Scheduling malleable jobs to minimize the Mean Flow Time

Yann Hendel[a] Wieslaw Kubiak[b] **Ruslan Sadykov**[a]

[a]LIX, Ecole Polytechnique, France

[b]Memorial University, Newfoundland, Canada

Bordeaux
January 31, 2008

Outline

1. Introduction: parallel jobs scheduling
2. Scheduling malleable jobs on 2 machines to minimize the Mean Flow Time
    1. A set of dominant schedules: π-schedules
    2. A polynomial dynamic programming algorithm
    3. Proof of the dominance of the π-schedules
3. Perspectives: the general case with m machines

Scheduling parallel jobs

Classic scheduling

A **classic job** can be executed on at most one processor (machine) at the same time.

Parallel scheduling

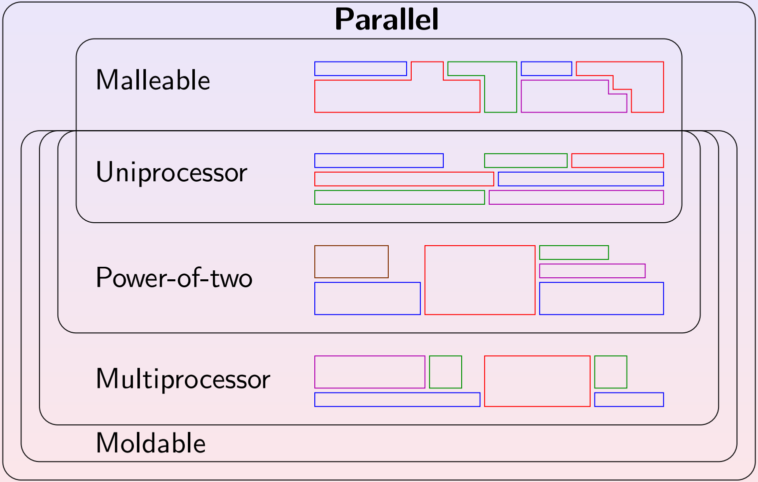
A **parallel job** can be executed on more than one processor at the same time.

δ_j — upper bound on the number of processors that may be used by job J_j.

Applications

- **Parallel computer applications**
- Reliable computing
- Bandwidth allocation
- Manufacturing
  - Printed Circuit Boards
  - Textile
  - ...

Types of parallel jobs



Cost of parallelism

Processing speed

The relation between the processing time p_j of job J_j and the number of assigned processors q:

- $p_j(q) = p_j/q$ (J_j is work preserving, no parallelism cost)
- $p_j(q) > p_j/q$ (parallelism costs)
  - $p_j(q) = f(q)$ (particular continuous function)
  - $p_j(q)$ is an arbitrary discrete function of q.

Outline

1. Introduction: parallel jobs scheduling
2. Scheduling malleable jobs on 2 machines to minimize the Mean Flow Time
   1. A set of dominant schedules: π-schedules
   2. A polynomial dynamic programming algorithm
   3. Proof of the dominance of the π-schedules
3. Perspectives: the general case with m machines

Problem definition

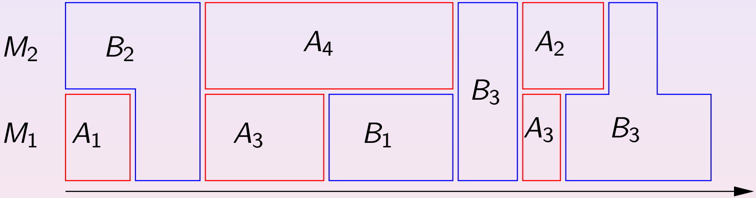
Notations

- 2 identical parallel machines: M_1 and M_2
- 2 kinds of jobs:
  - A set $A = \{A_1, A_2, \ldots, A_{n_A}\}$ of preemptive jobs ($\delta_j^A = 1$)
  - A set $B = \{B_1, B_2, \ldots, B_{n_B}\}$ of malleable jobs ($\delta_i^B = 2$)
- C_j^A and C_i^B are the completion times of jobs A_j and B_i
- p_j^A is the processing time of job A_j
- p_i^B is the processing time of job B_i, $p_i^B(2) = p_i^B/2$
- The objective is to minimize $\sum_{j=1}^{n_A} C_j^A + \sum_{i=1}^{n_B} C_i^B$

$\alpha|\beta|\gamma$ notation

$$P2 \mid var,\ p_j(q) = p_j/q,\ \delta_j \mid \sum C_j$$

An example



Outline

1. Introduction: parallel jobs scheduling
2. Scheduling malleable jobs on 2 machines to minimize the Mean Flow Time
   1. A set of dominant schedules: π-schedules
   2. A polynomial dynamic programming algorithm
   3. Proof of the dominance of the π-schedules
3. Perspectives: the general case with m machines

A set of dominant schedules

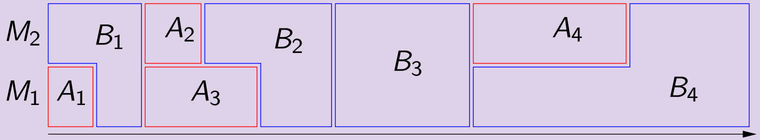
Definition

We say that a schedule σ is a π-schedule if it has the following properties:

1. the jobs in A are processed, non-preemptively, in SPT (Shortest Processing Time) order,
2. the jobs in B are processed, non-preemptively, in SPT order,
3. the jobs in B is completed on 2 machines
4. for every job B_i, there exists at most one job A_j such that $S_i^B < C_j^A \leq C_i^B$.

π-schedules

Example



Properties

- A π-schedule is fully described by a sequence of jobs.
- Completion time of a job B_i in a π-schedule depends only on its position in the corresponding sequence.
- Completion time of a job A_j in a π-schedule depends only on its position and on the position of the job in B which is the last before A_j in the corresponding sequence.

Outline

1. Introduction: parallel jobs scheduling
2. Scheduling malleable jobs on 2 machines to minimize the Mean Flow Time
    1. A set of dominant schedules: π-schedules
    2. A polynomial dynamic programming algorithm
    3. Proof of the dominance of the π-schedules
3. Perspectives: the general case with m machines

A dynamic programming algorithm

- (i, j, k) denotes the subproblem of scheduling jobs $A_1, \ldots, A_j$ and $B_1, \ldots, B_i$ such that A_k is the last job in A such that $C_k^A < C_i^B$.
- $f(i, j, k)$ denotes the optimal value of the subproblem (i, j, k).
- Since we build π-schedules, there are two possible transitions from the state (i, j, k):
  - $(i + 1, j, j)$ (we add B_{i+1} at the end of the schedule)
  - $(i, j + 1, k)$ (we add A_{j+1} at the end of the schedule)

A dynamic programming algorithm

1. $f(0, 0, 0) = 0$
2. $\forall i \in \{0, \dots, n_B\}$, $\forall j \in \{0, \dots, n_A\}$, $\forall k \in \{0, \dots, j\}$ do:
   make transitions from state (i, j, k)
   to states $(i + 1, j, j)$ and $(i, j + 1, k)$
3. return $\min_{0 \leq k \leq n_A} f(n_A, n_B, k)$

Theorem

DP finds an optimal π-schedule.

Theorem

The complexity of DP is in $O(n_A^2 n_B)$.

Outline

1. Introduction: parallel jobs scheduling
2. Scheduling malleable jobs on 2 machines to minimize the Mean Flow Time
    1. A set of dominant schedules: π-schedules
    2. A polynomial dynamic programming algorithm
    3. Proof of the dominance of the π-schedules
3. Perspectives: the general case with m machines

First dominance property

Lemma

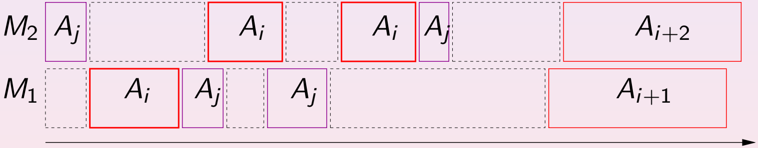
There exists an optimal schedule such that

1. $C_j^A \leq C_{j+1}^A, \forall 1 \leq j < n_A$
2. A_j *is not preempted*, $\forall 1 \leq j \leq n_A$
3. $C_i^B \leq S_{i+1}^B, \forall 1 \leq i < n_B$
4. *On each of the 2 machines,* B_i *is not preempted*, $\forall 1 \leq i \leq n_B$

Properties on A

We prove that

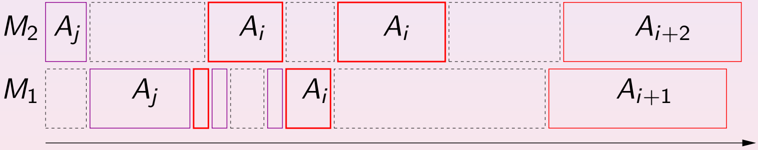
1. $C_j^A \leq C_{j+1}^A, \forall 1 \leq j < n_A$
2. A_j is not preempted, $\forall 1 \leq j \leq n_A$



Properties on A

We prove that

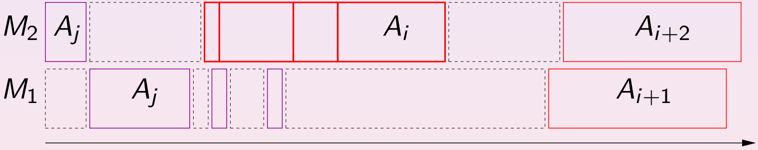
1. $C_j^A \leq C_{j+1}^A, \forall 1 \leq j < n_A$
2. A_j is not preempted, $\forall 1 \leq j \leq n_A$



Properties on A

We prove that

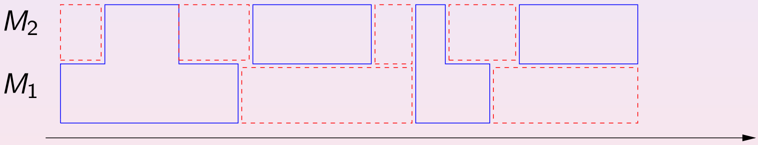
1. $C_j^A \leq C_{j+1}^A, \ \forall 1 \leq j < n_A$
2. A_j is not preempted, $\forall 1 \leq j \leq n_A$



Properties on B

We prove that

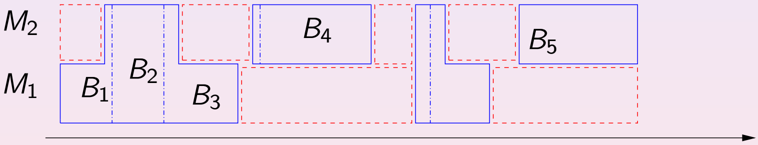
$C_i^B \leq S_{i+1}^B, \forall 1 \leq i < n_B$



Properties on B

We prove that

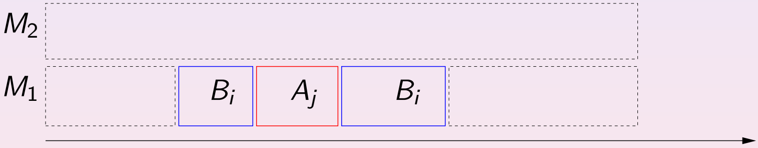
$C_i^B \leq S_{i+1}^B, \forall 1 \leq i < n_B$



Properties on B

We prove that

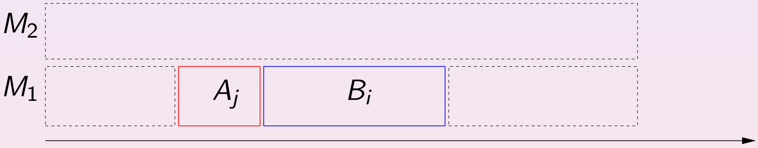
On each of the 2 machines, B_i is not preempted, $\forall 1 \leq i \leq n_B$



Properties on B

We prove that

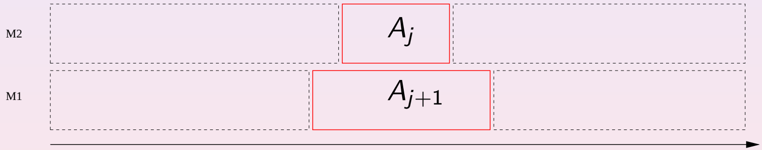
On each of the 2 machines, B_i is not preempted, $\forall 1 \leq i \leq n_B$



A: bad case!

Remains to prove

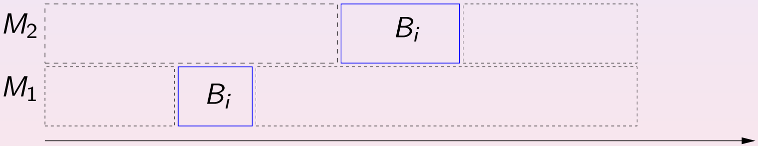
There exists on optimal schedule in which the jobs in A are started and completed in SPT order.



B: bad case!

Remains to prove

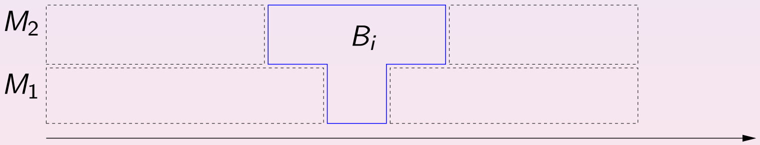
There exists on optimal schedule in which the jobs in B are completed on 2 machines.



B: Tetris, bad case!

Remains to prove

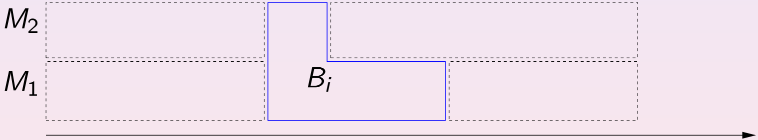
There exists on optimal schedule in which the jobs in B are completed on 2 machines.



B: Tetris, bad case!

Remains to prove

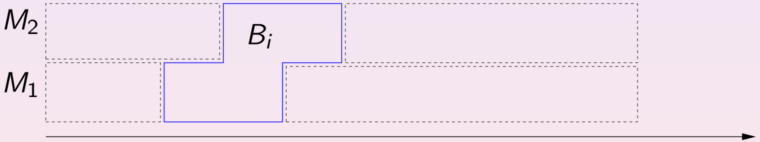
There exists on optimal schedule in which the jobs in B are completed on 2 machines.



B: Tetris, bad case!

Remains to prove

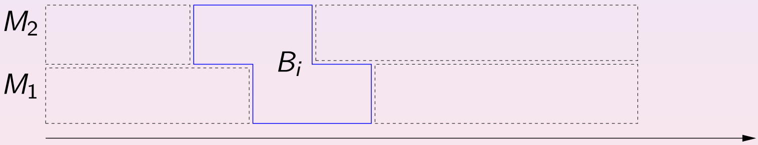
There exists on optimal schedule in which the jobs in B are completed on 2 machines.



B: Tetris, bad case!

Remains to prove

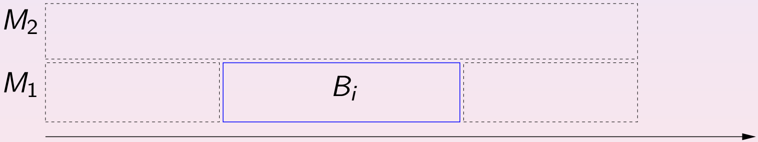
There exists on optimal schedule in which the jobs in B are completed on 2 machines.



B: Tetris, bad case!

Remains to prove

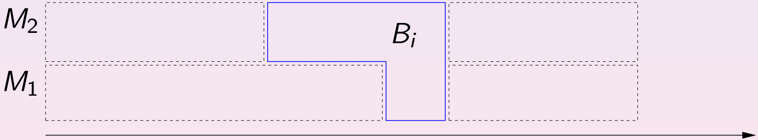
There exists on optimal schedule in which the jobs in B are completed on 2 machines.



B: Tetris, good case!

Remains to prove

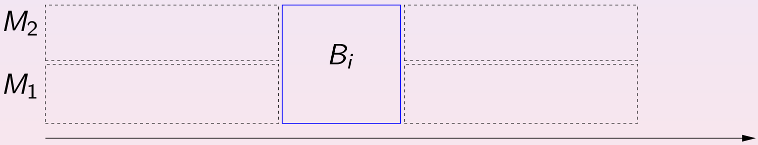
There exists on optimal schedule in which the jobs in B are completed on 2 machines.



B: Tetris, good case!

Remains to prove

There exists on optimal schedule in which the jobs in B are completed on 2 machines.

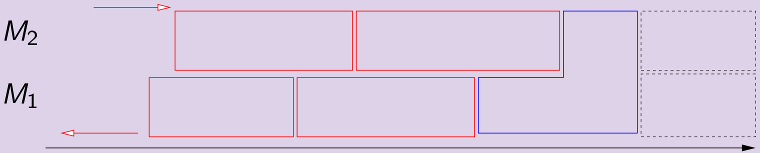


Auxiliary observation

Claim

Consider a partial π-schedule in which the first job on M_1 is started not later than the first job on M_2. Then, if we decrease the availability of M_1 by δ and increase the availability of M_2 by δ, the cost of the schedule does not increase.

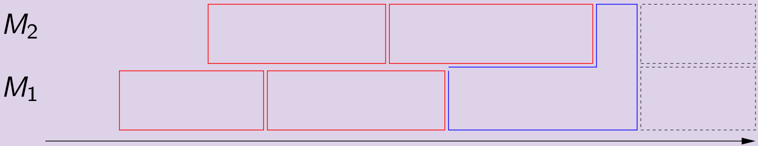
Proof. Case 1



Claim

Consider a partial π-schedule in which the first job on M_1 is started not later than the first job on M_2. Then, if we decrease the availability of M_1 by δ and increase the availability of M_2 by δ, the cost of the schedule does not increase.

Proof. Case 1

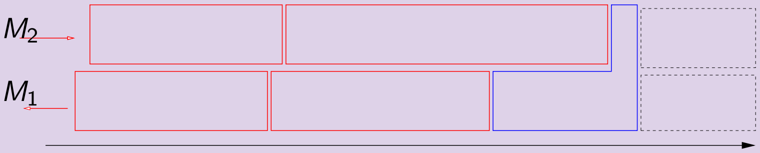


Auxiliary observation

Claim

Consider a partial π-schedule in which the first job on M_1 is started not later than the first job on M_2. Then, if we decrease the availability of M_1 by δ and increase the availability of M_2 by δ, the cost of the schedule does not increase.

Proof. Case 2

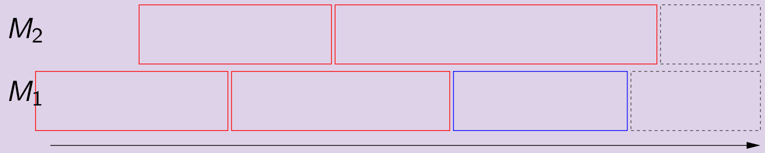


Auxiliary observation

Claim

Consider a partial π-schedule in which the first job on M_1 is started not later than the first job on M_2. Then, if we decrease the availability of M_1 by δ and increase the availability of M_2 by δ, the cost of the schedule does not increase.

Proof. Case 2



The main theorem

Theorem

There exists an optimal π-schedule.

Proof

It is possible to transform an optimal schedule ϵ satisfying the Lemma and which is not a π-schedule into another optimal schedule such that

- either the completion time of at least one job in B is strictly decreased while the completion times of other jobs in B are not increased.
- or the number of jobs in A processed in the SPT order is increased, and the completion times of all jobs in B are not increased.

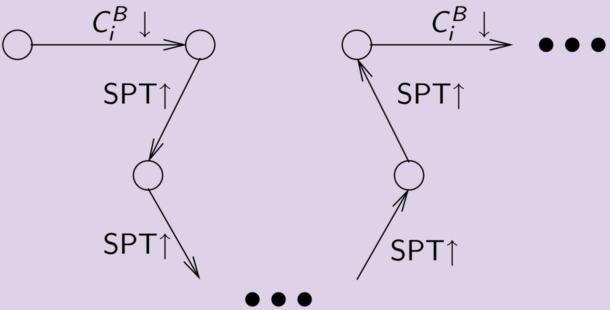
Applying this transformation a finite number of times, we can obtain an optimal π-schedule.

The main theorem

Theorem

There exists an optimal π-schedule.

Proof



Proof of the main theorem

Partial schedules of ϵ

$$\mathcal{A}_i = \{A_j : \ j \in N, \ C_i^B \leq C_j^A < C_{i+1}^B\}, \ 0 \leq i \leq m.$$

A partial schedule $\epsilon(i)$ contains jobs $\mathcal{A}_i \cup \cdots \cup \mathcal{A}_m \cup \{B_i, \ldots, B_m\}$.
$\exists i$: $\epsilon(i)$ is not a π-schedule, $\epsilon(i+1)$ is a π-schedule.

Proof of the main theorem

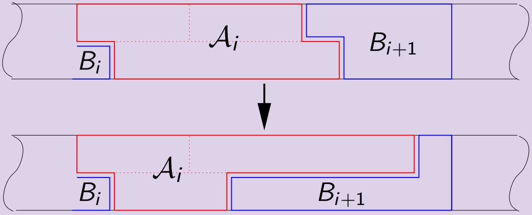
Partial schedules of ϵ

$$\mathcal{A}_i = \{A_j : \; j \in N, \; C_i^B \le C_j^A < C_{i+1}^B\}, \; 0 \le i \le m.$$

A partial schedule $\epsilon(i)$ contains jobs $\mathcal{A}_i \cup \cdots \cup \mathcal{A}_m \cup \{B_i, \ldots, B_m\}$.
$\exists i$: $\epsilon(i)$ is not a π-schedule, $\epsilon(i+1)$ is a π-schedule.

Case 1.1

$\mathcal{A}_i$ is not in SPT order



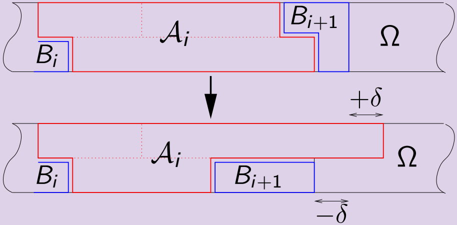
Partial schedules of ϵ

$$\mathcal{A}_i = \{A_j : j \in N,\ C_i^B \le C_j^A < C_{i+1}^B\},\ 0 \le i \le m.$$

A partial schedule $\epsilon(i)$ contains jobs $\mathcal{A}_i \cup \cdots \cup \mathcal{A}_m \cup \{B_i, \ldots, B_m\}$.
$\exists i$: $\epsilon(i)$ is not a π-schedule, $\epsilon(i+1)$ is a π-schedule.

Case 1.2

$\mathcal{A}_i$ is not in SPT order



Proof of the main theorem

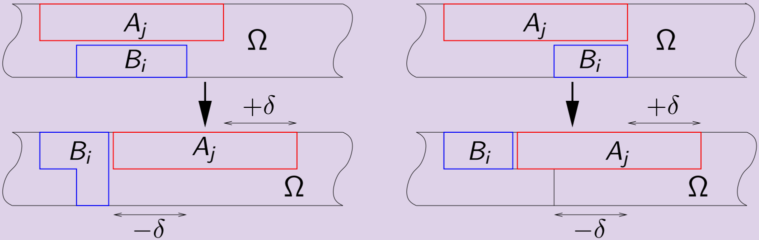
Partial schedules of ϵ

$$\mathcal{A}_i = \{A_j : \ j \in N, \ C_i^B \leq C_j^A < C_{i+1}^B\}, \ 0 \leq i \leq m.$$

A partial schedule $\epsilon(i)$ contains jobs $\mathcal{A}_i \cup \cdots \cup \mathcal{A}_m \cup \{B_i, \ldots, B_m\}$.
$\exists i$: $\epsilon(i)$ is not a π-schedule, $\epsilon(i+1)$ is a π-schedule.

Case 2

$\mathcal{A}_i$ is in SPT order, B_i is completed on one machine



Proof of the main theorem

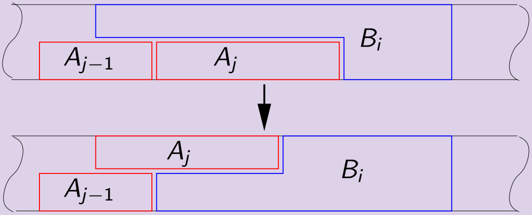
Partial schedules of ϵ

$$\mathcal{A}_i = \{A_j : \ j \in N, \ C_i^B \le C_j^A < C_{i+1}^B\}, \ 0 \le i \le m.$$

A partial schedule $\epsilon(i)$ contains jobs $\mathcal{A}_i \cup \cdots \cup \mathcal{A}_m \cup \{B_i, \ldots, B_m\}$.
$\exists i$: $\epsilon(i)$ is not a π-schedule, $\epsilon(i+1)$ is a π-schedule.

Case 3

$\mathcal{A}_i$ is in SPT order, B_i is completed on two machines



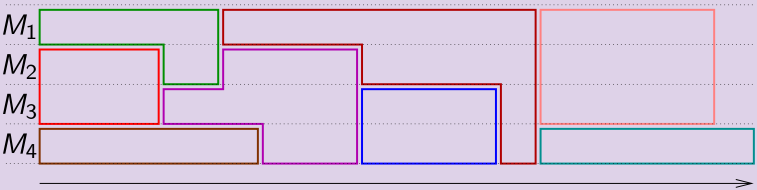
1. Introduction: parallel jobs scheduling
2. Scheduling malleable jobs on 2 machines to minimize the Mean Flow Time
    1. A set of dominant schedules: π-schedules
    2. A polynomial dynamic programming algorithm
    3. Proof of the dominance of the π-schedules
3. Perspectives: the general case with m machines

The general case with m machines: "Ascending property"

Theorem

For each instance of the problem $P \mid var,\ p_j(q) = p_j/q,\ \delta_j \mid \sum w_j C_j$ there exists an optimal schedule in which once a processor is assigned to a job, it remains assigned to this job until the job is completed (the number of processors assigned to a job cannot decrease over time while the job is not completed)

Example



The case with 3 machines

The simplest open case

$$P3 \mid var,\ p_j(q) = p_j/q,\ \delta_j \in \{1,3\} \mid \sum C_j$$

π-schedules

1. the jobs in A are processed, non-preemptively, in SPT order
2. the jobs in B are processed, non-preemptively, in SPT order
3. for every job B_i, there exists at most one job A_j such that $S_i^B < C_j^A \leq C_i^B$
4. the jobs of B is completed on 3 machines

The case with 3 machines

The simplest open case

$$P3 \mid var,\ p_j(q) = p_j/q,\ \delta_j \in \{1,3\} \mid \sum C_j$$

π-schedules

1. the jobs in A are processed, non-preemptively, in SPT order
2. the jobs in B are processed, non-preemptively, in SPT order
3. for every job B_i, there exists at most one job A_j such that $S_i^B < C_j^A \le C_i^B$
4. the jobs of B is completed on 3 machines

The case with 3 machines

The simplest open case

$$P3 \mid var,\ p_j(q) = p_j/q,\ \delta_j \in \{1,3\} \mid \sum C_j$$

π-schedules

1. the jobs in A are processed, non-preemptively, in SPT order
2. the jobs in B are processed, non-preemptively, in SPT order
3. for every job B_i, there exists at most one job A_j such that $S_i^B < C_j^A \leq C_i^B$
4. the jobs of B is completed on 3 machines

The case with 3 machines

The simplest open case

$$P3 \mid var,\ p_j(q) = p_j/q,\ \delta_j \in \{1,3\} \mid \sum C_j$$

π-schedules

1. the jobs in A are processed, non-preemptively, in SPT order
2. the jobs in B are processed, non-preemptively, in SPT order
3. for every job B_i, there exists at most one job A_j such that $S_i^B < C_j^A \leq C_i^B$
4. the jobs of B is completed on 3 machines

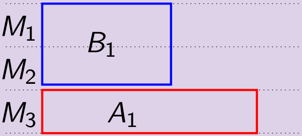
The case with 3 machines

The simplest open case

$$P3 \mid var,\ p_j(q) = p_j/q,\ \delta_j \in \{1,3\} \mid \sum C_j$$

π-schedules

1. the jobs in A are processed, non-preemptively, in SPT order
2. the jobs in B are processed, non-preemptively, in SPT order
3. for every job B_i, there exists at most one job A_j such that $S_i^B < C_j^A \leq C_i^B$
4. the jobs of B is completed on 3 machines



The case with 3 machines

The simplest open case

$$P3 \mid var,\ p_j(q) = p_j/q,\ \delta_j \in \{1, 3\} \mid \sum C_j$$

π-schedules

1. the jobs in A are processed, non-preemptively, in SPT order
2. the jobs in B are processed, non-preemptively, in SPT order
3. for every job B_i, there exists at most one job A_j such that $S_i^B < C_j^A \leq C_i^B$
4. the jobs of B is completed on the 3 machines

   The completion time of a job in a π-schedule depends now on the positions of all the preceding jobs in the corresponding sequence.

The end of the talk

Questions?