Optimisation dans les réseaux :

de l'approximation polynomiale à la théorie des jeux

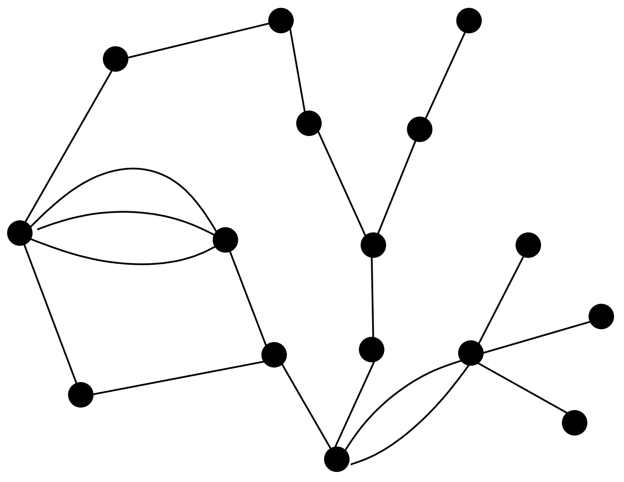
Fanny PASCUAL

Encadrants : Eric ANGEL et Evripidis BAMPIS

IBISC, université d'Évry Val d'Essonne

Networks

A network: “set of entities connected by links”.



Optimization problems:

e.g. routing problem, scheduling problem.

Outline

- Context
  - Classical optimization problems
  - Optimization problems with independent users
- Results
  - Scheduling
    - Performance vs stability
    - Performance vs truthfulness
  - Routing
    - Performance of distributed algorithms
- Future work

Classical combinatorial optimization problems

Given:

- A set of instances (data)
- For each instance: a set of feasible solutions
- An objective function

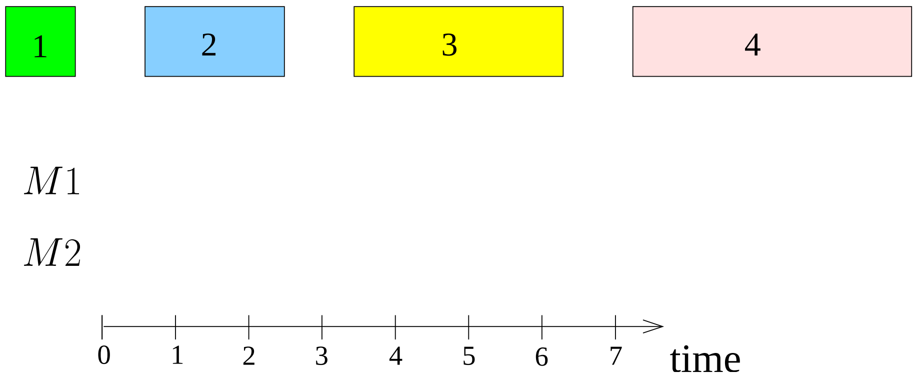
Our goal:

Find the best solution for the objective function.

Example

Given:

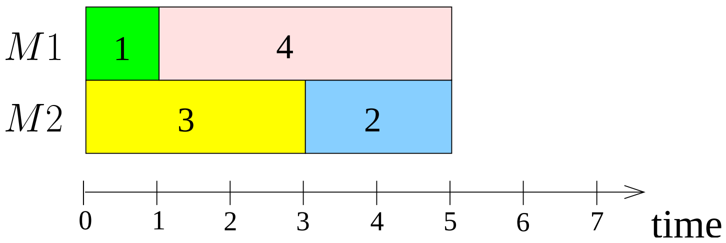
- a set of tasks:
  2 machines:

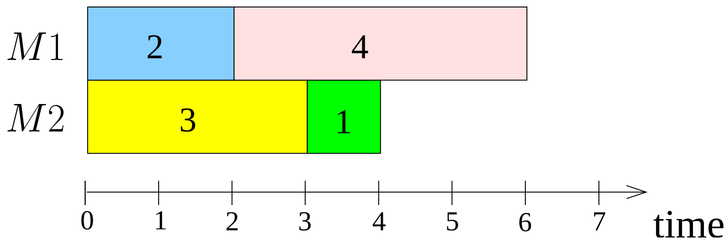


- we have to schedule the tasks on the machines

Goal:

Minimize the completion time of the last task (makespan).





Performance measure

Let $\mathcal{I}$ be the set of possible instances.

Let I be an instance.
$A(I)$ = obj. function's value in the solution returned by A.
$OPT(I)$ = obj. function's value in an optimal solution.

$$\text{Approximation ratio (A)} = \max_{I \in \mathcal{I}} \frac{A(I)}{OPT(I)}$$

Example: for a scheduling problem

$$\text{Approximation ratio (A)} = \max_{\mathcal{I}} \frac{\text{Makespan in the schedule returned by A}}{\text{Makespan in an optimal schedule}}$$

Taking into account independent users

Each independent user has:

- its own objective function
- a set of possible strategies (a degree of freedom)
- private data

Taking into account independent users

Each independent user has:

- its own objective function
- a set of possible strategies (a degree of freedom)
- private data

Nash equilibrium: a situation in which no user can improve its own objective function by unilaterally changing strategy.

Taking into account independent users

Each independent user has:

- its own objective function
- a set of possible strategies (a degree of freedom)
- private data

Nash equilibrium: a situation in which no user can improve its own objective function by unilaterally changing strategy.

Truthfulness: a situation in which no user has incentive to give false informations about its private data.

Taking into account independent users

Each independent user has:

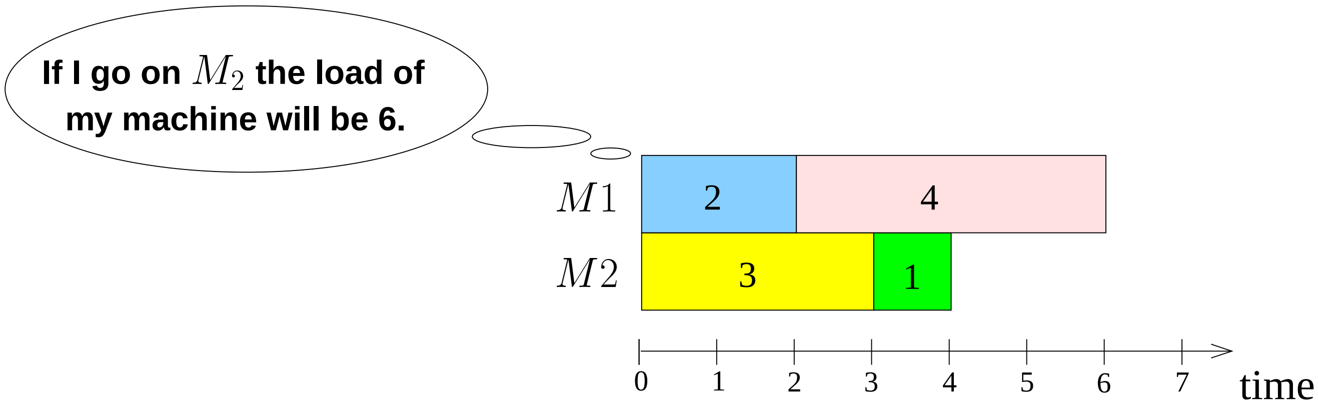
- its own objective function
- a set of possible strategies (a degree of freedom)
- private data

Nash equilibrium: a situation in which no user can improve its own objective function by unilaterally changing strategy.

Example:

- Each task wishes to minimize the load of its machine
- Each task can choose on which machine to be scheduled

Nash equilibrium:



Taking into account independent users

Each independent user has:

- its own objective function
- a set of possible strategies (a degree of freedom)
- private data

Truthfulness: a situation in which no user has incentive to give false informations about its private data.

Example:

- Each task wishes to minimize its completion time
- Private data = length of a task.
  Each task bids a value representing its length

Optimization problems with independent users

Given:
a combinatorial optimization problem

+

a set of independent users.

Our goal: to find an algorithm which optimizes the (global) objective function despite the behavior of the selfish users.

This algorithm:

Optimization problems with independent users

Given:
a combinatorial optimization problem

+

a set of independent users.

Our goal: to find an algorithm which optimizes the (global) objective function despite the behavior of the selfish users.

This algorithm:

returns stable solutions

Optimization problems with independent users

Given:
a combinatorial optimization problem

+

a set of independent users.

Our goal: to find an algorithm which optimizes the (global) objective function despite the behavior of the selfish users.

This algorithm:

returns stable solutions   and/or   is truthful

Distributed or centralized settings

Example:

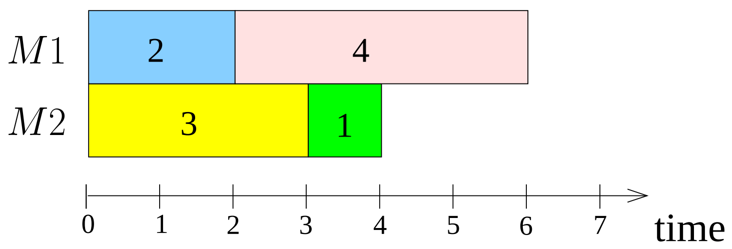
- Each task wishes to minimize the load of its machine
- Each task can choose on which machine to be scheduled

Distributed or centralized settings

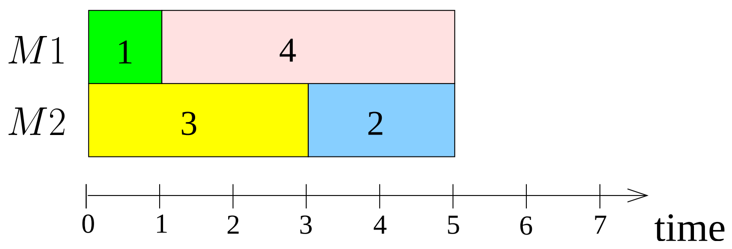
Example:

- Each task wishes to minimize the load of its machine
- Each task can choose on which machine to be scheduled

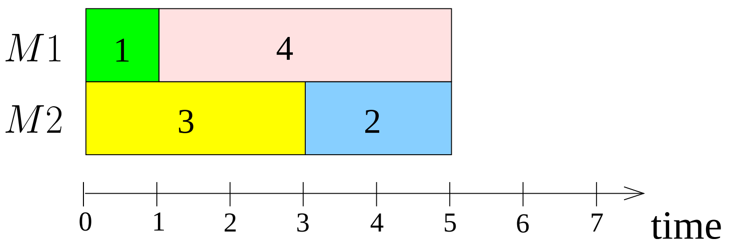
distributed setting



or :



centralized setting



Performance measures

- In a distributed setting:

  Introduced in [Koutsoupias et Papadimitriou, STACS 1999].

$$\text{Price of anarchy} = \max_{I \in \mathcal{I}} \frac{\text{Global obj. function in the } \textbf{worst} \text{ NE}(I)}{OPT(I)}$$

Performance measures

- In a distributed setting:

  Introduced in [Koutsoupias et Papadimitriou, STACS 1999].

$$\text{Price of anarchy} = \max_{I \in \mathcal{I}} \frac{\text{Global obj. function in the } \textbf{worst} \text{ NE}(I)}{OPT(I)}$$

- In a centralized setting:

  Introduced in [Schultz et al., SODA 2003] and [Anshelevich et al., FOCS 2004].

  Approximation ratio w.r.t stable solutions:

$$\text{Price of stability} = \max_{I \in \mathcal{I}} \frac{\text{Global obj. function in the } \textbf{best} \text{ NE}(I)}{OPT(I)}$$

Outline

- Context
  - Classical optimization problems
  - Optimization problems with independent users
- Results
  - Scheduling
    - Performance vs stability
    - Performance vs truthfulness
  - Routing
    - Performance of distributed algorithms
- Future work

Performance vs Stability: introduction

We wish to schedule selfish tasks on m machines.

- Each task is free to choose the machine on which it will be executed. It wishes to minimize its own completion time.

Performance vs Stability: introduction

We wish to schedule selfish tasks on m machines.

- Each task is free to choose the machine on which it will be executed. It wishes to minimize its own completion time.
- Each machine has a local policy to schedule its tasks.

Performance vs Stability: introduction

We wish to schedule selfish tasks on m machines.

- Each task is free to choose the machine on which it will be executed. It wishes to minimize its own completion time.
- Each machine has a local policy to schedule its tasks.

  e.g. the LPT policy ("for Longest Processing Time first"): each machine schedules its tasks from the largest one to the smallest one.

Performance vs Stability: introduction

We wish to schedule selfish tasks on m machines.

- Each task is free to choose the machine on which it will be executed. It wishes to minimize its own completion time.
- Each machine has a local policy to schedule its tasks.

  e.g. the LPT policy (“for Longest Processing Time first”): each machine schedules its tasks from the largest one to the smallest one.

Conjecture CKN: [Christodoulou et al., ICALP 2004]
There is no distributed algorithm which has a price of anarchy smaller than $\frac{4}{3} - \frac{1}{3\,m}$.

If this conjecture is true, in order to get a better approximation ratio, a centralized algorithm is necessary.

Performance vs Stability: introduction

We have:

- A policy per machine.
- A protocol which suggests an assignment of the tasks on the machines.

The tasks accept or refuse this assignment.

Performance vs Stability: introduction

We have:

- A policy per machine.
- A protocol which suggests an assignment of the tasks on the machines.

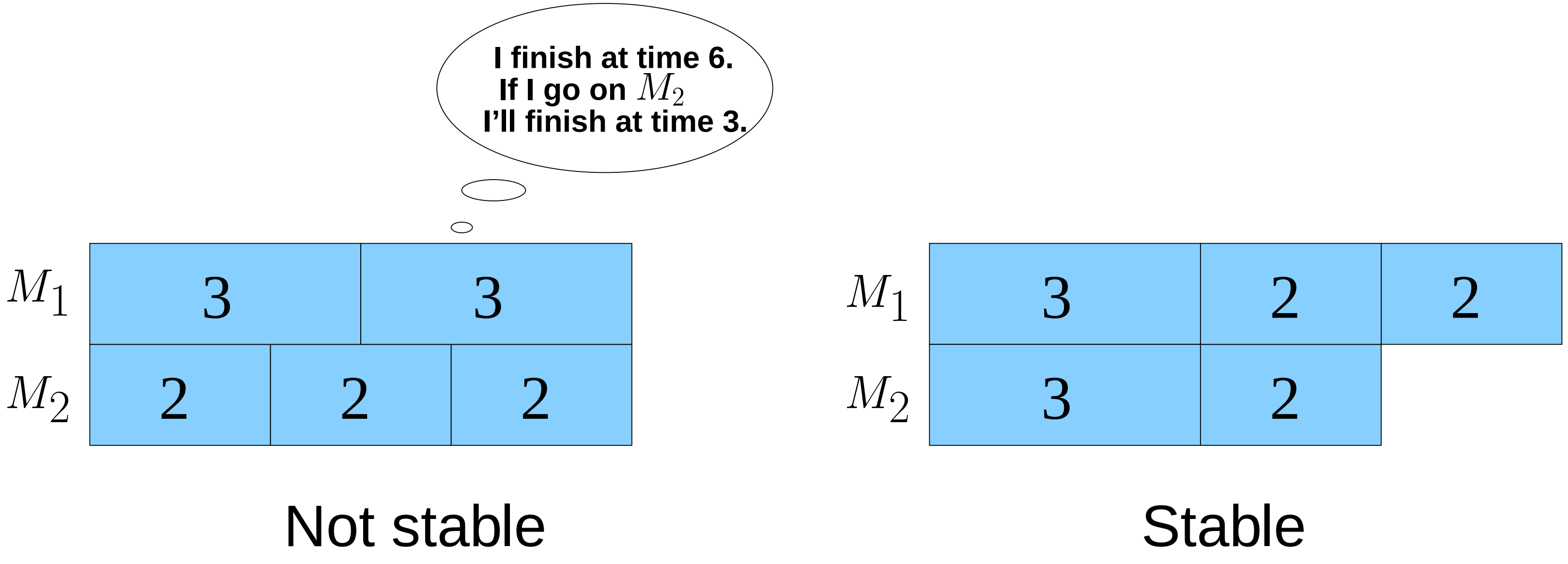
The tasks accept or refuse this assignment.

Goal: A protocol which returns a solution:

- which minimizes the makespan
- and which is stable.

Performance vs Stability: introduction

Example: The policy of each machine is LPT: each machine schedules its tasks from the largest one to the smallest one.



Price of stability

Recall:

$$\text{Price of stability} = \max_{\mathcal{I}} \frac{\text{Makespan in the best NE}}{\text{Makespan in an optimal solution}}.$$

Example: If the policy of each machine is LPT, then the price of stability is $\frac{4}{3} - \frac{1}{3\,m}$.

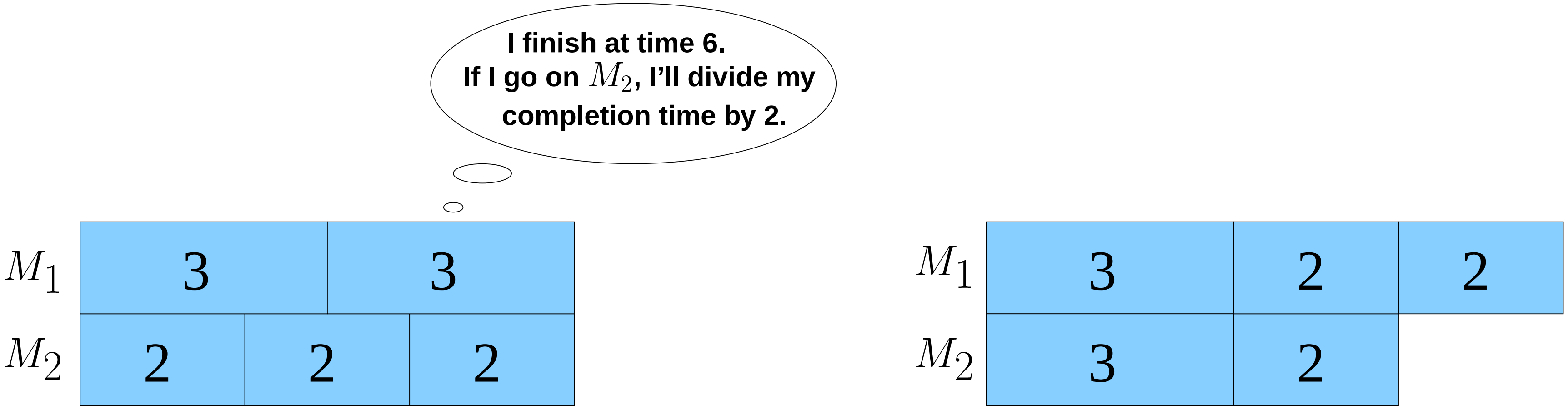
α-approximate Nash equilibrium

α-approximate Nash equilibrium = a situation in which no task can decrease its completion time by a factor larger than α by changing machine.

α-approximate Nash equilibrium

α-approximate Nash equilibrium = a situation in which no task can decrease its completion time by a factor larger than α by changing machine.

Example: Policy of each machine = LPT.



2-approximate Nash Eq. Nash Eq.

Price of α-approximate stability

$$\text{Price of } \alpha\text{-approximate stability} = \max_{\mathcal{I}} \frac{\text{Makespan in the best } \alpha\text{-approx. NE}}{\text{Makespan in an optimal solution}}$$

[Chen and Roughgarden, SPAA 2006]: study the tradeoff between stability (α-Nash equilibrium) and approximation ratio in a network problem.

Our goal

Goal: study the tradeoff between stability and approximation ratio.

Policy of the machines = LPT.

What is the price of α-approximate stability ?

Given (r, α), is there a r-approximate algorithm which returns α-approximate NE ?

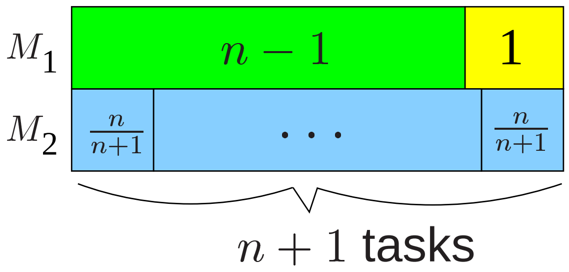
Lower bounds (policy = LPT)

Theorem: Let $n > 5$. There is no algorithm with an approx. ratio $< (1 + \frac{1}{n(n+1)})$ which returns α-approximate NE with $\alpha < n$.

Lower bounds (policy = LPT)

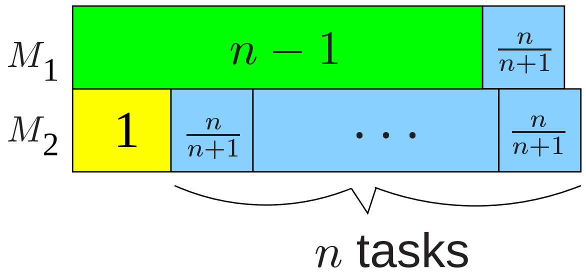
Theorem: Let $n > 5$. There is no algorithm with an approx. ratio $< (1 + \frac{1}{n(n+1)})$ which returns α-approximate NE with $\alpha < n$.

Sketch of the proof:



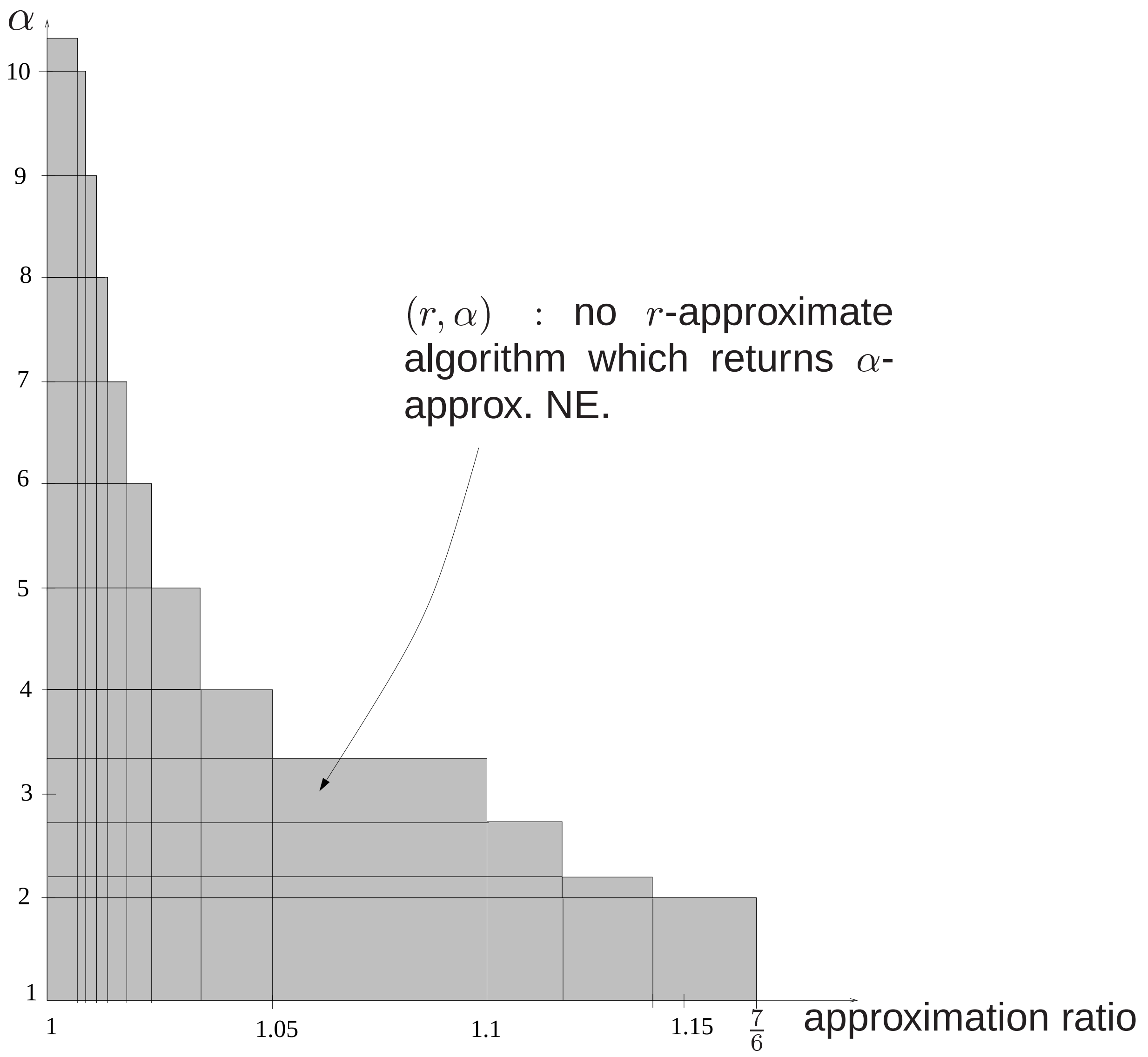
approx. ratio < $1 + \frac{1}{n(n+1)}$

n-approximate NE



approx. ratio = $1 + \frac{1}{n(n+1)}$

Lower bounds (policy = LPT)



Upper bounds (policy = LPT)

Theorem: There is a $\frac{8}{7}$-approximate algorithm which returns 3-approximate NE.

$\rightarrow$ Algorithm LPT_{swap}

Upper bounds (policy = LPT)

Theorem: There is a $\frac{8}{7}$-approximate algorithm which returns 3-approximate NE.

$\rightarrow$ Algorithm LPT_{swap}

Algorithm LPT: schedule greedily the tasks from the largest one to the smallest one.

Example: Tasks of lengths 8, 5, 4, 3, 3, 2

M_1

M_2

M_3

Upper bounds (policy = LPT)

Theorem: There is a $\frac{8}{7}$-approximate algorithm which returns 3-approximate NE.

$\rightarrow$ Algorithm LPT_{swap}

Algorithm LPT: schedule greedily the tasks from the largest one to the smallest one.

Example: Tasks of lengths 8, 5, 4, 3, 3, 2

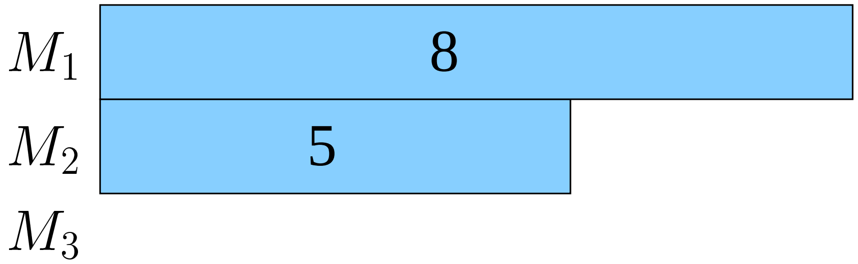
Upper bounds (policy = LPT)

Theorem: There is a $\frac{8}{7}$-approximate algorithm which returns 3-approximate NE.

$\rightarrow$ Algorithm LPT_{swap}

Algorithm LPT: schedule greedily the tasks from the largest one to the smallest one.

Example: Tasks of lengths 8, 5, 4, 3, 3, 2



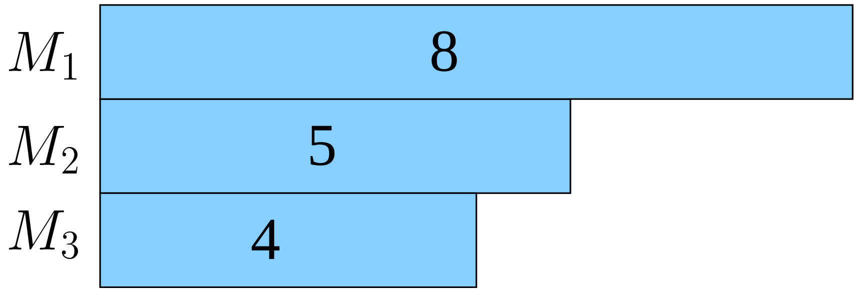
Upper bounds (policy = LPT)

Theorem: There is a $\frac{8}{7}$-approximate algorithm which returns 3-approximate NE.

$\rightarrow$ Algorithm LPT_{swap}

Algorithm LPT: schedule greedily the tasks from the largest one to the smallest one.

Example: Tasks of lengths 8, 5, 4, 3, 3, 2



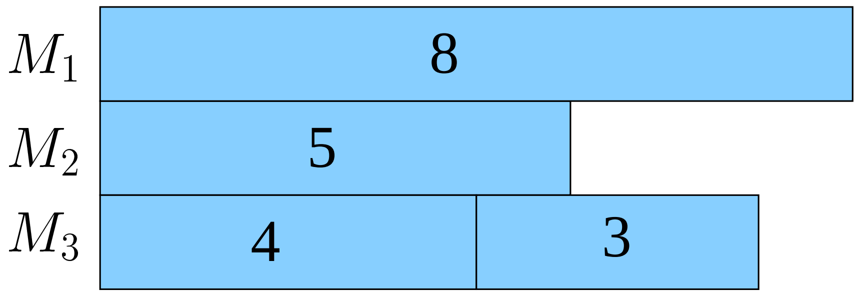
Upper bounds (policy = LPT)

Theorem: There is a $\frac{8}{7}$-approximate algorithm which returns 3-approximate NE.

$\rightarrow$ Algorithm LPT_{swap}

Algorithm LPT: schedule greedily the tasks from the largest one to the smallest one.

Example: Tasks of lengths 8, 5, 4, 3, 3, 2



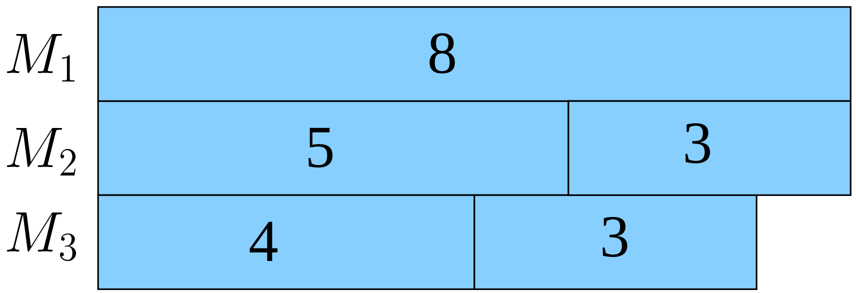
Upper bounds (policy = LPT)

Theorem: There is a $\frac{8}{7}$-approximate algorithm which returns 3-approximate NE.

$\rightarrow$ Algorithm LPT_{swap}

Algorithm LPT: schedule greedily the tasks from the largest one to the smallest one.

Example: Tasks of lengths 8, 5, 4, 3, 3, 2



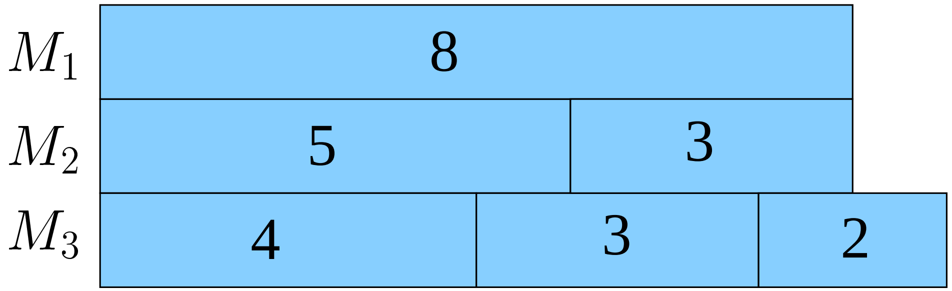
Upper bounds (policy = LPT)

Theorem: There is a $\frac{8}{7}$-approximate algorithm which returns 3-approximate NE.

$\rightarrow$ Algorithm LPT_{swap}

Algorithm LPT: schedule greedily the tasks from the largest one to the smallest one.

Example: Tasks of lengths 8, 5, 4, 3, 3, 2

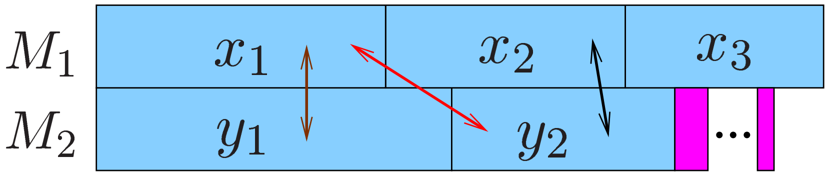


Upper bound: LPT_{swap}

- Build an LPT schedule

Upper bound: LPT_{swap}

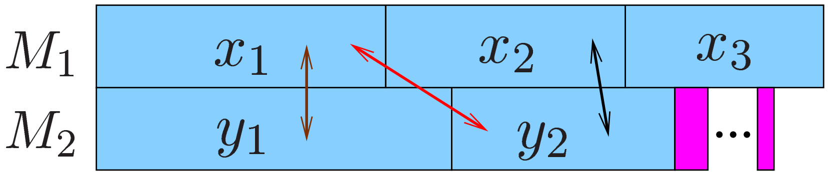
- Build an LPT schedule
- Look at this schedule:
  - 1st case:



Return the best schedule among the 4 possible ones.

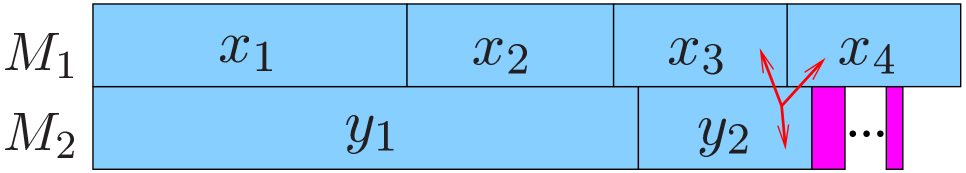
Upper bound: LPT_{swap}

- Build an LPT schedule
- Look at this schedule:
  - 1st case:



Return the best schedule among the 4 possible ones.

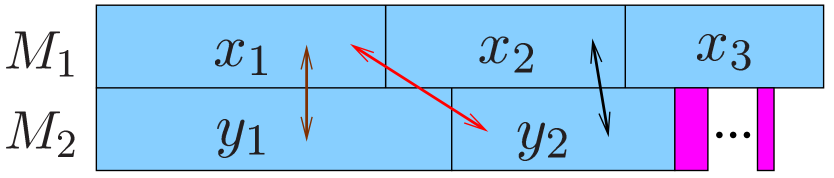
  - 2nd case:



Return the best schedule among the 2 possible ones.

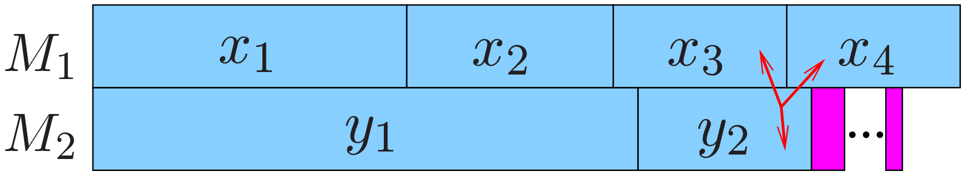
Upper bound: LPT_{swap}

- Build an LPT schedule
- Look at this schedule:
  - 1st case:



Return the best schedule among the 4 possible ones.

  - 2nd case:

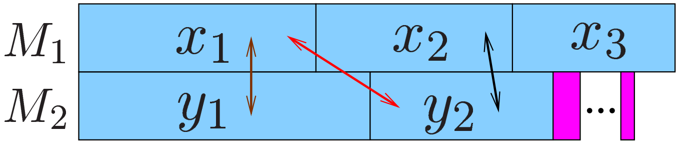


Return the best schedule among the 2 possible ones.

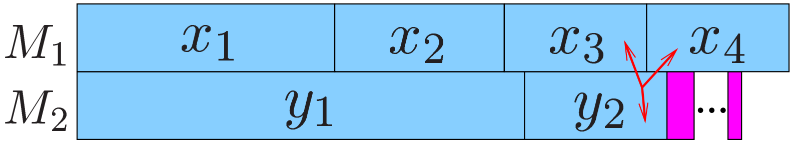
  - Other cases:
    Return the LPT schedule.

Upper bound: LPT_{swap}

- Build an LPT schedule
- Look at this schedule:
  - 1st case:



  - 2nd case:

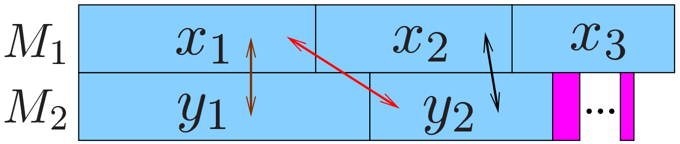


  - Other cases:
    Return the LPT schedule.

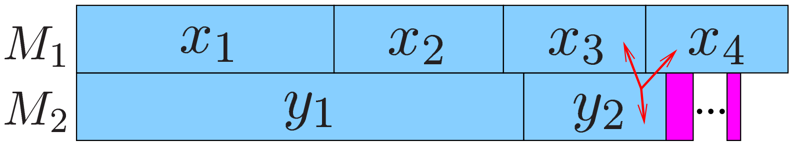
Sketch of the proof:

Upper bound: LPT_{swap}

- Build an LPT schedule
- Look at this schedule:
  - 1st case:



  - 2nd case:



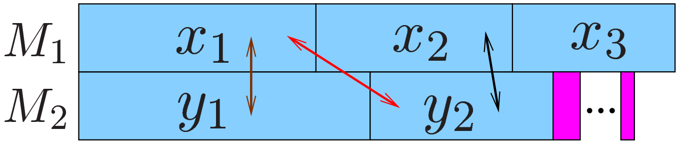
  - Other cases:
    Return the LPT schedule.

Sketch of the proof:

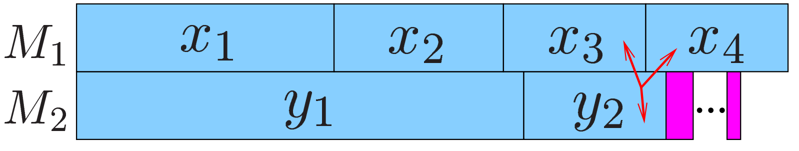
- $\frac{7}{6}$-approximate.

Upper bound: LPT_{swap}

- Build an LPT schedule
- Look at this schedule:
  - 1st case:



  - 2nd case:



  - Other cases:
    Return the LPT schedule.

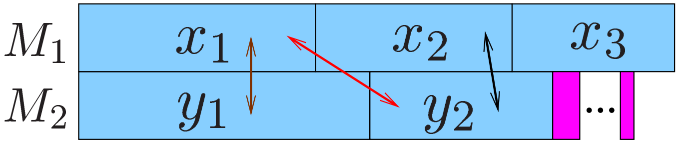
Sketch of the proof:

- $\frac{7}{6}$-approximate.

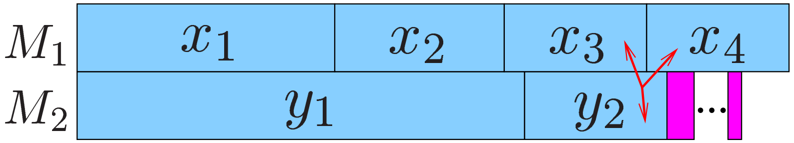
- LPT is $\frac{8}{7}$-approximate.

Upper bound: LPT_{swap}

- Build an LPT schedule
- Look at this schedule:
  - 1st case:



  - 2nd case:



  - Other cases:
    Return the LPT schedule.

Sketch of the proof:

- $\frac{7}{6}$-approximate.
- In both cases:
  - a swap returns an optimal solution of the large tasks.
  - $\sum$ (small tasks) $< \frac{1}{7} OPT$.

- LPT is $\frac{8}{7}$-approximate.

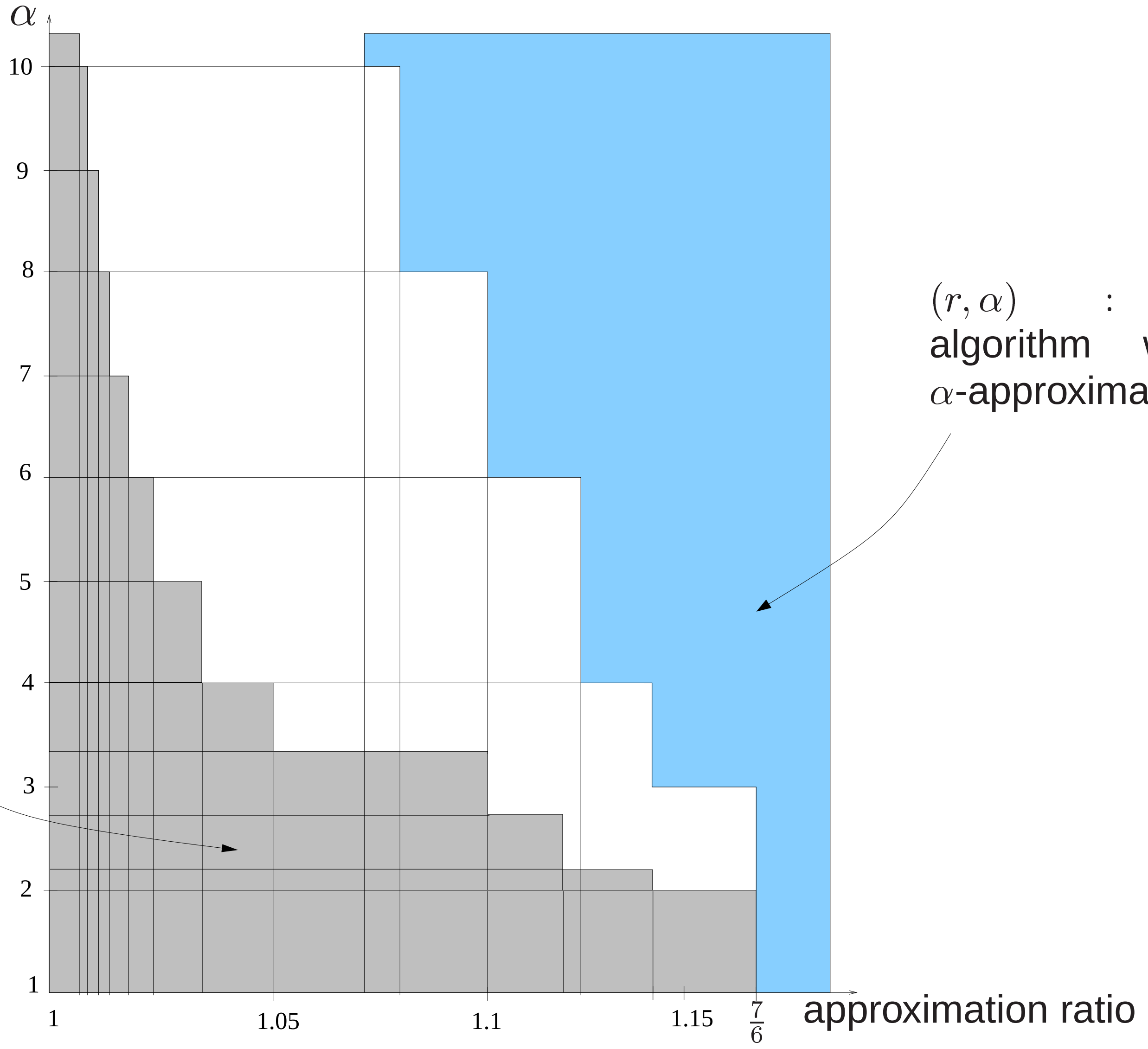
Upper bounds (policy = LPT)

Theorem: There is a $(1 + \frac{1}{\alpha})$-approximate algorithm which returns α-approximate NE.

$\rightarrow$ Approximation scheme [Graham, 1966]

Results: (policy = LPT)

(r, α) : no r-approximate algorithm which returns α-approximate NE.



(r, α) : r-approximate algorithm which returns α-approximate NE.

Other results

- The SPT policy (for “Shortest Processing Time first”) is not as good as the LPT policy.
- If randomized policies are allowed: each task wishes to reduce its expected completion time.

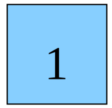
  The policy which schedules the tasks randomly is optimal.

Outline

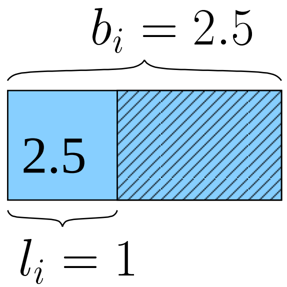
- Context
  - Classical optimization problems
  - Optimization problems with independent users
- Results
  - Scheduling
    - Performance vs stability
    - Performance vs truthfulness
  - Routing
    - Performance of distributed algorithms
- Future work

Performances of a truthful algorithm: introduction

- Task i has a secret real length (execution time) l_i.



- A task can add “fake” data to artificially increase its length: each task bids a value $b_i \geq l_i$.



- Each task knows the values bidded by the other tasks and the algorithm.

Each task wishes to reduce its completion time (and may lie if necessary).

Performances of a truthful algorithm: introduction

We have tasks to schedule on m machines.

Our goal: to minimize the makespan.

If the tasks lie, it is often not possible to have a guarantee of the approximation ratio of the makespan.

Performances of a truthful algorithm: introduction

We have tasks to schedule on m machines.

Our goal: to minimize the makespan.

If the tasks lie, it is often not possible to have a guarantee of the approximation ratio of the makespan.

A truthful algorithm: an algorithm in which no task has incentive to bid a false value.

Performances of a truthful algorithm: introduction

We have tasks to schedule on m machines.

Our goal: to minimize the makespan.

If the tasks lie, it is often not possible to have a guarantee of the approximation ratio of the makespan.

A truthful algorithm: an algorithm in which no task has incentive to bid a false value.

Aim: an algorithm (centralized or distributed) which is truthful and which minimizes the makespan.

Related work

- Distributed algorithms:
  - Not truthful: [Christodoulou et al., ICALP 2004], [Immorlica et al., WINE 2005]
- Truthful centralized algorithms:
  - Users are the tasks: they wish to minimize the load of their machine. [Auletta et al., SPAA 2004]
  - Users are the machines which bid their speeds. [Nisan, Ronen, STOC 1999], [Archer, Tardos, FOCS 2001], [Auletta et al., STACS 2004], etc.

A truthful algorithm

Algorithm SPT: schedule greedily the tasks from the smallest one to the largest one.

Example: Tasks of lengths 1, 1, 2, 3, 4, 5, 8

M_1

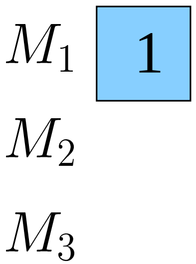
M_2

M_3

A truthful algorithm

Algorithm SPT: schedule greedily the tasks from the smallest one to the largest one.

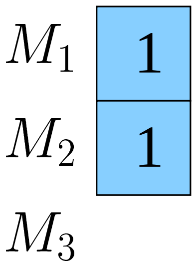
Example: Tasks of lengths 1, 1, 2, 3, 4, 5, 8



A truthful algorithm

Algorithm SPT: schedule greedily the tasks from the smallest one to the largest one.

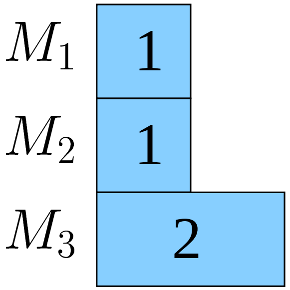
Example: Tasks of lengths 1, 1, 2, 3, 4, 5, 8



A truthful algorithm

Algorithm SPT: schedule greedily the tasks from the smallest one to the largest one.

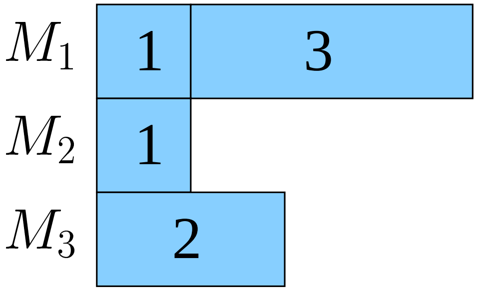
Example: Tasks of lengths 1, 1, 2, 3, 4, 5, 8



A truthful algorithm

Algorithm SPT: schedule greedily the tasks from the smallest one to the largest one.

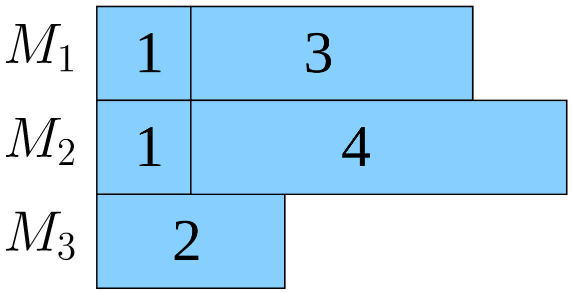
Example: Tasks of lengths 1, 1, 2, 3, 4, 5, 8



A truthful algorithm

Algorithm SPT: schedule greedily the tasks from the smallest one to the largest one.

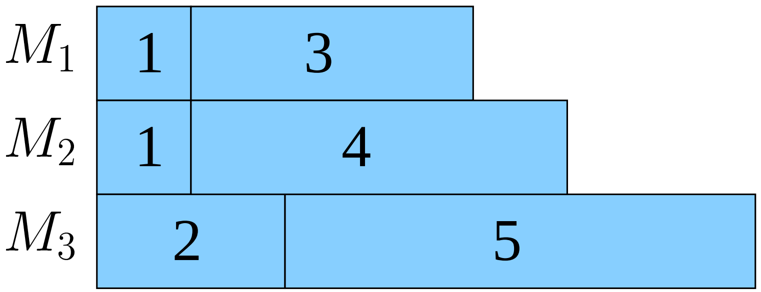
Example: Tasks of lengths 1, 1, 2, 3, 4, 5, 8



A truthful algorithm

Algorithm SPT: schedule greedily the tasks from the smallest one to the largest one.

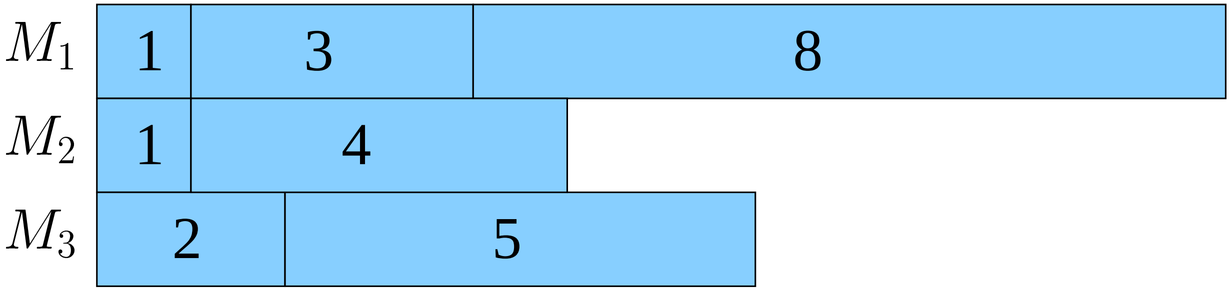
Example: Tasks of lengths 1, 1, 2, 3, 4, 5, 8



A truthful algorithm

Algorithm SPT: schedule greedily the tasks from the smallest one to the largest one.

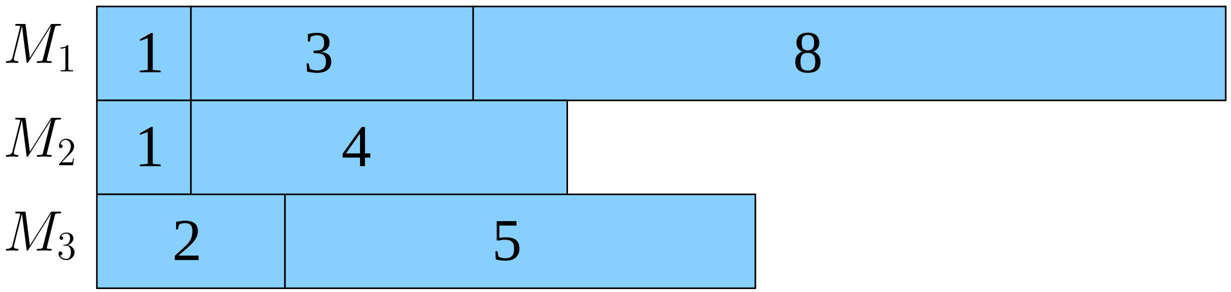
Example: Tasks of lengths 1, 1, 2, 3, 4, 5, 8



A truthful algorithm

Algorithm SPT: schedule greedily the tasks from the smallest one to the largest one.

Example: Tasks of lengths 1, 1, 2, 3, 4, 5, 8



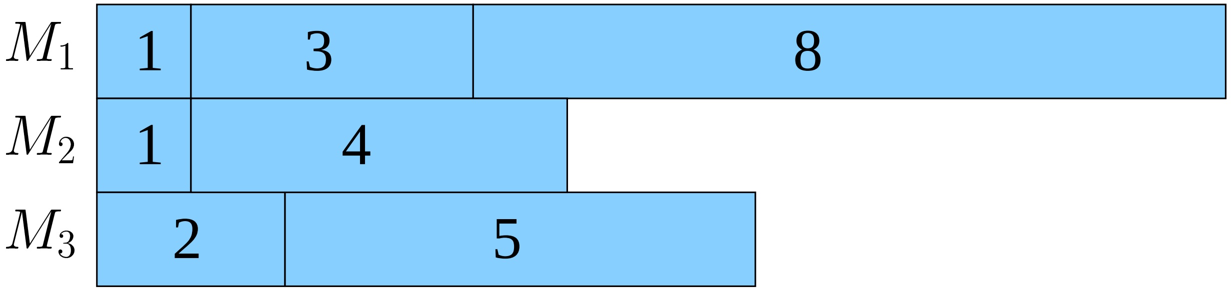
This algorithm is truthful.

Approx. ratio: $2 - \frac{1}{m}$. [Graham 1966]

A truthful algorithm

Algorithm SPT: schedule greedily the tasks from the smallest one to the largest one.

Example: Tasks of lengths 1, 1, 2, 3, 4, 5, 8



This algorithm is truthful.

Approx. ratio: $2 - \frac{1}{m}$. [Graham 1966]

Is there a better truthful algorithm ?

Performances of a truthful algorithm

Theorem: There is no truthful deterministic algorithm with an approx. ratio smaller than $2 - \frac{1}{m}$.

Is there a better truthful (randomized) algorithm ?

Performances of a truthful algorithm

Theorem: There is no truthful deterministic algorithm with an approx. ratio smaller than $2 - \frac{1}{m}$.

Is there a better truthful (randomized) algorithm ?

Theorem: There is no truthful randomized algorithm with an approx. ratio smaller than $\frac{3}{2} - \frac{1}{2m}$.

Performances of a truthful algorithm

Idea: to combine:

- A truthful algorithm
- an algorithm not truthful but with a good approximation ratio

Algorithm LPT: schedules greedily the tasks from the smallest one to the largest one.
Approx. ratio = $\frac{4}{3} - \frac{1}{3\,m}$. [Graham, 1966]

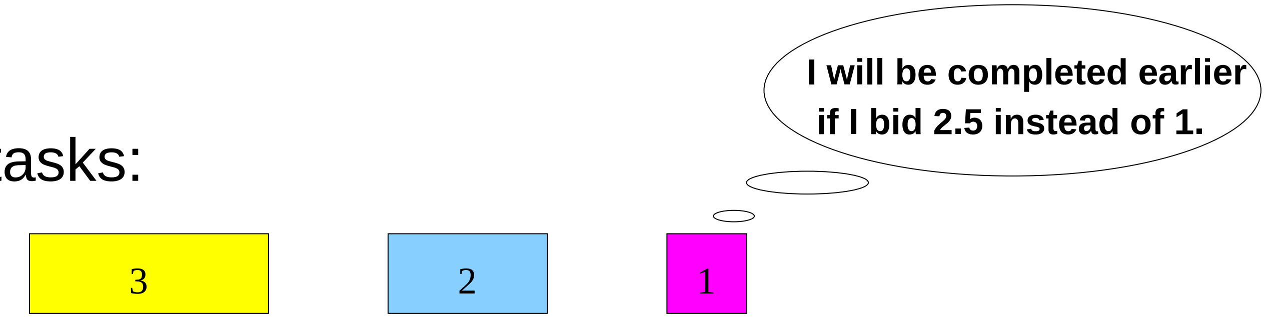
Algorithm $SPT \oplus LPT$:

- with a proba. p: SPT
- with a proba. $(1-p)$: LPT.

Performances of a truthful algorithm

Algorithm $SPT \oplus LPT$ is not truthful:

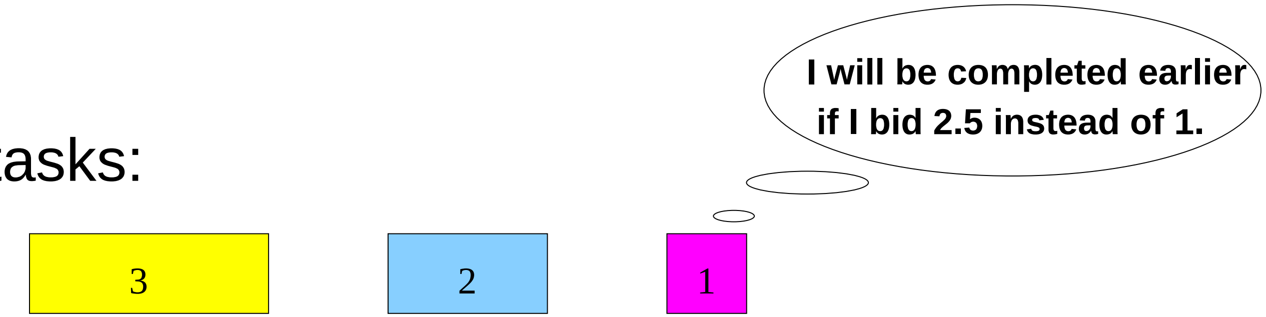
We have 3 tasks:



Performances of a truthful algorithm

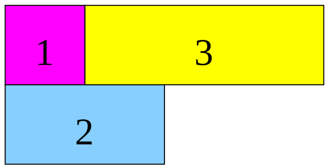
Algorithm $SPT \oplus LPT$ is not truthful:

We have 3 tasks:

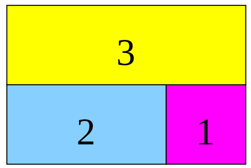


if task 1 bids its true value: 1

SPT :



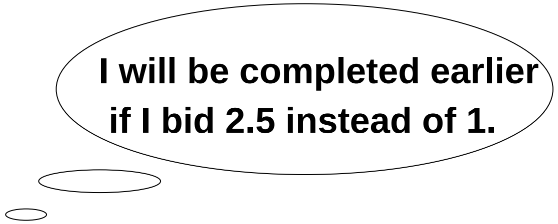
LPT :



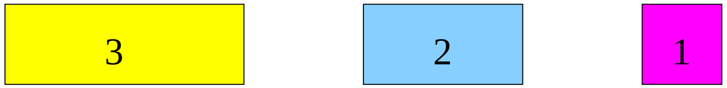
$$C_1 = p + 3(1-p) = 3 - 2p$$

Performances of a truthful algorithm

Algorithm $SPT \oplus LPT$ is not truthful:

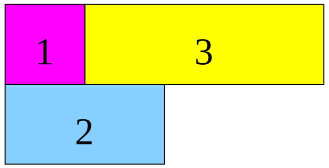


We have 3 tasks:

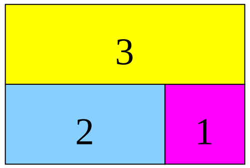


if task 1 bids its true value: 1

SPT :



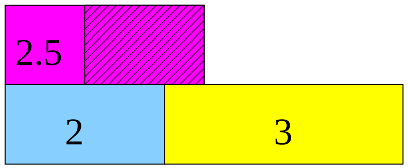
LPT :



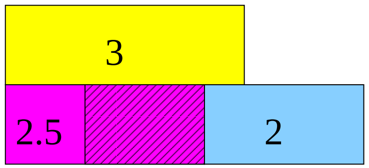
$$C_1 = p + 3(1-p) = 3 - 2p$$

if task 1 bids a false value: 2. 5

SPT :



LPT :



$$C_1 = 1$$

Algorithm DSPT

Algorithm DSPT (“delayed SPT”):

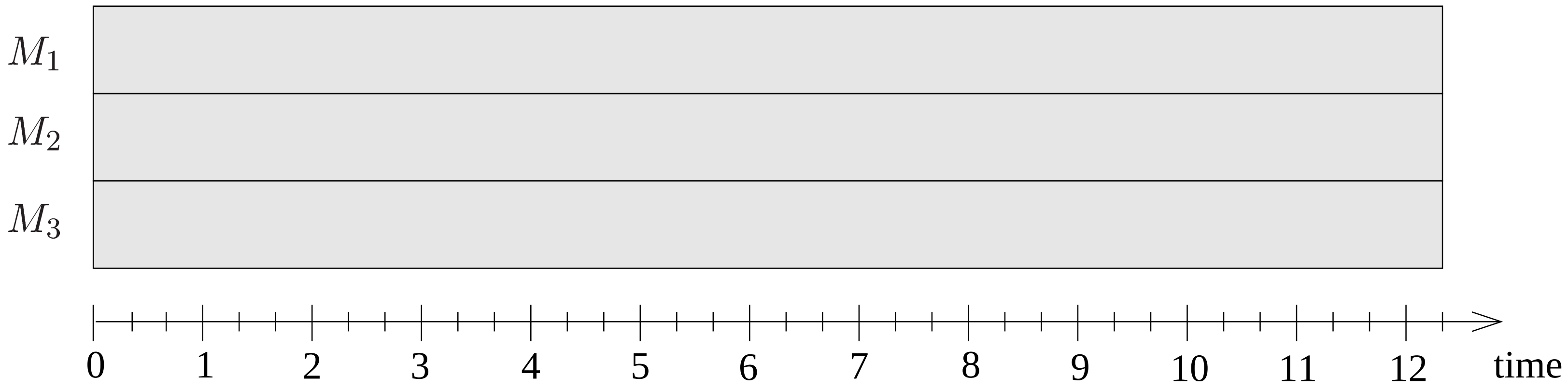
Schedules tasks $1, 2, \ldots, n$ such that $l_1 \leq l_2 \leq \ldots l_n$.
Task $(i+1)$ starts when $\frac{1}{m}$ of task i has been executed.

Algorithm DSPT

Algorithm DSPT (“delayed SPT”):

Schedules tasks $1, 2, \ldots, n$ such that $l_1 \leq l_2 \leq \ldots l_n$.
Task $(i+1)$ starts when $\frac{1}{m}$ of task i has been executed.

Example: $m = 3$, tasks of lengths 1, 1, 1, 1, 2, 3, 5, 5, 6.



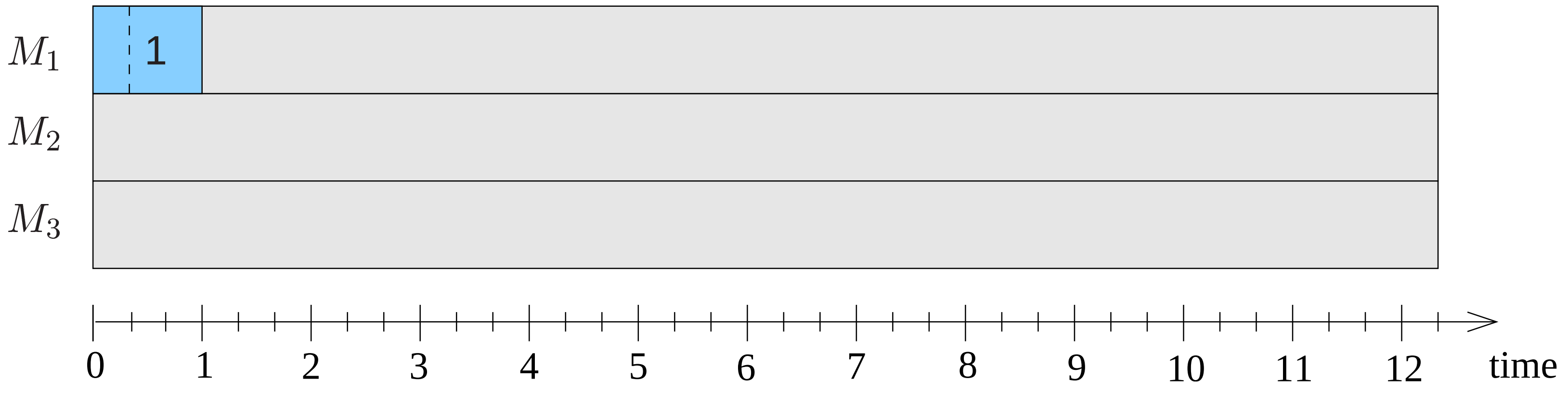
Algorithm DSPT

Algorithm DSPT ("delayed SPT"):

Schedules tasks $1, 2, \ldots, n$ such that $l_1 \leq l_2 \leq \ldots l_n$.

Task $(i+1)$ starts when $\frac{1}{m}$ of task i has been executed.

Example: $m = 3$, tasks of lengths 1, 1, 1, 1, 2, 3, 5, 5, 6.

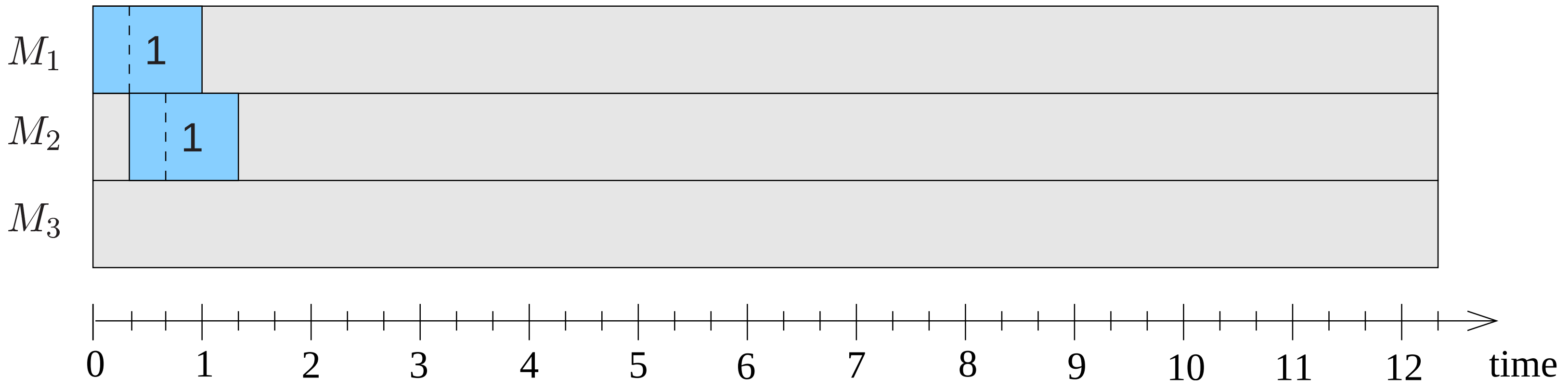


Algorithm DSPT

Algorithm DSPT ("delayed SPT"):

Schedules tasks $1, 2, \ldots, n$ such that $l_1 \leq l_2 \leq \ldots l_n$.
Task $(i+1)$ starts when $\frac{1}{m}$ of task i has been executed.

Example: $m = 3$, tasks of lengths 1, 1, 1, 1, 2, 3, 5, 5, 6.

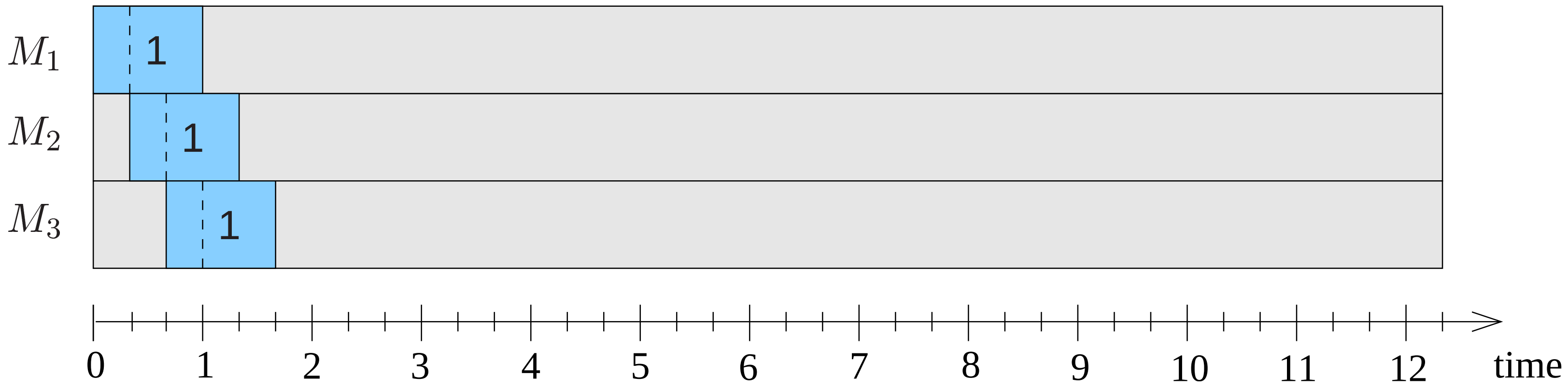


Algorithm DSPT

Algorithm DSPT ("delayed SPT"):

Schedules tasks $1, 2, \ldots, n$ such that $l_1 \leq l_2 \leq \ldots l_n$.
Task $(i + 1)$ starts when $\frac{1}{m}$ of task i has been executed.

Example: $m = 3$, tasks of lengths 1, 1, 1, 1, 2, 3, 5, 5, 6.

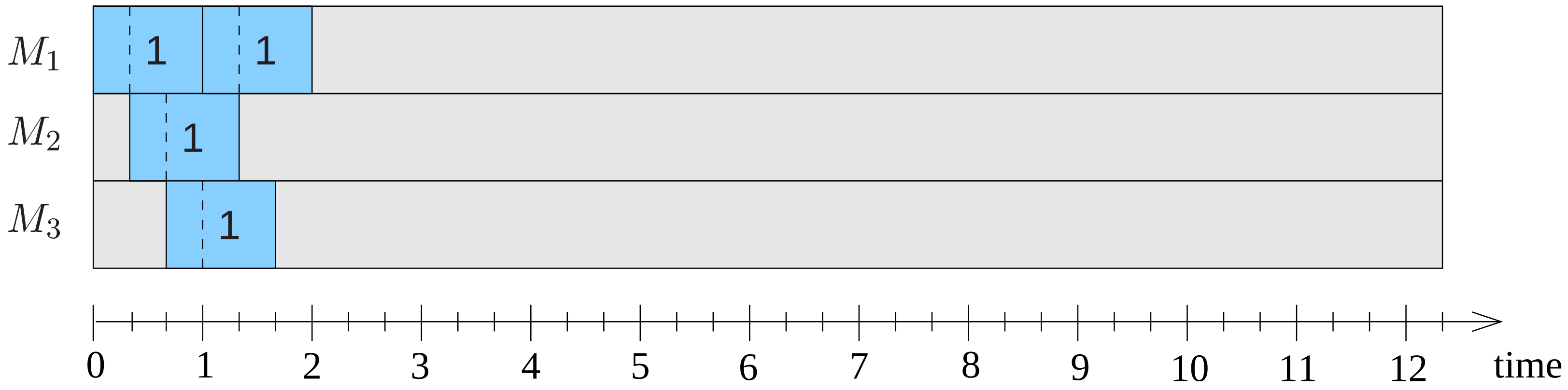


Algorithm DSPT

Algorithm DSPT (“delayed SPT”):

Schedules tasks $1, 2, \ldots, n$ such that $l_1 \leq l_2 \leq \ldots l_n$.
Task $(i + 1)$ starts when $\frac{1}{m}$ of task i has been executed.

Example: $m = 3$, tasks of lengths 1, 1, 1, 1, 2, 3, 5, 5, 6.

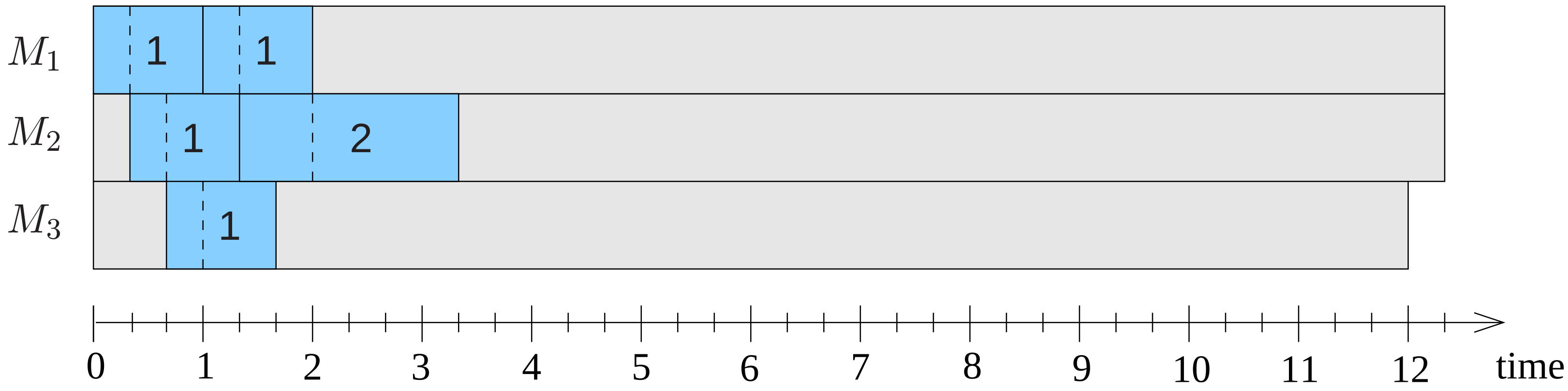


Algorithm DSPT

Algorithm DSPT ("delayed SPT"):

Schedules tasks $1, 2, \ldots, n$ such that $l_1 \leq l_2 \leq \ldots l_n$.
Task $(i+1)$ starts when $\frac{1}{m}$ of task i has been executed.

Example: $m = 3$, tasks of lengths 1, 1, 1, 1, 2, 3, 5, 5, 6.

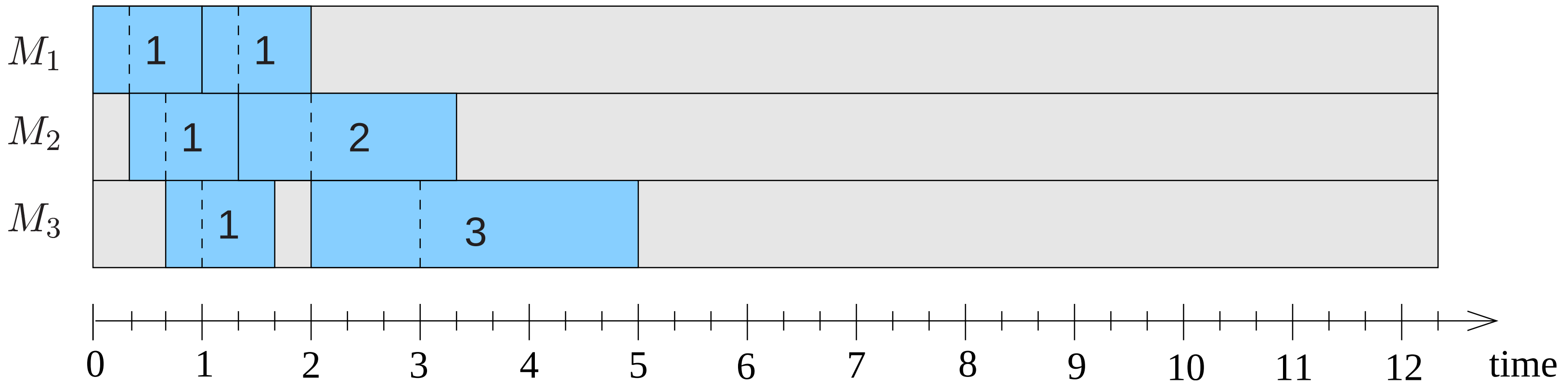


Algorithm DSPT

Algorithm DSPT ("delayed SPT"):

Schedules tasks $1, 2, \ldots, n$ such that $l_1 \leq l_2 \leq \ldots l_n$.
Task $(i+1)$ starts when $\frac{1}{m}$ of task i has been executed.

Example: $m = 3$, tasks of lengths 1, 1, 1, 1, 2, 3, 5, 5, 6.

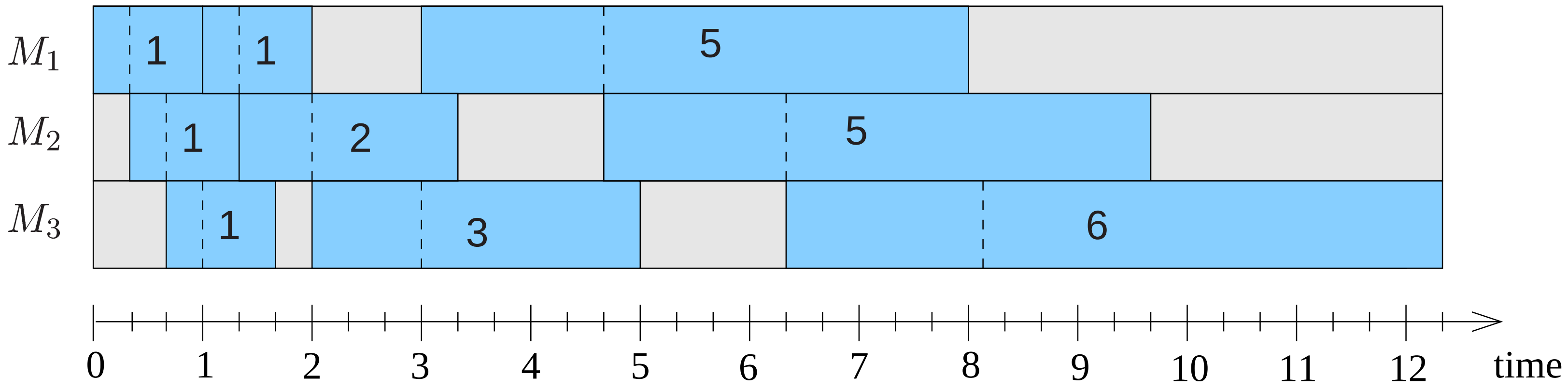


Algorithm DSPT

Algorithm DSPT (“delayed SPT”):

Schedules tasks $1, 2, \ldots, n$ such that $l_1 \leq l_2 \leq \ldots l_n$.
Task $(i+1)$ starts when $\frac{1}{m}$ of task i has been executed.

Example: $m = 3$, tasks of lengths 1, 1, 1, 1, 2, 3, 5, 5, 6.

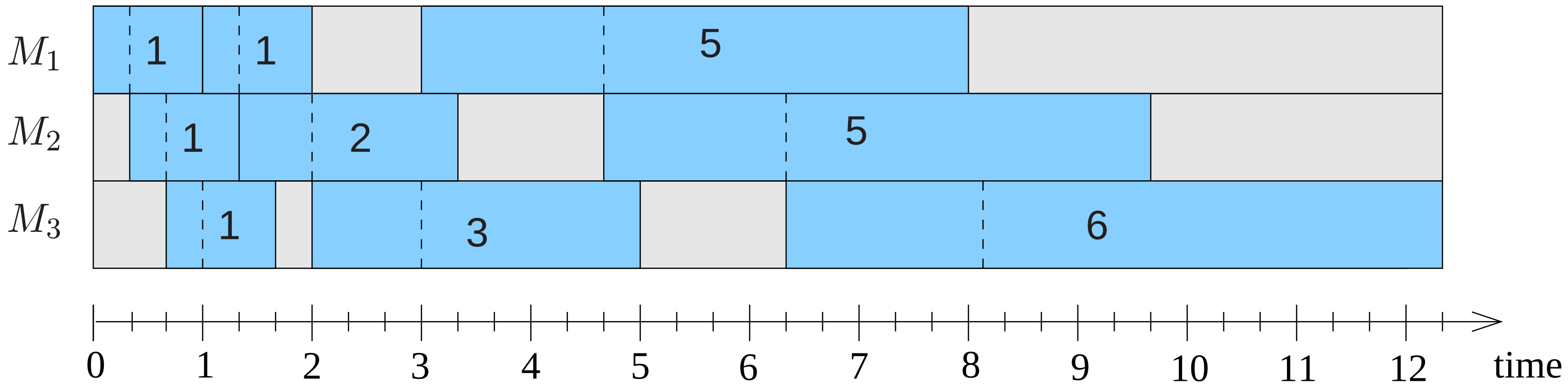


Algorithm DSPT

Algorithm DSPT (“delayed SPT”):

Schedules tasks $1, 2, \ldots, n$ such that $l_1 \leq l_2 \leq \ldots l_n$.
Task $(i+1)$ starts when $\frac{1}{m}$ of task i has been executed.

Example: $m = 3$, tasks of lengths 1, 1, 1, 1, 2, 3, 5, 5, 6.



Theorem: DSPT is $(2 - \frac{1}{m})$-approximate.

A truthful algorithm

Algorithm $DSPT \oplus LPT$:

- With a proba. $\frac{m}{m+1}$: DSPT
- With a proba. $\frac{1}{m+1}$: LPT

A truthful algorithm

Algorithm $DSPT \oplus LPT$:

- With a proba. $\frac{m}{m+1}$: DSPT
- With a proba. $\frac{1}{m+1}$: LPT

Theorem: Expected approximation ratio of $DSPT \oplus LPT$

$= \frac{m}{m+1} \left(2 - \frac{1}{m}\right) + \frac{1}{m+1} \left(\frac{4}{3} - \frac{1}{3\,m}\right)$

e.g. for $m = 2$: ratio($DSPT \oplus LPT$)<1.39, ratio(SPT)=1.5

Recall: there is no truthful $(1.25 - \varepsilon)$-approximate algorithm.

A truthful algorithm

Algorithm $DSPT \oplus LPT$:

- With a proba. $\frac{m}{m+1}$: DSPT
- With a proba. $\frac{1}{m+1}$: LPT

Theorem: Expected approximation ratio of $DSPT \oplus LPT$

$= \frac{m}{m+1} \left(2 - \frac{1}{m}\right) + \frac{1}{m+1} \left(\frac{4}{3} - \frac{1}{3\,m}\right)$

e.g. for $m = 2$: ratio($DSPT \oplus LPT$)<1.39, ratio(SPT)=1.5

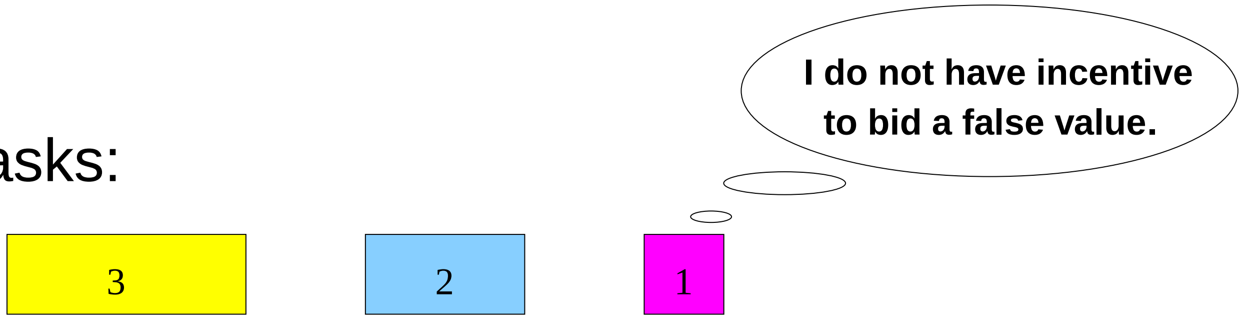
Recall: there is no truthful $(1.25 - \varepsilon)$-approximate algorithm.

Theorem: $DSPT \oplus LPT$ is truthful.

A truthful algorithm

Example:

We have 3 tasks:

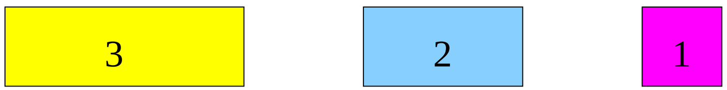


A truthful algorithm

Example:

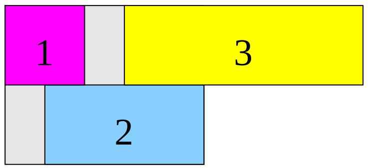
We have 3 tasks:

I do not have incentive to bid a false value.

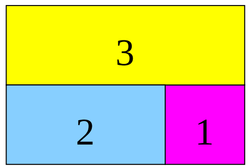


if task 1 bids its true value: 1

DSPT :



LPT :

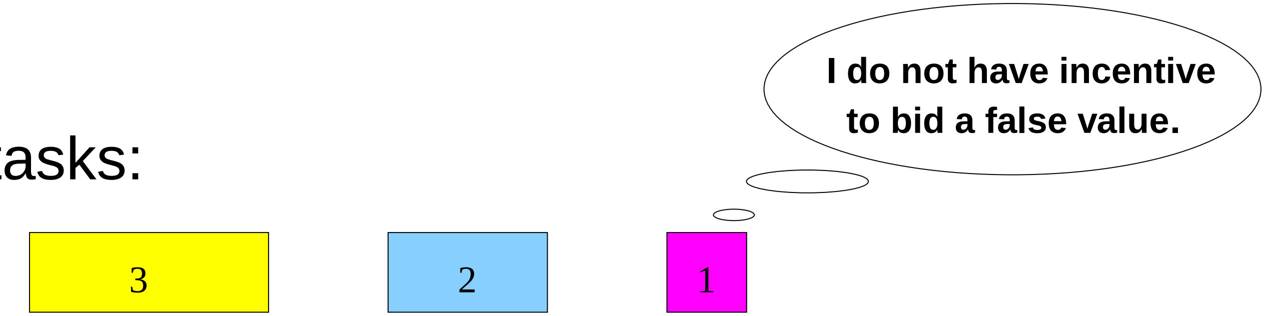


$C_1 = \frac{5}{3}$

A truthful algorithm

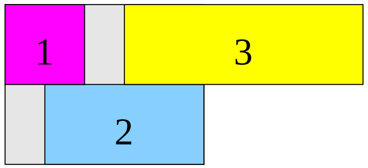
Example:

We have 3 tasks:

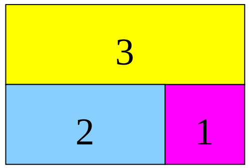


if task 1 bids its true value: 1

DSPT :



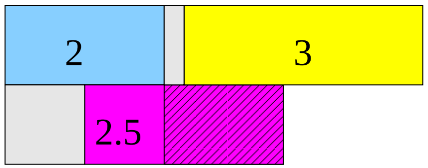
LPT :



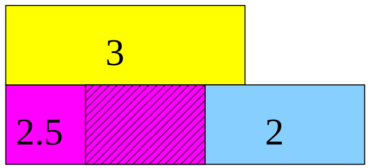
$C_1 = \frac{5}{3}$

if task 1 bids a false value: 2. 5

DSPT :



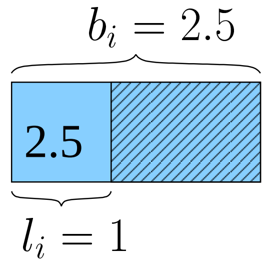
LPT :



$C_1 = \frac{5}{3}$

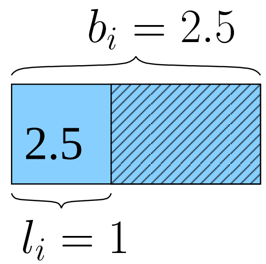
Other results

Until now: if task i bids $b_i > l_i$, its execution time is l_i (it gets its results l_i time units after its start).



Other results

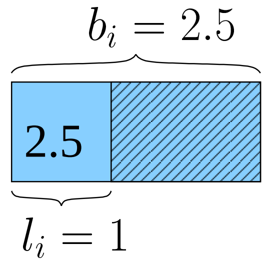
Until now: if task i bids $b_i > l_i$, its execution time is l_i (it gets its results l_i time units after its start).



Other model: if task i bids $b_i > l_i$, its execution time is b_i.

Other results

Until now: if task i bids $b_i > l_i$, its execution time is l_i (it gets its results l_i time units after its start).



Other model: if task i bids $b_i > l_i$, its execution time is b_i.

With this 2nd model:

- A deterministic $\left(\frac{4}{3} - \frac{1}{3\,m}\right)$-approximate truthful algorithm.
- No deterministic $(1.1 - \varepsilon)$ truthful algorithm.
- An optimal randomized truthful algorithm.

An optimal truthful algorithm

Algorithm BLOCK:

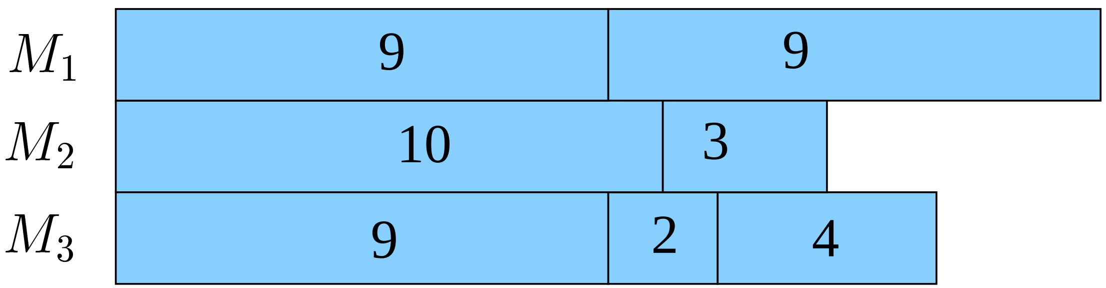
- Get an optimal schedule of the tasks.
  Let OPT be the makespan of the schedule.
  Let L_i be the sum of the tasks lengths on M_i.
- Add a fake task of length $OPT - L_i$ on M_i.
- On each machine, tasks are scheduled in a random order.

An optimal truthful algorithm

Algorithm BLOCK:

- Get an optimal schedule of the tasks.
  Let OPT be the makespan of the schedule.
  Let L_i be the sum of the tasks lengths on M_i.
- Add a fake task of length $OPT - L_i$ on M_i.
- On each machine, tasks are scheduled in a random order.

Example:

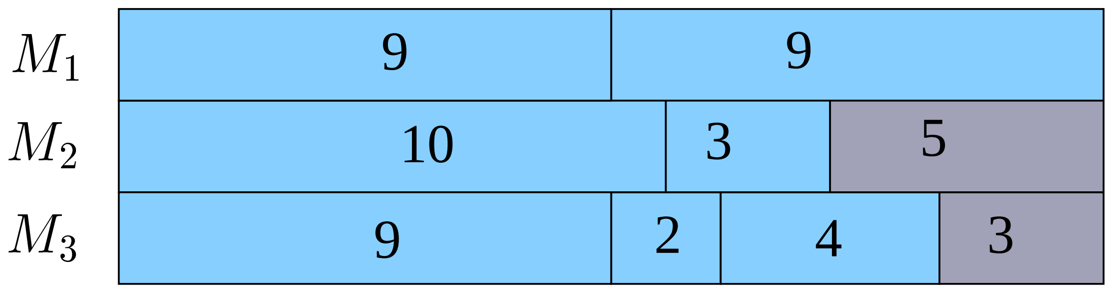


An optimal truthful algorithm

Algorithm BLOCK:

- Get an optimal schedule of the tasks.
  Let OPT be the makespan of the schedule.
  Let L_i be the sum of the tasks lengths on M_i.
- Add a fake task of length $OPT - L_i$ on M_i.
- On each machine, tasks are scheduled in a random order.

Example:



An optimal truthful algorithm

Lemma: Let a set of tasks scheduled in a random order on a single machine.
The expected completion time of task t is:

$$l_t + \frac{1}{2} \sum_{j \neq t} l_j$$

An optimal truthful algorithm

Lemma: Let a set of tasks scheduled in a random order on a single machine.
The expected completion time of task t is:

$$l_t + \frac{1}{2} \sum_{j \neq t} l_j$$

Theorem: Algorithm Block is truthful.

Proof: Let OPT be the makespan when i bids l_i, and OPT' be the makespan when it bids b_i: $OPT \leq OPT'$.

- if i bids $b_i = l_i$: expected comp. time = $l_i + \frac{1}{2}(OPT - l_i)$
- if i bids $b_i > l_i$: expected comp. time = $b_i + \frac{1}{2}(OPT' - b_i)$

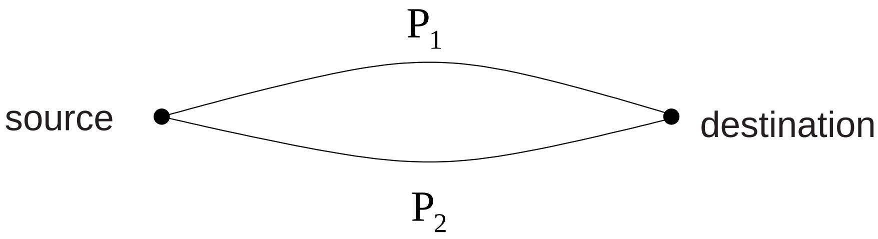
Outline

- Context
  - Classical optimization problems
  - Optimization problems with independent users
- Results
  - Scheduling
    - Performance vs stability
    - Performance vs truthfulness
  - Routing
    - Performance of distributed algorithms
- Future work

Performance of distributed algorithms

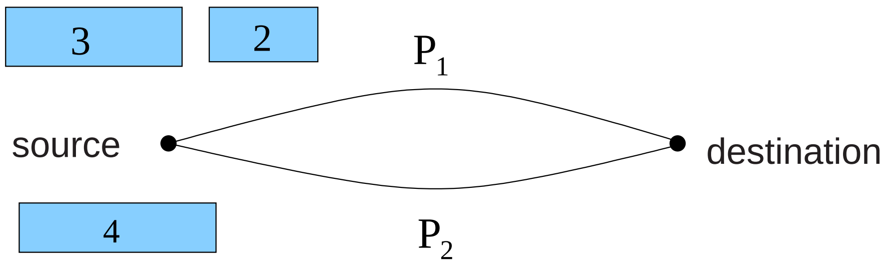
On a set of parallel links:

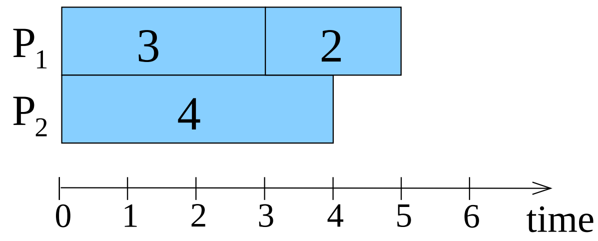
a set of packets:



Performance of distributed algorithms

On a set of parallel links:





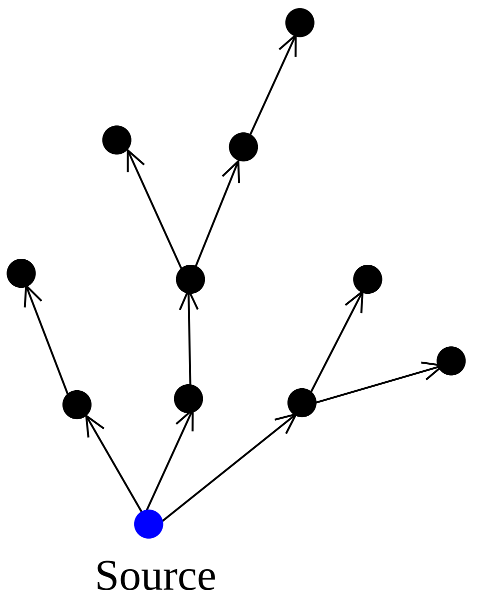
Performance of distributed algorithms

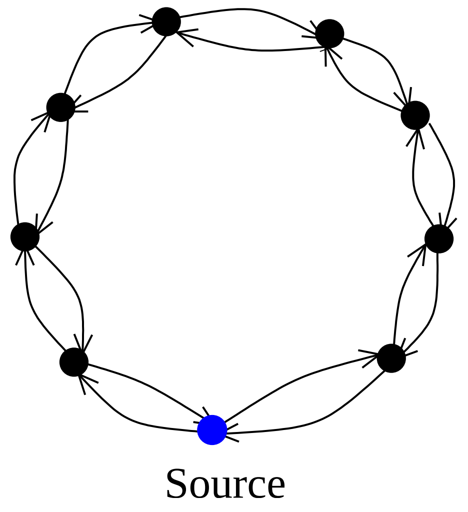
On a set of parallel links:

Best known distributed algorithm: LPT policy. [Christodoulou et al., ICALP 2004]

Distributed algorithms in trees and rings

We wish to route packets, released at the same time from a same source in:





- Each packet has: a length, a destination
- It wishes to minimize its arrival date at its destination
- “Store and forward” network

Related work

The goal is to minimize the maximal arrival date.

- Centralized algorithms in general graphs but with packets of same length. [Leighton, Maggs, Rao, FOCS 1988], [auf der Heide, Vöcking, STACS 1995], [Ostrovsky, Rabani, STOC 1997]

Related work

The goal is to minimize the maximal arrival date.

- Centralized algorithms in general graphs but with packets of same length. [Leighton, Maggs, Rao, FOCS 1988], [auf der Heide, Vöcking, STACS 1995], [Ostrovsky, Rabani, STOC 1997]

- Multicommodity flows over time problem: in a path, optimal solution if each link routes the packets in order of decreasing remaining distance. [Hall, Hippler, Skutella, ICALP 2003]

Distributed algorithms in trees and rings

Decentralized setting: each link knows only the packets it has to route and has a policy to route them. For example:

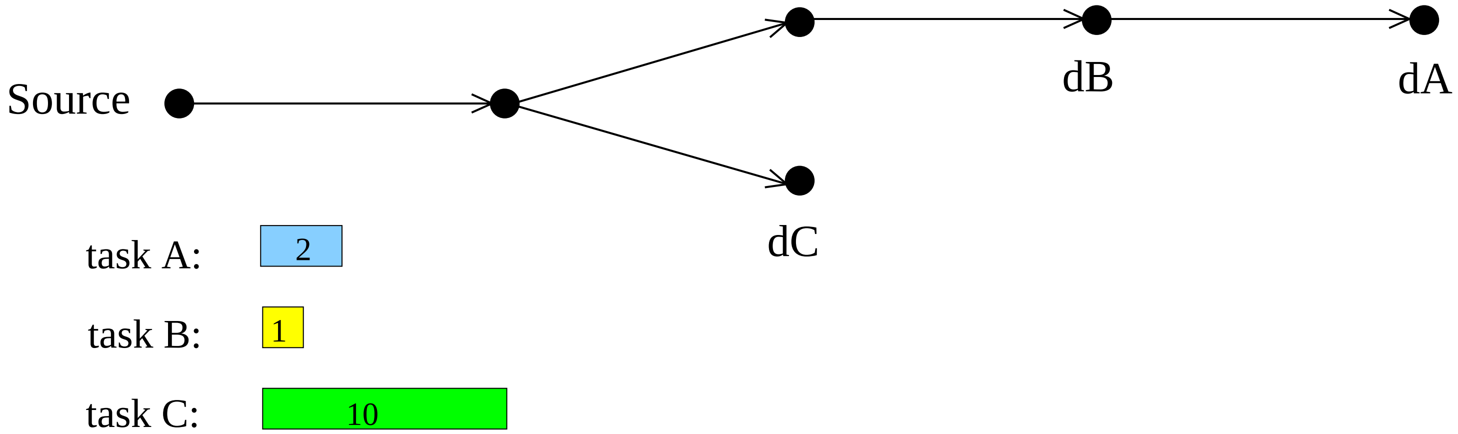
- SPT: *Shortest Processing Time first*
- LPT: *Longest Processing Time first*
- LRD: *Longest Remaining Distance first*

What is the performance of these policies for the following problems ?

- Minimize the maximum arrival date.
- Minimize the average arrival date.

Example

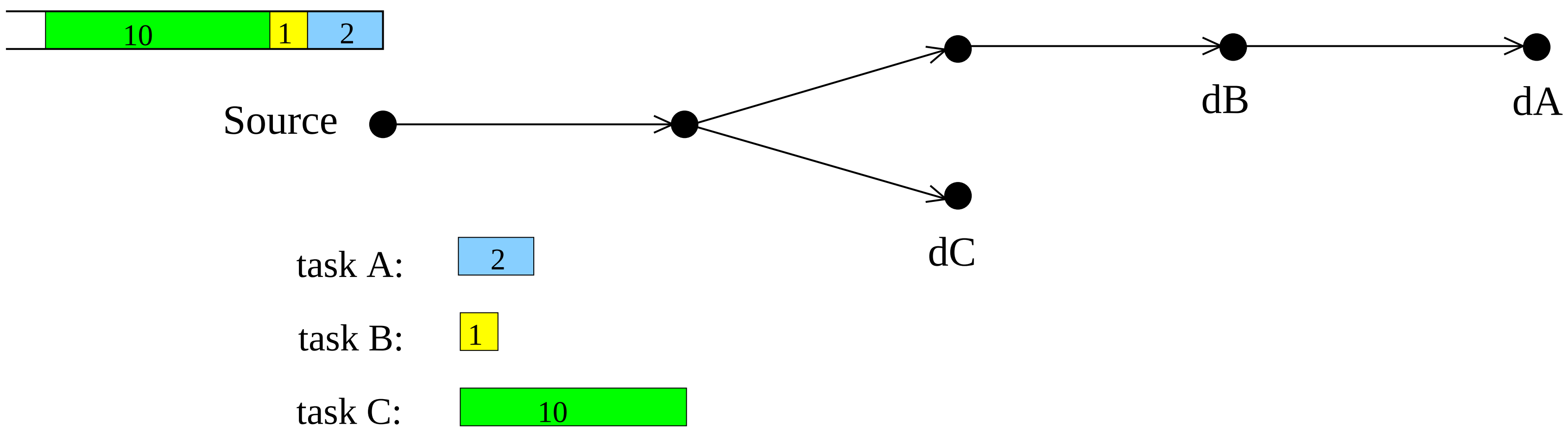
With the LRD policy: the more a task goes far, the earliest it is scheduled.



Example

With the LRD policy: the more a task goes far, the earliest it is scheduled.

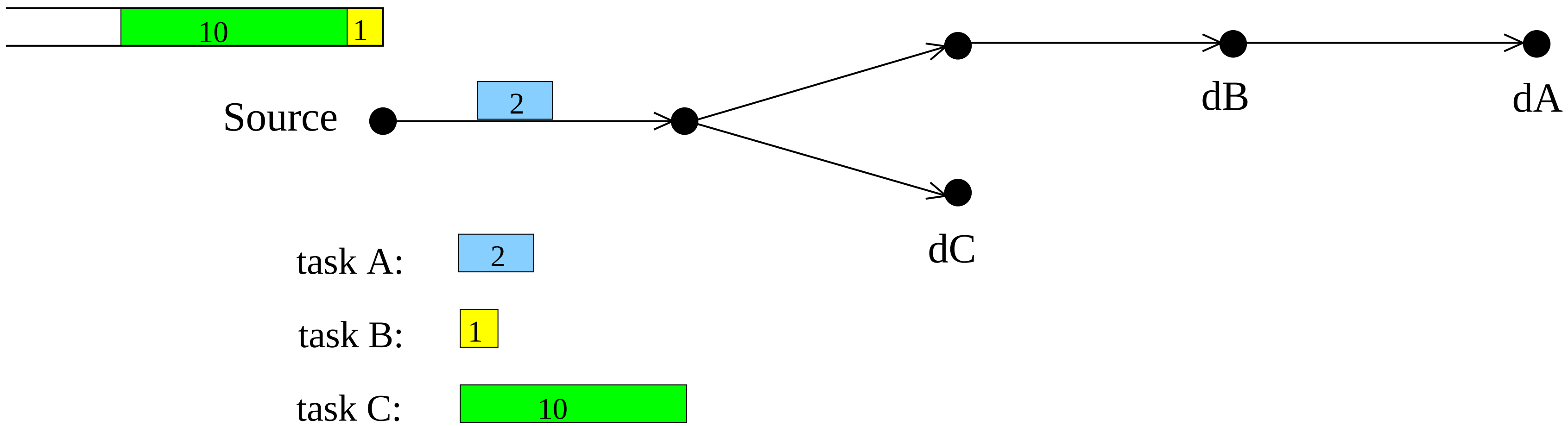
time = 0



Example

With the LRD policy: the more a task goes far, the earliest it is scheduled.

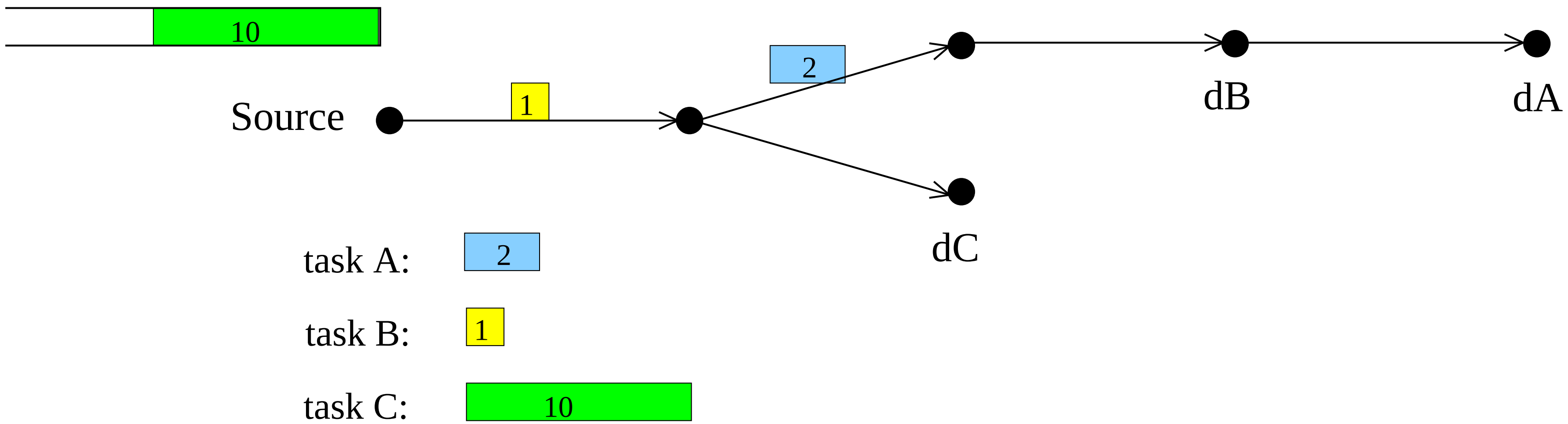
time interval= [0, 2)



Example

With the LRD policy: the more a task goes far, the earliest it is scheduled.

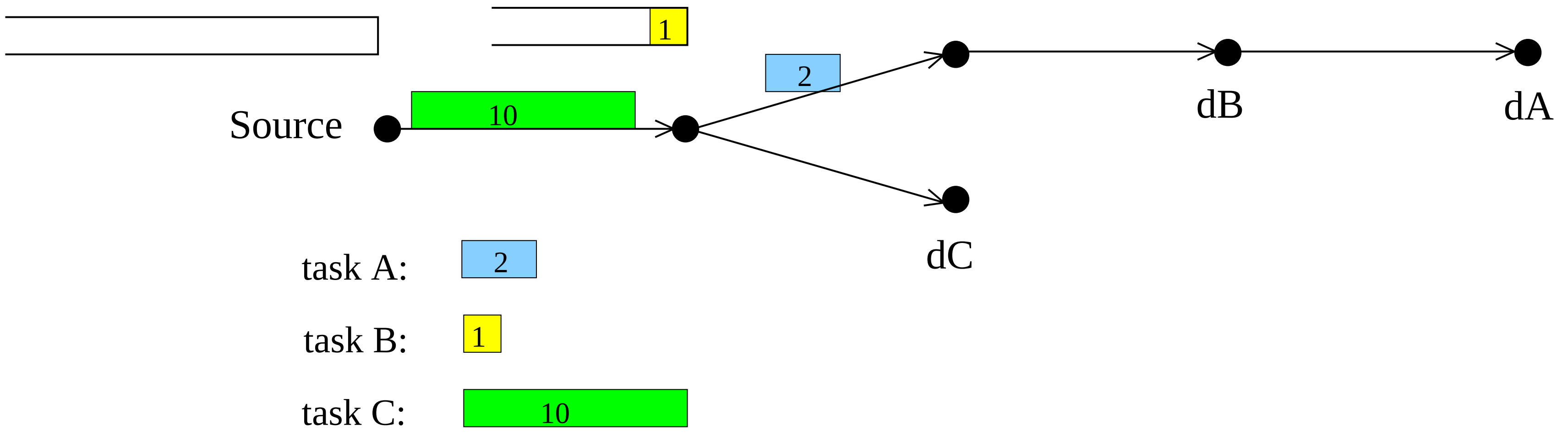
time interval= [2, 3)



Example

With the LRD policy: the more a task goes far, the earliest it is scheduled.

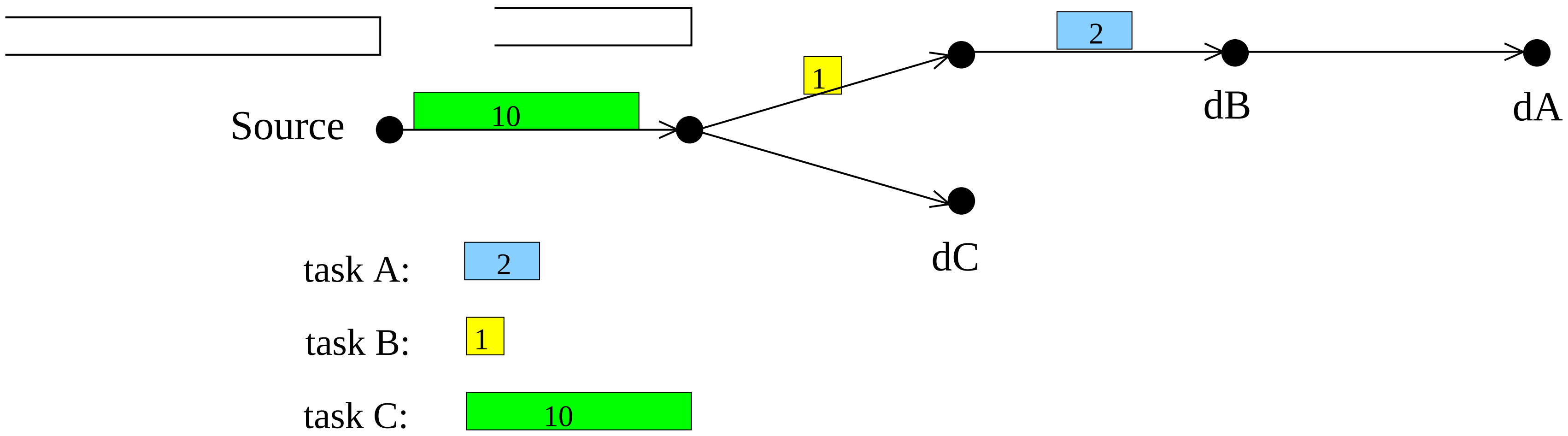
time interval= [3, 4)



Example

With the LRD policy: the more a task goes far, the earliest it is scheduled.

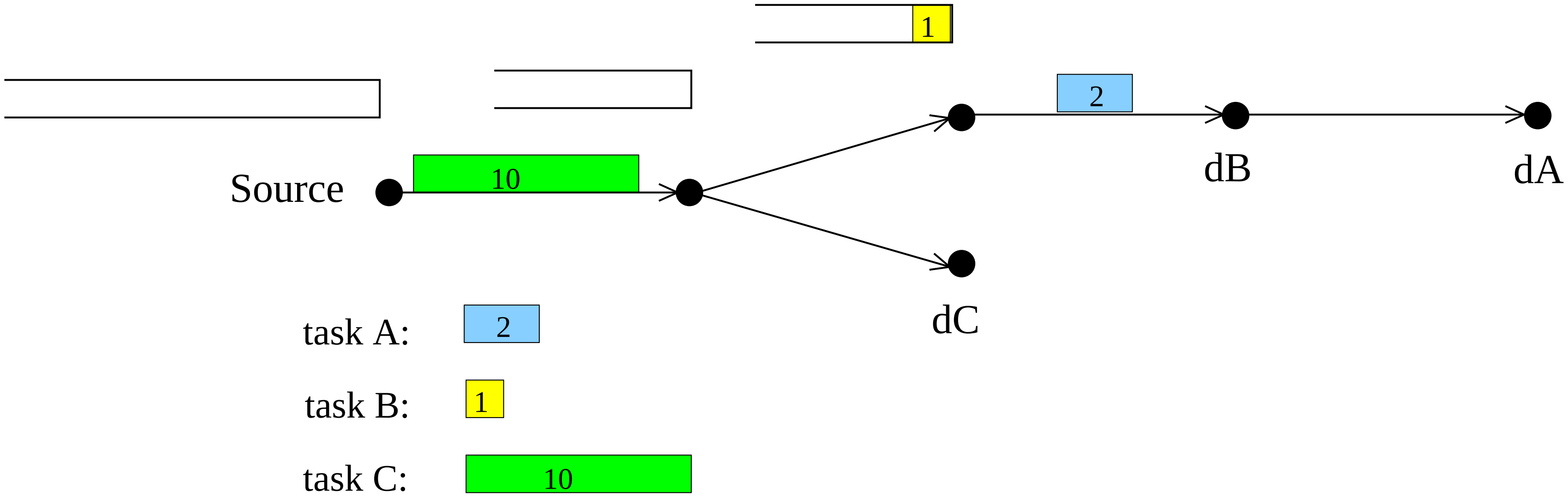
time interval= [4, 5)



Example

With the LRD policy: the more a task goes far, the earliest it is scheduled.

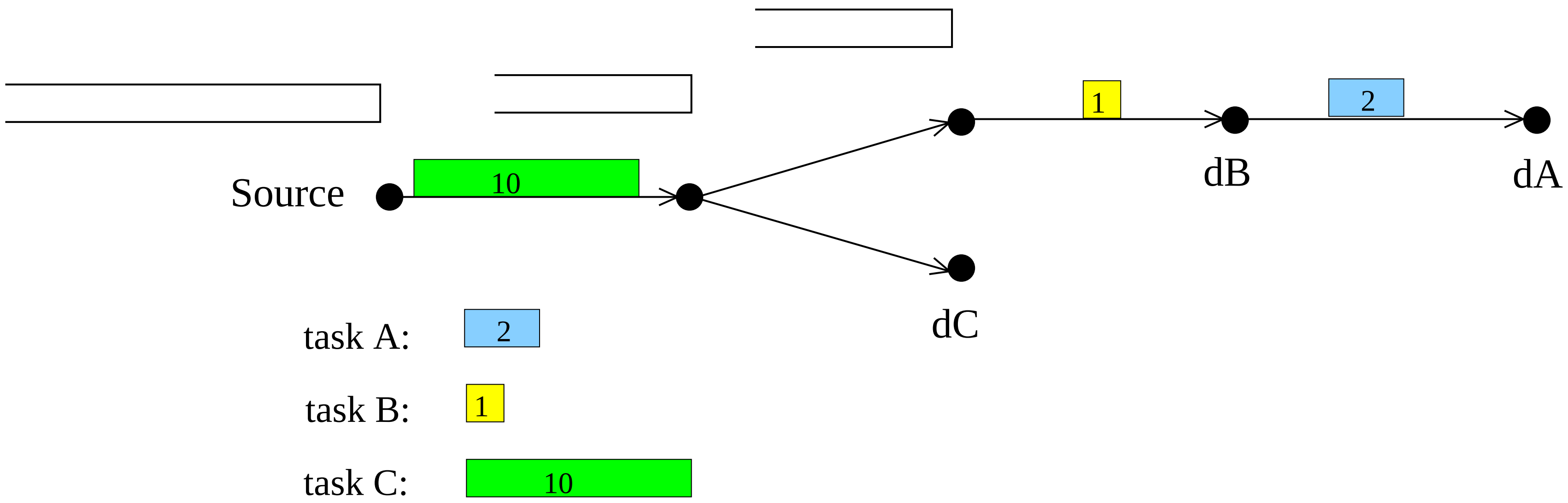
time interval= [5, 6)



Example

With the LRD policy: the more a task goes far, the earliest it is scheduled.

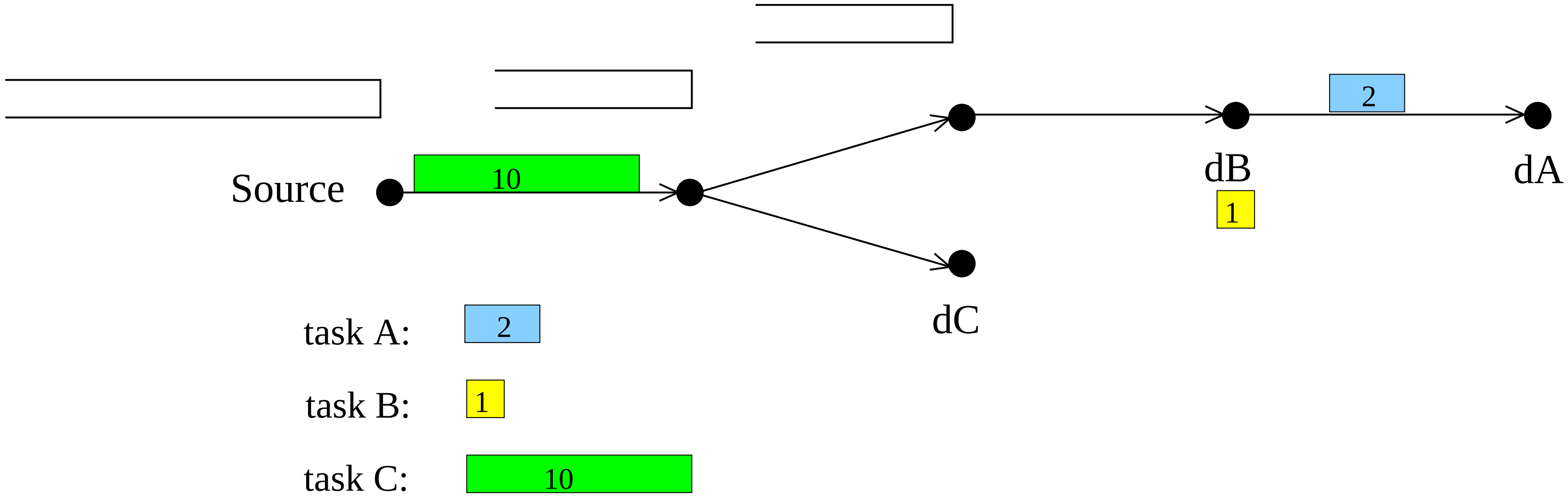
time interval= [6, 7)



Example

With the LRD policy: the more a task goes far, the earliest it is scheduled.

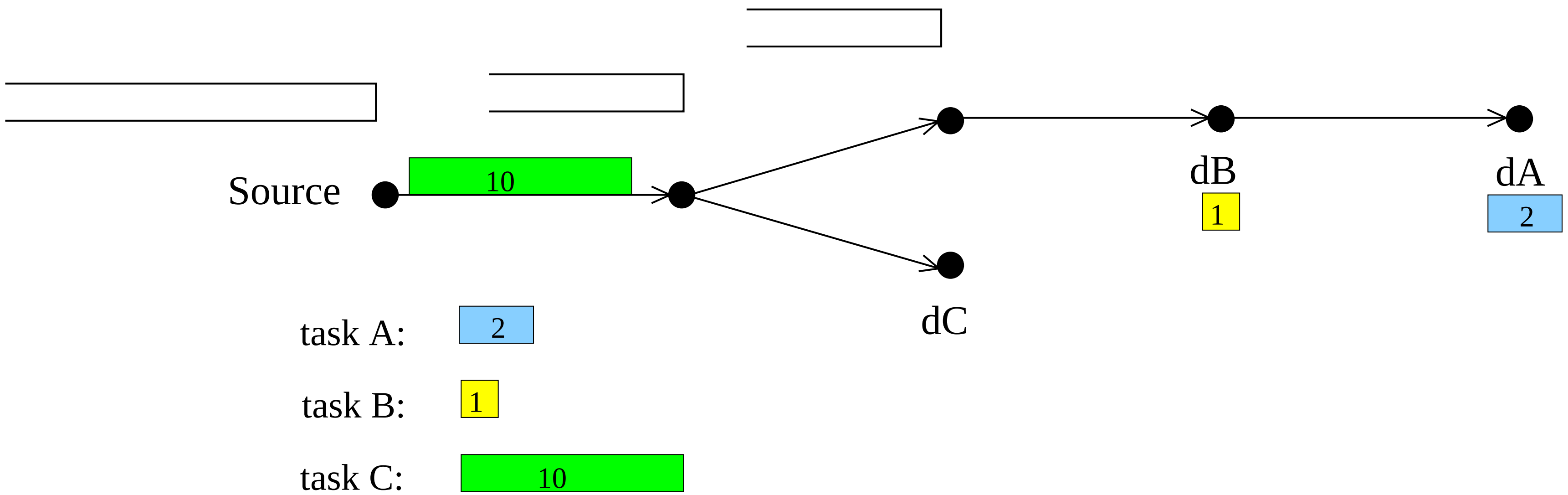
time interval= [7, 8)



Example

With the LRD policy: the more a task goes far, the earliest it is scheduled.

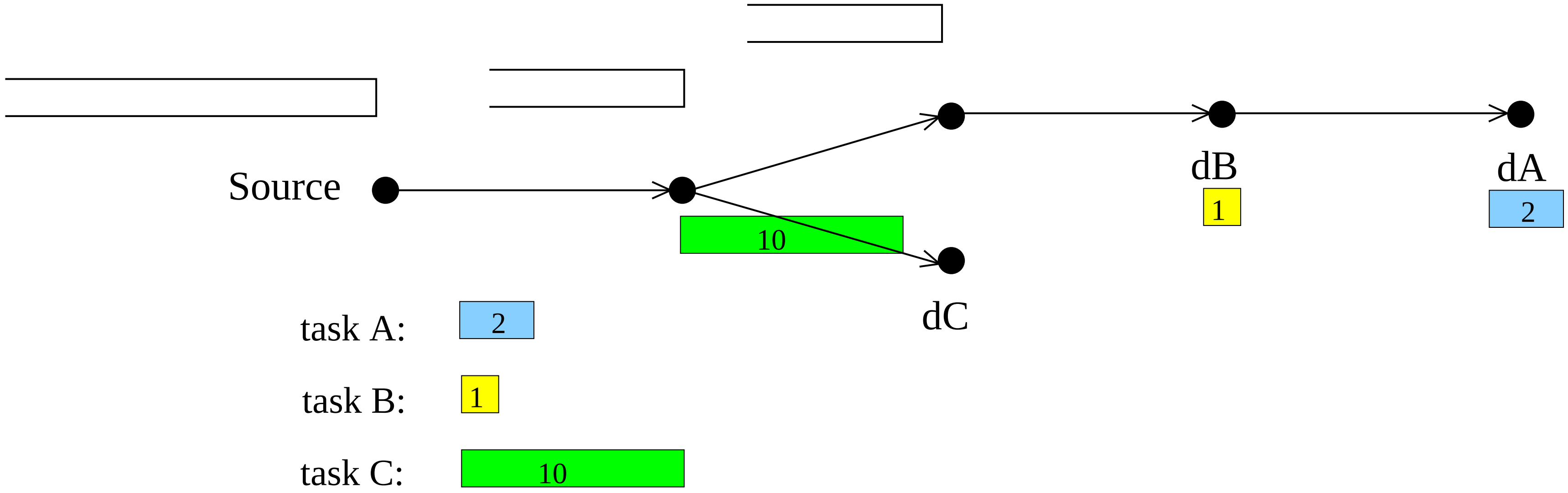
time interval= [8, 13)



Example

With the LRD policy: the more a task goes far, the earliest it is scheduled.

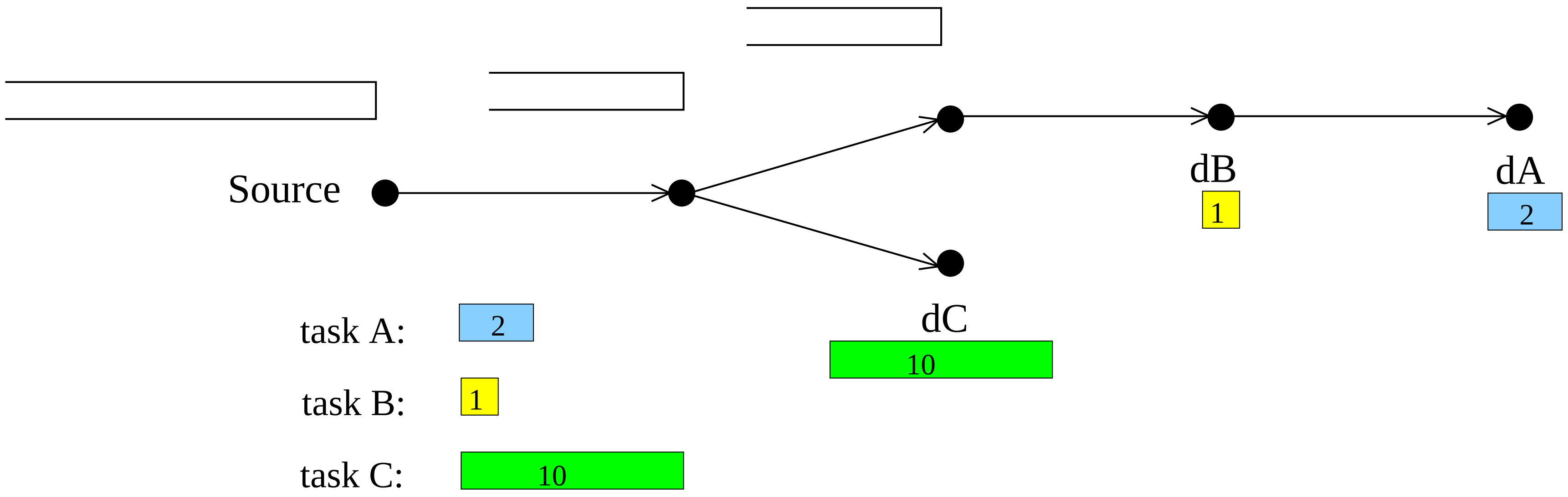
time interval= [13, 23)



Example

With the LRD policy: the more a task goes far, the earliest it is scheduled.

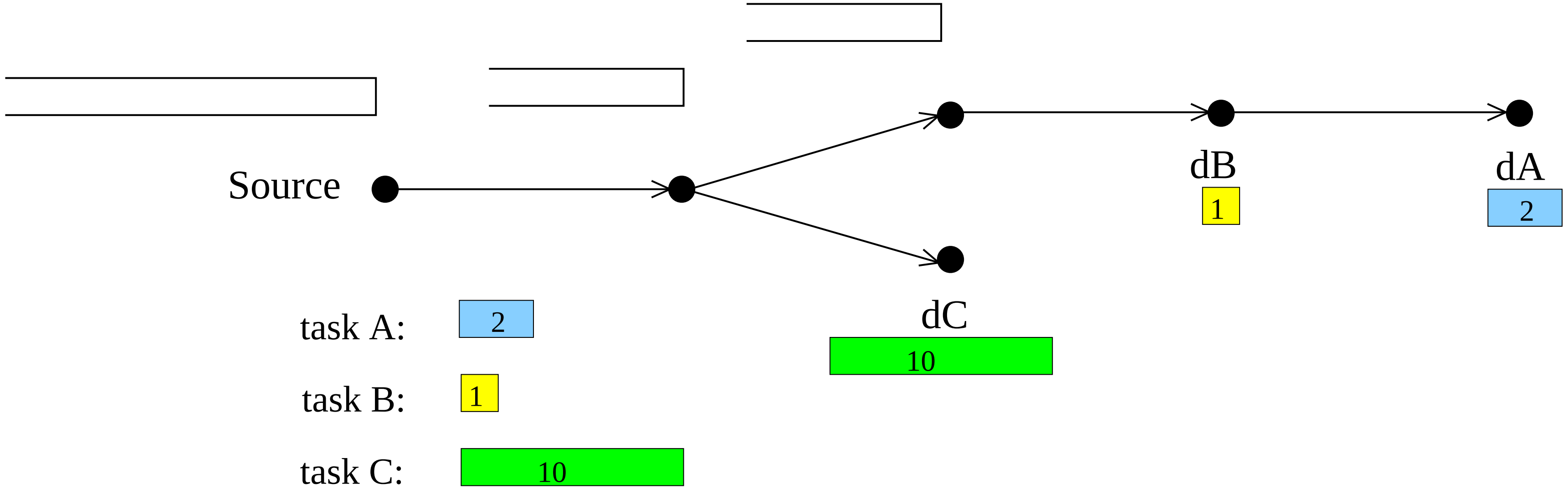
time = 23



Example

With the LRD policy: the more a task goes far, the earliest it is scheduled.

time = 23



In an optimal solution, maximum arrival date = 20.

$\rightarrow$ Approximation ratio $\geq 23/20$.

Rings

Tree: each packet has only one possible strategy.

Ring: choice between two paths at the source.

Rings

Tree: each packet has only one possible strategy.

Ring: choice between two paths at the source.

Nash equilibrium: No user has incentive to unilaterally change strategy.

Rings

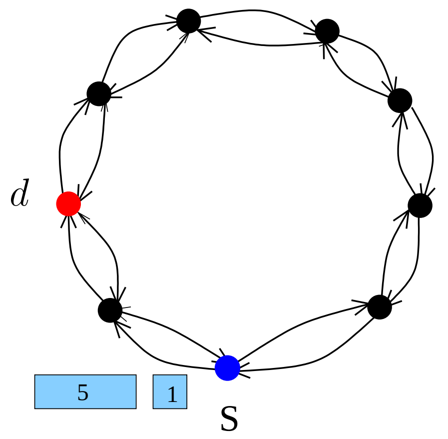
Tree: each packet has only one possible strategy.

Ring: choice between two paths at the source.

Nash equilibrium: No user has incentive to unilaterally change strategy.

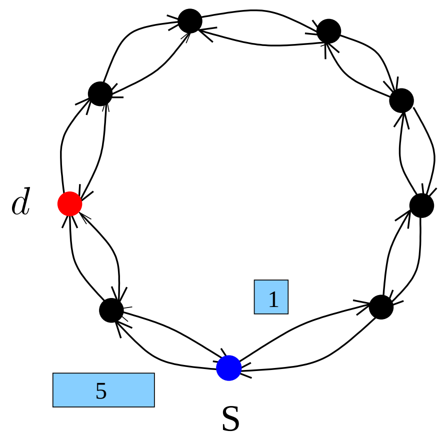
Example:

Policy = LPT



Arrival date of 5 : 10

Arrival date of 1 : 11



Arrival date of 5 : 10

Arrival date of 1 : 7

Results

Our goal: to minimize the maximum arrival date:

- LPT policy: ratio in $\Theta(\text{number of packets})$.
- SPT and LRD policies: in a tree: ratio = 2
  in a ring: ratio < 3

Results

Our goal: to minimize the maximum arrival date:

- LPT policy: ratio in $\Theta(\text{number of packets})$.
- SPT and LRD policies: in a tree: ratio = 2
  in a ring: ratio < 3

Our goal: to minimize the average arrival date:

- LPT and LRD policies: ratio in $\Theta(\text{number of packets})$.
- SPT policy: in a tree: optimal
  in a ring: ratio < 2

Outline

- Context
  - Classical optimization problems
  - Optimization problems with independent users
- Results
  - Scheduling
    - Performance vs stability
    - Performance vs truthfulness
  - Routing
    - Performance of distributed algorithms
- Future work

Future work

- Trade-off stability/performance
  - In a distributed setting: solve the CKN conjecture
  - In a centralized setting: is there a deterministic policy better than LPT ?
  - Mixed Nash equilibrium, correlated equilibrium

Future work

- Trade-off stability/performance
  - In a distributed setting: solve the CKN conjecture
  - In a centralized setting: is there a deterministic policy better than LPT ?
  - Mixed Nash equilibrium, correlated equilibrium
- Truthfulness
  - In a distributed setting: truthful algorithms
  - Consider related machines
  - Truthful algorithms when considering payments
  - Truthful algorithms when a task can bid $b_i < l_i$?

Future work

- Trade-off stability/performance
  - In a distributed setting: solve the CKN conjecture
  - In a centralized setting: is there a deterministic policy better than LPT ?
  - Mixed Nash equilibrium, correlated equilibrium
- Truthfulness
  - In a distributed setting: truthful algorithms
  - Consider related machines
  - Truthful algorithms when considering payments
  - Truthful algorithms when a task can bid $b_i < l_i$?
- Distributed algorithms for a routing problem
  - Several sources/destinations
  - Other topologies: in any graph
  - Online analysis

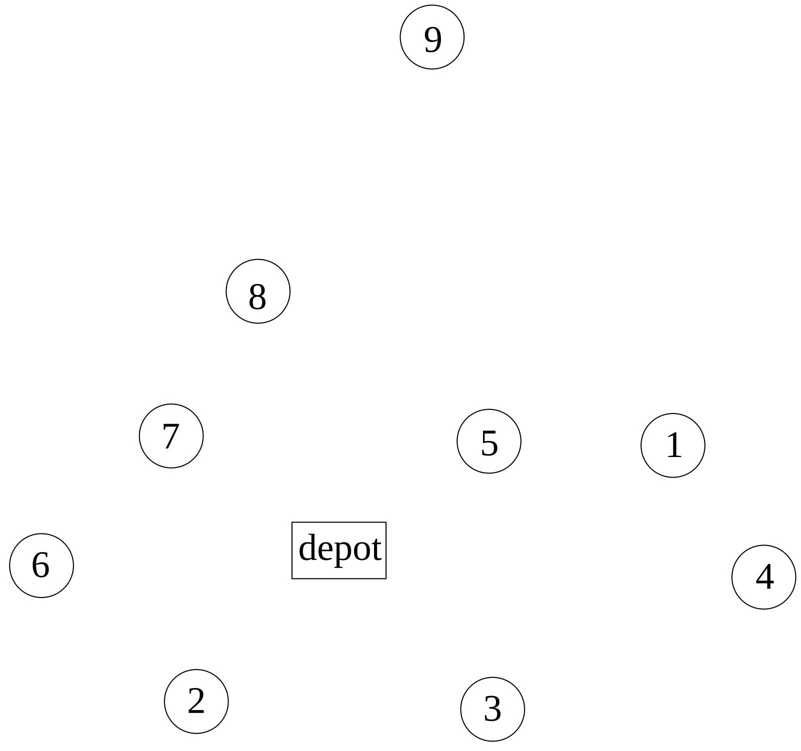
Annexe

Other problems

Examples : vehicule routing problem, traffic grooming problem, scheduling problem.

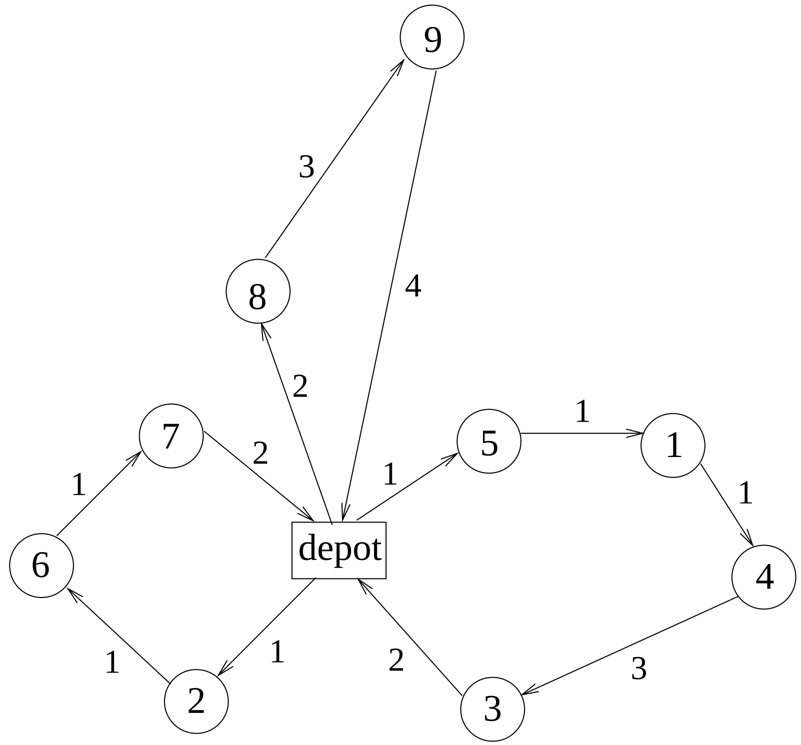
Other problems

Examples : vehicule routing problem, traffic grooming problem, scheduling problem.



Other problems

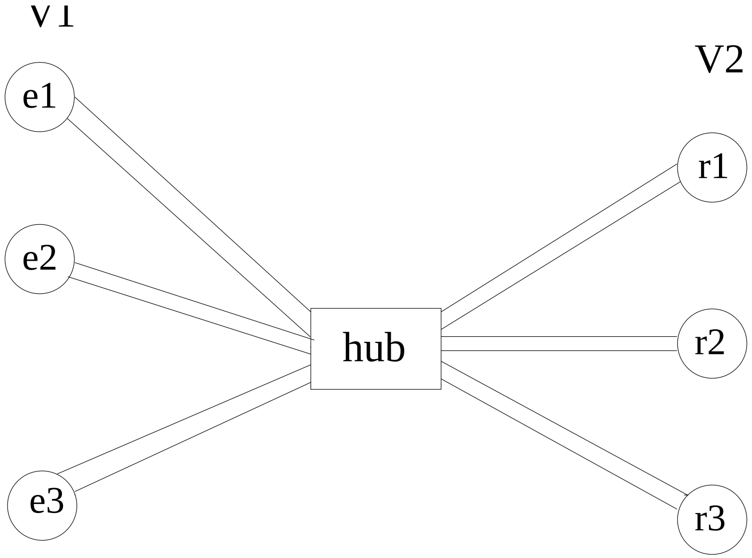
Examples : vehicule routing problem, traffic grooming problem, scheduling problem.



Other problems

Examples : vehicule routing problem, traffic grooming problem, scheduling problem.

Topology



number of links = 2, capacity of each link = 10

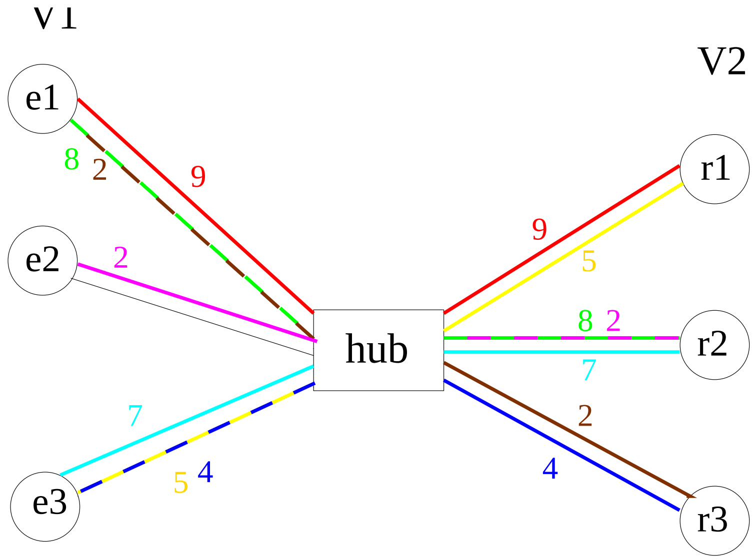
Traffic Matrix (requests)

$$\mathbf{T} = \begin{pmatrix} 9 & 8 & 2 \\ 0 & 2 & 0 \\ 5 & 7 & 4 \end{pmatrix}$$

Other problems

Examples : vehicule routing problem, traffic grooming problem, scheduling problem.

Topology



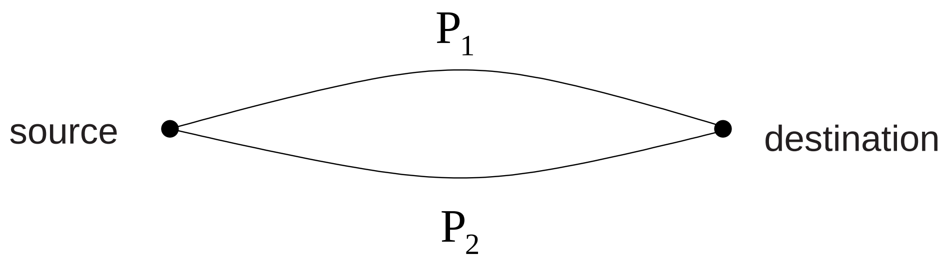
number of links = 2, capacity of each link = 10

Traffic Matrix (requests)

$$\mathbf{T} = \begin{pmatrix} 9 & 8 & 2 \\ 0 & 2 & 0 \\ 5 & 7 & 4 \end{pmatrix}$$

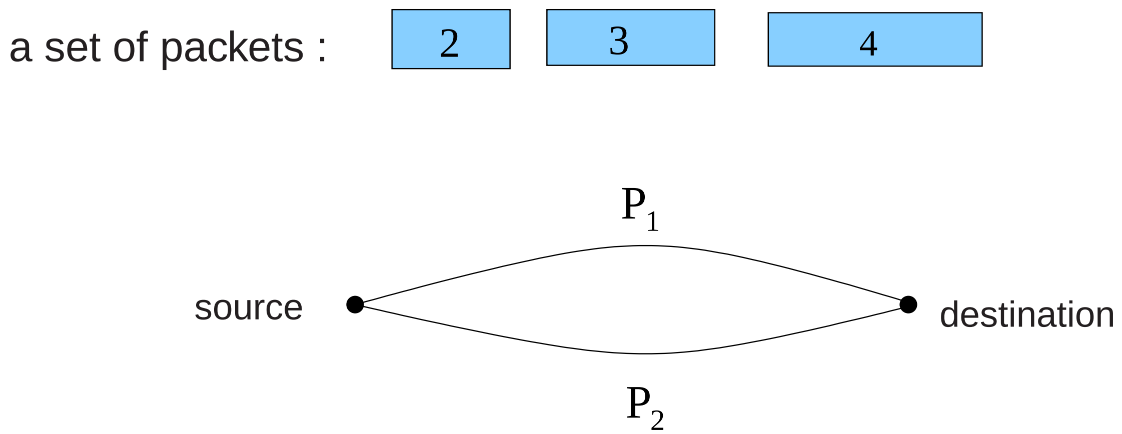
Other problems

Examples : vehicule routing problem, traffic grooming problem, scheduling problem.



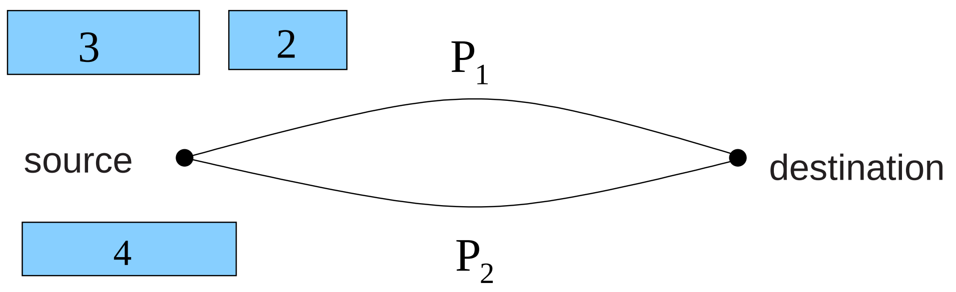
Other problems

Examples : vehicule routing problem, traffic grooming problem, scheduling problem.



Other problems

Examples : vehicule routing problem, traffic grooming problem, scheduling problem.



Other problems

Examples : vehicule routing problem, traffic grooming problem, scheduling problem.