Scheduling under Unavailability and Uncertainty Constraints

Louis-Claude Canon, Adel Essafi,
Grégory Mounié, Denis Trystram

Project-Team MOAIS
Grenoble INP / LIG / INRIA
UTIC

May 30, 2011

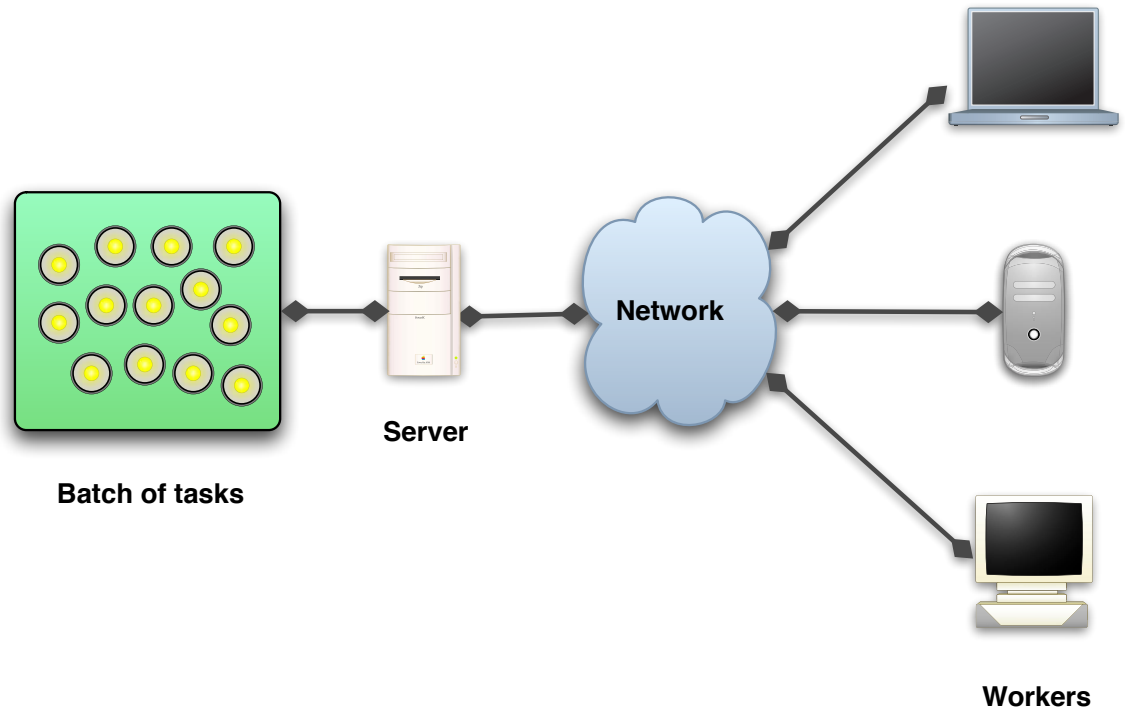
Outline

Outline

Desktop Grids

Distributed Computing

- The server assigns **tasks** to **workers**.
- Each worker returns a **result** for the corresponding job.
- Example: BOINC-based projects.



Examples

SETI@home

Goal: analyze radio signals, searching for signs of extra terrestrial intelligence.
Anybody can participate by running a free program that downloads and analyzes radio telescope data.
Participants gain credits for performing work.

Folding@home

Goal: understand protein folding, misfolding, and related diseases.

Examples

SETI@home

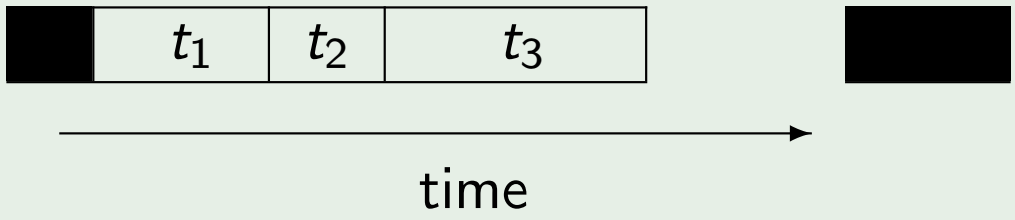
Goal: analyze radio signals, searching for signs of extra terrestrial intelligence.
Anybody can participate by running a free program that downloads and analyzes radio telescope data.
Participants gain credits for performing work.

Folding@home

Goal: understand protein folding, misfolding, and related diseases.

Unavailability and Uncertainty constraints

Example



Interruption

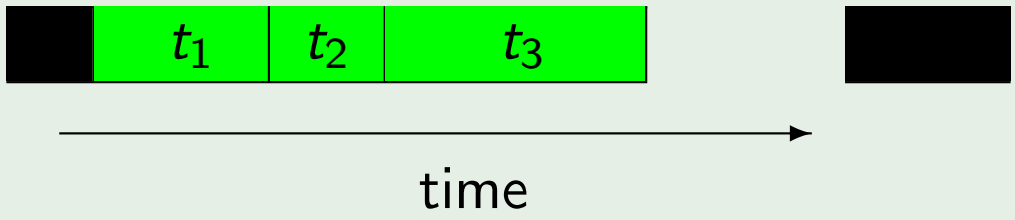
Interrupted tasks need to be re-executed.

Tackling Uncertainty

A proactive mechanism and a reactive re-execution strategy (each interrupted task is executed as soon as possible without delaying next scheduled tasks).

Unavailability and Uncertainty constraints

Example



Interruption

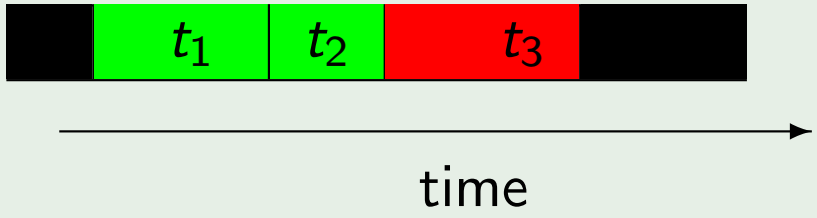
Interrupted tasks need to be re-executed.

Tackling Uncertainty

A proactive mechanism and a reactive re-execution strategy (each interrupted task is executed as soon as possible without delaying next scheduled tasks).

Unavailability and Uncertainty constraints

Example



Interruption

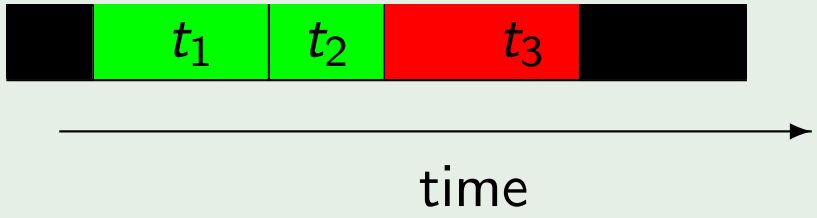
Interrupted tasks need to be re-executed.

Tackling Uncertainty

A proactive mechanism and a reactive re-execution strategy (each interrupted task is executed as soon as possible without delaying next scheduled tasks).

Unavailability and Uncertainty constraints

Example



Interruption

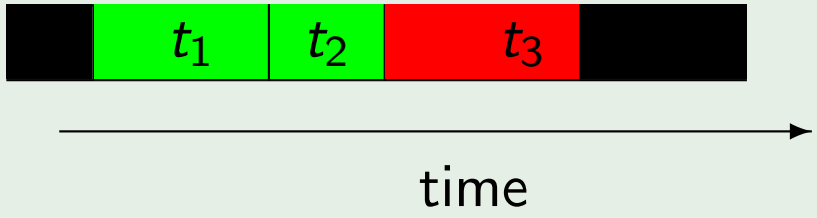
Interrupted tasks need to be re-executed.

Tackling Uncertainty

A proactive mechanism and a reactive re-execution strategy (each interrupted task is executed as soon as possible without delaying next scheduled tasks).

Unavailability and Uncertainty constraints

Example



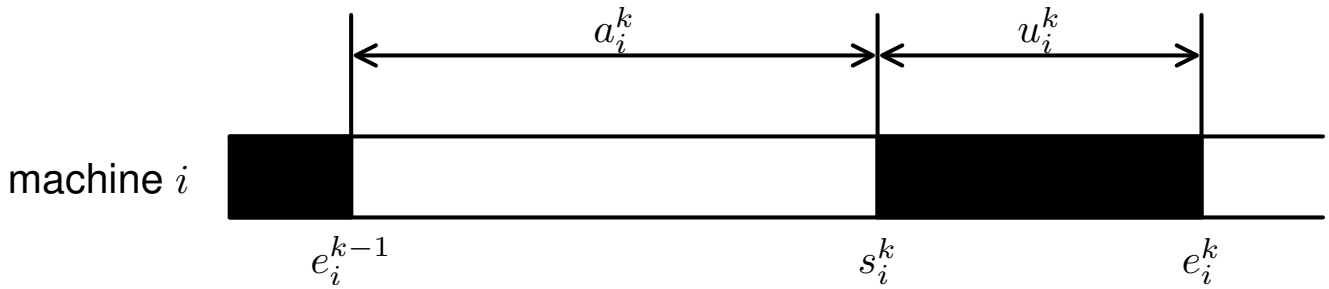
Interruption

Interrupted tasks need to be re-executed.

Tackling Uncertainty

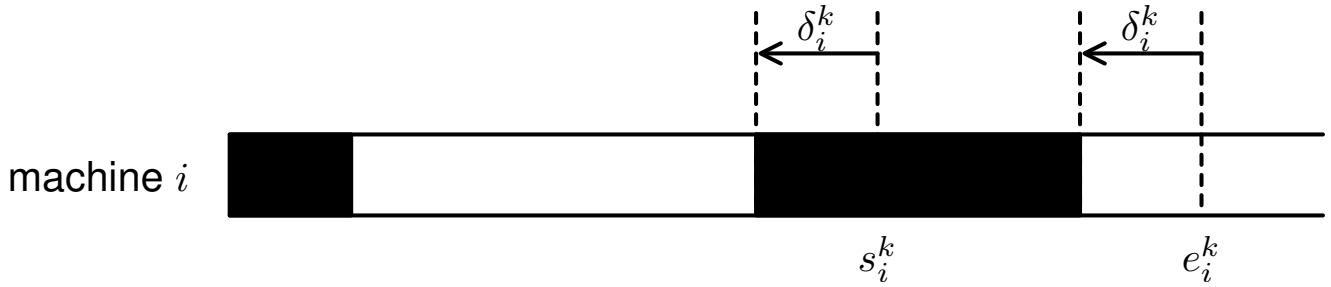
A proactive mechanism and a reactive re-execution strategy (each interrupted task is executed as soon as possible without delaying next scheduled tasks).

Assumptions



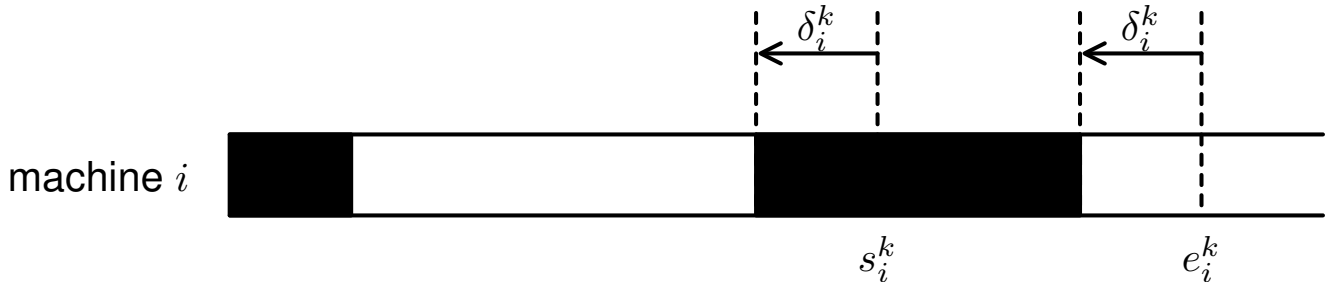
- Unavailability: u_k^i is certain
- Earliness: $\delta_i^k \leq a_i^k$
- Task durations: p_i with $2 \max p_i \leq \min a_i^k$

Assumptions



- Unavailability: u_k^i is certain
- Earliness: $\delta_i^k \leq a_i^k$
- Task durations: p_i with $2 \max p_i \leq \min a_i^k$

Assumptions



- Unavailability: u_k^i is certain
- Earliness: $\delta_i^k \leq a_i^k$
- Task durations: p_i with $2 \max p_i \leq \min a_i^k$

Checkpointing

Checkpoint

Often used in desktop grids to overcome the high volatility.

Limits of Checkpointing

- High cost.
- Cannot always be used (application-specific).

Our approach

Avoid the cost of checkpointing in desktop grids with good predictability (enterprise computing).

Checkpointing

Checkpoint

Often used in desktop grids to overcome the high volatility.

Limits of Checkpointing

- High cost.
- Cannot always be used (application-specific).

Our approach

Avoid the cost of checkpointing in desktop grids with good predictability (enterprise computing).

Checkpointing

Checkpoint

Often used in desktop grids to overcome the high volatility.

Limits of Checkpointing

- High cost.
- Cannot always be used (application-specific).

Our approach

Avoid the cost of checkpointing in desktop grids with good predictability (enterprise computing).

Problem Definition

Stability

The *stability* is the ratio between the worst disturbed makespan and the baseline makspan.

Bi-Objective Scheduling

Find an allocation function with minimal makespan and minimal value of stability.

Problem Definition

Stability

The *stability* is the ratio between the worst disturbed makespan and the baseline makspan.

Bi-Objective Scheduling

Find an allocation function with minimal makespan and minimal value of stability.

Outline

Slack Rule

Slack

The *slack* is the amount of idle time on each availability.

Slack rule

The slack must greater or equal to the maximum length of the jobs assigned to the corresponding availability.

Stability Condition

Any schedule respecting the slack rule is stable (stability equal to 1).

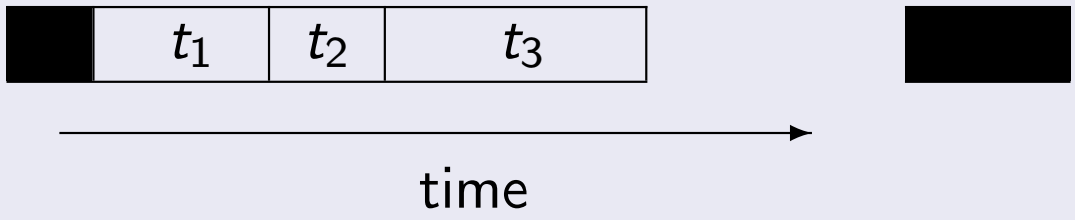
Slack Rule

Slack

The *slack* is the amount of idle time on each availability.

Slack rule

The slack must greater or equal to the maximum length of the jobs assigned to the corresponding availability.



Stability Condition

Any schedule respecting the slack rule is stable (stability equal to 1).

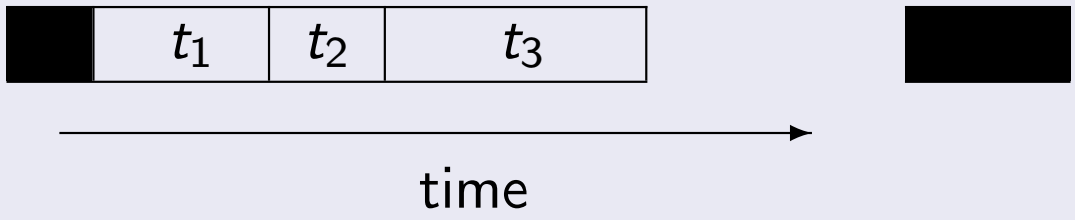
Slack Rule

Slack

The *slack* is the amount of idle time on each availability.

Slack rule

The slack must greater or equal to the maximum length of the jobs assigned to the corresponding availability.



Stability Condition

Any schedule respecting the slack rule is stable (stability equal to 1).

Complexity

Problem

Minimizing the makespan while respecting the slack rule.

Relation with a modified knapsack problem

On each availability, there must be as much elements as possible.
Let p_i be the profit of element i and $s_i = p_i$ its size. The capacity of the knapsack is C. Element i is selected if $x_i = 1$. Then, we want to maximize $\sum_i x_i p_i$ given the constraint $\sum_i x_i s_i + \max_i x_i s_i \leq C$.

Complexity

The problem is strongly NP-Hard (reduction from 3-Partition).

Complexity

Problem

Minimizing the makespan while respecting the slack rule.

Relation with a modified knapsack problem

On each availability, there must be as much elements as possible.
Let p_i be the profit of element i and $s_i = p_i$ its size. The capacity of the knapsack is C. Element i is selected if $x_i = 1$. Then, we want to maximize $\sum_i x_i p_i$ given the constraint $\sum_i x_i s_i + \max_i x_i s_i \leq C$.

Complexity

The problem is strongly NP-Hard (reduction from 3-Partition).

Complexity

Problem

Minimizing the makespan while respecting the slack rule.

Relation with a modified knapsack problem

On each availability, there must be as much elements as possible.
Let p_i be the profit of element i and $s_i = p_i$ its size. The capacity of the knapsack is C. Element i is selected if $x_i = 1$. Then, we want to maximize $\sum_i x_i p_i$ given the constraint $\sum_i x_i s_i + \max_i x_i s_i \leq C$.

Complexity

The problem is strongly NP-Hard (reduction from 3-Partition).

Parametrized Slack Rule

Parametrized Slack Rule

The slack is proportional to a parameter β (0 for performance, 1 for stability).

Stability Bound

The worst case stability using this approach is

$$r_S = \begin{cases} \frac{5}{2} - \beta + \gamma & \text{if } \beta \neq 1 \\ 1 & \text{otherwise} \end{cases}$$

where γ is the ratio between the longest unavailability over the shortest availability.

Parametrized Slack Rule

Parametrized Slack Rule

The slack is proportional to a parameter β (0 for performance, 1 for stability).

Stability Bound

The worst case stability using this approach is

$$r_S = \begin{cases} \frac{5}{2} - \beta + \gamma & \text{if } \beta \neq 1 \\ 1 & \text{otherwise} \end{cases}$$

where γ is the ratio between the longest unavailability over the shortest availability.

Parametrized Slack Rule

Parametrized Slack Rule

The slack is proportional to a parameter β (0 for performance, 1 for stability).

Stability Bound

The worst case stability using this approach is

$$r_S = \begin{cases} \frac{4+\gamma}{2+\beta} & \text{if } \beta \neq 1 \\ 1 & \text{otherwise} \end{cases}$$

where γ is the ratio between the longest unavailability over the shortest availability.

Outline

Schemes for Scheduling Tasks

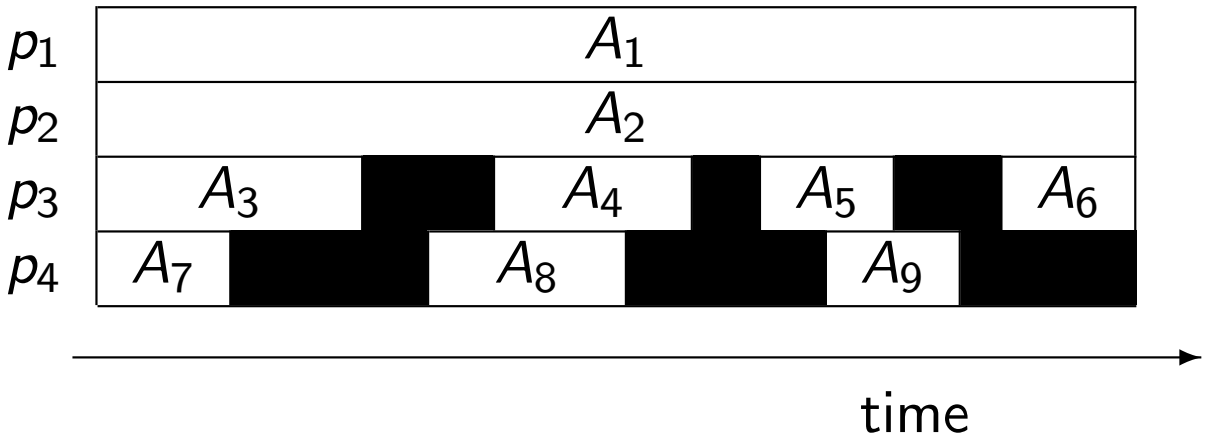
General Principle

- Consider each availability in a given order.
- Fill each availability with tasks while respecting the parameterized slack rule.

Schemes for Scheduling Tasks

General Principle

- Consider each availability in a given order.
- Fill each availability with tasks while respecting the parameterized slack rule.



Greedy Strategy

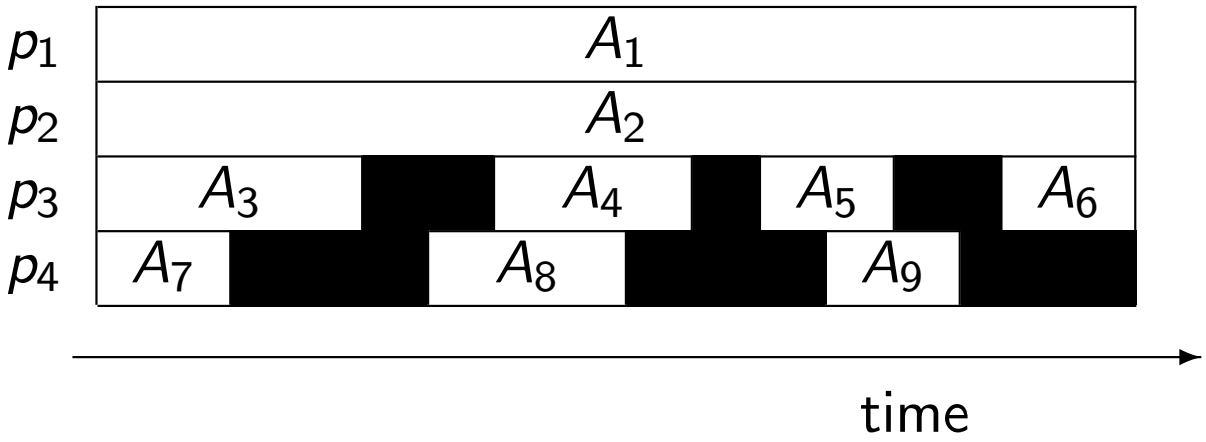
Principal Steps

- Sort availability by non-decreasing end date of following unavailability.
- Fill each availability until the non-scheduled tasks fit into the free processors (*i.e.*, with no unavailability).

Greedy Strategy

Principal Steps

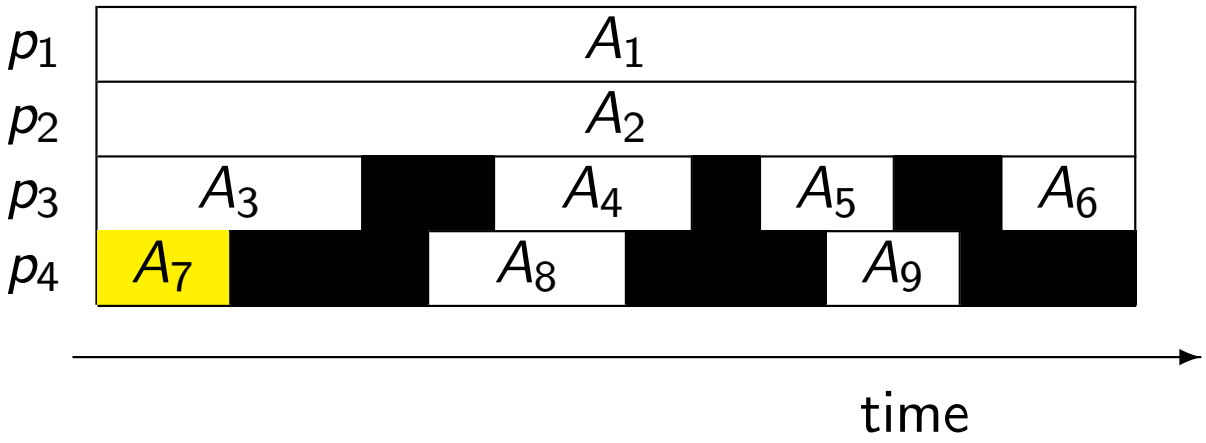
- Sort availability by non-decreasing end date of following unavailability.
- Fill each availability until the non-scheduled tasks fit into the free processors (*i.e.*, with no unavailability).



Greedy Strategy

Principal Steps

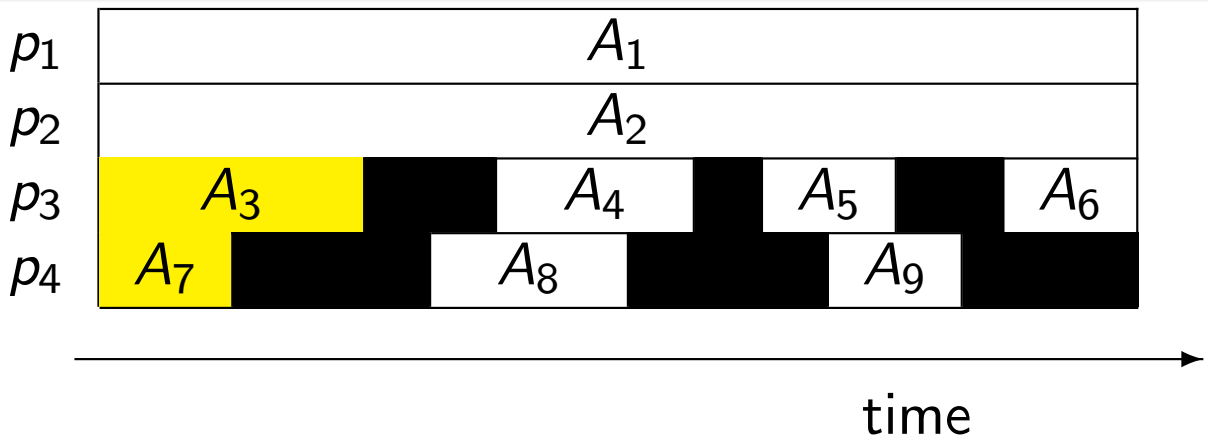
- Sort availability by non-decreasing end date of following unavailability.
- Fill each availability until the non-scheduled tasks fit into the free processors (*i.e.*, with no unavailability).



Greedy Strategy

Principal Steps

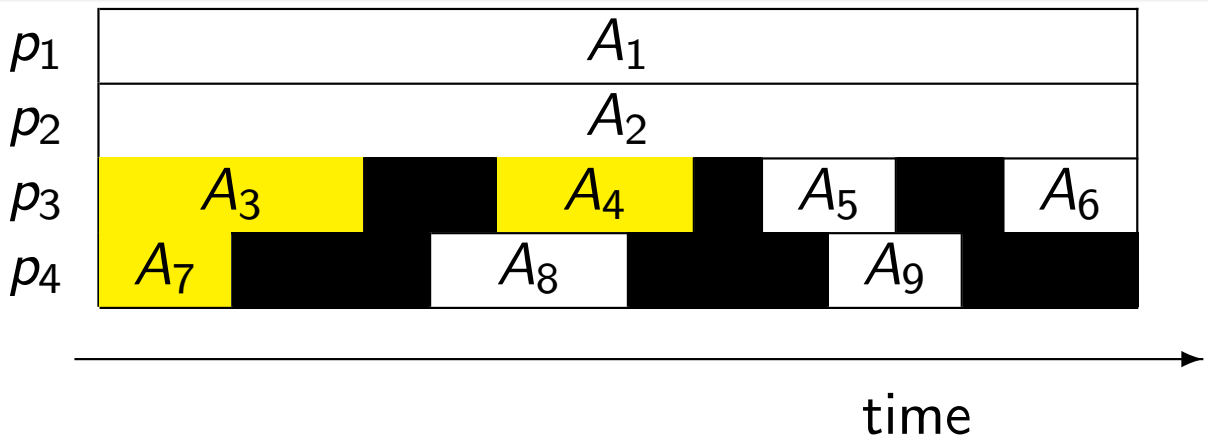
- Sort availability by non-decreasing end date of following unavailability.
- Fill each availability until the non-scheduled tasks fit into the free processors (*i.e.*, with no unavailability).



Greedy Strategy

Principal Steps

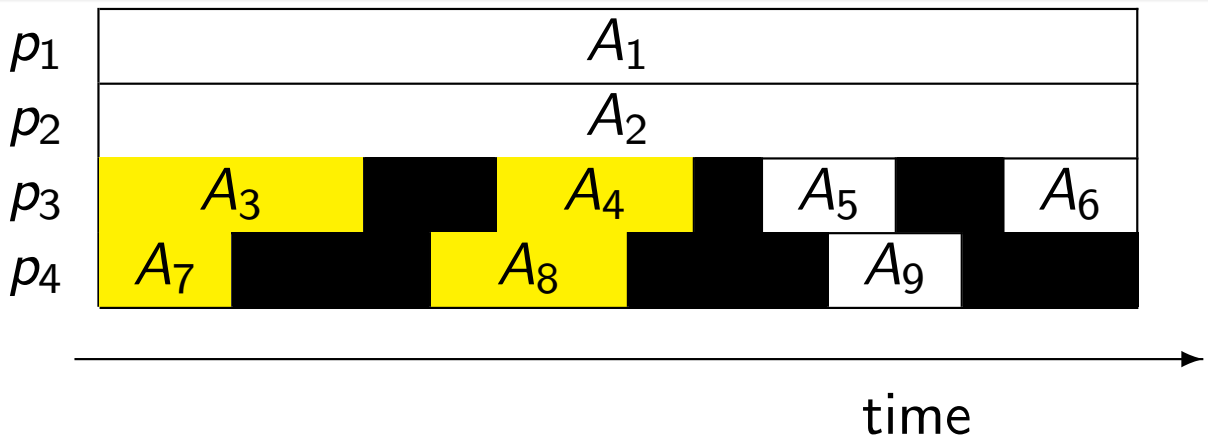
- Sort availability by non-decreasing end date of following unavailability.
- Fill each availability until the non-scheduled tasks fit into the free processors (*i.e.*, with no unavailability).



Greedy Strategy

Principal Steps

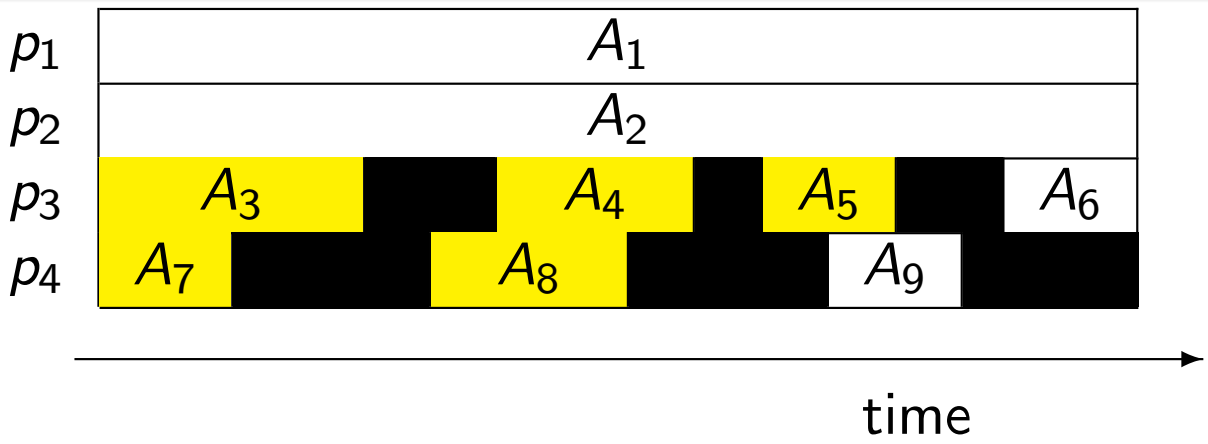
- Sort availability by non-decreasing end date of following unavailability.
- Fill each availability until the non-scheduled tasks fit into the free processors (*i.e.*, with no unavailability).



Greedy Strategy

Principal Steps

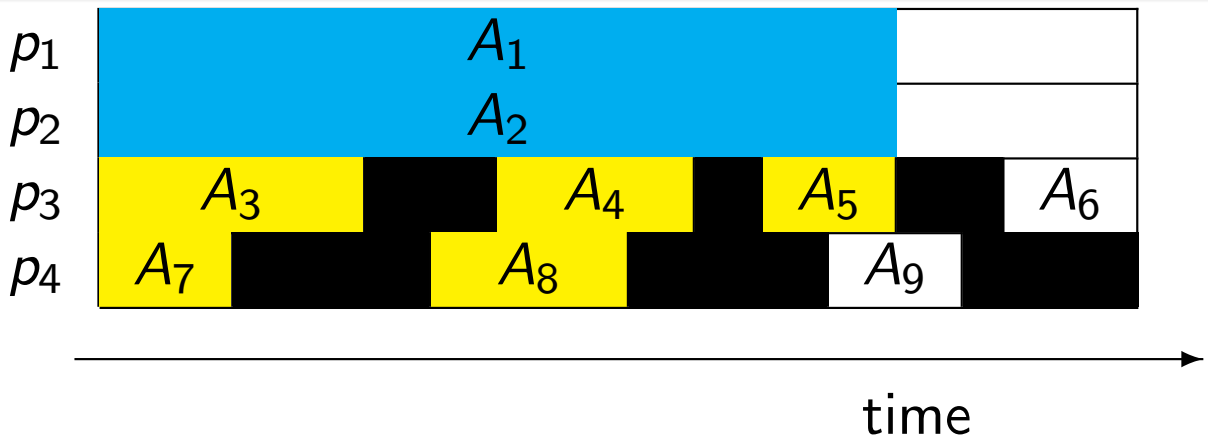
- Sort availability by non-decreasing end date of following unavailability.
- Fill each availability until the non-scheduled tasks fit into the free processors (*i.e.*, with no unavailability).



Greedy Strategy

Principal Steps

- Sort availability by non-decreasing end date of following unavailability.
- Fill each availability until the non-scheduled tasks fit into the free processors (*i.e.*, with no unavailability).



Dual Approximation Strategy

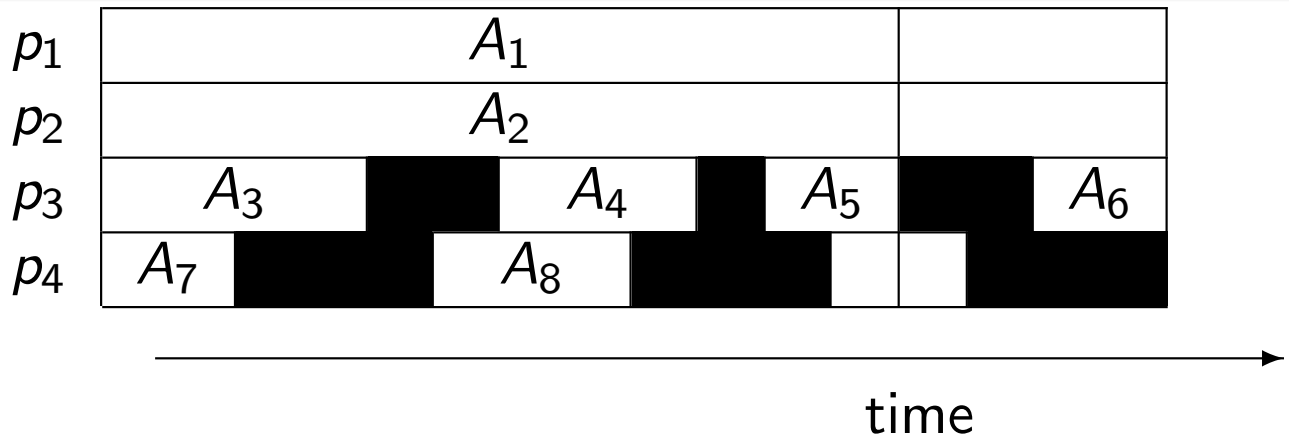
Principal Steps [Hochbaum, Shmoys, 1987]

- Schedule all tasks within a given horizon (several horizons are tested by dichotomy to obtain the minimum horizon).
- Fill each availability sorted by non-decreasing size.

Dual Approximation Strategy

Principal Steps [Hochbaum, Shmoys, 1987]

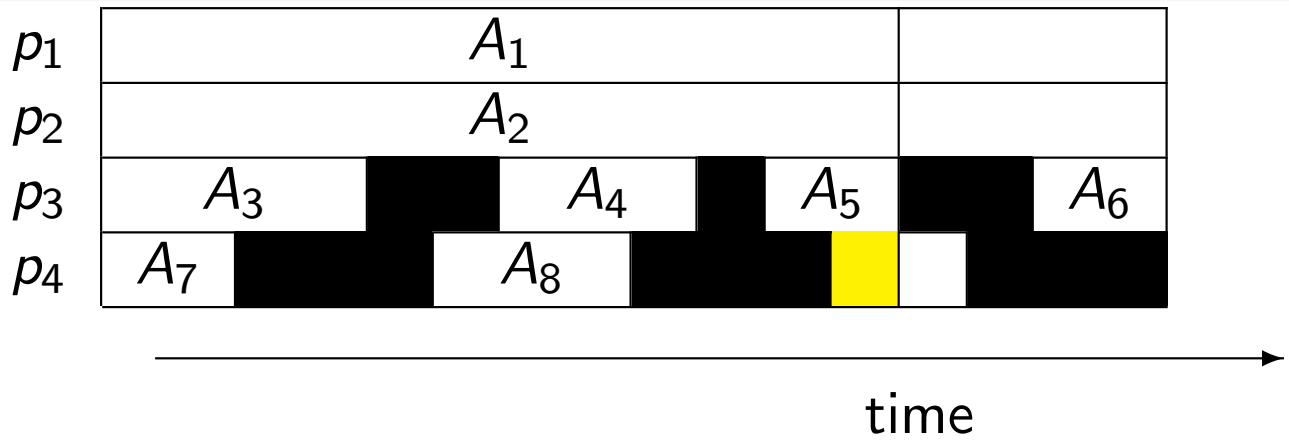
- Schedule all tasks within a given horizon (several horizons are tested by dichotomy to obtain the minimum horizon).
- Fill each availability sorted by non-decreasing size.



Dual Approximation Strategy

Principal Steps [Hochbaum, Shmoys, 1987]

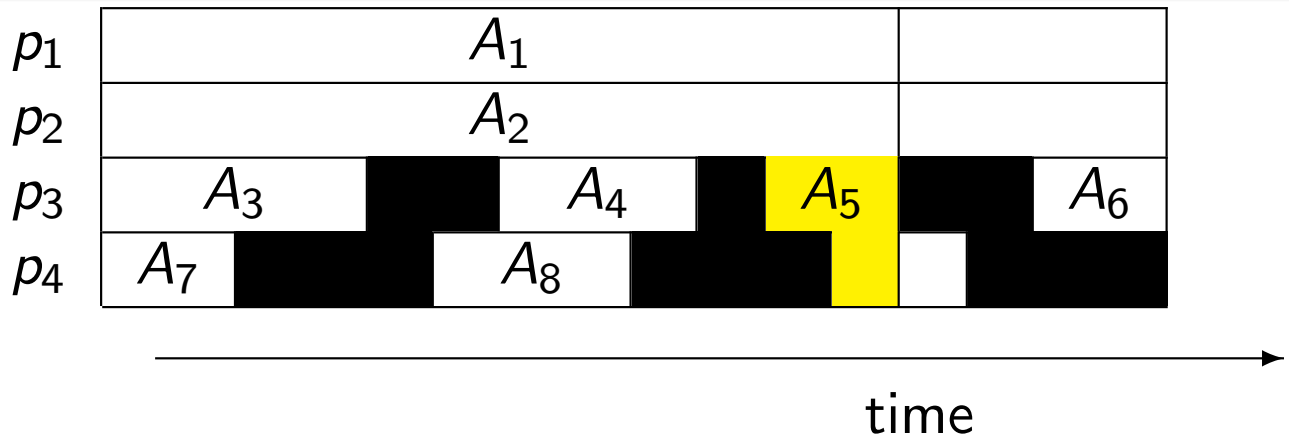
- Schedule all tasks within a given horizon (several horizons are tested by dichotomy to obtain the minimum horizon).
- Fill each availability sorted by non-decreasing size.



Dual Approximation Strategy

Principal Steps [Hochbaum, Shmoys, 1987]

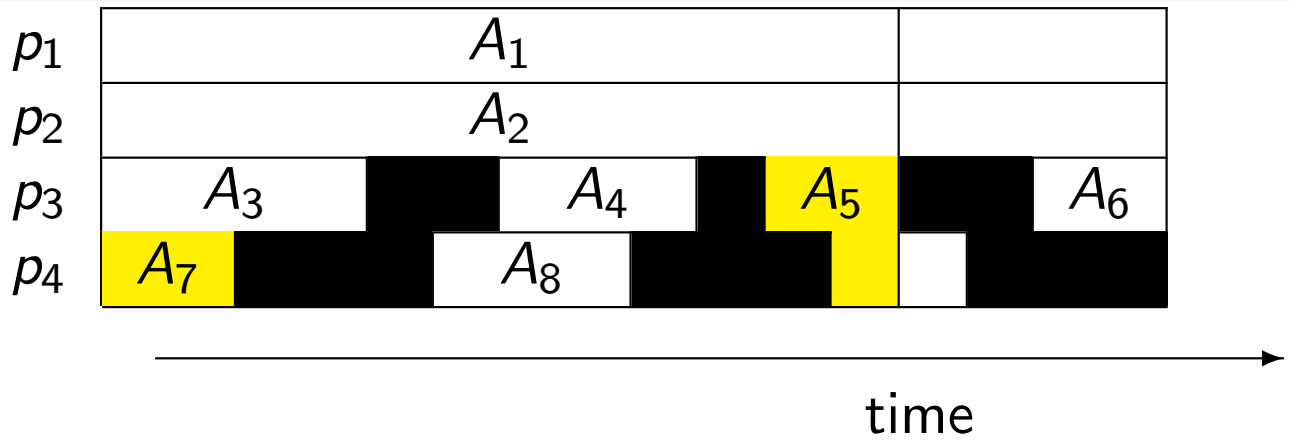
- Schedule all tasks within a given horizon (several horizons are tested by dichotomy to obtain the minimum horizon).
- Fill each availability sorted by non-decreasing size.



Dual Approximation Strategy

Principal Steps [Hochbaum, Shmoys, 1987]

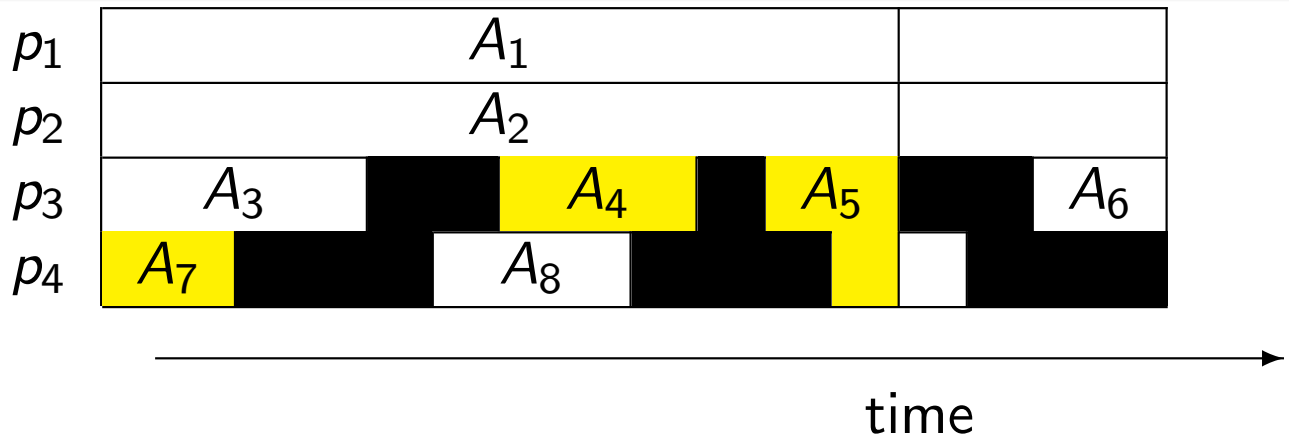
- Schedule all tasks within a given horizon (several horizons are tested by dichotomy to obtain the minimum horizon).
- Fill each availability sorted by non-decreasing size.



Dual Approximation Strategy

Principal Steps [Hochbaum, Shmoys, 1987]

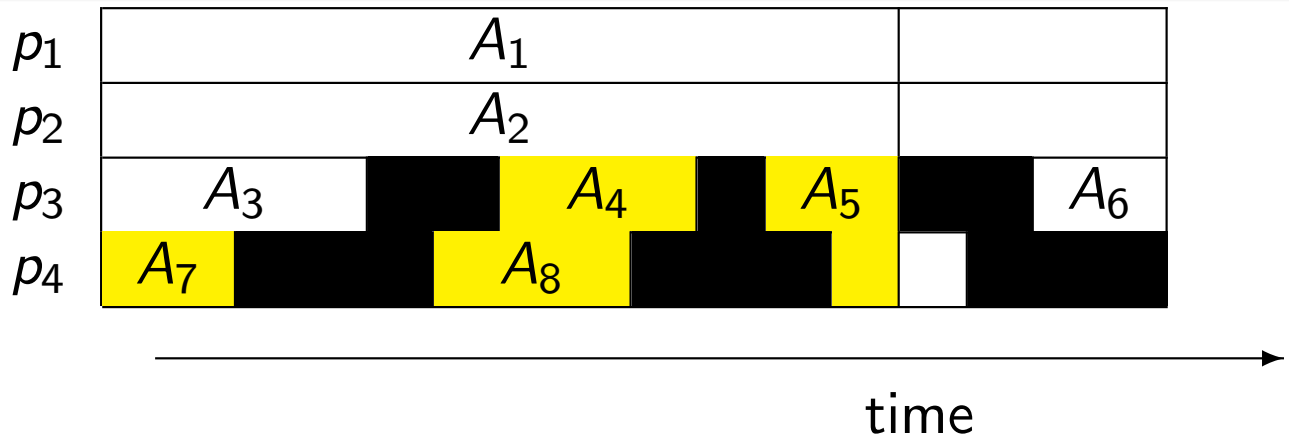
- Schedule all tasks within a given horizon (several horizons are tested by dichotomy to obtain the minimum horizon).
- Fill each availability sorted by non-decreasing size.



Dual Approximation Strategy

Principal Steps [Hochbaum, Shmoys, 1987]

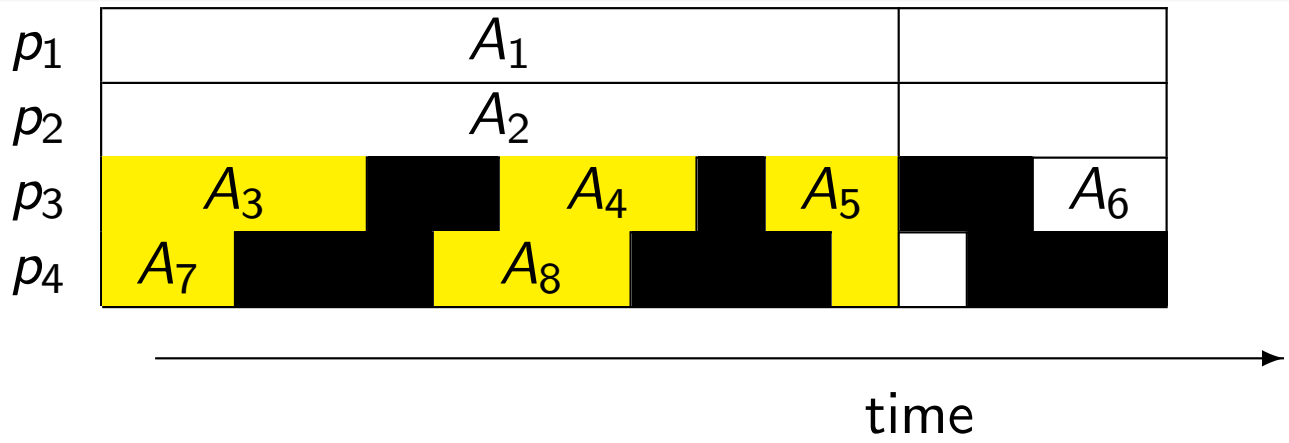
- Schedule all tasks within a given horizon (several horizons are tested by dichotomy to obtain the minimum horizon).
- Fill each availability sorted by non-decreasing size.



Dual Approximation Strategy

Principal Steps [Hochbaum, Shmoys, 1987]

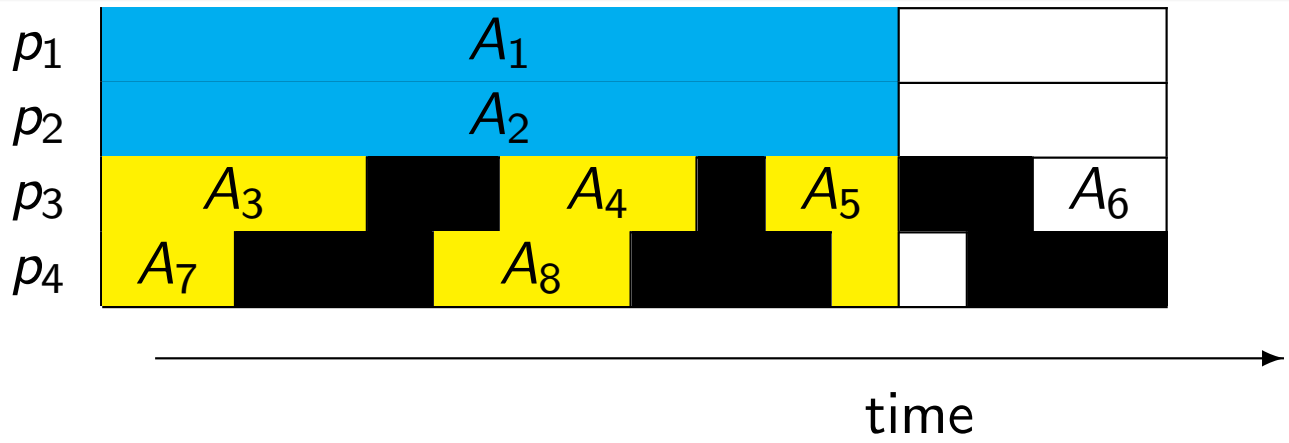
- Schedule all tasks within a given horizon (several horizons are tested by dichotomy to obtain the minimum horizon).
- Fill each availability sorted by non-decreasing size.



Dual Approximation Strategy

Principal Steps [Hochbaum, Shmoys, 1987]

- Schedule all tasks within a given horizon (several horizons are tested by dichotomy to obtain the minimum horizon).
- Fill each availability sorted by non-decreasing size.



Filling Heuristics

Filling Availabilities

Each availability is considered sequentially:

First Fit Decreasing Schedule from the largest task to the shortest.

First Fit Increasing Schedule from the shortest task to the largest.

Filling Free Processors

A set of availabilities are considered concurrently: LPT, SPT, FFD, FFI.

Filling Heuristics

Filling Availabilities

Each availability is considered sequentially:

First Fit Decreasing Schedule from the largest task to the shortest.

First Fit Increasing Schedule from the shortest task to the largest.

Filling Free Processors

A set of availabilities are considered concurrently: LPT, SPT, FFD, FFI.

Dynamic Programming

Main steps

1. Sort elements by non-decreasing size.
2. Solve the related knapsack problem without the slack constraint.
3. Add the slack constraint and use the previous results.

```
for i in 1:n
  for j in 1:C
    profit[i][j] = max(profit[i - 1][j],
                       profit[i - 1][j - s_i] + p_i)

for i in 1:n
  for j in 1:C
    profitSlack[i][j]
        = max(profitSlack[i - 1][j],
              profit[i - 1][j - s_i - β × s_i] + p_i)
```

Dynamic Programming

Main steps

1. Sort elements by non-decreasing size.
2. Solve the related knapsack problem without the slack constraint.
3. Add the slack constraint and use the previous results.

```
for i in 1:n
  for j in 1:C
    profit[i][j] = max(profit[i - 1][j],
                       profit[i - 1][j - s_i] + p_i)
```

```
for i in 1:n
  for j in 1:C
    profitSlack[i][j]
          = max(profitSlack[i - 1][j],
                profit[i - 1][j - s_i - β × s_i] + p_i)
```

Dynamic Programming

Main steps

1. Sort elements by non-decreasing size.
2. Solve the related knapsack problem without the slack constraint.
3. Add the slack constraint and use the previous results.

```
for i in 1:n
  for j in 1:C
    profit[i][j] = max(profit[i - 1][j],
                       profit[i - 1][j - s_i] + p_i)

for i in 1:n
  for j in 1:C
    profitSlack[i][j]
          = max(profitSlack[i - 1][j],
                profit[i - 1][j - s_i - β × s_i] + p_i)
```

Summary

Scheme	Availabilities	Free processors
Greedy	FFD/FFI/DP	LPT/SPT
Dual approximation	all	all

Outline

Settings

Traces

- Availability from SETI@home.
- Workload from Docking@home.
- GAPS: greedy strategy with FFD for the unavailabilities and LPT for the free processors.

Protocol

- 100 instances are generated from the trace (workload and availability).
- For each instances, a schedule is obtained.
- The makespan ratio is measured (baseline makespan over a lower bound).
- The schedule is disturbed 30 times (with each unavailability starting early such that every possible start dates are equiprobable) and the median stability is selected.

Settings

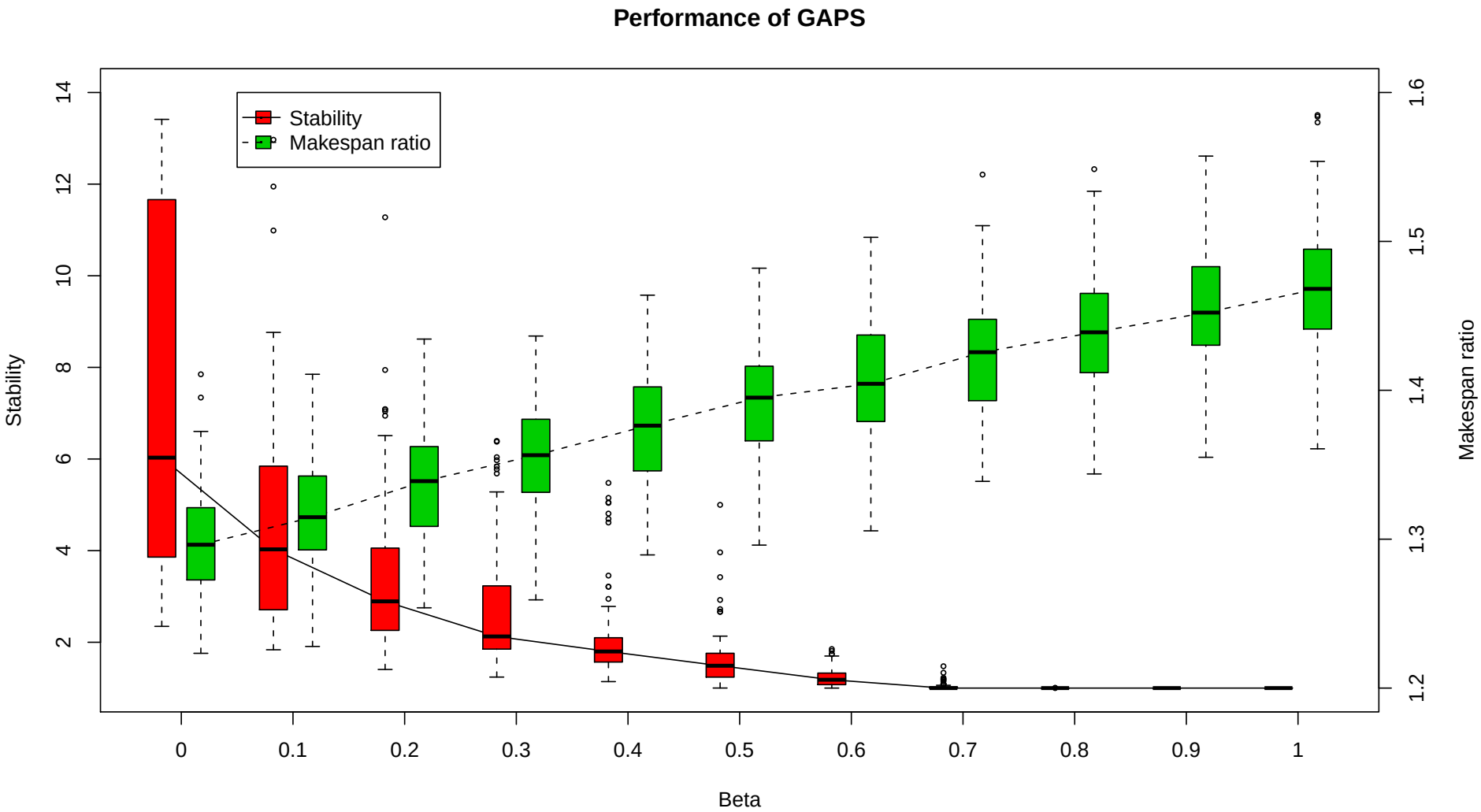
Traces

- Availability from SETI@home.
- Workload from Docking@home.
- GAPS: greedy strategy with FFD for the unavailabilities and LPT for the free processors.

Protocol

- 100 instances are generated from the trace (workload and availability).
- For each instances, a schedule is obtained.
- The makespan ratio is measured (baseline makespan over a lower bound).
- The schedule is disturbed 30 times (with each unavailability starting early such that every possible start dates are equiprobable) and the median stability is selected.

Effect of beta



Outline

Conclusion and Future Directions

Main contributions

- Study the stability when scheduling statically in environments with uncertainty on the unavailabilities.
- Propose a condition with bound on the stability.
- Several strategies (schemes and filling heuristics).
- Empirical validation of one approach.

Perspective

- Adapt a PTAS (from Hochbaum and Shmoys, 1988).
- Study hypotheses (worst case analysis, deterministic u_i^k).
- Compare with a checkpointing-based mechanism.
- Take into account fairness between multiple projects.

Conclusion and Future Directions

Main contributions

- Study the stability when scheduling statically in environments with uncertainty on the unavailabilities.
- Propose a condition with bound on the stability.
- Several strategies (schemes and filling heuristics).
- Empirical validation of one approach.

Perspective

- Adapt a PTAS (from Hochbaum and Shmoys, 1988).
- Study hypotheses (worst case analysis, deterministic u_i^k).
- Compare with a checkpointing-based mechanism.
- Take into account fairness between multiple projects.