been a winner had the input been (v_{-i}, b_i).

Computing Payments

How do we compute payments for each player i?

- Let (v_{-i}, v_i) be the reported values.
- If i does not win on input (v_{-i}, v_i), then we know his payment is zero.
- Otherwise, need to charge him the minimum b_i such that i would have been a winner had the input been (v_{-i}, b_i).

Observe

As i increases b_i from 0, he can only ever go from winning to not winning when he moves up in the density order.

Computing Payments

How do we compute payments for each player i?

- Let (v_{-i}, v_i) be the reported values.
- If i does not win on input (v_{-i}, v_i), then we know his payment is zero.
- Otherwise, need to charge him the minimum b_i such that i would have been a winner had the input been (v_{-i}, b_i).

Observe

As i increases b_i from 0, he can only ever go from winning to not winning when he moves up in the density order.

Therefore, the critical point must be such that $b_i/s_i = v_j/s_j$ for some $j \neq i$.

Only $n - 1$ such points, try running our algorithm on all of them!!

Computing Payments

How do we compute payments for each player i?

- Let (v_{-i}, v_i) be the reported values.
- If i does not win on input (v_{-i}, v_i), then we know his payment is zero.
- Otherwise, need to charge him the minimum b_i such that i would have been a winner had the input been (v_{-i}, b_i).

Observe

As i increases b_i from 0, he can only ever go from winning to not winning when he moves up in the density order.

Therefore, the critical point must be such that $b_i/s_i = v_j/s_j$ for some $j \neq i$.

Only $n-1$ such points, try running our algorithm on all of them!!

Note: this payment computation process requires $n-1$ runs of our allocation algorithm per player!!

Further Results for Knapsack

Theorem

There exists a monotone FPTAS for the Knapsack problem. The associated Myerson payments can be computed in polynomial-time, yielding a dominant-strategy truthful FPTAS.

Part of next homework. . .

Outline

Single-minded Combinatorial Allocation

- n players, m non-identical items
- For each player, publicly known subset A_i of items the player desires
- Allocations: partitions of items among players
- Each player has private value $v_i \in \mathbb{R}_+$, indicating his value for receiving a bundle including A_i (0 otherwise)

Objective: Maximize welfare (sum of values of players who receive their desired bundles)

Single-minded Combinatorial Allocation

- n players, m non-identical items
- For each player, publicly known subset A_i of items the player desires
- Allocations: partitions of items among players
- Each player has private value $v_i \in \mathbb{R}_+$, indicating his value for receiving a bundle including A_i (0 otherwise)

Objective: Maximize welfare (sum of values of players who receive their desired bundles)

Modeling

Ω is the set of indicator vectors of players who can be jointly satisfied, $T_i = \mathbb{R}_+$, and $v_i \in T_i$ is player i's value for receiving his desired set.

Design Goals

Want a mechanism (allocation rule and payment rule) satisfying the following properties:

1. Dominant strategy Truthfulness
2. Individual rationality: payment should be less than (reported) value for allocation
3. Polynomial time: The allocation algorithm must run in time polynomial in n,m, and the maximum number of bits in any of the real number inputs.
4. Worst-case approximation ratio: As small as possible, given computational complexity assumptions.

Dropping Polynomial Time

Claim

The welfare-maximizing allocation rule is monotone.

- Computable in time exponential in m (brute force: try all assignments of m items to n players)
- The Myerson payment rule can also be computed using brute force in time exponential in n.

Dropping Polynomial Time

Claim

The welfare-maximizing allocation rule is monotone.

- Computable in time exponential in m (brute force: try all assignments of m items to n players)
- The Myerson payment rule can also be computed using brute force in time exponential in n.

Proof of monotonicity is essentially identical to that for knapsack. In fact, welfare maximization is always monotone no matter the problem! (Check this)

Computational Complexity Facts

Fact

The problem of finding the allocation maximizing welfare is NP-hard. Moreover, there is no polynomial time approximation algorithm with ratio $O(m^{1/2-\epsilon})$ for any constant ϵ, unless $P = NP$.

Computational Complexity Facts

Fact

The problem of finding the allocation maximizing welfare is NP-hard. Moreover, there is no polynomial time approximation algorithm with ratio $O(m^{1/2-\epsilon})$ for any constant ϵ, unless $P = NP$.

Fact

There is a polynomial-time $\sqrt{m}$-approximation algorithm for welfare maximization in single-minded combinatorial allocation.

Approximation algorithms known prior to AMD research were based on linear programming and randomized rounding, and were non-monotone.

Computational Complexity Facts

Fact

The problem of finding the allocation maximizing welfare is NP-hard. Moreover, there is no polynomial time approximation algorithm with ratio $O(m^{1/2-\epsilon})$ for any constant ϵ, unless $P = NP$.

Fact

There is a polynomial-time $\sqrt{m}$-approximation algorithm for welfare maximization in single-minded combinatorial allocation.

Approximation algorithms known prior to AMD research were based on linear programming and randomized rounding, and were non-monotone.

Too complicated/messy to show you non-monotinicity, so let's not worry about it and just design a monotone $\sqrt{m}$-approximation algorithm.

Algorithm Attempt 1

1. Sort players in decreasing order of value v_i
2. Go through players in order, awarding i his desired set A_i so long as it hasn't been allocated already.
   - i.e. so long as there is no j with $v_j > v_i$ with $A_j \bigcap A_i \neq \emptyset$.

Algorithm Attempt 1

1. Sort players in decreasing order of value v_i
2. Go through players in order, awarding i his desired set A_i so long as it hasn't been allocated already.
   - i.e. so long as there is no j with $v_j > v_i$ with $A_j \bigcap A_i \neq \emptyset$.

Clearly Polynomial time. Remains to show monotonicity and approximation ratio.

Proof of Monotonicity

- Fix desired bundles $A_1, \ldots, A_n$, and reports v_{-i} of players other than i.
- Assume i wins when reporting v_i.
  - For all j with $v_j > v_i$, we have $A_j \bigcap A_i = \emptyset$.
- Consider what happens when reporting $\widehat{v}_i > v_i$.
  - i moves up in the order.
  - For all j with $v_j > \widehat{v}_i > v_i$, we have $A_j \bigcap A_i = \emptyset$.
  - Therefore i still wins.

Approximation: Bad Example 1

- $n = m + 1$
- $A_1 = [m]$, $v_1 = 1 + \epsilon$
- A_j is a (different) singleton for each $j \neq i$, and $v_j = 1$.

Approximation: Bad Example 1

- $n = m + 1$
- $A_1 = [m]$, $v_1 = 1 + \epsilon$
- A_j is a (different) singleton for each $j \neq i$, and $v_j = 1$.

Our algorithm chooses player 1 as the sole winner, whereas it is optimal to choose all the others. Ratio is roughly m.

Approximation: Bad Example 1

- $n = m + 1$
- $A_1 = [m]$, $v_1 = 1 + \epsilon$
- A_j is a (different) singleton for each $j \neq i$, and $v_j = 1$.

Our algorithm chooses player 1 as the sole winner, whereas it is optimal to choose all the others. Ratio is roughly m.

Problem: We didn't take into account that player 1 wanted too many items! Lets try to normalize by number of items demanded!

Algorithm Attempt 2

1. Sort players in decreasing order of value per item desired $v_i/|A_i|$
2. Go through players in order, awarding i his desired set A_i so long as it hasn't been allocated already.
   - i.e. so long as there is no j with $\frac{v_j}{|A_j|} > \frac{v_i}{|A_i|}$ with $A_j \bigcap A_i \neq \emptyset$.

Algorithm Attempt 2

1. Sort players in decreasing order of value per item desired $v_i/|A_i|$
2. Go through players in order, awarding i his desired set A_i so long as it hasn't been allocated already.
   - i.e. so long as there is no j with $\frac{v_j}{|A_j|} > \frac{v_i}{|A_i|}$ with $A_j \bigcap A_i \neq \emptyset$.

Clearly Polynomial time. Monotonicity proof identical to Algorithm 1.
What about approximation?

Approximation: Bad Example 2

- $n = 2$
- $A_1 = 1$, $v_1 = 1 + \epsilon$
- $A_2 = [m]$, $v_2 = m$

Approximation: Bad Example 2

- $n = 2$
- $A_1 = 1$, $v_1 = 1 + \epsilon$
- $A_2 = [m]$, $v_2 = m$

Our algorithm chooses player 1 as the sole winner, whereas it is optimal to choose player 2. Ratio is roughly m.

Approximation: Bad Example 2

- $n = 2$
- $A_1 = 1$, $v_1 = 1 + \epsilon$
- $A_2 = [m]$, $v_2 = m$

Our algorithm chooses player 1 as the sole winner, whereas it is optimal to choose player 2. Ratio is roughly m.

Problem: We didn't take into account that player 1's value was too small, and he excluded player 2 entirely.

Attempt 3

A happy medium...

Algorithm Attempt 3

1. Sort players in decreasing order of $v_i/\sqrt{|A_i|}$
2. Go through players in order, awarding i his desired set A_i so long as it hasn't been allocated already.
   - i.e. so long as there is no j with $\frac{v_j}{\sqrt{|A_j|}} > \frac{v_i}{\sqrt{|A_i|}}$ with $A_j \bigcap A_i \neq \emptyset$.

Attempt 3

A happy medium...

Algorithm Attempt 3

1. Sort players in decreasing order of $v_i/\sqrt{|A_i|}$
2. Go through players in order, awarding i his desired set A_i so long as it hasn't been allocated already.
   - i.e. so long as there is no j with $\frac{v_j}{\sqrt{|A_j|}} > \frac{v_i}{\sqrt{|A_i|}}$ with $A_j \bigcap A_i \neq \emptyset$.

Clearly Polynomial time. Monotonicity proof identical to Algorithms 1 and 2. What about approximation?

Attempt 3

A happy medium...

Algorithm Attempt 3

1. Sort players in decreasing order of $v_i/\sqrt{|A_i|}$
2. Go through players in order, awarding i his desired set A_i so long as it hasn't been allocated already.
   - i.e. so long as there is no j with $\frac{v_j}{\sqrt{|A_j|}} > \frac{v_i}{\sqrt{|A_i|}}$ with $A_j \bigcap A_i \neq \emptyset$.

Clearly Polynomial time. Monotonicity proof identical to Algorithms 1 and 2. What about approximation?

Theorem (Lehmann, O'Callahan, Shoham)

Algorithm 3 is a $\sqrt{m}$-approximation algorithm for welfare maximization in single-minded combinatorial allocation.

Proof of Approximation Ratio

- Let S^* be the set of satisfied players in the optimal solution, and let S be the set of satisfied players in algorithm's solution.
- For each $i \in S$, define the set B_i of players in S^* blocked by i
  - For $j \in S^*$, include $j \in B_i$ if i is the first player in the order s.t. $A_j \bigcap A_i \neq \emptyset$.
- Notice, the sets B_i partition S^*.

Convince Yourself

By a standard charging argument, it is sufficient to show that for each $i \in S$,

$$\sum_{j \in B_i} v_j \leq \sqrt{m}\, v_i$$

Proof of Approximation Ratio (Continued)

Will Show

For each i,

$$\sum_{j \in B_i} v_j \leq \sqrt{m}\, v_i$$

Computing Payments

How do we compute payments for each player i?

- Let (v_{-i}, v_i) be the reported values.
- If i does not win on input (v_{-i}, v_i), then we know his payment is zero.
- Otherwise, need to charge him the minimum b_i such that i would have been a winner had the input been (v_{-i}, b_i).

Computing Payments

How do we compute payments for each player i?

- Let (v_{-i}, v_i) be the reported values.
- If i does not win on input (v_{-i}, v_i), then we know his payment is zero.
- Otherwise, need to charge him the minimum b_i such that i would have been a winner had the input been (v_{-i}, b_i).

Observe

As i increases b_i from 0, he can only ever go from winning to not winning when he moves up in the order according to $v_i/\sqrt{|A_i|}$.

Computing Payments

How do we compute payments for each player i?

- Let (v_{-i}, v_i) be the reported values.
- If i does not win on input (v_{-i}, v_i), then we know his payment is zero.
- Otherwise, need to charge him the minimum b_i such that i would have been a winner had the input been (v_{-i}, b_i).

Observe

As i increases b_i from 0, he can only ever go from winning to not winning when he moves up in the order according to $v_i/\sqrt{|A_i|}$.

Therefore, the critical point must be such that $b_i/\sqrt{|A_i|} = v_j/\sqrt{|A_j|}$ for some $j \neq i$.

Only $n - 1$ such points, try running our algorithm on all of them!!

Computing Payments

How do we compute payments for each player i?

- Let (v_{-i}, v_i) be the reported values.
- If i does not win on input (v_{-i}, v_i), then we know his payment is zero.
- Otherwise, need to charge him the minimum b_i such that i would have been a winner had the input been (v_{-i}, b_i).

Observe

As i increases b_i from 0, he can only ever go from winning to not winning when he moves up in the order according to $v_i/\sqrt{|A_i|}$.

Therefore, the critical point must be such that $b_i/\sqrt{|A_i|} = v_j/\sqrt{|A_j|}$ for some $j \neq i$.

Only $n-1$ such points, try running our algorithm on all of them!!

Note: this payment computation process requires $n-1$ runs of our allocation algorithm per player!!

Outline