CS599: Algorithm Design in Strategic Settings
Fall 2012
Lecture 6: Prior-Free Single-Parameter Mechanism Design (Continued)

Instructor: Shaddin Dughmi

Administrivia

- Homework 1 due today.
- Homework 2 out sometime next week

Outline

Outline

Single-parameter Problems

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."

Single-parameter Problems

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

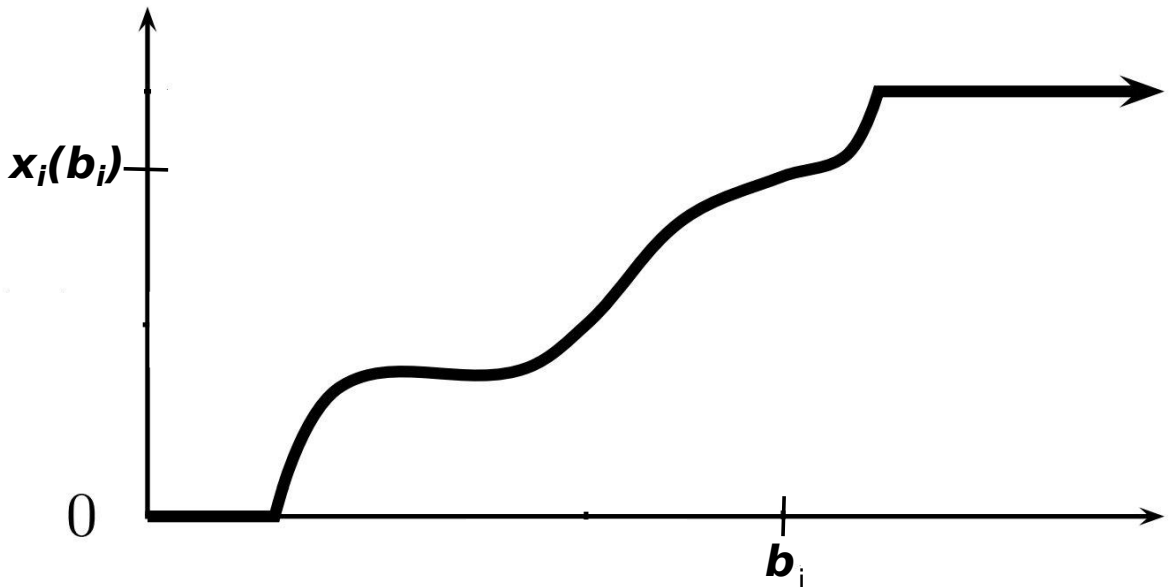
- Set Ω of allocations is common knowledge.
- Each player i's type is a single real number t_i. Player i's type-space T_i is an interval in $\mathbb{R}$.
- Each allocation $x \in \Omega$ is a vector in $\mathbb{R}^n$.
- A player's utility for allocation x and payment p_i is $t_i x_i - p_i$.

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, there is some pivot term $h_i(b_{-i})$ such that

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

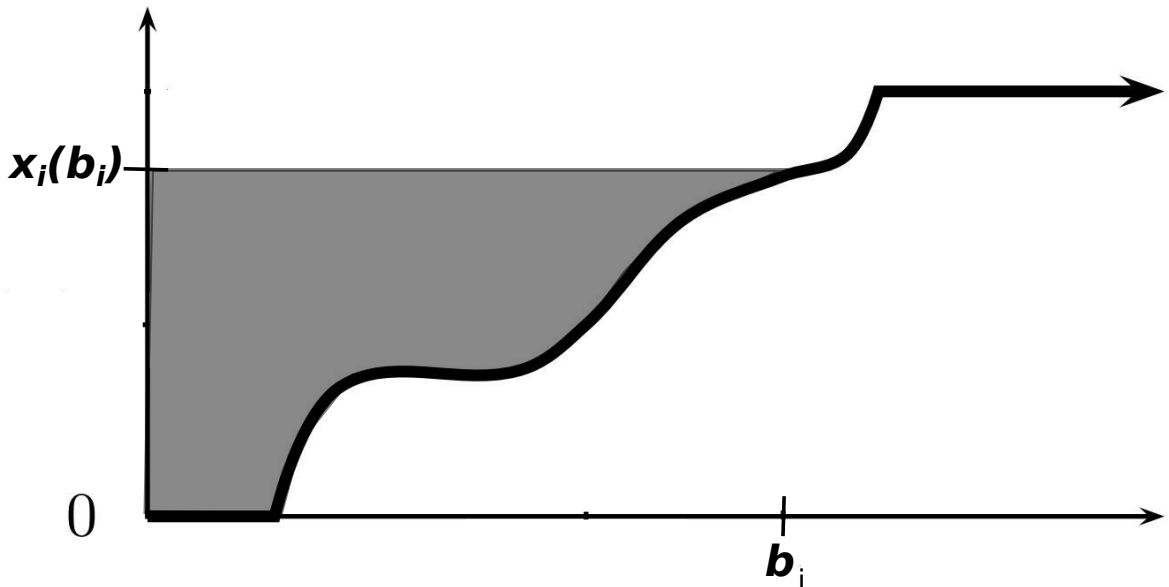


Myerson's Lemma (Dominant Strategy)

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, there is some pivot term $h_i(b_{-i})$ such that

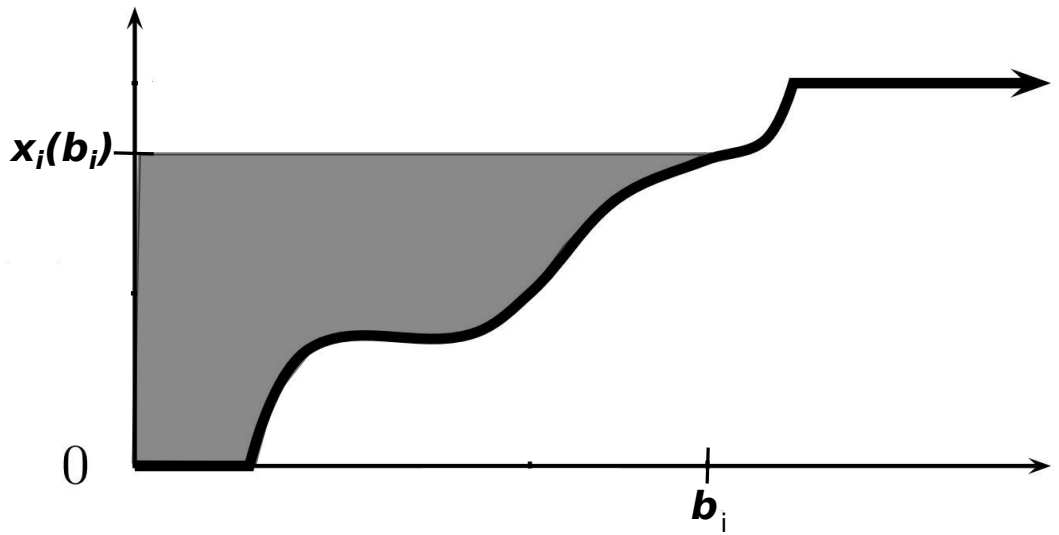
$$p_i(b_i) = h_i(b_{-i}) + b_i \cdot x_i(b_i) - \int_{b=0}^{b_i} x_i(b)db$$



Interpretation of Myerson's Lemma

General Interpretation

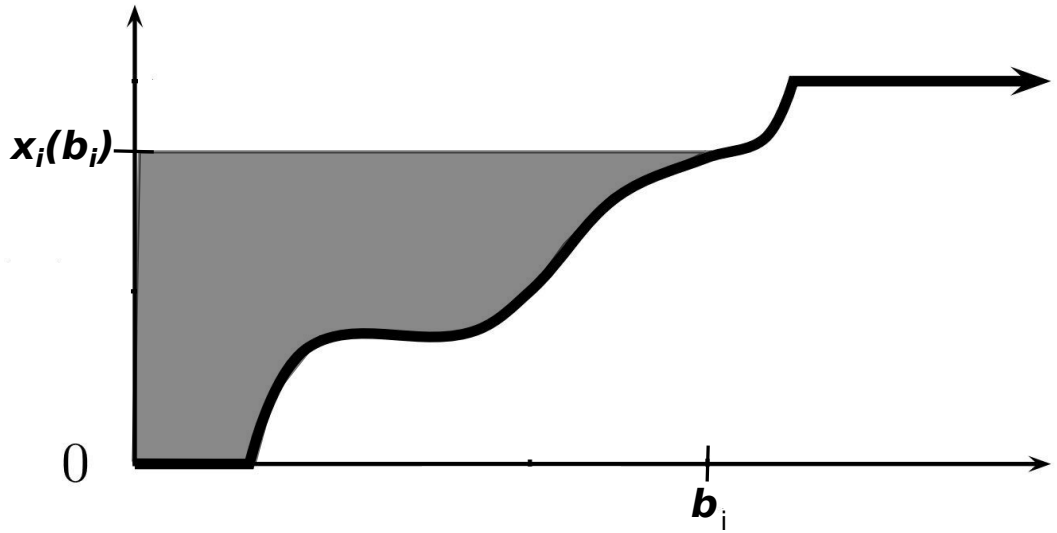
As player increases his reported value per unit of resource, he pays for each additional chunk of resource at a rate equal to the minimum report needed to win that chunk.



Interpretation of Myerson's Lemma

General Interpretation

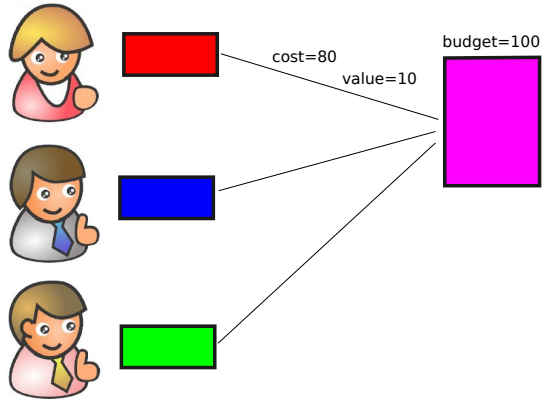
As player increases his reported value per unit of resource, he pays for each additional chunk of resource at a rate equal to the minimum report needed to win that chunk.



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.

Recap: Knapsack Allocation



We Showed

- Exact solution of the problem (in exponential time) gives a monotone algorithm, yielding a truthful mechanism by Myerson's Lemma.
- The canonical FPTAS is non-monotone, and therefore cannot be turned into a truthful mechanism.
- We showed a monotone, polynomial-time 2-approximation algorithm, and a corresponding truthful mechanism.
- Next HW: A truthful FPTAS.

Recap: Single-minded Combinatorial Allocation

We Showed

- Exact solution of the problem (in exponential time) gives a monotone algorithm, yielding a truthful mechanism by Myerson's Lemma.
- We showed a monotone, polynomial-time $\sqrt{m}$-approximation algorithm, and a corresponding truthful mechanism.

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
  - Non-binary
  - Mechanism will be randomized

Outline

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 $p_1, \ldots, p_m$
- 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)

Modeling

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

Design Goals

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

1. Dominant strategy Truthfulness
2. Payment to a machine receiving no work should be 0

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. Payment to a machine receiving no work should be 0

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:

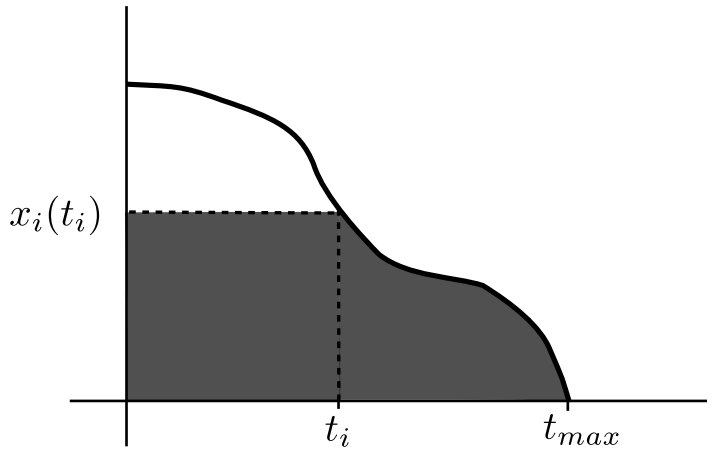
1. Dominant strategy Truthfulness
2. Payment to a machine receiving no work should be 0

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 makespan of the schedule out by the algorithm to the optimum makespan.

Reinterpreting Myerson's Lemma



Myerson's Lemma (Cost Version)

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

- The workload $x_i(t_i)$ of machine i is a non-increasing function of t_i.
- $p_i(t_i)$ is an integral of $t_i dx_i$. Assuming some $t_{\max}$ such that the machine gets no work, and requiring $p_i(t_{\max}) = 0$, gives

$$p_i(t_i) = t_i x_i(t_i) + \int_{t=t_i}^{t_{\max}} x_i(t)dt.$$

Dropping Polynomial Time

Claim

The allocation rule that computes a makespan-minimizing schedule, breaking ties via some fixed global order on schedules, is monotone.

- Computable in time $O(m^n)$ (brute force: try all schedules)
- The Myerson payment rule can also be computed using brute force in time $O(m)^{O(n)}$.

Proof of Monotonicity

- Fix reports t_{-i} of machines other than i
- For every fixed schedule σ with workloads $(w_1, \ldots, w_n)$, makespan of σ as a function of t_i is

$$makespan_\sigma(t_i) = \max_j w_j t_j = \max(C_\sigma(t_{-i}), w_i t_i)$$

for $C_\sigma(t_{-i}) = \max_{j \neq i} w_j t_j$

Proof of Monotonicity

- Fix reports t_{-i} of machines other than i
- For every fixed schedule σ with workloads $(w_1, \ldots, w_n)$, makespan of σ as a function of t_i is

$$makespan_\sigma(t_i) = \max_j w_j t_j = \max(C_\sigma(t_{-i}), w_i t_i)$$

  for $C_\sigma(t_{-i}) = \max_{j \neq i} w_j t_j$
- Assume σ with workloads $(w_1, \ldots, w_n)$ is output when machine i bids t_i.
- Consider machine i slowing down from t_i to $t'_i = t_i + \epsilon$, and algorithm outputting σ' with loads $(w'_1, \ldots, w'_n)$, two cases

Proof of Monotonicity

- Fix reports t_{-i} of machines other than i
- For every fixed schedule σ with workloads $(w_1, \ldots, w_n)$, makespan of σ as a function of t_i is

$$makespan_\sigma(t_i) = \max_j w_j t_j = \max(C_\sigma(t_{-i}), w_i t_i)$$

  for $C_\sigma(t_{-i}) = \max_{j \neq i} w_j t_j$
- Assume σ with workloads $(w_1, \ldots, w_n)$ is output when machine i bids t_i.
- Consider machine i slowing down from t_i to $t_i' = t_i + \epsilon$, and algorithm outputting σ' with loads $(w_1', \ldots, w_n')$, two cases
  1. Machine i is not the "bottleneck" in σ (i.e. $C_\sigma(t_i) > w_i t_i$): makespan of σ doesn't change, and makespan of every other schedule gets no better, so by consistent tie-breaking $\sigma' = \sigma$

Proof of Monotonicity

- Fix reports t_{-i} of machines other than i
- For every fixed schedule σ with workloads $(w_1, \ldots, w_n)$, makespan of σ as a function of t_i is

$$makespan_\sigma(t_i) = \max_j w_j t_j = \max(C_\sigma(t_{-i}), w_i t_i)$$

  for $C_\sigma(t_{-i}) = \max_{j \neq i} w_j t_j$
- Assume σ with workloads $(w_1, \ldots, w_n)$ is output when machine i bids t_i.
- Consider machine i slowing down from t_i to $t_i' = t_i + \epsilon$, and algorithm outputting σ' with loads $(w_1', \ldots, w_n')$, two cases
  1. Machine i is not the "bottleneck" in σ (i.e. $C_\sigma(t_i) > w_i t_i$): makespan of σ doesn't change, and makespan of every other schedule gets no better, so by consistent tie-breaking $\sigma' = \sigma$
  2. Machine i is the "bottleneck" in σ:

$$w_i t_i' = maksp an_\sigma(t_i') > makespan_{\sigma'}(t_i') \geq w_i' t_i'$$

Computing Payments

Observe

As t_i changes, schedule (and hence load on machine i) changes only when two curves $makespan_\sigma(t_i)$ and $makespan_{\sigma'}(t_i)$ cross, and any pair of such curves cross at most once.

Computing Payments

Observe

As t_i changes, schedule (and hence load on machine i) changes only when two curves $makespan_\sigma(t_i)$ and $makespan_{\sigma'}(t_i)$ cross, and any pair of such curves cross at most once.

To integrate the curve, simply enumerate the crossing points and vary t_i over all of them.

Computational Complexity Facts

Fact

Scheduling on related machines is strongly NP-hard. (Reduction from 3D Matching)

i.e. unless $P = NP$, there is no optimal algorithm, or even an FPTAS, 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

Theorem (Hochbaum and Shmoys)

Scheduling on related machines admits a polynomial-time approximation scheme (PTAS).

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

But, as usual, the original PTAS was non-monotone!

Next Up

A randomized, monotone, polynomial-time, 3-approximation algorithm.

But first, a note about randomized mechanisms.

A Note about Randomized Mechanisms

So far, we have implicitly only spoken of deterministic mechanisms (x, p). In general, since player utility is linear in amount of work, the following analogues of Myerson's lemma hold by immediately the same analysis.

Myerson's Lemma (Value Version)

A randomized 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

- $\widetilde{x}_i(b_i)$ is a monotone non-decreasing function of b_i
- $\widetilde{p}_i(b_i)$ is an integral of $b_i \; d\widetilde{x}_i$.

Where $\widetilde{x}_i(b_i)$ and $\widetilde{p}_i(b_i)$ are the expectations of $x_i(b_i)$ and $p_i(b_i)$ respectively.

A Note about Randomized Mechanisms

So far, we have implicitly only spoken of deterministic mechanisms (x, p). In general, since player utility is linear in amount of work, the following analogues of Myerson's lemma hold by immediately the same analysis.

Myerson's Lemma (Cost Version)

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

- The expected workload $\widetilde{x}_i(t_i)$ of machine i is a non-increasing function of t_i.
- $\widetilde{p}_i(t_i)$ is an integral of $t_i d\widetilde{x}_i$.

Fractional Schedules

A fractional schedule is one that assigns jobs to machines fractionally.

We will discuss later how to interpret a fractional schedule as a “real” schedule via randomized rounding.

Fractional Schedules

A fractional schedule is one that assigns jobs to machines fractionally.

We will discuss later how to interpret a fractional schedule as a "real" schedule via randomized rounding.

Valid fractional schedule

Given reports $t_1, \ldots, t_n$ (times per unit job), we say a fractional schedule σ is T-valid if the makespan of σ is at most T, and moreover whenever part of job j is assigned to machine i, we have $p_j t_j \geq T$.

In other words, machine i has time to process job j in its entirety within the makespan time.

Greedy Fractional Scheduling

Fix job sizes and machine reports. Given a target makespan of T, the following is the "obvious" way to construct a fractional schedule satisfying the target

Algorithm Greedy-Fractional(T)

1. Sort jobs in decreasing order of size, and sort machines in increasing order of time per unit job
2. Think of machine i as a bin of capacity $c_i = T/t_i$. Bins are sorted in decreasing order of size.
3. Greedily place jobs, in order, in bins, also in order. Fractionally cut jobs when a bin overflows and continue.

Greedy Fractional Scheduling

Algorithm Greedy-Fractional(T)

1. Sort jobs in decreasing order of size, and sort machines in increasing order of time per unit job
2. Think of machine i as a bin of capacity $c_i = T/t_i$. Bins are sorted in decreasing order of size.
3. Greedily place jobs, in order, in bins, also in order. Fractionally cut jobs when a bin overflows and continue.

Claim

If there is a T-valid fractional schedule, then Greedy-Fractional(T) finds one.

Greedy Fractional Scheduling

Proof

- Assume it doesn't find one.
- If total capacity of bins is insufficient, then there is no fractional schedule of makespan T.
- Otherwise, a job j is partially assigned to a machine i on which it does not fit whole — i.e. $c_i < p_j$.
- By greedy nature of algorithm, total size of jobs p_j or larger exceeds total capacity of bins p_j or larger.
- Therefore, no T-valid fractional schedule exists.

A Monotone, Polynomial-time 2-Approximation Algorithm

Algorithm

1. Input: job sizes $p_1, \ldots, p_m$, costs per unit load $t_1, \ldots, t_n$
2. Calculate T^*: the minimum time T such that a T-valid fractional schedule exists. (Will see how later)
3. Compute $\sigma_{frac} =$ Greedy-fractional(T^*).
4. Randomized Rounding: Assign each job j to machines randomly, with probability proportional to the fractions of the job on each machine in σ_{frac}. Let σ be the resulting (integral) schedule.
5. Output σ

Proof of Approximation

Claim

$T^* \leq OPT$

The optimal integral schedule is OPT-valid.

Proof of Approximation

Claim

$T^* \leq OPT$

The optimal integral schedule is OPT-valid.

Claim

There are at most two jobs partially to each machine.

In particular, the first job and the last job assigned to the machine.

Proof of Approximation

Claim

$T^* \leq OPT$

The optimal integral schedule is OPT-valid.

Claim

There are at most two jobs partially to each machine.

In particular, the first job and the last job assigned to the machine.

Therefore,

Fix any machine. Since each partial job fits on the machine whole (i.e. in time T^*), then even if both partial jobs end up on the machine whole, the total time spent processing is at most $3T^* \leq 3OPT$.

Proof of Monotonicity

Observe

The expected load on each machine is the fractional load in the chosen fractional schedule.

For all but possibly the slowest machine, this is $c_i = T^*/t_i$.

Proof of Monotonicity

Observe

The expected load on each machine is the fractional load in the chosen fractional schedule.

For all but possibly the slowest machine, this is $c_i = T^*/t_i$.

We are now ready to prove monotonicity of load. We will do it for non-slowest machines (slowest is an easy exercise).

- Fix t_{-i}. Let T^* be the fractional makespan on report t_i, and T' be fractional makespan on report $t'_i = (1+\epsilon)t_i$.
- Expected load before slowing down was T^*/t_i
- Observe $T' \geq T^*$ because a machine slowed down.
- $T' \leq (1+\epsilon)T^*$: Slowing down one machine by a factor of $(1+\epsilon)$ increases makespan of any schedule by at most $(1+\epsilon)$ factor.
- New load on machine i after slowing down is at most $T'/t'_i \leq (1+\epsilon)T^*/(1+\epsilon)t_i = T^*/t_i$

Loose End: Computing T^*

We still need to show that we can calculate T^*, the minimum time such that a T^*-valid fractional schedule exists.

Lemma

Let jobs be sorted such that $p_1 \geq \ldots \geq p_n$, and machines sorted such that $t_1 \leq \ldots \leq t_m$.

$$T^* = \max_{j=1}^{n} \min_{i=1}^{m} \max \left\{ p_j t_i, \frac{\sum_{k=1}^{j} p_k}{\sum_{\ell=1}^{i} \frac{1}{t_i}} \right\}$$

Exercise: Prove this!

Computing Payments

Part of next homework...

Further Results for Related Scheduling

Theorem (Dhangwatnotai, Dobzinski, Dughmi, Roughgarden '08)

There exists a monotone PTAS for the related scheduling. The associated Myerson payments can be computed in polynomial-time, yielding a dominant-strategy truthful PTAS.

Further Results for Related Scheduling

Theorem (Dhangwatnotai, Dobzinski, Dughmi, Roughgarden '08)

There exists a monotone PTAS for the related scheduling. The associated Myerson payments can be computed in polynomial-time, yielding a dominant-strategy truthful PTAS.

HW: Improve 3 to 2.