# Single machine scheduling with delivery dates and cumulative payoffs

Yasmina Seddik, Christophe Gonzales, Safia Kedad-Sidhoum

Laboratoire d'Informatique de Paris 6, Université Pierre et Marie Curie

Séminaire GOThA
15 mars 2012

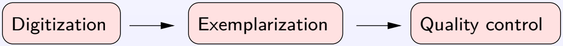
# Outline

1. Introduction
2. Complexity
   - The two delivery dates problem
   - General problem
   - Polynomial cases
3. Solving the two delivery dates problem
4. Branch and Bound

# The industrial problem

- The Bibliothèque Nationale de France (BNF) is digitizing its entire collection.
- The digitization firm :
  - receives the books every 6 weeks,
  - handles each kind of book in a specific way (specific processing time)
  - wishes to provide at each delivery date the corresponding demanded quantity of digitized books, set by BNF

Digitization process

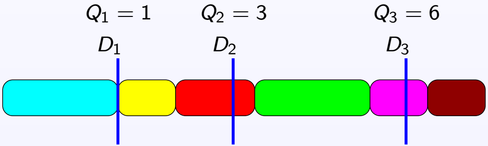



*Flowshop*

# Problem definition (1/3) : the parameters

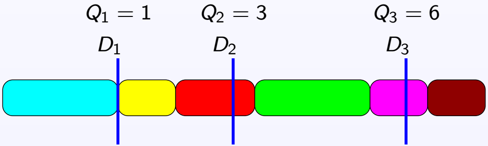
- $N$ jobs $J_1, \ldots, J_N$
- a job $J_i$ has :
  - a processing time $p_i$
  - a release date $r_i$
- $K$ delivery dates $D_1, \ldots, D_K$

# Problem definition (2/3) : the cumulative payoffs



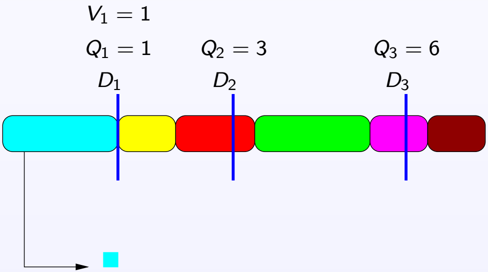
# Problem definition (2/3) : the cumulative payoffs

$V_k$ : number of jobs executed before $D_k$



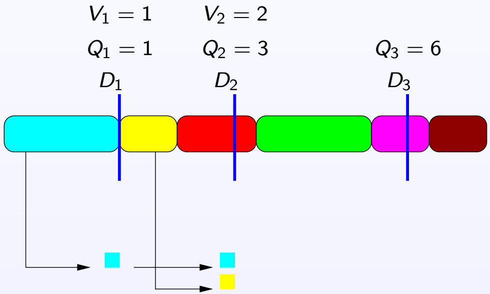
# Problem definition (2/3) : the cumulative payoffs

$V_k$ : number of jobs executed before $D_k$



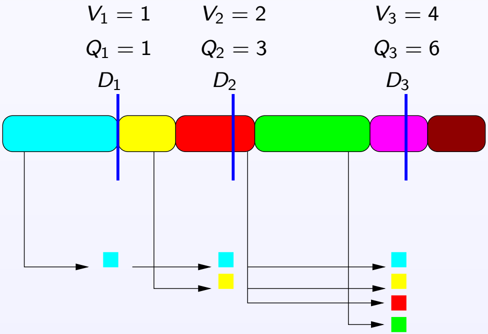
# Problem definition (2/3) : the cumulative payoffs

$V_k$ : number of jobs executed before $D_k$



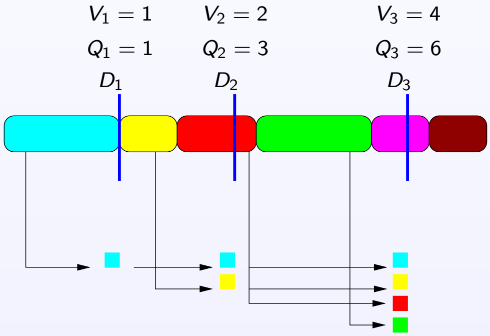
# Problem definition (2/3) : the cumulative payoffs

$V_k$ : number of jobs executed before $D_k$



# Problem definition (2/3) : the cumulative payoffs

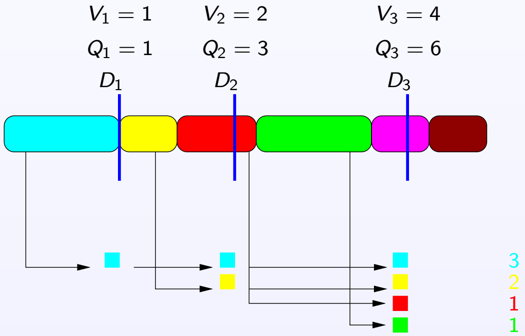
$V_k$ : number of jobs executed before $D_k$




Maximizing $\sum_{k=1}^{K} V_k - Q_k \rightarrow \max \sum_{k=1}^{K} V_k$

# Problem definition (2/3) : the cumulative payoffs

$V_k$ : number of jobs executed before $D_k$




Maximizing $\sum_{k=1}^{K} V_k - Q_k \rightarrow \max \sum_{k=1}^{K} V_k$

# Problem definition (3/3)

$$1|r_i|\sum_{k=1}^{K} V_k$$

**State of the art**

Without release dates:

- Unrelated parallel machines: Detienne et al., JOS (11)
- Single machine: Detienne et al.,C&OR (11) ; Tseng et al. (10)
- Single machine, common breakpoints: Yang (09)
- Single and parallel machines: Janiak and Krysiak, JOS (07)

With release dates:

- Unrelated parallel machines: Detienne et al., JOS (11)
- Single machine: Sahin and Ahuja (11)

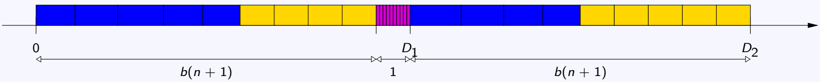
Online arriving jobs with rescheduling:

- Parallel machines: Curry and Peters, IJPR (05)

1. Introduction
2. Complexity
   - The two delivery dates problem
   - General problem
   - Polynomial cases
3. Solving the two delivery dates problem
4. Branch and Bound

# The two delivery dates problem

- $K = 2$
- $1|r_i|V_1 + V_2$ is weakly NP-hard.
- Reduction from Partition $(A = \{a_1, \ldots, a_n\}, \sum_{a_i \in A} a_i = 2b)$




| | $p_i$ | $r_i$ |
|---|---|---|
| $n$ jobs $\tilde{J}_i$ | $b + a_i$ | 0 |
| $n$ jobs $\widehat{J}_i$ | $b$ | 0 |
| $n$ jobs $\overline{J}_i$ | $\frac{1}{n}$ | $b(n+1)$ |

- $V = 5n$

# General problem (1/2)

- The general problem is strongly NP-hard
- Reduction from 3-Partition

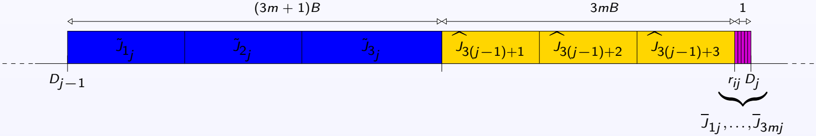
3-Partition :

- $A = \{a_1, a_2, \ldots, a_{3m}\}$ s.t. $\sum_{i=1}^{3m} a_i = mB$ and $B/4 < a_i < B/2$ for $i = 1, \ldots, 3m$
- Does there exist a partition $< A_1, A_2, \ldots, A_m >$ s.t. $\forall i, \sum_{a \in A_i} a = B$ ?

# General problem (2/2)

Instance of $1|r_i|\sum V_k$ :

- $m$ delivery dates




| | $p_i$ | $r_i$ |
|---|---|---|
| $3m$ jobs $\tilde{J}_i$ | $mB + a_i$ | 0 |
| $3m$ jobs $\widehat{J}_i$ | $mB$ | 0 |
| $3m^2$ jobs $\bar{J}_{ij}$ | $\frac{1}{3m}$ | $D_j - 1$ |

- $V = (6+3m)m(m+1)/2$

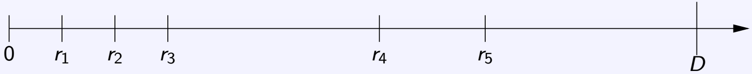
# Polynomial cases

| | |
|---|---|
| No release date | SPT ($O(NlogN)$) |
| Preemptive | SRPT ($O(NlogN)$) |
| Identical processing times | nondecreasing order of the release dates $O(NlogN)$ |

# Polynomial cases

| | |
|---|---|
| No release date | SPT ($O(NlogN)$) |
| Preemptive | SRPT ($O(NlogN)$) |
| Identical processing times | nondecreasing order of the release dates $O(NlogN)$ |

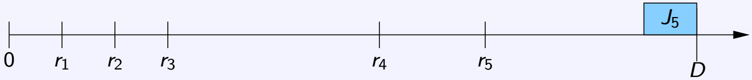
- If $\boldsymbol{K = 1}$, the problem can be solved in time $O(NlogN)$.
- Moore-Hodgson algorithm for $1||\sum U_i$ : Moore, MS (98).



# Polynomial cases

| | |
|---|---|
| No release date | SPT ($O(NlogN)$) |
| Preemptive | SRPT ($O(NlogN)$) |
| Identical processing times | nondecreasing order of the release dates $O(NlogN)$ |

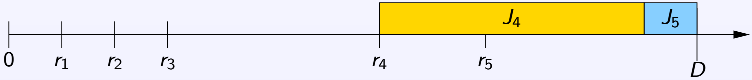
- If $\boldsymbol{K = 1}$, the problem can be solved in time $O(NlogN)$.
- Moore-Hodgson algorithm for $1||\sum U_i$ : Moore, MS (98).



# Polynomial cases

| | |
|---|---|
| No release date | SPT ($O(NlogN)$) |
| Preemptive | SRPT ($O(NlogN)$) |
| Identical processing times | nondecreasing order of the release dates $O(NlogN)$ |

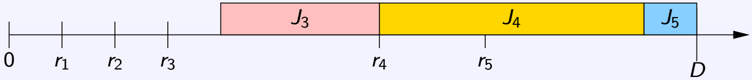
- If $\boldsymbol{K = 1}$, the problem can be solved in time $O(NlogN)$.
- Moore-Hodgson algorithm for $1||\sum U_i$ : Moore, MS (98).



# Polynomial cases

| | |
|---|---|
| No release date | SPT ($O(NlogN)$) |
| Preemptive | SRPT ($O(NlogN)$) |
| Identical processing times | nondecreasing order of the release dates $O(NlogN)$ |

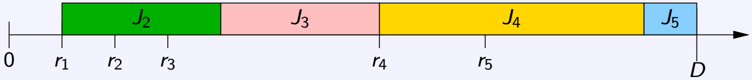
- If $\boldsymbol{K = 1}$, the problem can be solved in time $O(NlogN)$.
- Moore-Hodgson algorithm for $1||\sum U_i$ : Moore, MS (98).



# Polynomial cases

| | |
|---|---|
| No release date | SPT ($O(NlogN)$) |
| Preemptive | SRPT ($O(NlogN)$) |
| Identical processing times | nondecreasing order of the release dates $O(NlogN)$ |

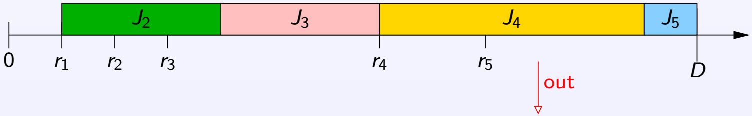
- If $\boldsymbol{K = 1}$, the problem can be solved in time $O(NlogN)$.
- Moore-Hodgson algorithm for $1||\sum U_i$ : Moore, MS (98).



# Polynomial cases

| | |
|---|---|
| No release date | SPT ($O(NlogN)$) |
| Preemptive | SRPT ($O(NlogN)$) |
| Identical processing times | nondecreasing order of the release dates $O(NlogN)$ |

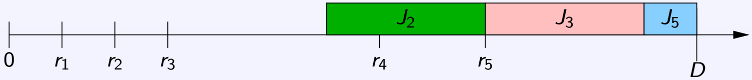
- If $K = 1$, the problem can be solved in time $O(NlogN)$.
- Moore-Hodgson algorithm for $1||\sum U_i$ : Moore, MS (98).



# Polynomial cases

| | |
|---|---|
| No release date | SPT ($O(NlogN)$) |
| Preemptive | SRPT ($O(NlogN)$) |
| Identical processing times | nondecreasing order of the release dates $O(NlogN)$ |

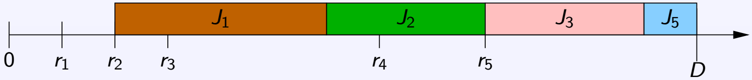
- If $\boldsymbol{K = 1}$, the problem can be solved in time $O(NlogN)$.
- Moore-Hodgson algorithm for $1||\sum U_i$ : Moore, MS (98).



# Polynomial cases

| | |
|---|---|
| No release date | SPT ($O(NlogN)$) |
| Preemptive | SRPT ($O(NlogN)$) |
| Identical processing times | nondecreasing order of the release dates $O(NlogN)$ |

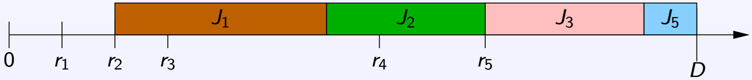
- If $\boldsymbol{K = 1}$, the problem can be solved in time $O(NlogN)$.
- Moore-Hodgson algorithm for $1||\sum U_i$ : Moore, MS (98).



# Polynomial cases

| | |
|---|---|
| No release date | SPT ($O(NlogN)$) |
| Preemptive | SRPT ($O(NlogN)$) |
| Identical processing times | nondecreasing order of the release dates $O(NlogN)$ |

- If $\boldsymbol{K = 1}$, the problem can be solved in time $O(NlogN)$.
- Moore-Hodgson algorithm for $1||\sum U_i$ : Moore, MS (98).




- $1|r_i|V$ equivalent to $1|r_i, d_i = d|\sum U_i$.

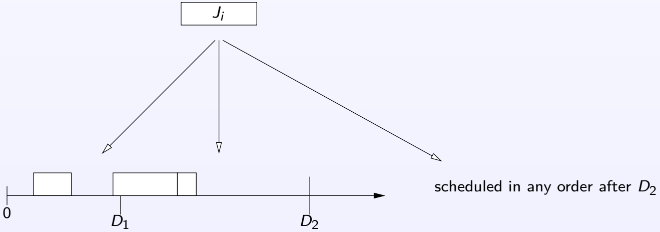
# Dominance rules

There exists an optimal solution such that :

1. there is no idle time between any pair of consecutive jobs scheduled between $D_1$ and $D_2$,
2. the jobs scheduled between two consecutive delivery dates (with $D_0 = 0$) are ordered following their nondecreasing release dates.

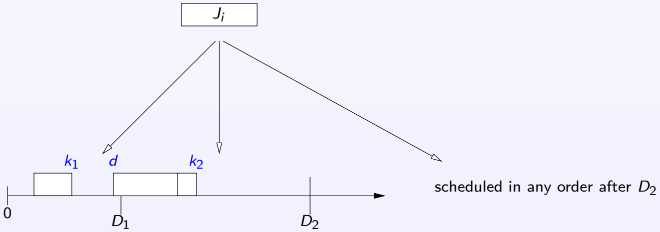
# Dynamic programming algorithm

- Dynamic programming algorithm for $1|r_i|V_1 + V_2$.
- Rationale of the algorithm :
  - Jobs are ordered following their nondecreasing release dates : $J_1, \ldots, J_N$
  - $N$ steps
  - Step $i$ :



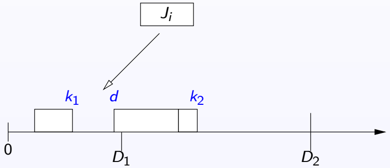
# Dynamic programming algorithm

- Dynamic programming algorithm for $1|r_i|V_1 + V_2$.
- Rationale of the algorithm :
  - Jobs are ordered following their nondecreasing release dates : $J_1, \ldots, J_N$
  - $N$ steps
  - Step $i$ :



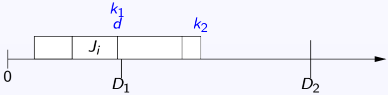
# Dynamic programming algorithm

Case 1 :



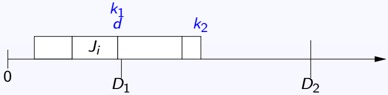
# Dynamic programming algorithm

Case 1 :

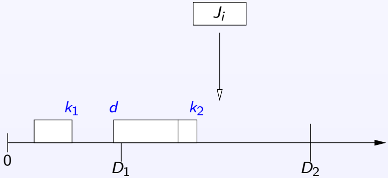


# Dynamic programming algorithm

Case 1 :

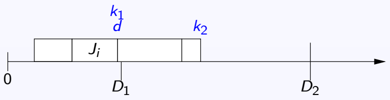



Case 2 :

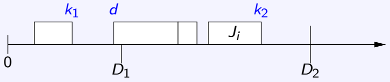


# Dynamic programming algorithm

Case 1 :

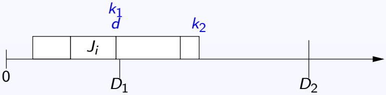



Case 2 :

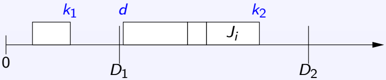


# Dynamic programming algorithm

Case 1 :

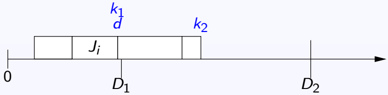



Case 2 :

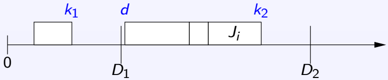


# Dynamic programming algorithm

Case 1 :



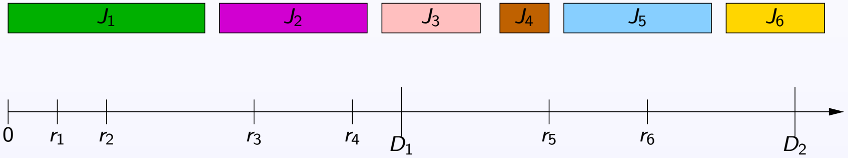

Case 2 :




Complexity : $O(N(D_1(D_2)^2) + N \log N)$

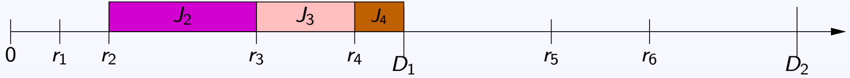
# Additional dominance rule

Given an instance of $1|r_i|V_1 + V_2$:



# Additional dominance rule

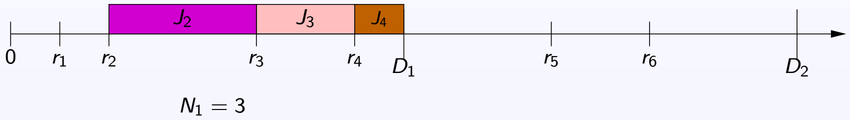
Given an instance of $1|r_i|V_1 + V_2$:




$N_1 = 3$

# Additional dominance rule

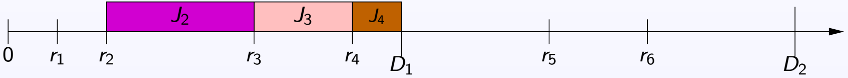
Given an instance of $1|r_i|V_1 + V_2$:




Dominance rule: there exists an optimal solution in which $N_1$ jobs complete before $D_1$.

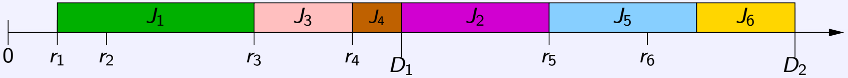
# Additional dominance rule

Given an instance of $1|r_i|V_1 + V_2$:




$N_1 = 3$

Dominance rule: there exists an optimal solution in which $N_1$ jobs complete before $D_1$.



# Preliminary results

*Instance generation scheme:*

- Processing times $p_i \in [10, 100[$
- Delivery dates: $\{T, 2T\}$, with $T = \alpha \sum p_i/2$ ($\alpha \in \{0.8, 1, 1.2\}$)
- Release dates in $[0, r_{amp}T]$ or $[T, T + r_{amp}T]$ ($r_{amp} \in \{0.1, 0.3, 0.5\}$)

# Preliminary results

*Instance generation scheme:*

- Processing times $p_i \in [10, 100[$
- Delivery dates: $\{T, 2T\}$, with $T = \alpha \sum p_i/2$ $(\alpha \in \{0.8, 1, 1.2\})$
- Release dates in $[0, r_{amp}T]$ or $[T, T + r_{amp}T]$ $(r_{amp} \in \{0.1, 0.3, 0.5\})$

*Simple dynamic programming:*

Mean CPU time on 45 30-jobs instances (time limit: 15 min CPU):

344.7 (41/45)

# Preliminary results

*Instance generation scheme:*

- Processing times $p_i \in [10, 100[$
- Delivery dates: $\{T, 2T\}$, with $T = \alpha \sum p_i/2$ ($\alpha \in \{0.8, 1, 1.2\}$)
- Release dates in $[0, r_{amp}T]$ or $[T, T + r_{amp}T]$ ($r_{amp} \in \{0.1, 0.3, 0.5\}$)

*Simple dynamic programming:*

Mean CPU time on 45 30-jobs instances (time limit: 15 min CPU):

344.7 (41/45)

*Dynamic programming + additional dominance rule:*

Mean CPU time on 45 N-jobs instances (time limit: 30 min CPU):

| $N$ | 30 | 40 | 50 | 60 | 70 | 80 |
|---|---|---|---|---|---|---|
| CPU time (s) | 9.3 | 56.7 | 274.8 | 668.4 (44) (693.5) | 797.8 (28) (1176.4) | 1186.6 (9) (1677.3) |

3.33 GHz Intel Core2-Duo processor, 8 GB RAM, running Debian wheezy/sid

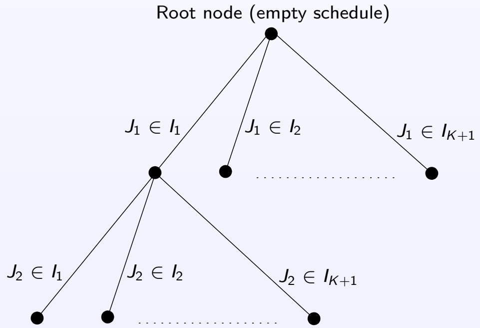
# Dominance rules

There exists an optimal solution such that :

1. there is no idle time between any pair of consecutive jobs scheduled between $D_k$ and $D_{k+1}$, $k = 0, \ldots, K-1$
2. the jobs scheduled between two consecutive delivery dates (with $D_0 = 0$) are ordered following their nondecreasing release dates.

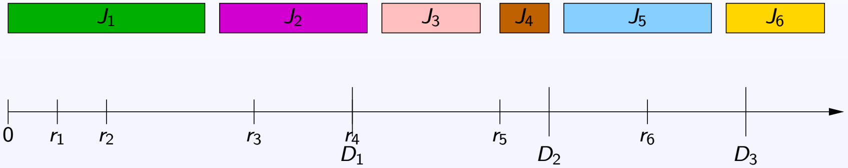
# Branching rule

- Jobs numbered by nondecreasing release dates : $J_1, \ldots, J_N$
- $I_k = ]D_{k-1}, D_k]$, $k = 1, \ldots, K$
- $I_{K+1} = ]D_K, \max_{i=1,\ldots,N} r_i + \sum_{i=1}^{N} p_i]$



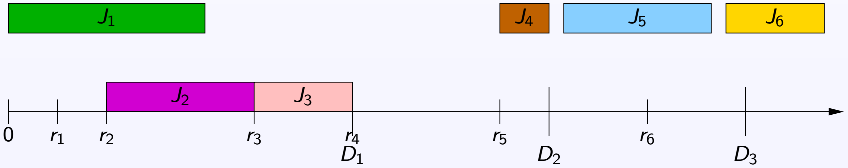
# An initial lower bound

- Algorithm for $1|r_i|V$ applied on each $I_k$
- on not yet scheduled jobs



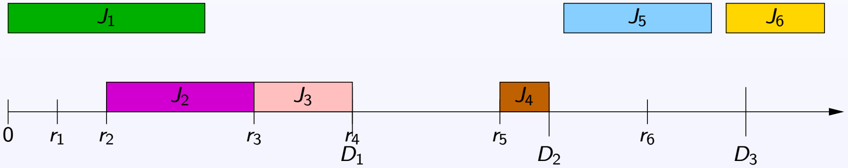
# An initial lower bound

- Algorithm for $1|r_i|V$ applied on each $I_k$
- on not yet scheduled jobs



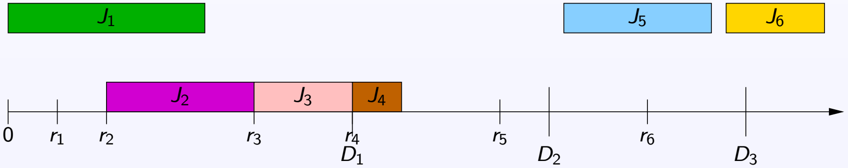
# An initial lower bound

- Algorithm for $1|r_i|V$ applied on each $I_k$
- on not yet scheduled jobs



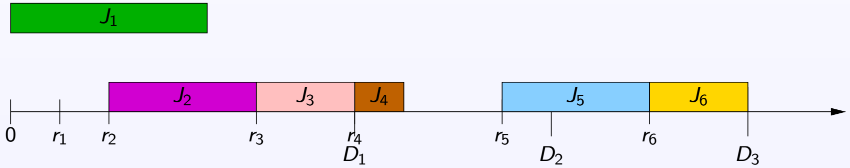
# An initial lower bound

- Algorithm for $1|r_i|V$ applied on each $I_k$
- on not yet scheduled jobs



# An initial lower bound

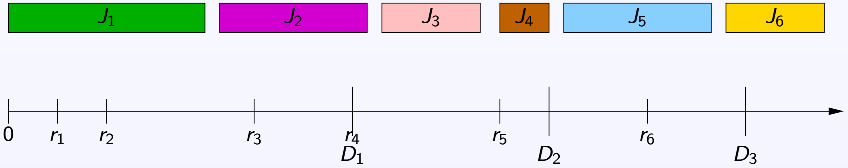
- Algorithm for $1|r_i|V$ applied on each $I_k$
- on not yet scheduled jobs




- 2 jobs in $I_1 \rightarrow 2 \times 3 = 6$
- 1 job in $I_2 \rightarrow 1 \times 2 = 2$
- 2 jobs in $I_3 \rightarrow 2 \times 1 = 2$
- Payoff : 10

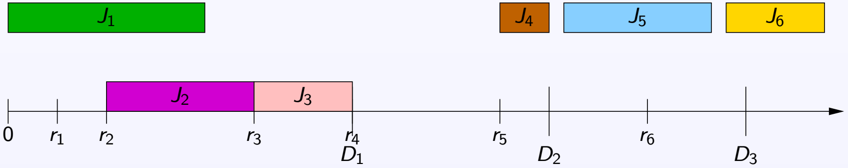
# An initial upper bound

- Algorithm for $1|r_i|V$ applied on each interval $[0, D_k]$, $k = 1, \ldots, K$
- always on the initial set of jobs



# An initial upper bound

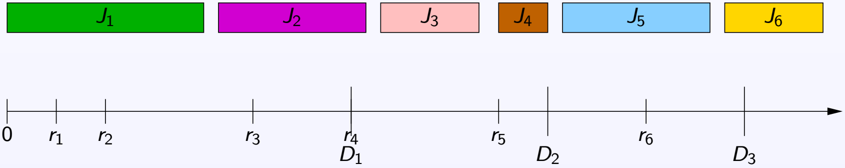
- Algorithm for $1|r_i|V$ applied on each interval $[0, D_k]$, $k = 1, \ldots, K$
- always on the initial set of jobs




Maximum number of jobs in $[0, D_1]$ : $N_1 = 2$

# An initial upper bound

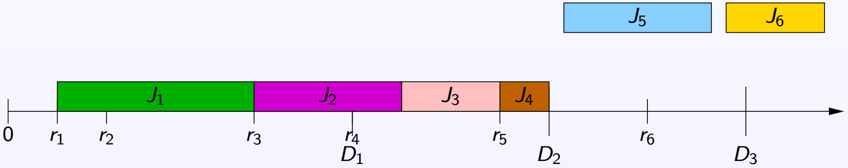
- Algorithm for $1|r_i|V$ applied on each interval $[0, D_k]$, $k = 1, \ldots, K$
- always on the initial set of jobs




Maximum number of jobs in $[0, D_1]$ : $N_1 = 2$

# An initial upper bound

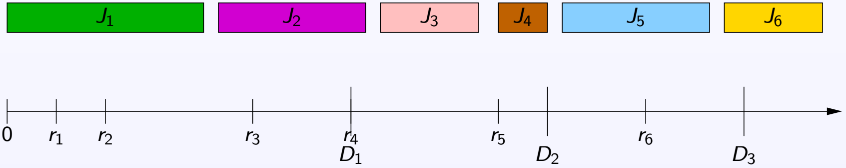
- Algorithm for $1|r_i|V$ applied on each interval $[0, D_k]$, $k = 1, \ldots, K$
- always on the initial set of jobs




Maximum number of jobs in $[0, D_1]$ : $N_1 = 2$
Maximum number of jobs in $[0, D_2]$ : $N_2 = 4$

# An initial upper bound

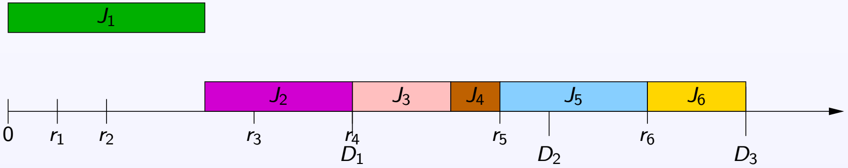
- Algorithm for $1|r_i|V$ applied on each interval $[0, D_k]$, $k = 1, \ldots, K$
- always on the initial set of jobs




Maximum number of jobs in $[0, D_1]$ : $N_1 = 2$
Maximum number of jobs in $[0, D_2]$ : $N_2 = 4$

# An initial upper bound

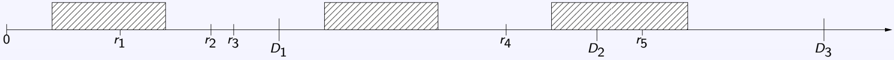
- Algorithm for $1|r_i|V$ applied on each interval $[0, D_k]$, $k = 1, \ldots, K$
- always on the initial set of jobs




Maximum number of jobs in $[0, D_1]$ : $N_1 = 2$
Maximum number of jobs in $[0, D_2]$ : $N_2 = 4$
Maximum number of jobs in $[0, D_3]$ : $N_3 = 5$
Upper bound : $N_1$ jobs in $I_1$, $N_2 - N_1$ jobs in $I_2$, $N_3 - N_2$ jobs in $I_3$
Payoff : $2 \times 3 + 2 \times 2 + 1 \times 1 = 11$

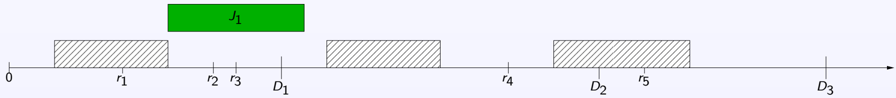
# An upper bound on a partial schedule

Completion with SRPT rule :



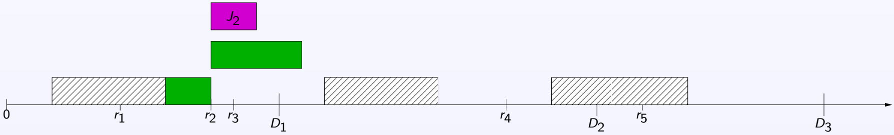
# An upper bound on a partial schedule

Completion with SRPT rule :



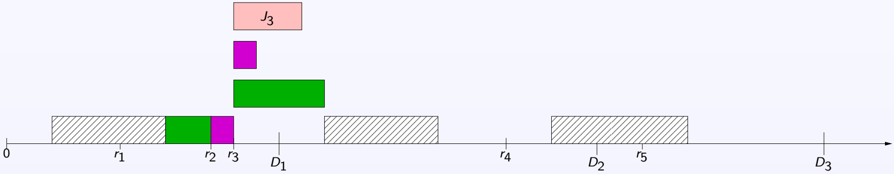
# An upper bound on a partial schedule

Completion with SRPT rule :



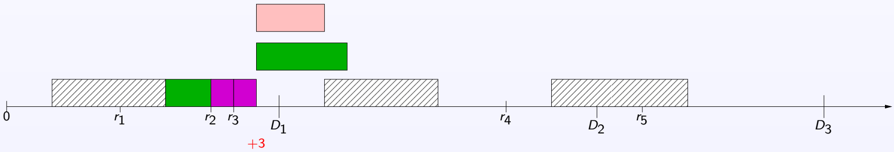
# An upper bound on a partial schedule

Completion with SRPT rule :



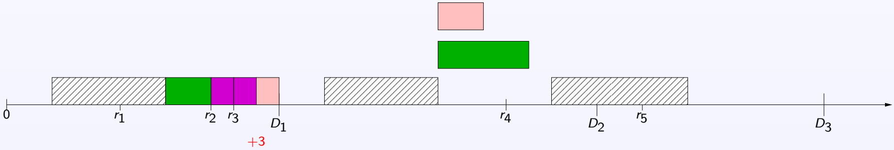
# An upper bound on a partial schedule

Completion with SRPT rule :



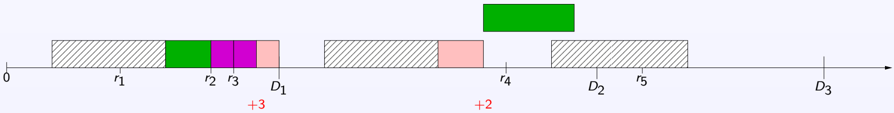
# An upper bound on a partial schedule

Completion with SRPT rule :



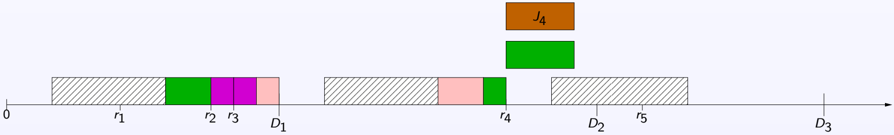
# An upper bound on a partial schedule

Completion with SRPT rule :



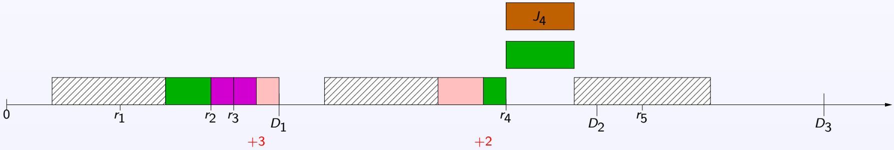
# An upper bound on a partial schedule

Completion with SRPT rule :



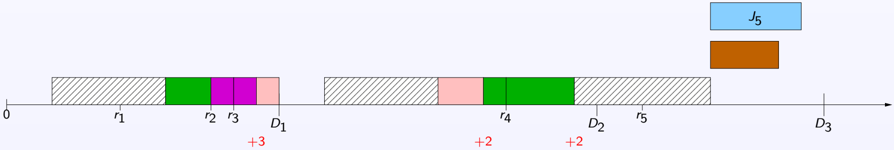
# An upper bound on a partial schedule

Completion with SRPT rule :



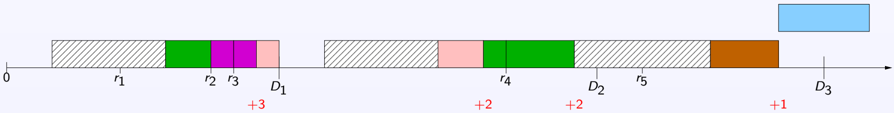
# An upper bound on a partial schedule

Completion with SRPT rule :



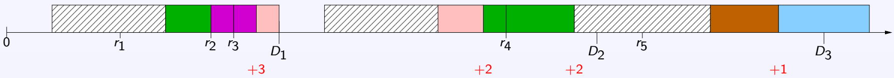
# An upper bound on a partial schedule

Completion with SRPT rule :



# An upper bound on a partial schedule

Completion with SRPT rule :



# Preliminary results

*Instance generation scheme:*

- Processing times $p_i \in [10, 100[$
- Delivery dates: $\{T, 2T, \ldots, KT\}$, with $T = \lfloor \alpha \sum p_i / K \rfloor$ $(\alpha \in \{0.8, 1, 1.2\})$
- Release dates in $[(k-1)T, (k-1)T + r_{amp}T]$, $k = 1, \ldots, K$ $(r_{amp} \in \{0.1, 0.3, 0.5, 0.7\})$

# Preliminary results

*Instance generation scheme:*

- Processing times $p_i \in [10, 100[$
- Delivery dates: $\{T, 2T, \ldots, KT\}$, with $T = \lfloor \alpha \sum p_i / K \rfloor$ $(\alpha \in \{0.8, 1, 1.2\})$
- Release dates in $[(k-1)T, (k-1)T + r_{amp}T]$, $k = 1, \ldots, K$ $(r_{amp} \in \{0.1, 0.3, 0.5, 0.7\})$

Number of instances solved in less than 3600 s. CPU :

| $K \backslash N$ | 100 | 200 | 300 | 500 |
|---|---|---|---|---|
| 2 (200 inst.) | 196 (194) | 193 (193) | 198 (198) | 199 (199) |
| 3 (60 inst.) | 58 (57) | 58 (58) | 59 (59) | 60 (59) |
| 4 (60 inst.) | 55 (54) | 54 (54) | 54 (54) | 56 (56) |

Mean CPU time for $N = 100$ :

| $K$ | 2 (2 inst.) | 3 (1 inst.) | 4 (1 inst.) |
|---|---|---|---|
| Time | 1032.00 | 1049.56 | 5.87 |

3.33 GHz Intel Core2-Duo processor, 8 GB RAM, running Debian wheezy/sid

# Travaux en cours

- Dans le Branch and Bound, calcul de la borne supérieure d'une solution partielle avec un algorithme analogue à celui calculant une borne supérieure initiale
- Comparaison avec les résultats de Detienne et al. (2011)
- Recherche d'un algorithme approché avec garantie de performance pour $1|r_i|V_1 + V_2$

B. Detienne, S. Dauzères-Pérès, and C. Yugma.
Scheduling jobs on parallel machine to minimize a regular step total cost function.
*Journal of Scheduling*, 2011.

N. G. Hall, M. Lesaoana, and C. N. Potts.
Scheduling with fixed delivery dates.
*Operations Research*, 49(1), 2001.

N. G. Hall, S. P. Sethi, and C. Sriskandarajah.
On the complexity of generalized due date scheduling problems.
*European Journal of Operational Research*, 51(1):100–109, March 1991.

A. Janiak and T. Krysiak.
Single processor scheduling with job values depending on their completion times.
*Journal of Scheduling*, 10(2):129–138, April 2007.

J. M. Moore.
An n Job, One Machine Sequencing Algorithm for Minimizing the Number of Late Jobs.
*Management Science*, 15(1):102–109, 1968.