Scheduling Problems and Solutions

Uwe Schwiegelshohn

IRF-IT Dortmund University
Summer Term 2006

Textbook

- Scheduling – Theory, Algorithms, and Systems
  Michael Pinedo
  2nd edition, 2002
  Prentice-Hall Inc.
  Pearson Education

- The lecture is based on this textbook.

- These slides are an extract from this book. They are to be used only for this lecture and as a complement to the book.

Scheduling Problem

Constraints

Tasks $\longrightarrow$ Time

Resources

(Jobs) (Machines)

Objective(s)

Areas:

- Manufacturing and production
- Transportations and distribution
- Information - processing

Example 1 Paper Bag Factory

- different types of paper bags
- 3 production stages
  - printing of the logo
  - gluing of the side
  - sewing of one or both ends
- several machines for each stage
  - differences in speed and function
  - processing speed and processing quantity
  - setup time for a change of the bag type
- due time and late penalty
- minimization of late penalties, setup times

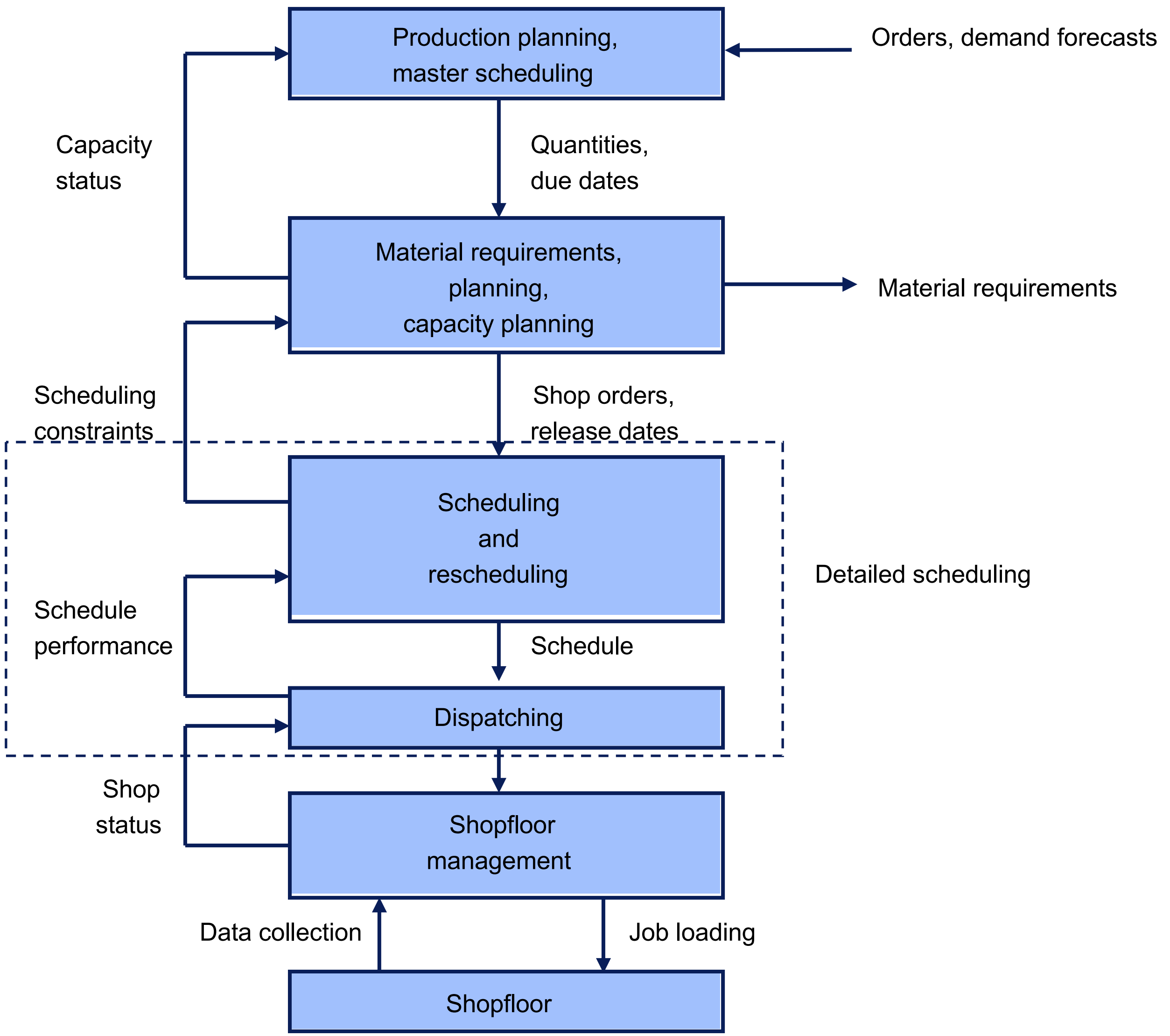
Example 2
Gate Assignments at Airport

- different types of planes (size)
- different types of gates (size, location)
- flight schedule
  - randomness (weather, take off policy)
- service time at gate
  - deplaning of passengers
  - service of airplane
  - boarding of passengers
- minimization of work for airline personnel
- minimization of airplane delay

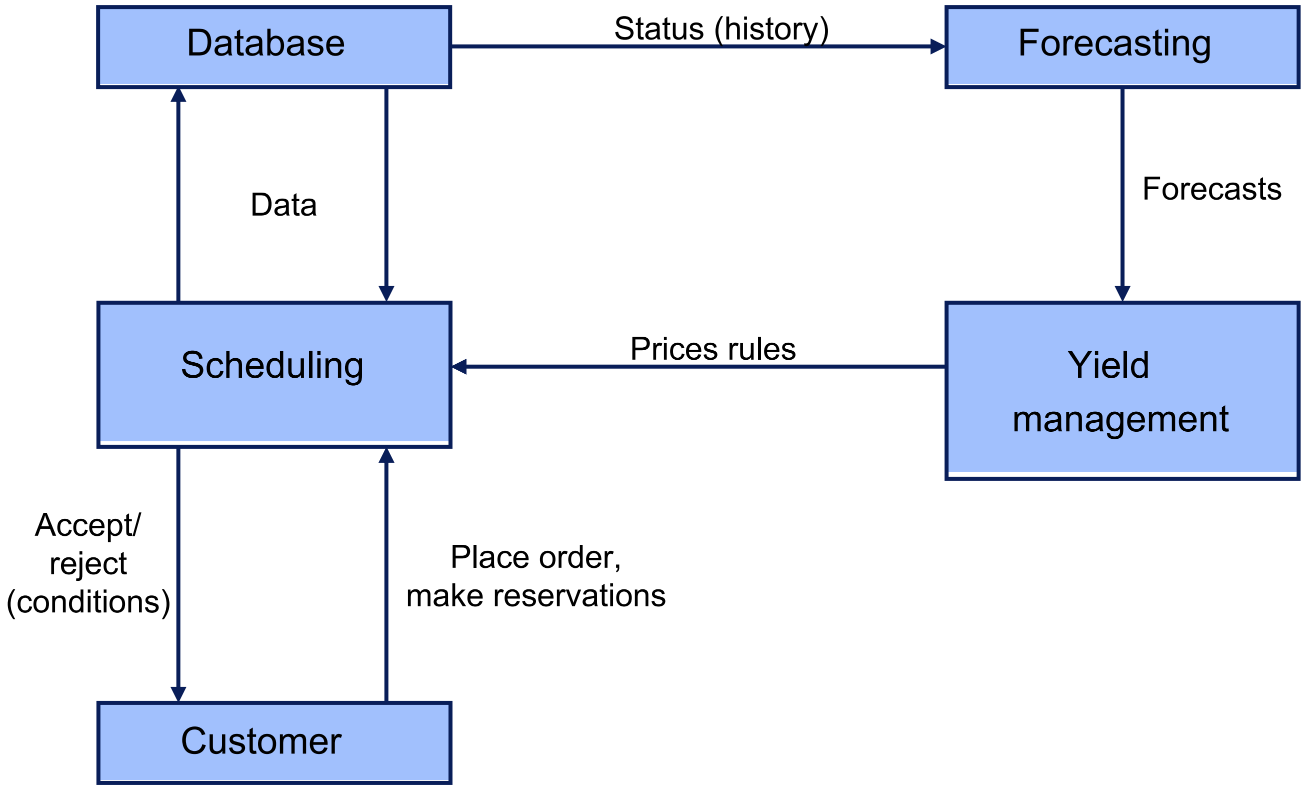
Example 3 Tasks in a CPU

- different applications
  - unknown processing time
  - known distributions (average, variance)
  - priority level
- multitasking environment
  - preemption
- minimization of the sum of expected weighted completion times

Information Flow Diagram in a Manufacturing System



Information Flow Diagram in a Service System



Job Properties

p_{ij}: processing time of job j on machine i

(p_j: identical processing time of job j on all machines)

r_{ij}: release date of job j (earliest starting time)

d_j: due date of job j (completion of job j after d_j results in a late penalty)

$\bar{d}_j$: deadline ($\bar{d}_j$ must be met)

w_j: weight of job j (indicates the importance of the job)

Machine Environment

1 : single machine

P_m : **m** identical machines in parallel

Q_m : **m** machines in parallel with different speeds

R_m : **m** unrelated machines in parallel

F_m : flow shop with **m** machines in series

- each job must be processed on each machine using the same route.
- queues between the machines

  FIFO queues, see also permutation flow shop

FF_c: flexible flow shop with **c** stages in series and several identical machines at each stage, one job needs processing on only one (arbitrary) machine at each stage.

Machine Environment

J_m : job show with **m** machines with a separate predetermined route for each job

A machine may be visited more than once by a job. This is called *recirculation.*

FJ_c : flexible job shop with **c** stages and several identical machines at each stage, see FF_c

O_m : Open shop with **m** machines

Each job must be processed on each machine.

Restrictions and Constraints

- release dates, see also job properties
- sequence dependent setup times
  - S_{ijk} : setup time between job j and job k on machine i
  - (S_{jk} : identical setup times for all machines)
  - (S_{0j} : startup for job j)
  - (S_{j0} : cleanup for job j)
- preemption (prmp)
  - The processing of a job can be interrupted and later resumed (on the same or another machine).
- precedence constraints (prec)
  - Certain jobs must be completed before another job can be started.
    - representation as a directed acyclic graph (DAG)

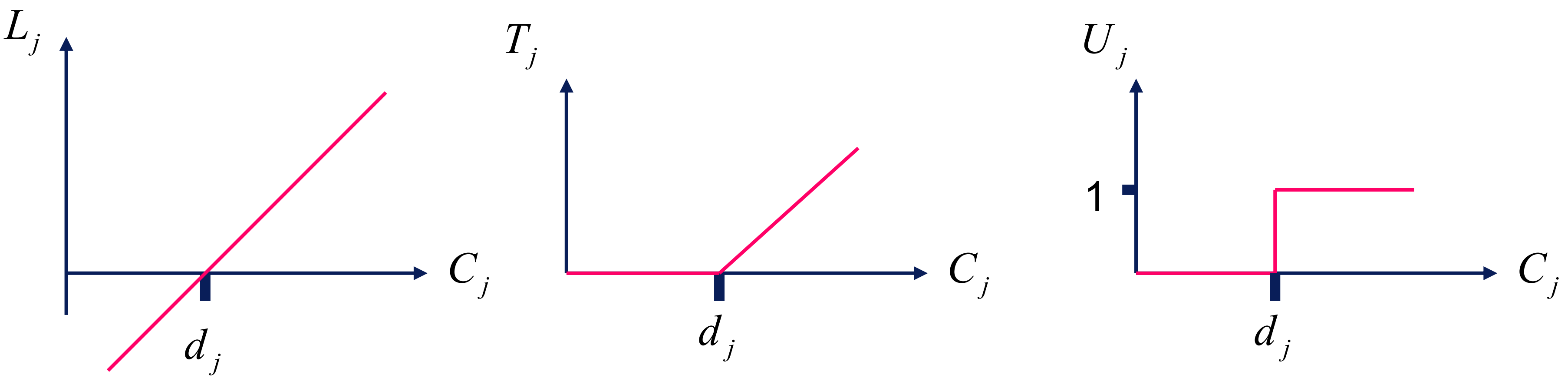
Restrictions and Constraints

- machine breakdowns (brkdwn)
  - machines are not continuously available: For instance, m(t) identical parallel machines are available at time t.
- machine eligibility restrictions (M_j)
  - M_j denotes the set of parallel machines that can process job j (for P_m and Q_m).
- permutation (prmu), see F_m
- blocking (block)
  - A completed job cannot move from one machine to the next due to limited buffer space in the queue. Therefore, it blocks the previous machine (F_m , FF_c)
- no – wait (nwt)
  - A job is not allowed to wait between two successive executions on different machines (F_m , FF_c).
- recirculation (recirc)

Objective Functions

- Completion time of job j: C_j
- Lateness of job j: $L_j = C_j – d_j$
  - The lateness may be positive or negative.
- Tardiness: $T_j = \max (L_j , 0)$
- Number of late jobs: $U_j = \begin{cases} 1, & \text{if } C_j > d_j, \\ 0, & \text{otherwise} \end{cases}$

Objective Functions



- Makespan: $C_{max} = \max(C_1, \ldots, C_n)$
  - completion time of the last job in the system
- Maximum lateness: $L_{max} = \max(L_1, \ldots, L_n)$

Objective Functions

- Total weighted completion time: $\Sigma\, w_j C_j$
- Total weighted flow time: $(\Sigma\, w_j (C_j - r_j)) = \Sigma\, w_j C_j - \underbrace{\Sigma\, w_j r_j}_{\textbf{const.}}$
- Discounted total weighted completion time:
  - $(\Sigma\, w_j (1 - e^{-rC_j}))$ $\quad 0<r<1$
- Total weighted tardiness: $\Sigma\, w_j T_j$
- Weighted number of tardy jobs: $\Sigma\, w_j U_j$
- Regular objective functions:
  - non decreasing in $C_1, \ldots, C_n$
  - Earliness: $E_j = \max(-L_j, 0)$
    - non increasing in C_j
- $\Sigma\, E_j + \Sigma\, T_j$, $\Sigma\, w_j' E_j + \Sigma\, w_j'' T_j \longrightarrow$ not regular obj. functions

Description of a Scheduling Problem

$$\alpha \mid \beta \mid \gamma$$

machine environment

constraints, processing, characteristics

objective (to be minimized)

Examples:

- Paper bag factory — $FF_3 \mid r_j , s_{jk} \mid \Sigma\, w_j T_j$
- Gate assignment — $P_m \mid r_j , M_j \mid \Sigma\, w_j T_j$
- Tasks in a CPU — $l \mid r_j , prmp \mid \Sigma\, w_j C_j$
- Traveling Salesman — $l \mid s_{jk} \mid C_{max}$

Classes of Schedules

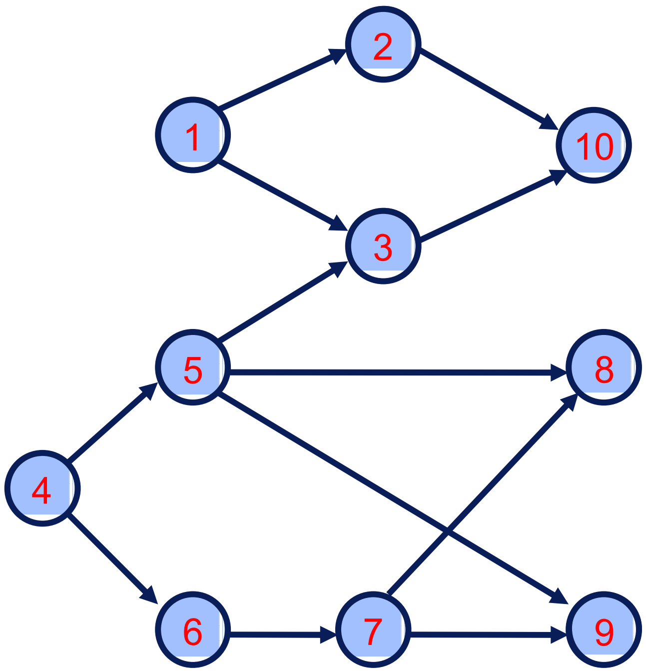
- Nondelay (greedy) schedule
  - No machine is kept idle while a task is waiting for processing.

  An optimal schedule need not be nondelay!

Example: P2 | prec | C_{max}

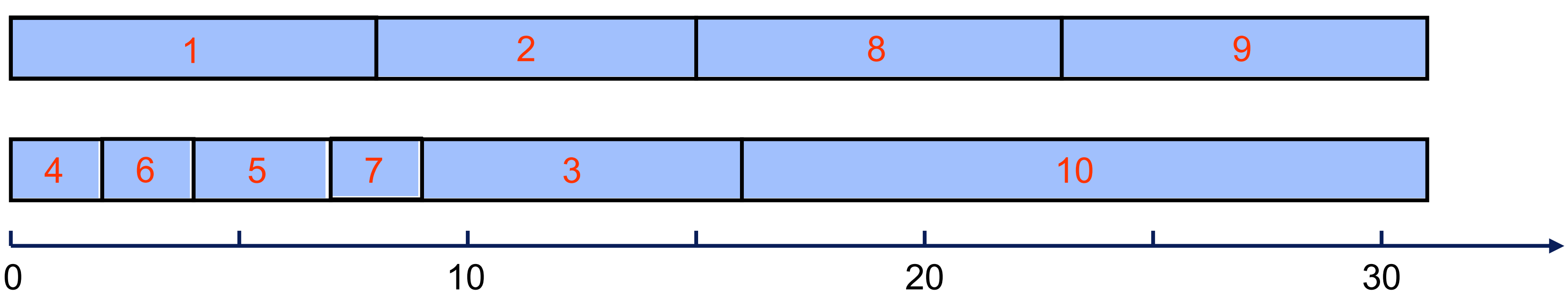
jobs	1	2	3	4	5	6	7	8	9	10
p_j	8	7	7	2	3	2	2	8	8	15

Precedence Constraints
Original Schedule

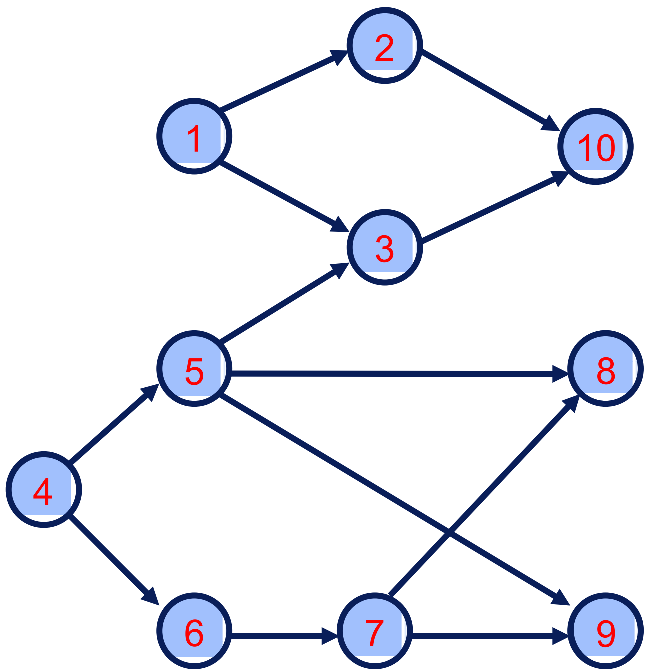


jobs	1	2	3	4	5	6	7	8	9	10
p_j	8	7	7	2	3	2	2	8	8	15

= job completed



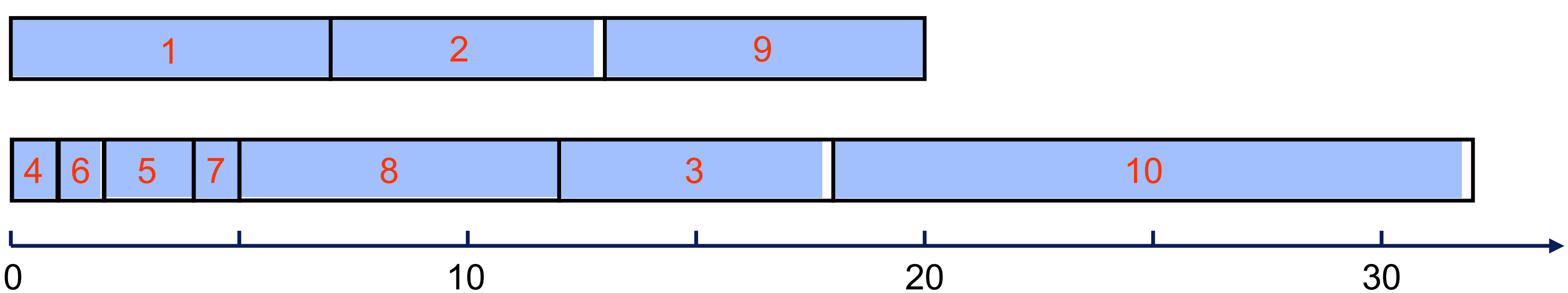
Precedence Constraints
Reduced Processing Time



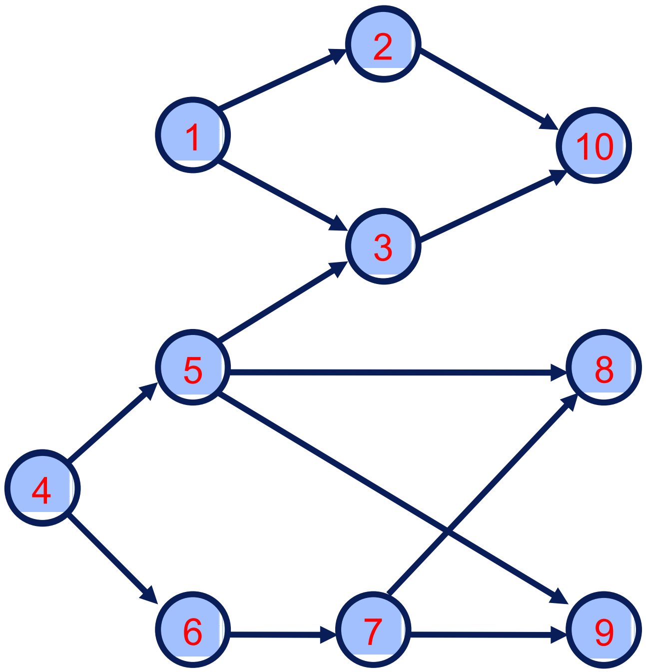
jobs	1	2	3	4	5	6	7	8	9	10
p_j	7	6	6	1	2	1	1	7	7	14

= job completed

The processing time of each job is reduced by 1 unit.



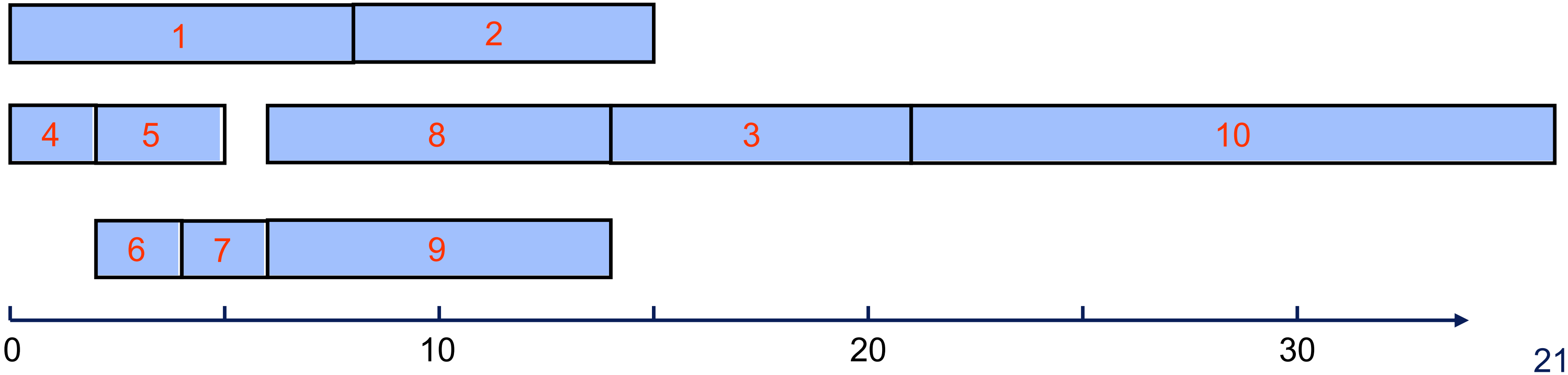
Precedence Constraints
Use of 3 Machines



jobs	1	2	3	4	5	6	7	8	9	10
p_j	8	7	7	2	3	2	2	8	8	15

= job completed

3 machines are used instead of 2 with the original processing times



Active Schedule

It is not possible to construct another schedule by changing the order of processing on the machines and having at least one task finishing earlier without any task finishing later.

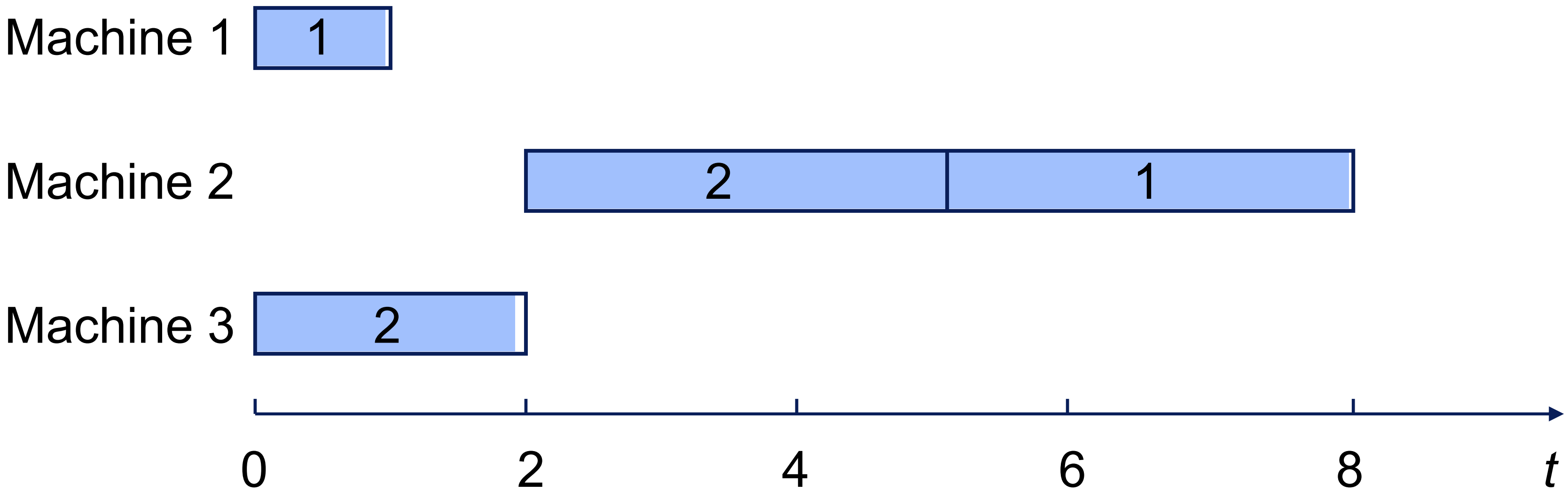
There is at least one optimal and active schedule for $J_m||\gamma$ if the objective function is regular.

Example :

Consider a job shop with three machines and two jobs.

- Job 1 needs 1 time unit on machine 1 and 3 time units on machine 2.
- Job 2 needs 2 time units on machine 3 and 3 time units on machine 2.
- Both jobs have to be processed last on machine 2.

Example of an Active Schedule



It is clear that this schedule is active as reversing the sequence of the two jobs on machine 2 postpones the processing of job 2. However, the schedule is neither nondelay nor optimal. Machine 2 remains idle until time 2 while there is a job available for processing at time 1.

Semi – active Schedule

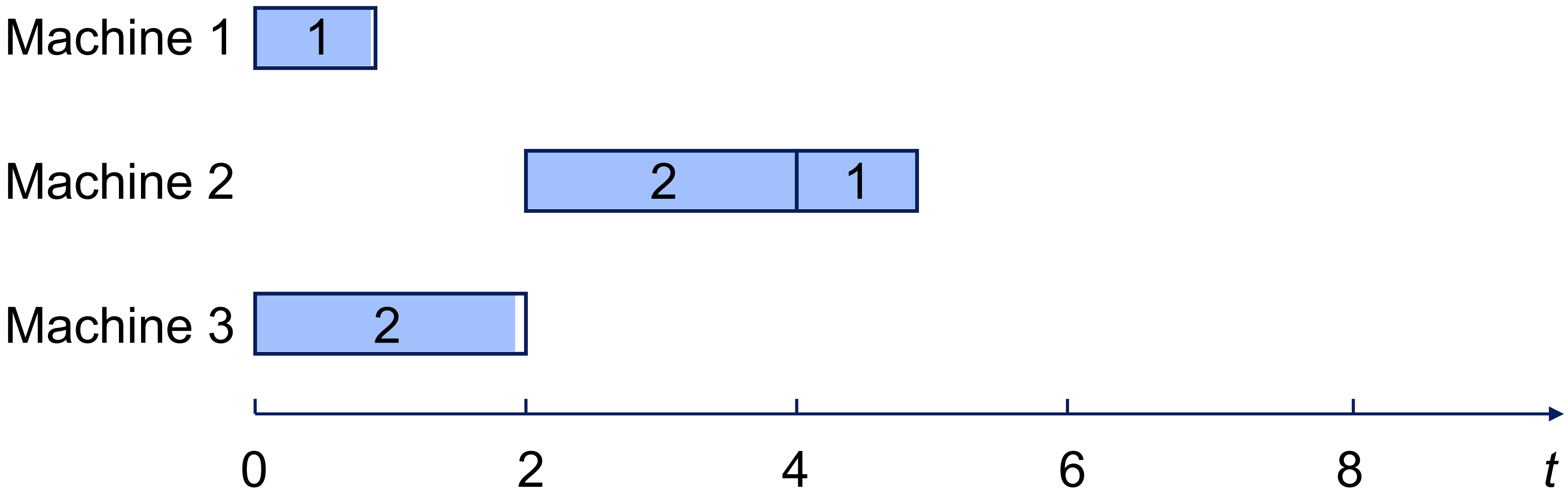
No task can be completed earlier without changing the order of processing on any one of the machines.

Example:

Consider again a schedule with three machines and two jobs. The routing of the two jobs is the same as in the previous example.

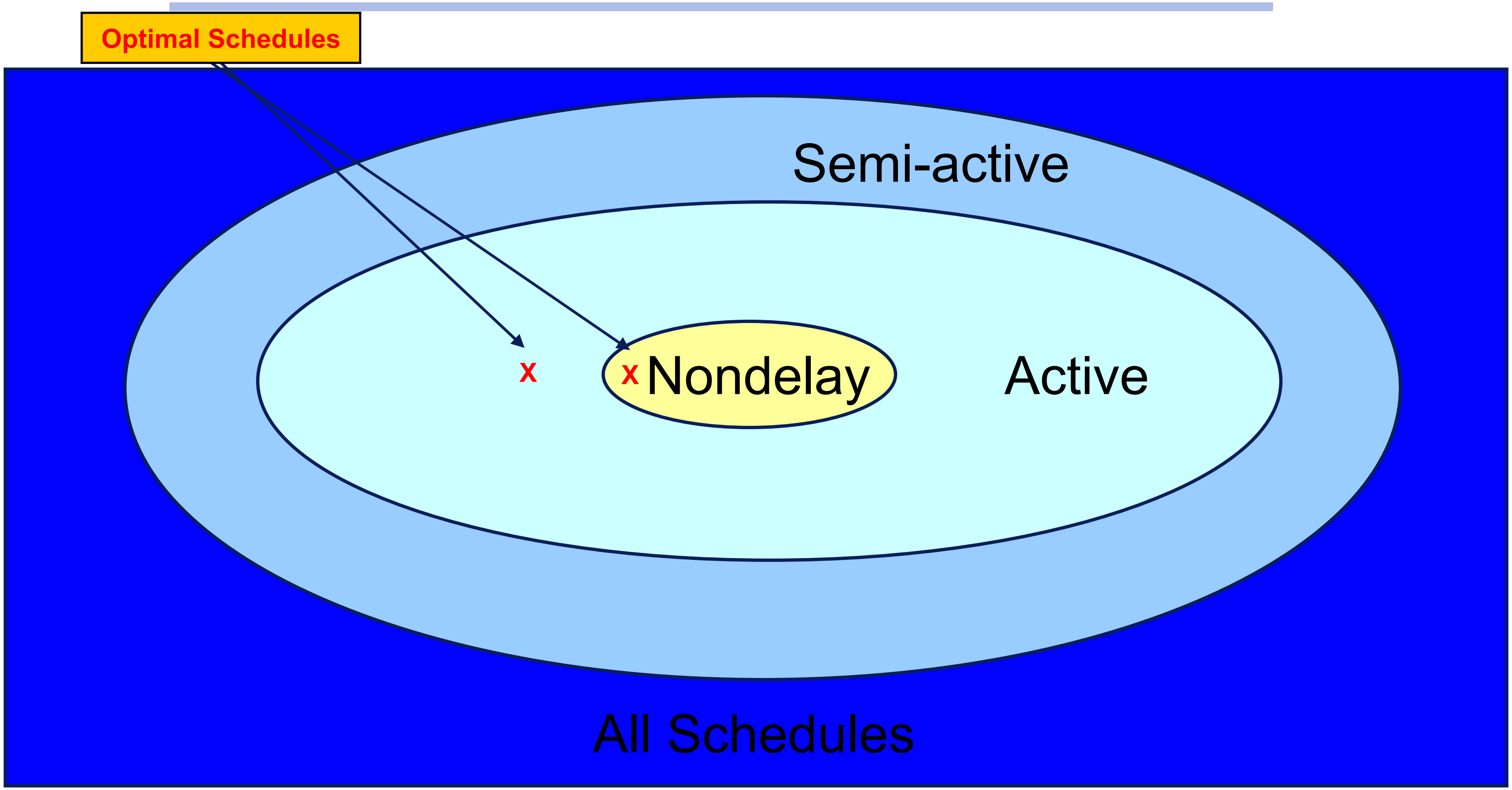
- The processing times of job 1 on machines 1 and 2 are both equal to 1.
- The processing times of job 2 on machines 2 and 3 are both equal to 2.

Example of a Semi – active Schedule



Consider the schedule under which job 2 is processed on machine 2 before job 1. This implies that job 2 starts its processing on machine 2 at time 2 and job 1 starts its processing on machine 2 at time 4. This schedule is semi-active. However, it is not active as job 1 can be processed on machine 2 without delaying the processing of job 2 on machine 2.

Venn Diagram of Classes of Schedules for Job Shops



A Venn diagramm of the three classes of nonpreemptive schedules; the nondelay schedules, the active schedules, and the semi-active schedules

Complexity Preliminaries

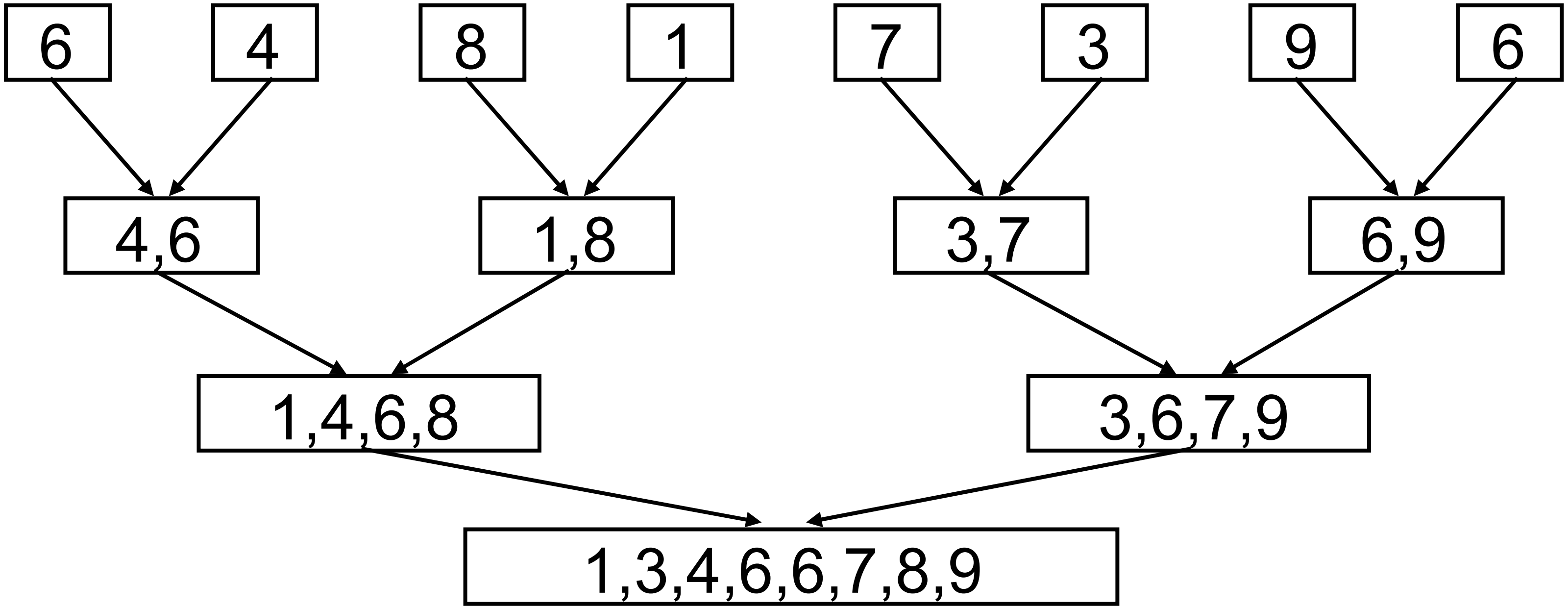
- $T(n)=O(f(n))$ if $T(n)<c\cdot f(n)$ holds for some $c>0$ and all $n>n_0$.
- Example $1500 + 100n^2 + 5n^3=O(n^3)$
- Input size of a simple scheduling problem

$$\underbrace{n}_{\text{number of jobs}} \ \underbrace{\log_2(\max p_j)}_{\text{maximal processing time In binary encoding}}$$

number of jobs

maximal processing time
In binary encoding

Mergesort



n input values

at most $n \log_2 n$ comparison steps

time complexity of mergesort: $O(n \log n)$

Complexity Hierarchies of Deterministic Scheduling Problems

Some problems are special cases of other problems:

Notation: $\alpha_1 \mid \beta_1 \mid \gamma_1 \propto$ (reduces to) $\alpha_2 \mid \beta_2 \mid \gamma_2$

Examples:

$$1 \mid\mid \Sigma C_j \propto 1 \mid\mid \Sigma w_j C_j \propto P_m \mid\mid \Sigma w_j C_j \propto Q_m \mid prec \mid \Sigma w_j C_j$$

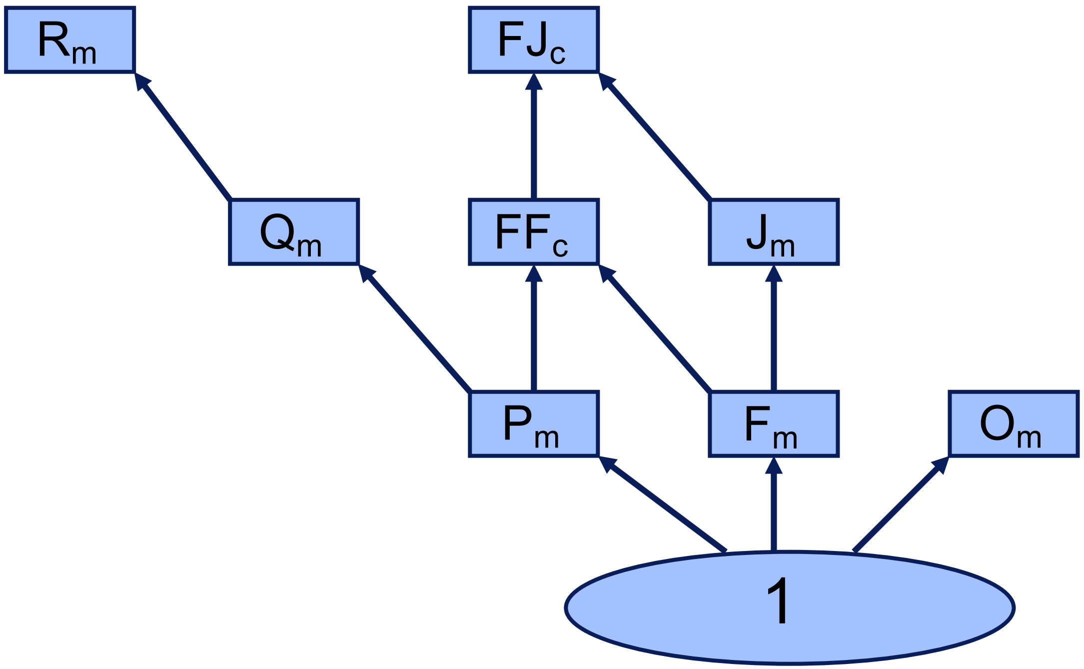
Complex reduction cases:

$$\alpha \mid \beta \mid L_{max} \propto \alpha \mid \beta \mid \Sigma U_j$$

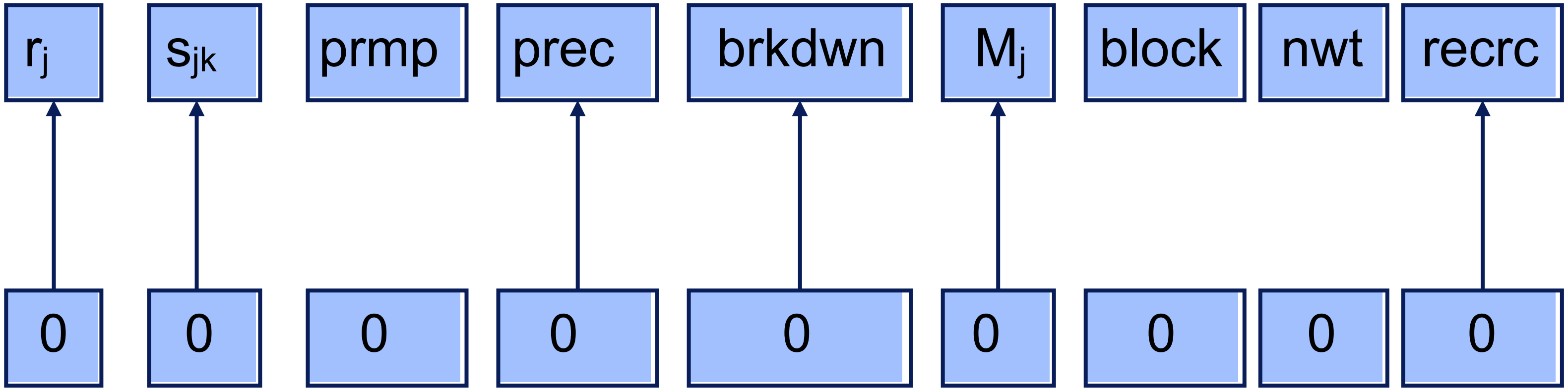
$$\alpha \mid \beta \mid L_{max} \propto \alpha \mid \beta \mid \Sigma T_j$$

Variation of d_j and logarithmic search

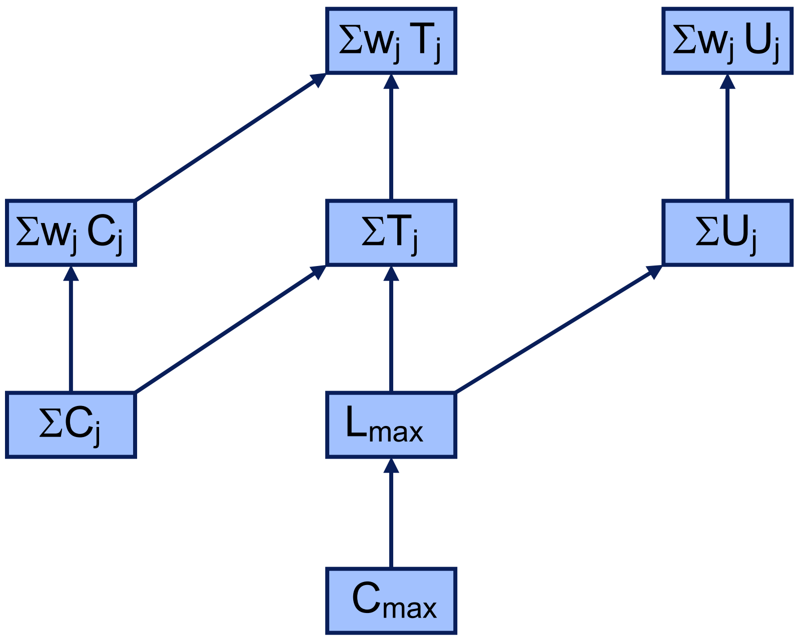
Machine Environment



Processing Restrictions and Constraints



Objective Functions



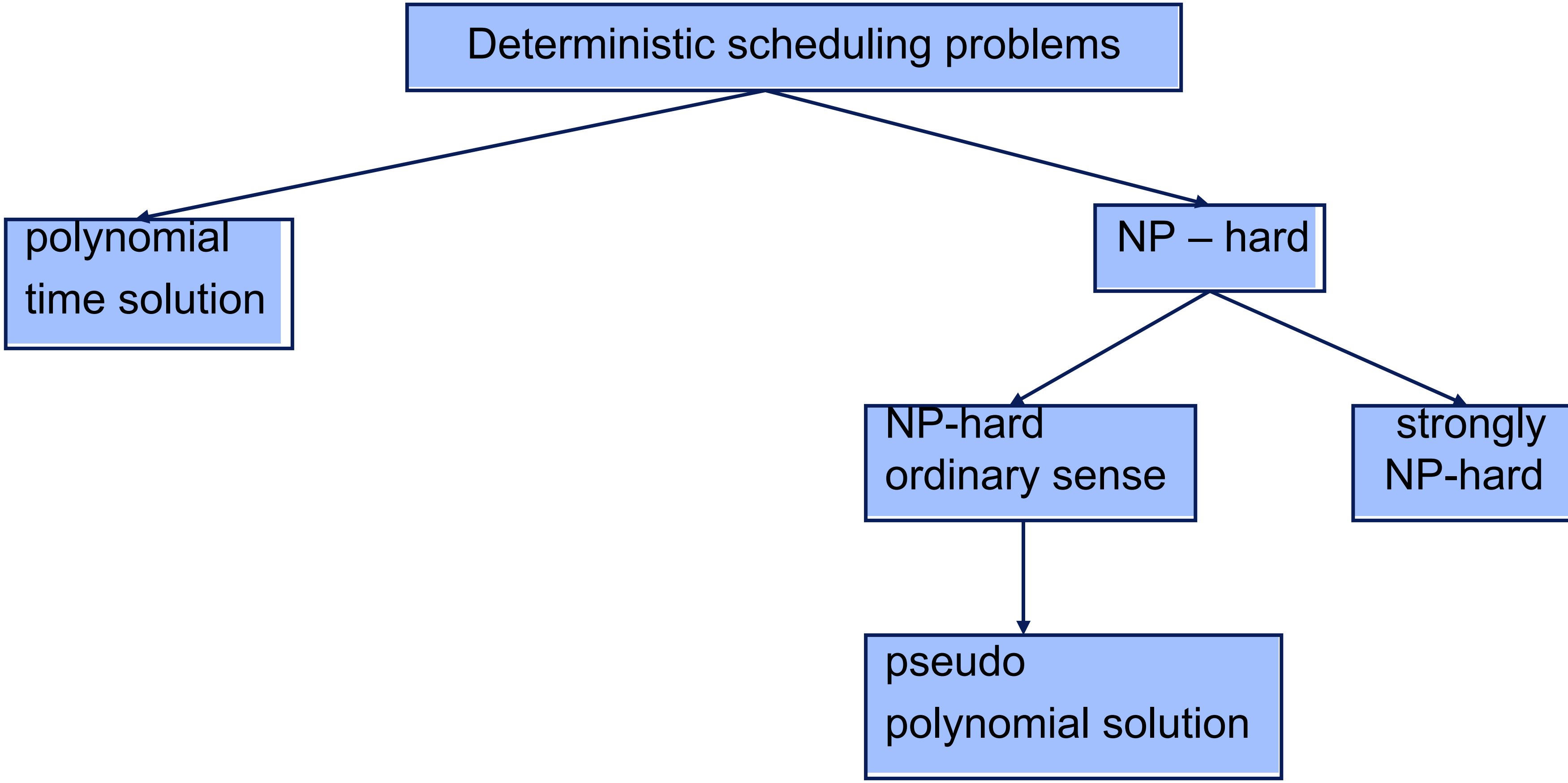
Time Complexity of Algorithms

- Easy (polynomial time complexity):
  There is an algorithm that optimally solves the problem with time complexity $O((n \log(\max p_j))^k)$ for some fixed k.

- NP-hard in the ordinary sense
  (pseudo polynomial time complexity):
  The problem cannot be optimally solved by an algorithm with polynomial time complexity but with an algorithm of time complexity $O((n \max p_j)^k)$.

- NP-hard in the strong sense:
  The problem cannot be optimally solved by an algorithm with pseudo polynomial complexity.

Problem Classification



Partition

Given positive integers $a_1,\ldots, a_t$ and $b = \frac{1}{2}\sum_{j=1}^{t} a_j$,

do there exist two disjoint subsets S_1 and S_2 such that

$$\sum_{j \in S_i} a_j = b$$

for i=1,2?

Partition is NP-hard in the ordinary sense.

3-Partition

Given positive integers $a_1,\dots, a_{3t}$, b with

$$\frac{b}{4} < a_j < \frac{b}{2} \quad , \qquad j = 1,\dots , 3t,$$

and

$$\sum_{j=1}^{3t} a_j = tb$$

do there exist t pairwise disjoint three element subsets $S_i \subset \{1,\dots , 3t\}$ such that

$$\sum_{j \in S_i} a_j = b \qquad \text{for } i=1,\dots , t?$$

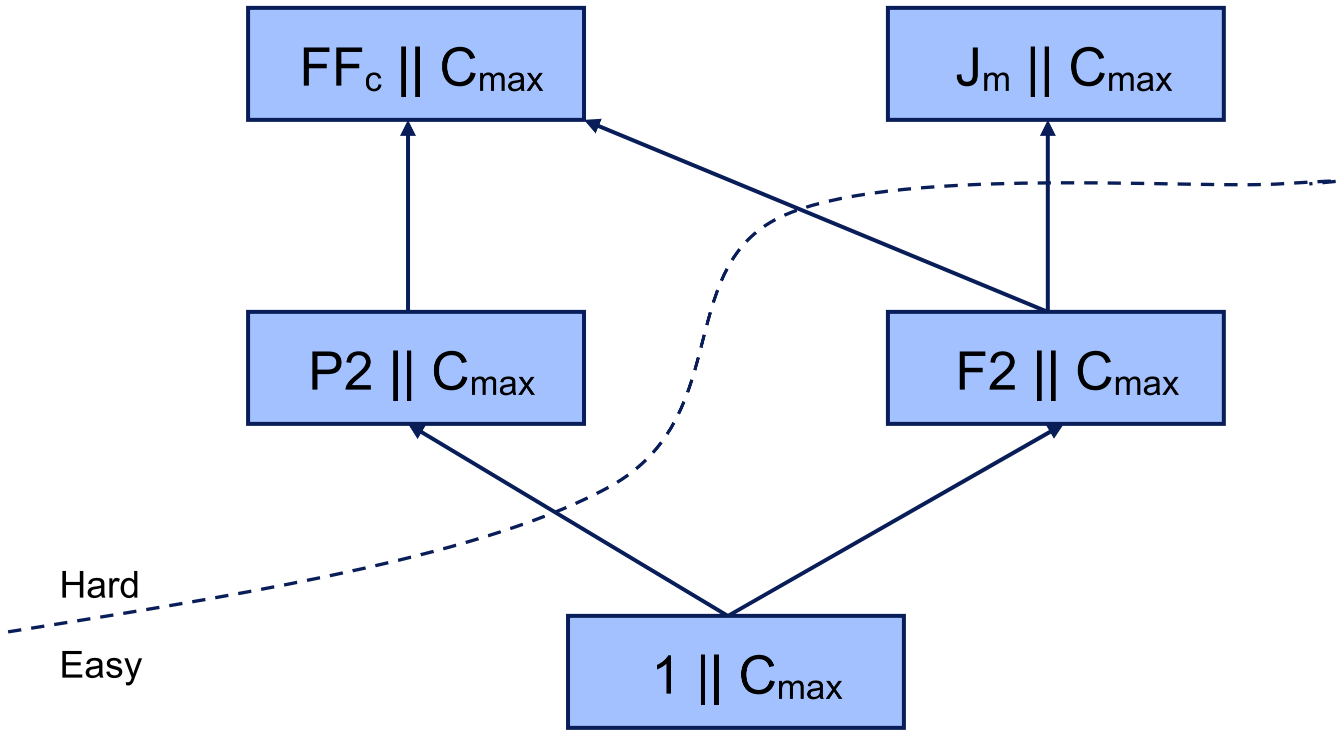
3-Partition is strongly NP-hard.

Proof of NP-Hardness

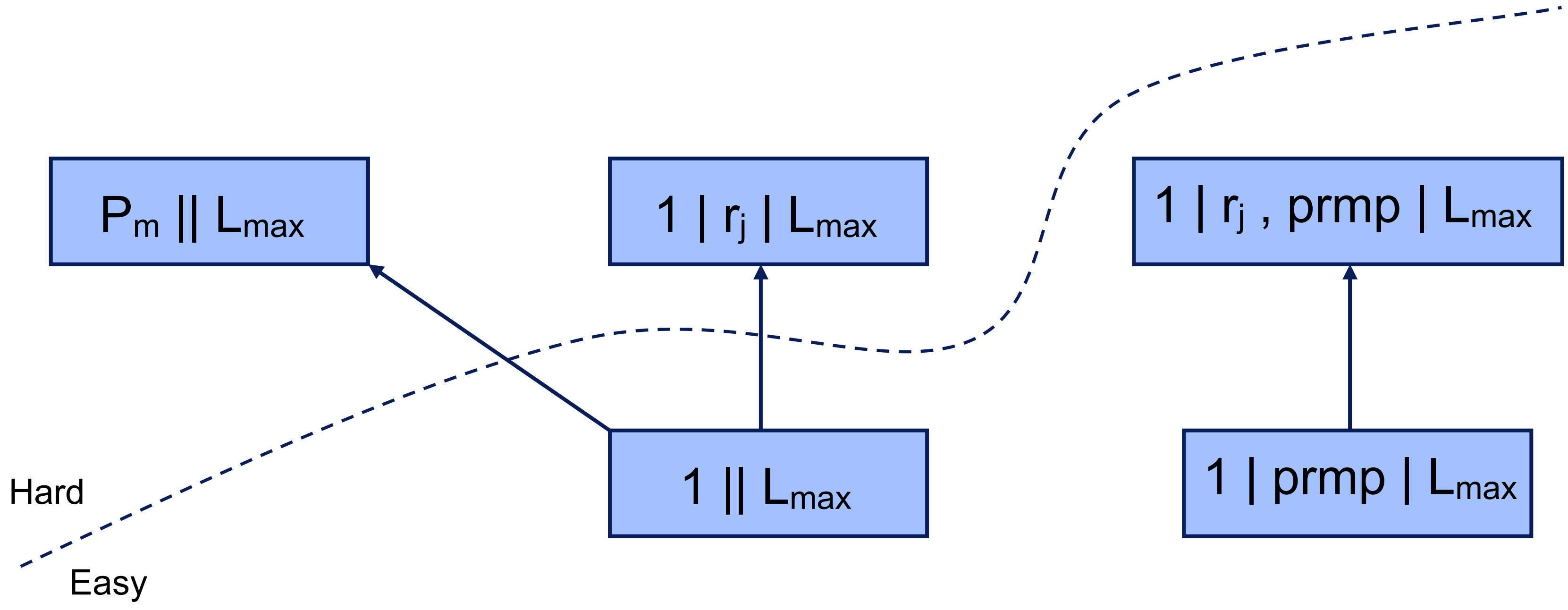
- A scheduling problem is NP-hard in the ordinary sense if
  - partition (or a similar problem) can be reduced to this problem with a polynomial time algorithm and
  - there is an algorithm with pseudo polynomial time complexity that solves the scheduling problem.

- A scheduling problem is strongly NP-hard if
  - 3-partition (or a similar problem) can be reduced to this problem with a polynomial time algorithm.

Complexity of Makespan Problems



Complexity of Maximum Lateness Problems



Total Weighted Completion Time

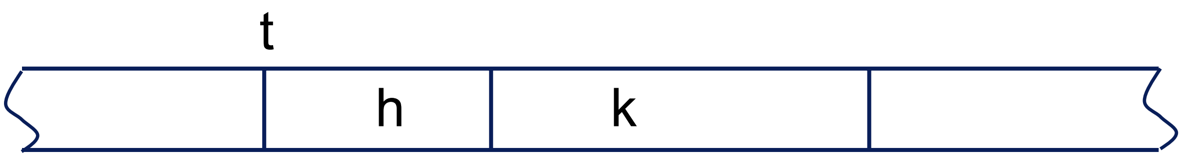
$1 \mid\mid \Sigma\, w_j C_j$: Schedule the jobs in Smith order $\frac{w_j}{p_j}$.

The Weighted Shortest Processing Time first (WSPT) rule is optimal for $1 \mid\mid \Sigma\, w_j C_j$.

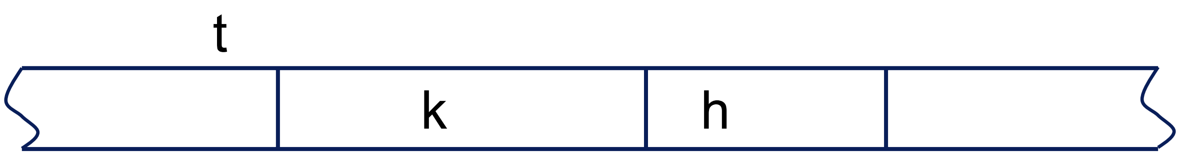
Proof by contradiction and localization:

If the WSPT rule is violated then it is violated by a pair of neighboring task h and k.

Total Weighted Completion Time



S_1: $\Sigma\, w_j\, C_j = \ldots + w_h(t+p_h) + w_k(t + p_h + p_k)$



S_2: $\Sigma\, w_j\, C_j = \ldots + w_k(t+p_k) \quad + w_h(t + p_k + p_h)$

Difference between both schedules S_1 und S_2:

$w_k\, p_h - w_h\, p_k > 0$ (improvement by exchange)

The complexity is dominated by sorting $\Longrightarrow$ O (n log(n)) $\Longleftrightarrow \dfrac{w_k}{p_k} > \dfrac{w_h}{p_h}$

Total Weighted Completion Time

Use of precedence constraints: $1 \mid prec \mid \Sigma\, w_j C_j$
Only independent chains are allowed at first!

Chain of jobs 1, ... , k

l* satisfies

δ factor of this chain

$$\frac{\sum_{j=1}^{l^*} w_j}{\sum_{j=1}^{l^*} p_j} = \max_{1 \le l \le k} \left(\frac{\sum_{j=1}^{l} w_j}{\sum_{j=1}^{l} p_j} \right)$$

l* determines the δ-factor of the chain 1, ... , k

Total Weighted Completion Time with Chains

Whenever the machine is available, select among the remaining chains the one with the highest δ-factor.

Schedule all jobs from this chain without interruption until the job that determines the δ-factor.

Proof concept

There is an optimal schedule that processes
all jobs 1, ... , l^* in succession +
Pairwise interchange of chains

Example: Total Weighted Completion Time with Chains

Consider the following two chains:

$$1 \rightarrow 2 \rightarrow 3 \rightarrow 4$$

and

$$5 \rightarrow 6 \rightarrow 7$$

The weights and processing times of the jobs are given in the following table.

jobs	1	2	3	4	5	6	7
w_j	6	18	12	8	8	17	18
p_j	3	6	6	5	4	8	10

Example: Total Weighted Completion Time with Chains

- δ-factor of first chain $(6+18)/(3+6)=\frac{24}{9}$ $\longrightarrow$ Job 2
- δ-factor of second chain $(8+17)/(4+8)=\frac{25}{12}<\frac{24}{9}$ $\longrightarrow$ Job 6
  - Jobs 1 and 2 are scheduled first.
- δ-factor of remaining part of first chain $\frac{12}{6}<\frac{25}{12}$ $\longrightarrow$ Job 3
  - Jobs 5 and 6 are scheduled next.
- $\frac{w_7}{p_7}=\frac{18}{10}<\frac{12}{6}$ $\longrightarrow$ Job 3 is scheduled next.
- $\frac{w_4}{p_4}=\frac{8}{5}<\frac{18}{10}$ $\longrightarrow$ Job 7 is scheduled next and finally job 4

Other Total Completion Time Problems

- $1 \mid prec \mid \Sigma\, w_j C_j$ is strongly NP hard for arbitrary precedence constraints.
- $1 \mid r_j, prmp \mid \Sigma\, w_j C_j$ is strongly NP hard.
  - The WSPT (remaining processing time) rule is not optimal.
    <u>Example:</u> Select another job that can be completed before the release date of the next job.
- $1 \mid r_j, prmp \mid \Sigma\, C_j$ is easy.
- $1 \mid r_j \mid \Sigma\, C_j$ is strongly NP hard.
- $1 \mid\mid \Sigma\, w_j (1 - e^{-rC_j})$ can be solved optimally with the Weighted Discounted Shortest Processing Time first (WDSPT) rule:

$$\frac{w_j \cdot e^{-rp_j}}{1 - e^{-rp_j}}$$

Maximum Cost

- General problem: $1 \mid prec \mid h_{max}$
  - $h_j(t)$: nondecreasing cost function
  - $h_{max} = \max(h_1(C_1), \ldots, h_n(C_n))$
- Backward dynamic programming algorithm
  - makespan $C_{max} = \Sigma p_j$
  - J: set of all jobs already scheduled (backwards) in

    $$[C_{max} - \sum_{j \in J} p_j, C_{max}]$$
  - $J^c = \{1, \ldots, n\} \setminus J$: set of jobs still to be scheduled
  - $J' \subseteq J^c$: set jobs that can be scheduled under consideration of precedence constraints.

Algorithm: Minimizing Maximum Cost

- Step 1 Set $J = \varnothing$, let $J^c = \{1, \dots, n\}$ and J' be the set of all jobs with no successors.
- Step 2 Let $j^* \in J'$ be such that

$$h_{j^*}\left(\sum_{j \in J^c} p_j\right) = \min_{j \in J'}\left(h_j\left(\sum_{k \in J^c} p_k\right)\right)$$

  Add j* to J.
  Delete j* from J^c.
  Modify J' to represent the new set of schedulable jobs.

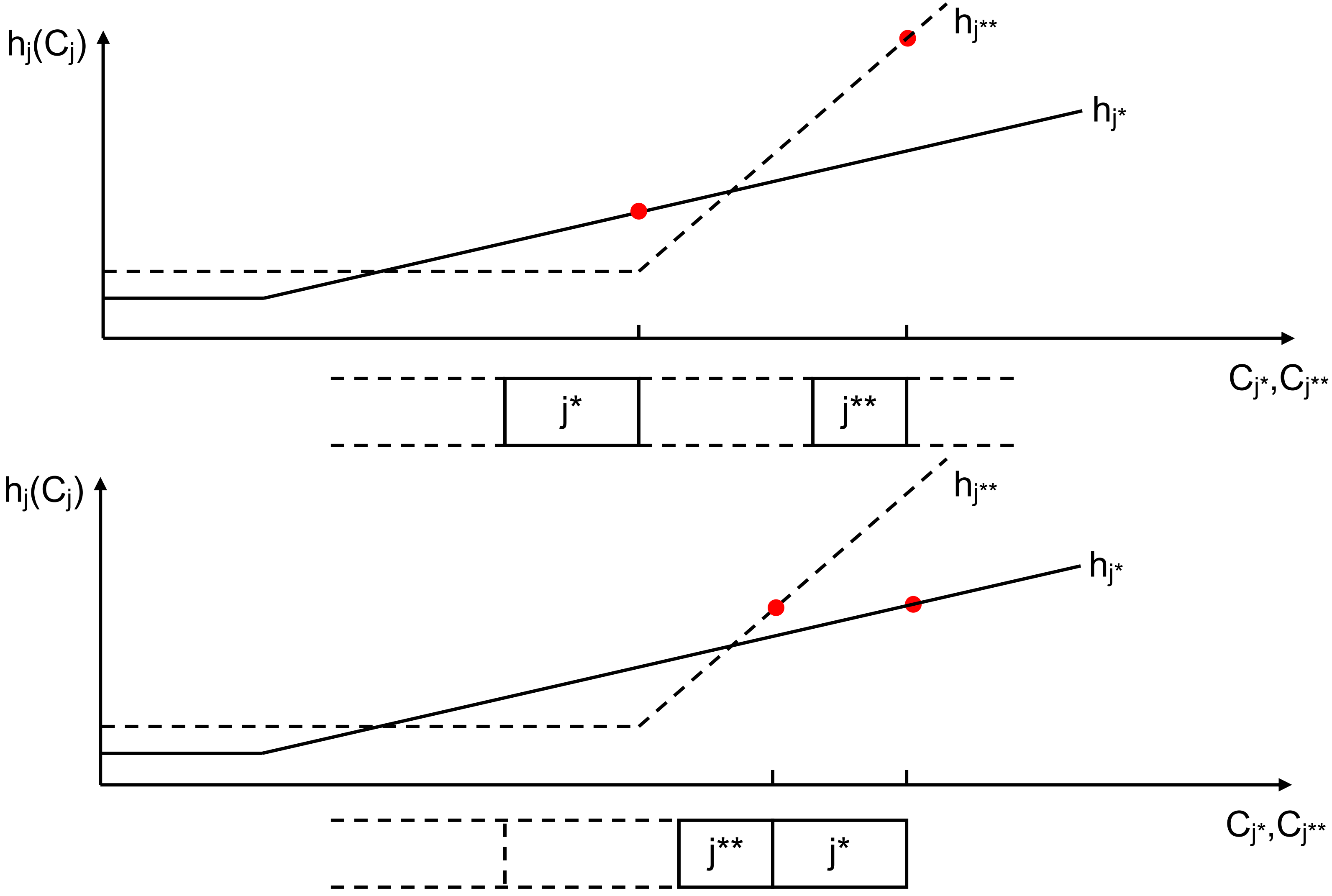
- Step 3 If $J^c = \varnothing$ then STOP otherwise go to Step 2.

This algorithm yields an optimal schedule for $1 \mid prec \mid h_{max}$.

Minimizing Maximum Cost: Proof of Optimality

- Assumption: The optimal schedule S_{opt} and the schedule S of the previous algorithm are identical at positions k+1,... , n.
- At position k with completion time t, there is job j** in S_{opt} and job j* with $h_{j^{**}}(t) \geq h_{j^*}(t)$ in S.
  - ➡ Job j* is at position k‘ < k in S_{opt}.
- Create schedule S’ by removing job j* in S_{opt} and putting it at position k.
  - ➡ $h_j(C_j)$ does not increase for all jobs {1, ... , n} \ {j*}.
  - ➡ $h_{j^*}(t) \leq h_{j^{**}}(t) \leq h_{max}(S_{opt})$ holds due to the algorithm.
- Therefore, schedule S’ is optimal as $h_{max}(S') \leq h_{max}(S_{opt})$ holds.
  - ➡ An optimal schedule and schedule S are identical at positions k, k+1, ..., n.

Minimizing Maximum Cost: Proof of Optimality



Minimizing Maximum Cost: Example

jobs	1	2	3
p_j	2	3	5
$h_j(C_j)$	$1 + C_j$	$1.2\ C_j$	10

- $C_{max} = 2+3+5 = 10$

- $h_3(10) = 10 < h_1(10) = 11 < h_2(10) = 12$
  - ➔ Job 3 is scheduled last.

- $h_2(10 - p_3) = h_2(5) = 6 = h_1(5)$
  - ➔ **Optimal schedules 1,2,3 and 2,1,3**

Maximum Lateness

- $1 \,||\, L_{max}$ is a special case of $1 \mid prec \mid h_{max}$.
  - ➡ $h_j = C_j - d_j \longrightarrow$ *Earliest Due Date first*

- $1 \mid r_j \mid L_{max}$ is strongly NP complete.

Proof:

Reduction of 3-Partition to $1 \mid r_j \mid L_{max}$

integers $a_1, \ldots , a_{3t}, b$

➡ $n = 4t - 1$ jobs

$$\frac{b}{4} < a_j < \frac{b}{2} \qquad \sum_{j=1}^{3t} a_j = t \cdot b$$

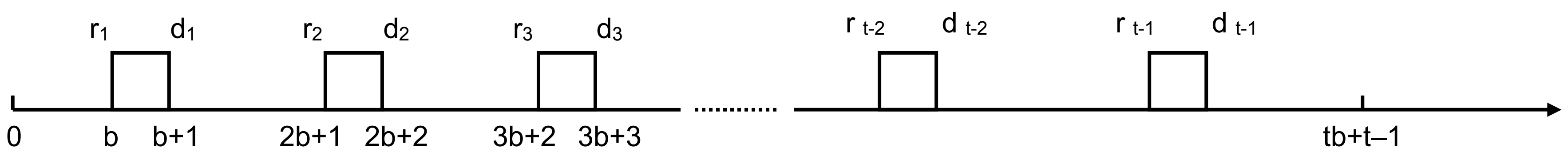
$r_j = j \cdot b + (j - 1)$, $\quad p_j = 1$, $\quad d_j = j \cdot b + j$, $\quad \forall\, j = 1, \ldots, t - 1$

$r_j = 0$, $\quad p_j = a_{j-t+1}$, $\quad d_j = t \cdot b + (t - 1)$, $\quad \forall\, j = t, \ldots, 4t - 1$

Maximum Lateness

$L_{max} = 0$ if every job $j \in \{1, \dots, t-1\}$ can be processed from r_j to $r_j + p_j = d_j$ and all other jobs can be partitioned over t intervals of length b.

- ➡ 3 – Partition has a solution.



$1 \mid r_j \mid L_{max}$ is strongly NP – hard.

Optimal Solution for 1 | r_j | L_{max}

Optimal solution for 1 | r_j | L_{max}: Branch and bound method

- Tree with n+1 levels

- Level 0: 1 root node
- Level 1: n nodes:
  A specific job scheduled at the first position of the schedule.
- Level 2: n(n-1) nodes:
  from each node of level 1 there are n – 1 edges to nodes of level 2:
  a second specific job scheduled at the second position of the schedule.

- n!/(n-k)! nodes at level k:
  each node specifies the first k positions of the schedule.

Optimal Solution for $1 \mid r_j \mid L_{max}$

Assumption: $$r_{j_k} \geq \min_{l \in J} \left(\max(t, r_l) + p_l \right)$$

J: jobs that are not scheduled at the father node of level k – 1

t: makespan at the father node of level k – 1

Job j_k need not be considered at a node of level k with this specific father at level k – 1.

Finding bounds:

If there is a better schedule than the one generated by a branch then the branch can be ignored.

$1 \mid r_j, prmp \mid L_{max}$ can be solved by the

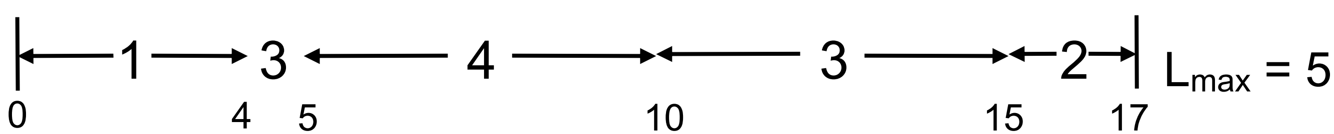
preemptive *Earliest Due Date* (EDD) *first* rule.

- This produces a nondelay schedule.
- The resulting schedule is optimal if it is nonpreemptive.

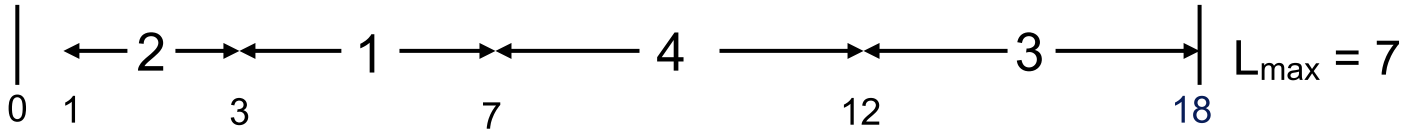
Branch and Bound Applied to Minimizing Maximum Lateness

jobs	1	2	3	4
p_j	4	2	6	5
r_j	0	1	3	5
d_j	8	12	11	10

- Level 1 (1, ?, ?, ?) (2, ?, ?, ?) (3, ?, ?, ?) (4, ?, ?, ?)
  - Disregard (3, ?, ?, ?) and (4, ?, ?, ?) as job 2 can be completed at r_3 and r_4 at the latest.
- Lower bound for node (1, ?, ?, ?):



- Lower bound for node (2, ?, ?, ?):



Branch and Bound Applied to Minimizing Maximum Lateness

- Lower bound for node (1, 2, ?, ?):
  - 1, 2, 3, 4 (nonpreemptive, $L_{max} = 6$)
  - Disregard (2, ?, ?, ?)
- Lower bound for node (1, 3, ?, ?):
  - 1, 3, 4, 2 (nonpreemptive, $L_{max} = 5$) ← optimal
  - Disregard (1, 2, ?, ?)
- Lower bound for node (1, 4, ?, ?):
  - 1, 4, 3, 2 (nonpreemptive, $L_{max} = 5$) ← optimal

A similar approach can be used for $1 \mid r_j, prec \mid L_{max}$.

- The additional precedence constraints may lead to less nodes in the branch and bound tree.

Number of Tardy Jobs: $1 \mid\mid \Sigma U_j$

- The jobs are partitioned into 2 sets.
  set A: all jobs that meet their due dates
  - These jobs are scheduled according to the EDD rule.

  set B: all jobs that do not meet their due dates
  - These jobs are not scheduled!

- The problem is solved with a forward algorithm.
  J: Jobs that are already scheduled
  J^d: Jobs that have been considered and are assigned to set B
  J^c: Jobs that are not yet considered

Algorithm for Solving $1 \mid\mid \Sigma U_j$

- Step 1 — Set $J = \varnothing$, $J^d = \varnothing$, and $J^c = \{1, \dots, n\}$.
- Step 2 — Let j* denote the job that satisfies $d_{j^*} = \min_{j \in J^c} (d_j)$
  Add j* to J.
  Delete j* from J^c.
  Go to Step 3.
- Step 3 — If $\sum_{j \in J} p_j \leq d_{j^*}$ then go to Step 4,
  otherwise
  let k* denote the job which satisfies $p_{k^*} = \max_{j \in J} (p_j)$
  Delete k* from J.
  Add k* to J^d.
- Step 4 — If $J^c = \varnothing$ then STOP, otherwise go to Step 2.

$1 \mid\mid \Sigma U_j$: Proof of Optimality

The computational complexity is determined by sorting $O(n \cdot \log(n))$.

We assume that all jobs are ordered by their due dates.

➡ $d_1 \le d_2 \le \ldots \le d_n$

J_k is a subset of jobs $\{1, \ldots, k\}$ such that

(I) it has the maximum number N_k of jobs in $\{1, \ldots, k\}$ completed by their due dates,

(II) of all sets with N_k jobs in $\{1, \ldots, k\}$ completed by their due dates J_k is the set with the smallest total processing time.

➡ J_n corresponds to an optimal schedule.

$1 \,||\, \Sigma\, U_j$: Proof of Optimality

Proof by induction

The claim is correct for k=1.

- We assume that it is correct for an arbitrary k.

1. Job k+1 is added to set J_k and it is completed by its due date.
   - $J_{k+1} = J_k \cup \{k+1\}$ and $|J_{k+1}| = N_k+1=N_{k+1}$.

2. Job k+1 is added to set J_k and it is not completed on time.
   - The job with the longest processing time is deleted
   - $N_{k+1} = N_k$
   - The total processing time of J_k is not increased.
   - No other subset of {1, ... ,k+1} can have N_k on-time completions and a smaller processing time.

$1 \mid\mid \Sigma U_j$: Example

jobs	1	2	3	4	5
p_j	7	8	4	6	6
d_j	9	17	18	19	21

- Job 1 fits: $J_1 = \{1\}$
- Job 2 fits: $J_2 = \{1, 2\}$
- Job 3 does not fit: $J_3 = \{1, 3\}$
- Job 4 fits: $J_4 = \{1, 3, 4\}$
- Job 5 does not fit: $J_5 = \{3, 4, 5\}$
  - ➡ **schedule order 3, 4, 5, (1, 2)** **$\Sigma U_j = 2$**

$1 \mid\mid \Sigma w_j U_j$ is NP-hard in the ordinary sense.

- ➡ This is even true if all due dates are the same: $1 \mid d_j = d \mid \Sigma w_j U_j$
- ➡ Then the problem is equivalent to the knapsack problem.

$1 \mid\mid \Sigma w_j U_j$: Example

- Heuristic approach: Jobs are ordered by the WSPT rule (w_j / p_j).

  ➜ The ratio $\dfrac{\sum w_j U_j(\text{WSPT})}{\sum w_j U_j(\text{OPT})}$ may be very large.

- Example: WSPT: 1, 2, 3 $\Sigma w_j U_j = 89$
  OPT: 2, 3, 1 $\Sigma w_j U_j = 12$

jobs	1	2	3
p_j	11	9	90
w_j	12	9	89
d_j	100	100	100

Total Tardiness

$1 \,||\, \Sigma\, T_j$: NP hard in the ordinary sense.

- There is a pseudo polynomial time algorithm to solve the problem.

Properties of the solution:

1. If $p_j \leq p_k$ and $d_j \leq d_k$ holds then there exists an optimal sequence in which job j is scheduled before job k.
   - This is an **Elimination criterion** or **Dominance result**.
     A large number of sequences can be disregarded.
     $\Rightarrow$ New precedence constraints are introduced.
     $\Rightarrow$ The problem becomes easier.

Total Tardiness

2 problem instances with processing times $p_1, ..., p_n$

First instance: $d_1, ..., d_n$

C'_k: latest possible completion time of job k in an optimal sequence (S')

Second instance:

$d_1, ..., d_{k-1}, \max\{d_k, C'_k\} \; d_{k+1}, ..., d_n$

S'': an optimal sequence

C_j'': completion time of job j in sequence S''

2. Any sequence that is optimal for the second instance is optimal for the first instance as well.

Total Tardiness

<u>Assumption</u>: $d_1 \le \ldots \le d_n$ and $p_k = \max(p_1, \ldots, p_n)$

- ➡ k^{th} smallest due date has the largest processing time.

3. There is an integer δ, $0 \le \delta \le n - k$ such that there is an optimal sequence S in which job k is preceded by all other jobs j with $j \le k+\delta$ and followed by all jobs j
   with $j > k+\delta$.

- ➡ An optimal sequence consists of

1. jobs 1, ..., k-1, k+1, ..., $k+\delta$ in some order
2. job k
3. jobs $k+\delta+1$, ... , n in some order

The completion time of job k is given by $C_k(\delta) = \sum_{j \le k+\delta} p_j$.

Minimizing Total Tardiness

- $J(j, l, k)$: all jobs in the set $\{j, ..., l\}$ with a processing time $\leq p_k$ but job k is not in $J(j, l, k)$.
- $V(J(j, l, k), t)$ is the total tardiness of $J(j, l, k)$ in an optimal sequence that starts at time t.

Algorithm: Minimizing Total Tardiness

Initial conditions: $V(\varnothing, t) = 0$

$V(\{j\}, t) = \max(0, t + p_j - d_j)$

Recursive relation:

$$V(J(j,l,k),t) = \min_{\delta} \left(V(J(j,k'+\delta,k'),t) + \max(0, C_{k'}(\delta) - d_{k'}) + V(J(k'+\delta+1,l,k'), C_{k'}(\delta)) \right)$$

where k‘ is such that $p_{k'} = \max(p_{j'} \mid j' \in J(j,l,k))$

Optimal value function: $V(\{1, ..., n\},0)$

Minimizing Total Tardiness

- At most $O(n^3)$ subsets $J(j, l, k)$ and $\Sigma\, p_j$ points in t
  - ➜ $O(n^3 \cdot \Sigma\, p_j)$ recursive equations

- Each recursion takes $O(n)$ time
  - ➜ Running time $O(n^4\, \Sigma\, p_j)$

polynomial in n

pseudo polynomial

Algorithm [PTAS Minimizing Total Tardiness]

Minimizing Total Tardiness Example

jobs	1	2	3	4	5
p_j	121	79	147	83	130
d_j	260	266	266	336	337

- k=3 (largest processing time) $\Rightarrow 0 \leq \delta \leq 2 = 5 - 3$

- $$V(\{1, 2, ..., 5\}, 0) = \min \begin{cases} V(J(1, 3, 3), 0) + 81 + V(J(4, 5, 3), 347), & \delta=0 \\ V(J(1, 4, 3), 0) + 164 + V(J(5, 5, 3), 430), & \delta=1 \\ V(J(1, 5, 3), 0) + 294 + V(\varnothing, 560), & \delta=2 \end{cases}$$

- V(J(1, 3, 3), 0) = 0 for sequences 1, 2 and 2, 1

- V(J(4, 5, 3), 347) = 347 +83 – 336 +347 + 83 +130 – 337 = 317 for sequence 4, 5

Minimizing Total Tardiness Example

- V(J(1, 4, 3), 0) = 0 for sequences 1, 2, 4 and 2, 1, 4
- V(J(5, 5, 3), 430) = 430 + 130 – 337 =223
- V(J(1, 5, 3), 0) = 76 for sequences 1, 2, 4, 5 and 2, 1, 4, 5

$$\Rightarrow V(\{1, ..., 5\}, 0) = \min \left\{ \begin{array}{l} 0 + 81 + 317 \\ 0 + 164 + 223 \\ 76 + 294 + 0 \end{array} \right\} = 370$$

1, 2, 4, 5, 3 and 2, 1, 4, 5, 3 are optimal sequences.

Total Weighted Tardiness

- $1 \mid\mid \Sigma\, w_j T_j$ is strongly NP complete.
  - Proof by reduction of 3 – Partition

- Dominance result
  If there are two jobs j and k with $d_j \leq d_k$, $p_j \leq p_k$ and $w_j \geq w_k$,
  then there is an optimal sequence in which job j appears before job k.

- The Minimizing Total Tardiness algorithm can solve this problem if $w_j \leq w_k$ holds for all jobs j and k with $p_j \geq p_k$.

Total Tardiness
An Approximation Scheme

For NP – hard problems, it is frequently interesting to find in polynomial time a (approximate) solution that is close to optimal.

Fully Polynomial Time Approximation Scheme A for $1 \mid\mid \Sigma T_j$:

$$\sum T_j(A) \leq (1+\varepsilon) \underbrace{\sum T_j(OPT)}_{\text{optimal schedule}}$$

The running time is bounded by a polynomial (fixed degree) in n and $1/\varepsilon$.

Total Tardiness
An Approximation Scheme

a) n jobs can be scheduled with 0 total tardiness iff (if and only if) the EDD schedule has 0 total tardiness.

➡ $T_{max}(EDD) \leq \sum T_j(OPT) \leq \sum T_j(EDD) \leq n \cdot T_{max}(EDD)$

maximum tardiness of any job in the EDD schedule

Total Tardiness
An Approximation Scheme

b) V(J,t): Minimum total tardiness of job subset J assuming processing starts at t.

- There is a time t* such that

  $V(J, t)=0$ for $t \leq t^*$ and

  $V(J, t)>0$ for $t > t^*$

  $\Rightarrow V(J, t^* + \delta) \geq \delta$ for $\delta \geq 0$

- The pseudo polynomial algorithm is used to compute V(J, t) for

  $$\max\{0,t^*\} \leq t \leq n \cdot T_{max}(EDD)$$

- Running time bound $O(n^5 \cdot T_{max}(EDD))$

Total Tardiness
An Approximation Scheme

c) Rescale $p'_j = \lfloor p_j / K \rfloor$ and $d'_j = d_j / K$ with some factor *K*.

S is the optimal sequence for rescaled problem.

$\sum T_j^*(S)$ is the total tardiness of sequence *S* for processing times $K \cdot p'_j \le p_j$ and due dates d_j.

$\sum T_j(S)$ is the total tardiness of sequence *S* for $p_j < K \cdot (p'_j + 1)$ and d_j.

$$\Rightarrow \sum T_j^*(S) \le \sum T_j(OPT) \le \sum T_j(S) < \sum T_j^*(S) + K \cdot \frac{n(n+1)}{2}$$

$$\sum T_j(S) - \sum T_j(OPT) < K \cdot \frac{n(n+1)}{2}$$

Select $K = \frac{2\varepsilon}{n(n+1)} \cdot T_{max}(EDD)$

$$\Rightarrow \sum T_j(S) - \sum T_j(OPT) \le \varepsilon \cdot T_{max}(EDD)$$

PTAS Minimizing Total Tardiness

Algorithm: PTAS Minimizing Total Tardiness

- Step 1 — Apply EDD and determine T_{max}.
  If $T_{max} = 0$, then $\sum T_j = 0$ and EDD is optimal; STOP.
  Otherwise set

$$K = \left(\frac{2\varepsilon}{n(n+1)}\right) T_{max}(EDD)$$

- Step 2 — Rescale processing times and due dates as follows:

$$p'_j = \lfloor p_j / K \rfloor \qquad d'_j = \frac{d_j}{K}$$

- Step 3 — Apply Algorithm Minimizing Total Tardiness to the rescaled data.

Running time complexity: $O(n^5 \cdot T_{max}(EDD)/K) = O(n^7/\varepsilon)$

PTAS Minimizing Total Tardiness Example

jobs	1	2	3	4	5
p_j	1210	790	1470	830	1300
d_j	1996	2000	2660	3360	3370

- Optimal sequence 1,2,4,5,3 with total tardiness 3700.
  - Verified by dynamic programming
- $T_{max}(EDD)=2230$
  - If ε is chosen 0.02 then we have K=2.973.
- Optimal sequences for the rescaled problem: 1,2,4,5,3 and 2,1,4,5,3.
  - Sequence 2,1,4,5,3 has total tardiness 3704 for the original data set.
  - $\sum T_j(2,1,4,5,3) \leq 1.02 \cdot \sum T_j(1,2,4,5,3)$

Total Earliness and Tardiness

Objective $\Sigma E_j + \Sigma T_j$

- This problem is harder than total tardiness.
- A special case is considered with $d_j = d$ for all jobs j.

Properties of the special case

- No idleness between any two jobs in the optimal schedule
  - The first job does not need to start at time 0.
- Schedule S is divided into 2 disjoint sets

early completion	late completion
$C_j \leq d$	$C_j > d$
job set J_1	job set J_2

Total Earliness and Tardiness

- Optimal Schedule:
  Early jobs (J_1) use *Longest Processing Time first* (LPT)
  Late jobs (J_2) use *Shortest Processing Time first* (SPT)

- There is an optimal schedule such that one job completes exactly at time **d.**

Proof: Job j* starts before and completes after **d.**

If $|J_1| \leq |J_2|$ then
shift schedule to the left until j* completes at **d.**

If $|J_1| > |J_2|$ then
shift schedule to the right until j* starts at **d.**

Minimizing Total Earliness and Tardiness with a Loose Due Date

Assume that the first job can start its processing after $t = 0$ and $p_1 \geq p_2 \geq \ldots \geq p_n$ holds.

- Step 1 Assign job 1 to set J_1.
  Set $k = 2$.
- Step 2 Assign job k to set J_1 and job $k + 1$ to set J_2 or vice versa.
- Step 3 If $k+2 \leq n - 1$, set $k = k+2$ and go to Step 2
  If $k+2 = n$, assign job n to either set J_1 or set J_2 and STOP.
  If $k+2 = n+1$, all jobs have been assigned;
  STOP.

Minimizing Total Earliness and Tardiness with a Tight Due Date

The problem becomes NP-hard if job processing must start at time 0 and the schedule is nondelay.

It is assumed that $p_1 \geq p_2 \geq \dots \geq p_n$ holds.

- Step 1 — Set $\tau_1 = d$ and $\tau_2 = \Sigma\, p_j - d$.
  Set $k = 1$.

- Step 2 — If $\tau_1 \geq \tau_2$, assign job k to the first unfilled position in the sequence and set $\tau_1 = \tau_1 - p_k$.
  If $\tau_1 < \tau_2$, assign job k to the last unfilled position in the sequence and set $\tau_2 = \tau_2 - p_k$.

- Step 3 — If $k < n$, set $k = k + 1$ and go to Step 2.
  If $k = n$, STOP.

Minimizing Total Earliness and Tardiness with a Tight Due Date

- 6 jobs with d = 180

jobs	1	2	3	4	5	6
p_j	106	100	96	22	20	2

- Applying the heuristic yields the following results.

τ_1	τ_2	**Assignment**	**Sequence**
180	166	Job 1 Placed First	1xxxxx
74	166	Job 2 Placed Last	1xxxx2
74	66	Job 3 Placed First	13xxx2
-22	66	Job 4 Placed Last	13xx42
-22	44	Job 5 Placed Last	13x542
-22	24	Job 6 Placed Last	136542

Minimizing Total Earliness and Tardiness

- Objective $\Sigma\, w'E_j + \Sigma\, w''T_j$ with $d_j = d$.
  - All previous properties and algorithms for $\Sigma\, E_j + \Sigma\, T_j$ can be generalized using the difference of w' and w''.

- Objective $\Sigma\, w_j'E_j + \Sigma\, w_j''T_j$ with $d_j = d$.
  - The LPT/SPT sequence is not necessarily optimal in this case.
  - WLPT and WSPT are used instead.
    - The first part of the sequence is ordered in increasing order of w_j / p_j.
    - The second part of the sequence is ordered in decreasing order of w_j / p_j.

Minimizing Total Earliness and Tardiness

- Objective $\Sigma\, w'E_j + \Sigma\, w''T_j$ with different due dates
  - ➡ The problem is NP – hard.
  - a) Sequence of the jobs
  - b) Idle times between the jobs
    - ➡ dependent optimization problems

- Objective $\Sigma\, w_j'E_j + \Sigma\, w_j''T_j$ with different due dates
  - ➡ The problem is NP – hard in the strong sense.
    - It is more difficult than total weighted tardiness.

If a predetermined sequence is given then the timing can be determined in polynomial time.

Primary and Secondary Objectives

A scheduling problem is usually solved with respect to the **primary** objective. If there are several optimal solutions, the **best of those solutions** is selected according to the **secondary** objective.

$$\alpha \mid \beta \mid \gamma_1 \text{ (opt)}, \gamma_2$$

primary objective

secondary objective

- We consider the problem $1 \mid\mid \Sigma C_j$ (opt), L_{max}.
  - All jobs are scheduled according to SPT.
  - If several jobs have the same processing time EDD is used to order these jobs.
    - SPT/EDD rule

Reversal of Priorities

- We consider the problem with reversed priorities:

$$1 \;||\; L_{max}\; (opt), \Sigma C_j$$

➡ L_{max} is determined with EDD.
➡ $z := L_{max}$

Transformation of this problem:

$$\bar{d}_j = d_j + z$$

new deadline old due dates

Reversal of Priorities

- After the transformation, both problems are equivalent.
  - The optimal schedule minimizes $\Sigma\, C_j$ and guarantees that each job completes by its deadline.
  - In such a schedule, job k is scheduled last if

$$\bar{d}_k \geq \sum_{j=1}^{n} p_j \quad \text{and} \quad p_k \geq p_l$$

for all l such that $\bar{d}_k \geq \sum_{j=1}^{n} p_j$ hold.

- **Proof:** If the first condition is not met, the schedule will miss a deadline.
  - A pairwise exchange of job l and job k (not necessarily adjacent) decreases $\Sigma\, C_j$ if the second condition is not valid for l and k.

Minimizing Total Completion Time with Deadlines

- Step 1 Set $k = n$, $\tau = \sum_{j=1}^{n} p_j$, $J^c = \{1, \ldots, n\}$
- Step 2 Find k^* in J^c such that $\bar{d}_{k^*} \geq \tau$ and $p_{k^*} \geq p_l$

  for all jobs l in J^c such that $\bar{d}_l \geq \tau$.
- Step 3 Decrease k by 1.
  Decrease τ by p_{k^*}
  Delete job k^* from J^c.
- Step 4 If $k \geq 1$ go to Step 2, otherwise STOP.

The optimal schedule is always nonpreemptive even if preemptions are allowed.

Minimizing Total Completion Time with Deadlines

jobs	1	2	3	4	5
p_j	4	6	2	4	2
$\bar{d}_j$	10	12	14	18	18

$\tau = 18 \Rightarrow d_4 = d_5 = 18 \geq \tau$

$p_4 = 4 > 2 = p_5$

- Last job : 4
- $\tau = 18 - p_4 = 14 \Rightarrow d_3 = 14 \geq 14$ $\quad d_5 = 18 \geq 14$

  $p_5 = 2 = p_3$
- Either job can go in the now last position : 3
- $\tau = 14 - p_3 = 12 \Rightarrow d_5 = 18 \geq 12$ $\quad d_2 = 12 \geq 12$

  $p_2 = 6 > 2 = p_5$
- Next last job: 2
- $\tau = 12 - p_2 = 6 \Rightarrow d_5 = 18 \geq 6$ $\quad d_1 = 10 \geq 12$

  $p_1 = 4 > 2 = p_5$
- **Sequence: 5 1 2 3 4**

Multiple Objectives

In a generalized approach, multiple objectives are combined in a linear fashion instead of using a priority ordering.

- Objectives: γ_1, γ_2

- Problem with a weighted sum of two (or more) objectives:

$$1 \mid \beta \mid \Theta_1 \gamma_1 + \Theta_2 \gamma_2$$

- The weights are normalized: $\Theta_1 + \Theta_2 = 1$

Pareto-Optimal Schedule

A schedule is called pareto-optimal if it is not possible to decrease the value of one objective without increasing the value of the other.

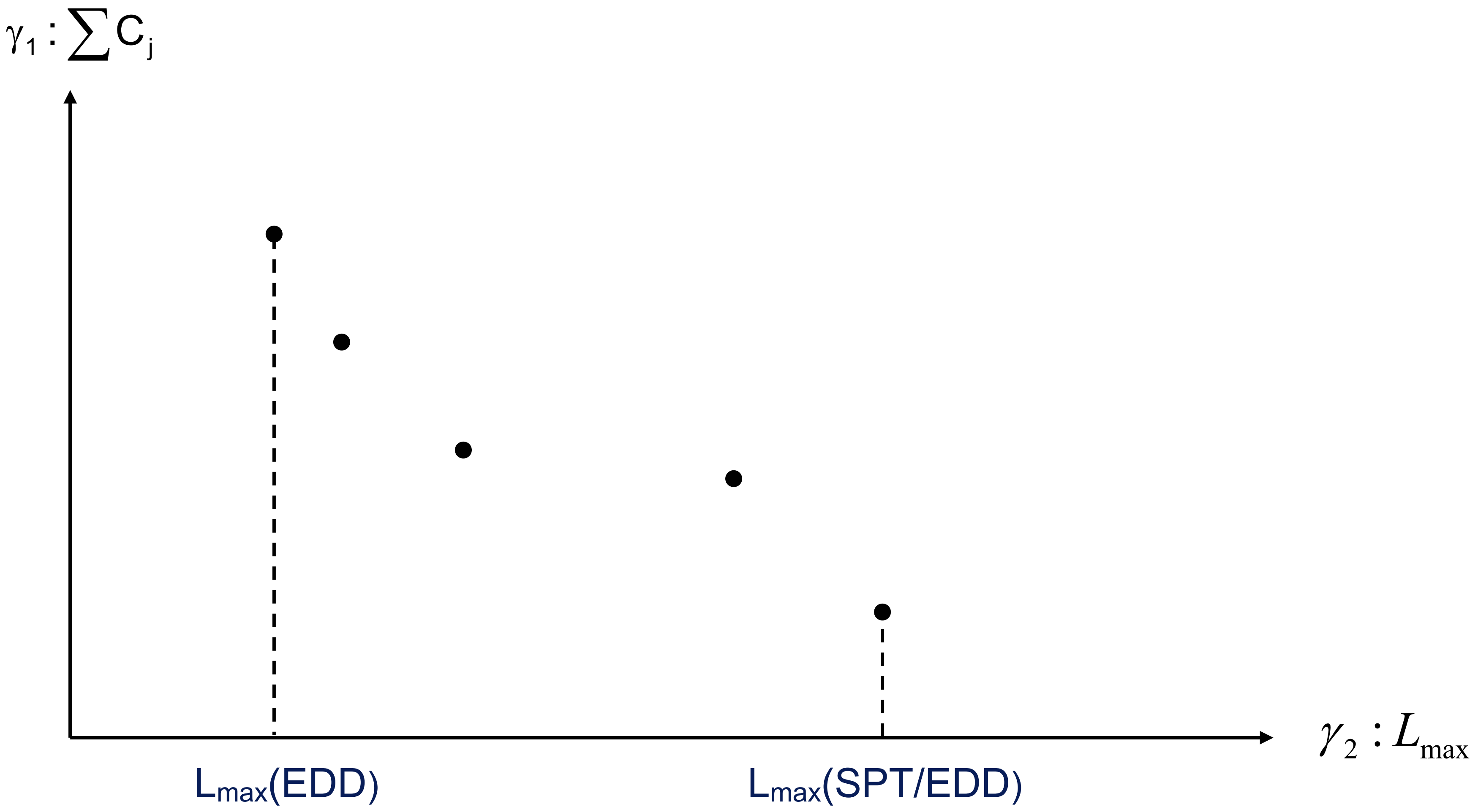
$\Theta_1 \to 0$ and $\Theta_2 \to 1$

➡ $1 \mid \beta \mid \Theta_1\gamma_1 + \Theta_2\gamma_2 \to 1 \mid \beta \mid \gamma_2(\mathrm{opt}), \gamma_1$

$\Theta_1 \to 1$ and $\Theta_2 \to 0$

➡ $1 \mid \beta \mid \Theta_1\gamma_1 + \Theta_2\gamma_2 \to 1 \mid \beta \mid \gamma_1(\mathrm{opt}), \gamma_2$

Pareto-Optimal Schedule



Pareto-Optimal Solutions

Generation of all pareto-optimal solutions

Find a new pareto-optimal solution:

Determine the optimal schedule for L_{max}.

Determine the minimum increment of L_{max} to decrease the minimum $\Sigma\ C_j$.

Similar to the minimization of the total weighted completion time with deadlines

Start with the EDD schedule,
end with the SPT/EDD schedule.

Pareto-Optimal Solutions

- Step 1 Set $r = 1$
  Set $L_{max} = L_{max}(EDD)$ and $\bar{d}_j = d_j + L_{max}$.
- Step 2 Set $k = n$ and $J^c = \{1, \ldots, n\}$.
  Set $\tau = \sum_{j=1}^{n} p_j$ and $\delta = \tau$.
- Step 3 Find j^* in J^c such that $\bar{d}_{j^*} \geq \tau$ and $p_{j^*} \geq p_l$
  for all jobs in J^c such that $\bar{d}_l \geq \tau$.
  Put job j^* in position k of the sequence.
- Step 4 If there is no job l such that $\bar{d}_l < \tau$ and $p_l > p_{j^*}$,
  go to Step 5.
  Otherwise find j^{**} such that $\tau - \bar{d}_{j^{**}} = \min_l (\tau - \bar{d}_l)$
  for all l such that $\bar{d}_l < \tau$ and $p_l > p_{j^*}$.
  Set $\delta^{**} = \tau - \bar{d}_{j^{**}}$.
  If $\delta^{**} < \delta$, then $\delta = \delta^{**}$.

Pareto-Optimal Solutions

- Step 5 Decrease k by 1.
  Decrease τ by p_{j^*}.
  Delete job j^* from J^c.
  If $k \geq 1$, go to Step 3
  Otherwise go to Step 6.

- Step 6 Set $L_{max} = L_{max} + \delta$.
  If $L_{max} > L_{max}$(SPT/EDD), then STOP.
  Otherwise set $r = r + 1$, $\bar{d}_j = \bar{d}_j + \delta$, and go to Step 2.

Maximum number of pareto – optimal points

- $n(n-1)/2 = O(n^2)$

Complexity to determine one pareto – optimal schedule

- $O(n \log(n))$
- Total complexity $O(n^3 \log(n))$

Pareto-Optimal Solutions

jobs	1	2	3	4	5
p_j	1	3	6	7	9
d_j	30	27	20	15	12

- EDD sequence 5,4,3,2,1 $\Rightarrow L_{max}$ (EDD) = 2
  $c_3 = 22 \quad d_3=20$
- SPT/EDD sequence 1,2,3,4,5 $\Rightarrow L_{max}$ (SPT/EDD) =14
  $c_5 = 26 \quad d_5 = 12$

Pareto-Optimal Solutions

Iteration r	$(\sum C_j, L_{max})$	Pareto – optimal schedule	current $\tau + \delta$	δ
1	96, 2	5,4,3,1,2	32 29 22 17 14	1
2	77, 3	1,5,4,3,2	33 30 23 18 15	2
3	75, 5	1,4,5,3,2	35 32 25 20 17	1
4	64, 6	1,2,5,4,3	36 33 26 21 18	2
5	62, 8	1,2,4,5,3	38 35 28 23 20	3
6	60, 11	1,2,3,5,4	41 38 31 26 23	3
7	58, 14	1,2,3,4,5	44 41 34 29 26	Stop

- $1 \mid\mid \Theta_1 \sum w_j C_j + \Theta_2 L_{max}$
  Extreme points (WSPT/EDD and EDD) can be determined in polynomial time.
  - The problem with arbitrary weights Θ_1 and Θ_2 is NP – hard.

Parallel Machine Models

- A scheduling problem for parallel machines consists of 2 steps:
  - Allocation of jobs to machines
  - Generating a sequence of the jobs on a machine
- A minimal makespan represents a balanced load on the machines.
- Preemption may improve a schedule even if all jobs are released at the same time.
- Most optimal schedules for parallel machines are nondelay.
  - Exception: $R_m \mid\mid \sum C_j$
- General assumption for all problems: $p_1 \geq p_2 \geq \ldots \geq p_n$

$P_m \,||\, C_{max}$

The problem is NP-hard.

- $P_2 \,||\, C_{max}$ is equivalent to Partition.

Heuristic algorithm: Longest processing time first (LPT) rule

Whenever a machine is free, the longest job among those not yet processed is put on this machine.

- Upper bound: $$\frac{C_{max}(LPT)}{C_{max}(OPT)} \leq \frac{4}{3} - \frac{1}{3m}$$
- The optimal schedule $C_{max}(OPT)$ is not necessarily known but the following bound holds:

$$C_{max}(OPT) \geq \frac{1}{m}\sum_{j=1}^{n} p_j$$

Proof of the Bound

- If the claim is not true, then there is a counterexample with the smallest number *n* of jobs.

- The shortest job *n* in this counterexample is the last job to start processing (LPT) and the last job to finish processing.
  - If *n* is not the last job to finish processing, then deletion of *n* does not change C_{max} *(LPT)* while C_{max} *(OPT)* cannot increase.
  - A counter example with *n* – 1 jobs

- Under LPT, job n starts at time $C_{max}(LPT)-p_n$.
  - In time interval $[0, C_{max}(LPT) - p_n]$, all machines are busy.

$$C_{max}(LPT) - p_n \leq \frac{1}{m}\sum_{j=1}^{n-1} p_j$$

Proof of the Bound

$$C_{max}(LPT) \le p_n + \frac{1}{m}\sum_{j=1}^{n-1} p_j = p_n(1-\frac{1}{m}) + \frac{1}{m}\sum_{j=1}^{n} p_j$$

$$\frac{4}{3}-\frac{1}{3m} < \frac{C_{max}(LPT)}{C_{max}(OPT)} \le \frac{p_n(1-\frac{1}{m})}{C_{max}(OPT)} + \frac{\frac{1}{m}\sum_{j=1}^{n} p_j}{C_{max}(OPT)} \le \frac{p_n(1-1/m)}{C_{max}(OPT)} + 1$$

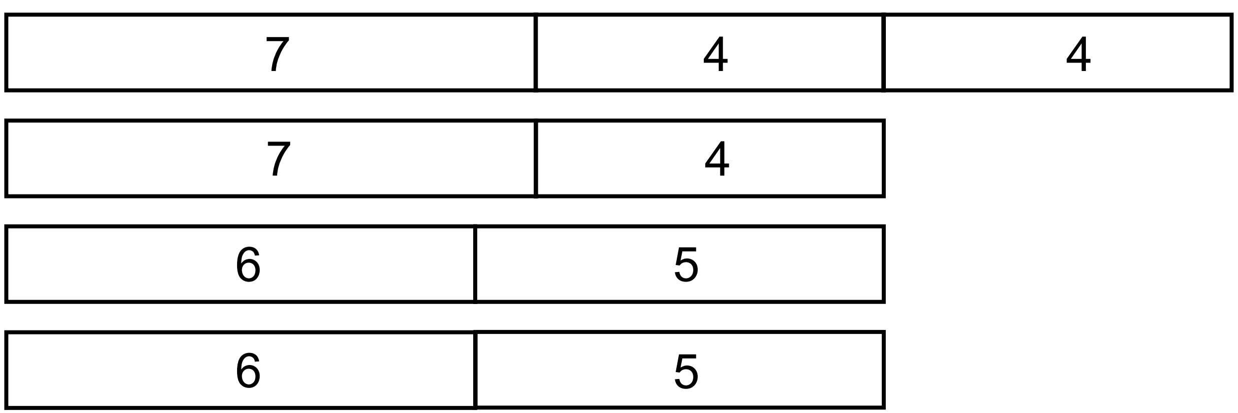
$$C_{max}(OPT) < 3p_n$$

At most two jobs are scheduled on each machine.
For such a problem, LPT is optimal.

A Worst Case Example for LPT

jobs	1	2	3	4	5	6	7	8	9
p_j	7	7	6	6	5	5	4	4	4

- 4 parallel machines
- $C_{max}(OPT) = 12 = 7+5 = 6+6 = 4+4+4$
- $C_{max}(LPT) = 15 = \quad (4/3 - 1/(3 \cdot 4)) \cdot 12$



Other Makespan Results

- Arbitrary nondelay schedule $$\frac{C_{max}(LIST)}{C_{max}(OPT)} \leq 2 - \frac{1}{m}$$

- $P_m \mid prec \mid C_{max}$ with $2 \leq m < \infty$ is strongly NP hard even for chains.

- Special case $m \geq n$: $P_\infty \mid prec \mid C_{max}$

a) Start all jobs without predecessor at time 0.

b) Whenever a job finishes, immediately start all its successors for which all predecessors have been completed.
   - Critical Path Method (CPM)
   - Project Evaluation and Review Technique (PERT)

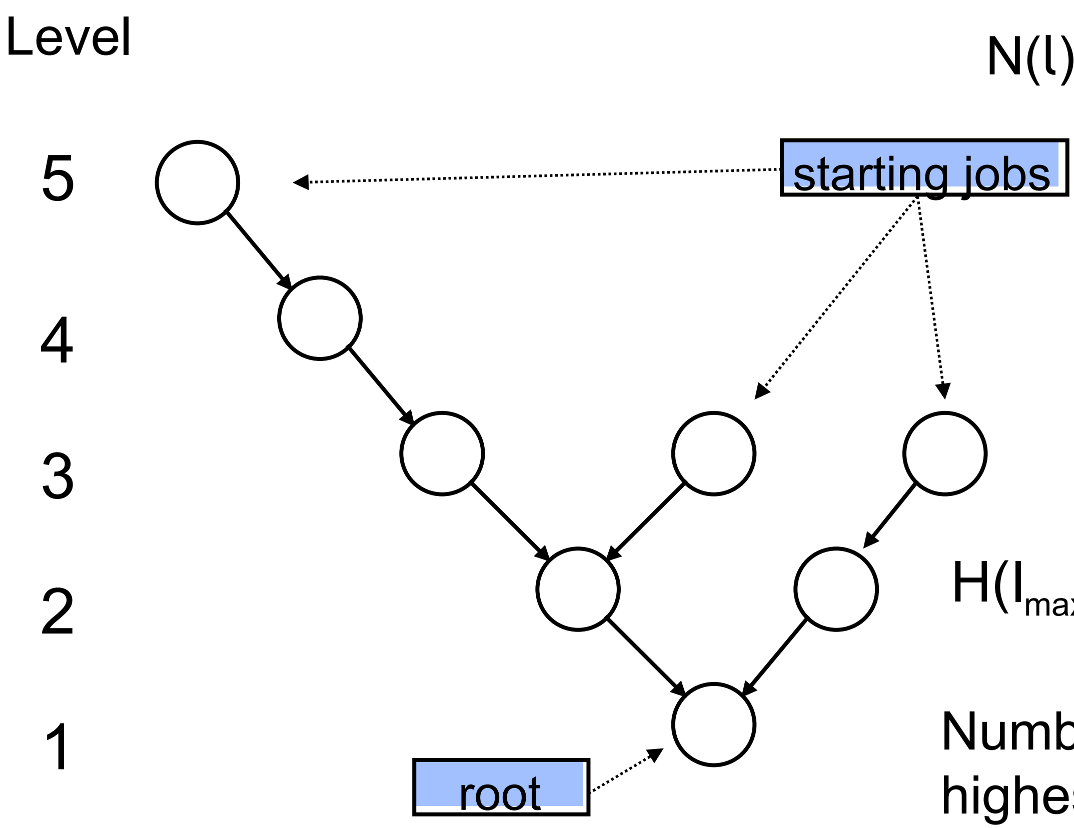
Heuristics Algorithms

- Critical Path (CP) rule
  - The job at the head of the longest string of jobs in the precedence constraints graph has the highest priority.
  - $P_m \mid p_j = 1, tree \mid C_{max}$ is solvable with the CP rule.
- Largest Number of Successors first (LNS)
  - The job with the largest total number of successors in the precedence constraints graph has the highest priority.
  - For intrees and chains, LNS is identical to the CP rule
  - LNS is also optimal for $P_m \mid p_j = 1, outtree \mid C_{max}$.
- Generalization for problems with arbitrary processing times
  - Use of the total amount of processing remaining to be done on the jobs in question.

$P_m \mid p_j = 1, \text{tree} \mid C_{max}$

highest level l_{max}

N(l) number of jobs at level l

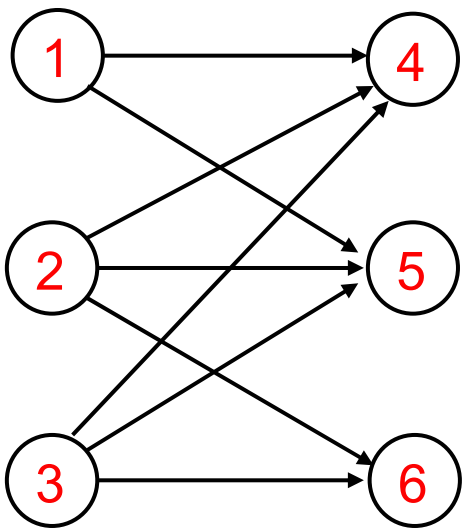


$$H(l_{max}+1-r)=\sum_{k=1}^{r} N(l_{max}+1-k)$$

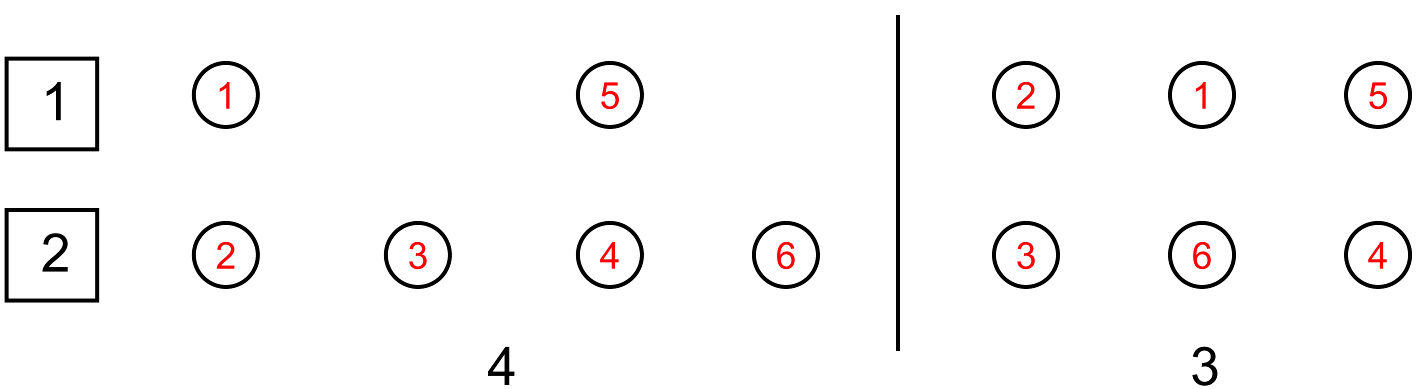
Number of nodes at the r highest levels

CP for $P_2|p_j=1,prec|C_{max}$

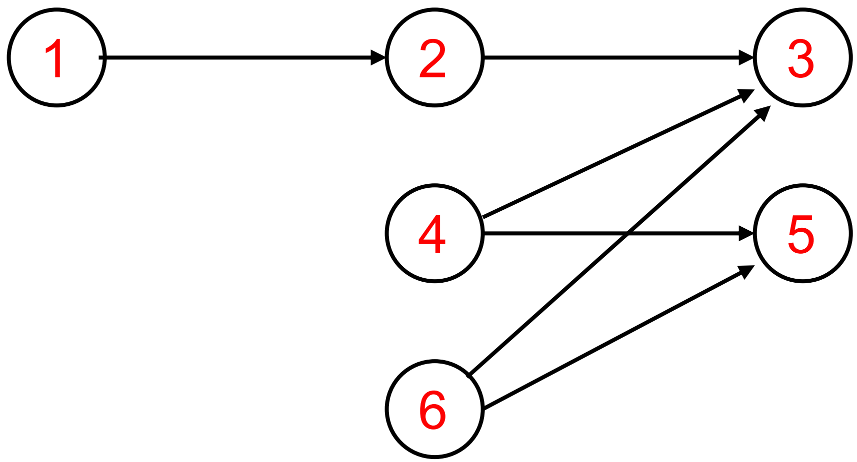
$$\frac{C_{max}(CPM)}{C_{max}(OPT)} \leq \frac{4}{3}$$ for two machines

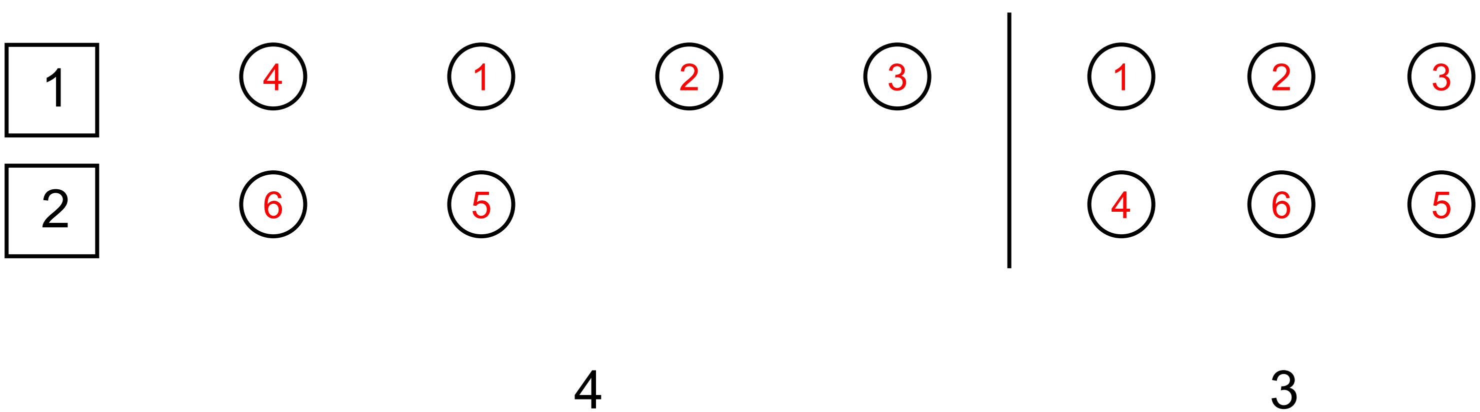


almost fully connected bipartite graph



LNS for $P_2|p_j=1,prec|C_{max}$





$P_m \mid p_j = 1, M_j \mid C_{max}$

A job can only be processed on subset M_j of the *m* parallel machines.

Here, the sets M_j are nested.

- Exactly 1 of 4 conditions is valid for jobs j and k.
  - M_j is equal to M_k ($M_j = M_k$)
  - M_j is a subset of M_k ($M_j \subset M_k$)
  - M_k is a subset of M_j ($M_j \supset M_k$)
  - M_j and M_k do not overlap. ($M_j \cap M_k = \emptyset$)

Every time a machine is freed, the job is selected that can be processed on the smallest number of machines.

- Least Flexible Job first (LFJ) rule
- LFJ is optimal for $P_2 \mid p_j = 1, M_j \mid C_{max}$ and for $P_m \mid p_j = 1, M_j \mid C_{max}$ when the M_j sets are nested (pairwise exchange).

$P_m \mid p_j = 1, M_j \mid C_{max}$

- Consider $P_4 \mid p_j = 1, M_j \mid C_{max}$ with eight jobs. The eight M_j sets are:
  - $M_1 = \{1,2\}$
  - $M_2 = M_3 = \{1,3,4\}$
  - $M_4 = \{2\}$
  - $M_5 = M_6 = M_7 = M_8 = \{3,4\}$

Machines	**1**	**2**	**3**	**4**
LFJ	1	4	5	6
	2		7	8
	3			
optimal	2	1	5	7
	3	4	6	8

- LFM (Least Flexible Machine) and LFM-LFJ do not guarantee optimality for this example either.

Makespan with Preemptions

Linear programming formulation for $P_m \mid prmp \mid C_{max}$

The variable x_{ij} represents the total time job j spends on machine i.

Minimize C_{max} subject to

$$\sum_{i=1}^{m} x_{ij} = p_j$$

processing time of job j

$$\sum_{i=1}^{m} x_{ij} \leq C_{max}$$

processing time of each job is less than makespan

$$\sum_{j=1}^{n} x_{ij} \leq C_{max}$$

processing on each machine is less than makespan

$$x_{ij} \geq 0$$

non-negative execution fragments

Makespan with Preemptions

- The solution of a linear program yields the processing of each job on each machine.
  - ➡ A schedule must be generated in addition.
- Lower bound: $$C_{max} \geq \max\left\{p_1, \sum_{j=1}^{n} p_j / m\right\} = C^*_{max}$$

Algorithm Minimizing Makespan with Preemptions

1. Nondelay processing of all jobs on a single machine without preemption $\Rightarrow$ makespan $\leq m \cdot C^*_{max}$
2. Cutting of this schedule into m parts
3. Execution of each part on a different machine

LRPT Rule

Longest Remaining Processing Time first (LRPT)

- Preemptive version of Longest Processing Time first (LPT)
- This method may generate an infinite number of preemptions.

Example: 2 jobs with $p_1 = p_2 = 1$ and 1 machine

The algorithm uses the time period ε.

- Time ε after the previous decision the situation is evaluated again.

The makespan of the schedule is 2 while the total completion time is $4 - \varepsilon$.

- The optimal (non preemptive) total completion time is 3.

The following proofs are based on a discrete time framework.

- Machines are only preempted at integer times.

Vector Majorization

Vector of remaining processing times at time t

$(p_1(t), p_2(t), \ldots\ldots\ldots. p_n(t)) = \bar{p}(t)$.

A vector $\bar{p}(t)$ majorizes a vector $\bar{q}(t)$, $\bar{p}(t) \geq_m \bar{q}(t)$, if

$$\sum_{j=1}^{k} p_{(j)}(t) \geq \sum_{j=1}^{k} q_{(j)}(t) \text{ holds for all } k = 1, \ldots, n.$$

$p_j(t)$ is the j^{th} largest element of $\bar{p}(t)$.

Example

Consider the two vectors $\bar{p}(t) = (4, 8, 2, 4)$ and $\bar{q}(t) = (3, 0, 6, 6)$.

Rearranging the elements within each vector and putting these in decreasing order results in vectors (8, 4, 4, 2) and (6, 6, 3, 0).

It can be easily verified that $\bar{p}(t) \geq_m \bar{q}(t)$.

LRPT Property

If $\bar{p}(t) \geq_m \bar{q}(t)$ then LRPT applied to $\bar{p}(t)$ results in a larger or equal makespan than obtained by applying LRPT to $\bar{q}(t)$.

Induction hypothesis: The lemma holds for all pairs of vectors with total remaining processing time less than or equal to $\sum_{j=1}^{n} p_j(t) - 1$ and $\sum_{j=1}^{n} q_j(t) - 1$, respectively.

Induction base: Vectors 1, 0, …, 0 and 1, 0, … 0.

After LRPT is applied for one time unit on $\bar{p}(t)$ and $\bar{q}(t)$, respectively, then we obtain at time t+1 the vectors $\bar{p}(t+1)$ and $\bar{q}(t+1)$ with

$$\sum_{j=1}^{n} p_j(t+1) \leq \sum_{j=1}^{n} p_j(t) - 1 \quad \text{and} \quad \sum_{j=1}^{n} q_j(t+1) \leq \sum_{j=1}^{n} q_j(t) - 1 \quad .$$

If $\bar{p}(t) \geq_m \bar{q}(t)$, then $\bar{p}(t+1) \geq_m \bar{q}(t+1)$.

Result of the LRPT Rule

LPRT yields an optimal schedule for $P_m \mid prmp \mid C_{max}$ in discrete time.

We consider only problems with more than m jobs remaining to be processed.

Induction hypothesis: The lemma holds for any vector $\bar{p}(t)$ with $\sum_{j=1}^{n} p_j(t) \leq N-1$.

We consider a vector $\bar{p}(t)$ with $\sum_{j=1}^{n} p_j(t) = N$.

If LRPT is not optimal for $\bar{p}(t)$, then another rule R must be optimal. R produces vector $\bar{q}(t+1)$ with $\bar{q}(t+1) \geq_m \bar{p}(t+1)$.

From time t+1 on, R uses LRPT as well due to our induction hypothesis.

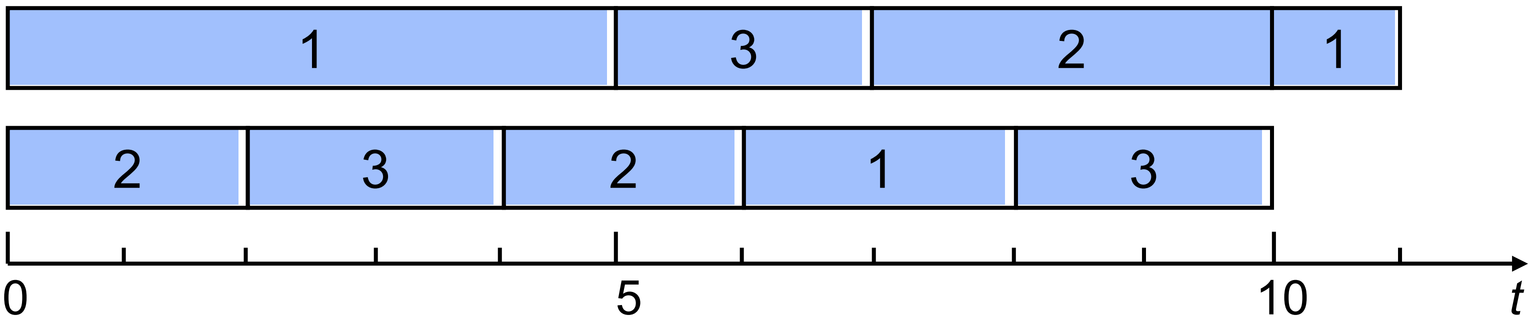
Due to the LRPT property, R cannot produce a smaller makespan than LRPT.

LRPT in Discrete Time

Consider two machines and three jobs 1, 2 and 3, with processing times 8, 7, and 6.

$C_{max}(LRPT)=C_{max}(OPT)=11$.

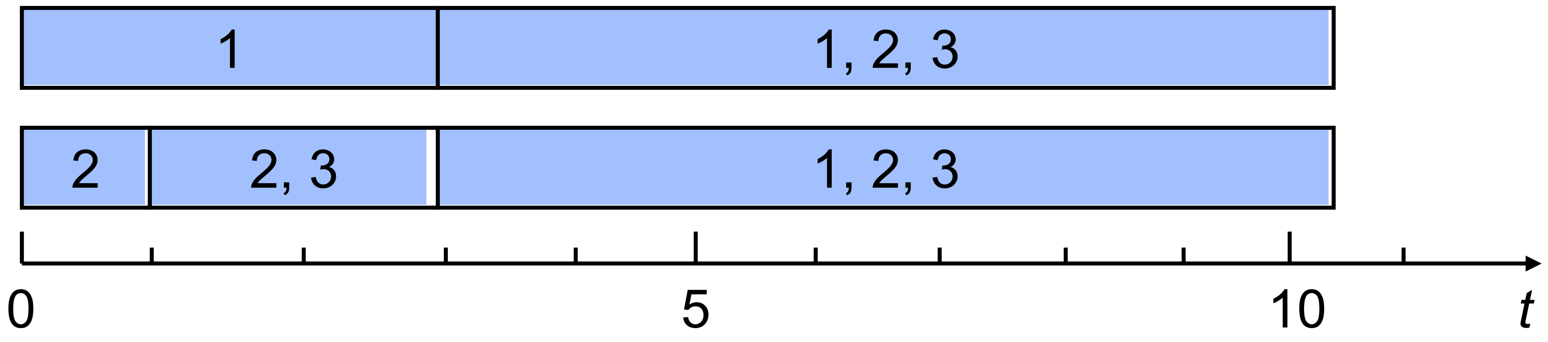
Remember: Ties are broken arbitrarily!



LRPT in Continuous Time

Consider the same jobs as in the previous example.
As preemptions may be done at any point in time, processor sharing takes place.
$C_{max}(LRPT)=C_{max}(OPT)=10.5$.

To prove that LRPT is optimal in continuous time, multiply all processing times by a very large integer K and let K go to ∞.



Lower Bound for Uniform Machines

$Q_m \mid prmp \mid C_{max}$

note n!

$$C_{max} \geq \max\left(\frac{p_1}{v_1}, \frac{p_1+p_2}{v_1+v_2}, \ldots, \frac{\sum_{j=1}^{m-1} p_j}{\sum_{j=1}^{m-1} v_j}, \frac{\sum_{j=1}^{n} p_j}{\sum_{j=1}^{m} v_j}\right)$$

for $v_1 \geq v_2 \geq \ldots \geq v_m$

Comparison: $P_m \mid prmp \mid C_{max}$

$$C_{max} \geq \max\left\{p_1, \sum_{j=1}^{n} p_j / m\right\}$$

LRPT-FM

- Longest Remaining Processing Time on the Fastest Machine first (LRPT – FM) yields an optimal schedule with infinitely many preemptions for $Q_m \mid prmp \mid C_{max}$:
  - At any point in time the job with the largest remaining processing time is assigned to the fastest machine.

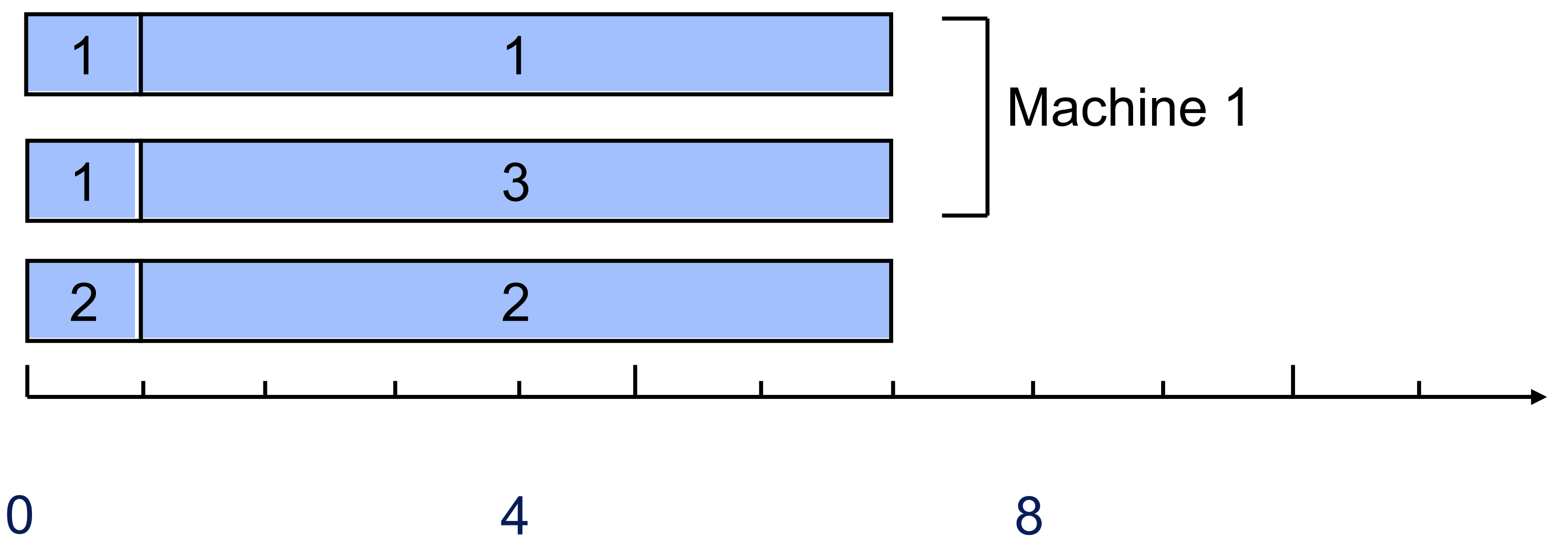
- Proof for a discrete framework with respect to speed and time
  - Replace machine j by v_j machines of unit speed.
  - A job can be processed on more than one machine in parallel, if the machines are derived from the same machine.

- Continuous time:
  - All processing times are multiplied by a large number K.
  - The speeds of the machines are multiplied by a large number V.

- The LRPT-FM rule also yields optimal schedules if applied to $Q_m \mid r_j, prmp \mid C_{max}$.

Application of LRPT-FM

- 2 machines with speed $v_1 = 2$, $v_2 = 1$
- 3 jobs with processing times 8, 7, and 6



$\sum C_j$ without Preemptions (1)

Different argument for SPT for total completion time without preemptions on a single machine.

$p_{(j)}$: processing time of the job in position j on the machine

$$\sum C_j = n \cdot p_{(1)} + (n-1) \cdot p_{(2)} + ... + 2 \cdot p_{(n-1)} + p_{(n)}$$

➡ $p_{(1)} \leq p_{(2)} \leq p_{(3)} \leq ..... \leq p_{(n-1)} \leq p_{(n)}$ must hold for an optimal schedule.

$\sum C_j$ without Preemptions (2)

SPT rule is optimal for $P_m \mid\mid \sum C_j$

- The proof is based on the same argument as for single machines.

- Dummy jobs with processing time 0 are added until n is a multiple of m.
  - ➡ The sum of the completion time has n additive terms with one coefficient each:

    m coefficients with value n/m
    m coefficients with value n/m – 1
    :
    m coefficients with value 1

- The SPT schedule is not the only optimal schedule.

$\sum w_j C_j$ without Preemptions

jobs	1	2	3
p_j	1	1	3
w_j	1	1	3

- 2 machines and 3 jobs
- With the given values any schedule is WSPT.
- If w_1 and w_2 are increased by ε
  - ➡ WSPT is not necessarily optimal.
- Tight approximation factor $\dfrac{\sum w_j C_j(\mathrm{WSPT})}{\sum w_j C_j(\mathrm{OPT})} \le \dfrac{1}{2}(1+\sqrt{2})$
- $P_m \,||\, \sum w_j\, C_j$ is NP hard.

$P_m \mid prec \mid \sum C_j$

- $P_m \mid prec \mid \sum C_j$ is strongly NP-hard.

- The CP rule is optimal for $P_m \mid p_j = 1, outtree \mid \sum C_j$.
  - The rule is valid if at most m jobs are schedulable.
  - t_1 is the last time the CP rule is not applied but rule R.
    - String 1 is the longest string not assigned at t_1
    - String 2 is the shortest of the longest strings assigned at t_1
    - C_1' is the completion time of the last job of string 1 under R
    - C_2' is the completion time of the last job of string 2 under R
  - If $C_1' \geq C_2'+1$ and machines are idle before $C_1' - 1$, then CP is better than R, otherwise CP is as good as R.

- However, the CP rule is not always optimal for intrees.

Other $\sum C_j$ Problems

- The LFJ rule is optimal for $P_m|p_j=1,M_j|\sum C_j$ when the M_j sets are nested.
- The $R_m||\sum C_j$ problem can be formulated as an integer program
  - Although linear integer programming is NP-hard this program has a special structure that allows a solution in polynomial time.
  - $x_{ikj}=1$ if job j is scheduled as the k^{th} to last job on machine i.
    - x_{ikj} are 0-1 integer variables.

Minimize $\sum_{i=1}^{m}\sum_{j=1}^{n}\sum_{k=1}^{n} k p_{ij} x_{ikj}$ subject to

$$\sum_{i=1}^{m}\sum_{k=1}^{n} x_{ikj} = 1 \qquad j = 1,\ldots, n$$

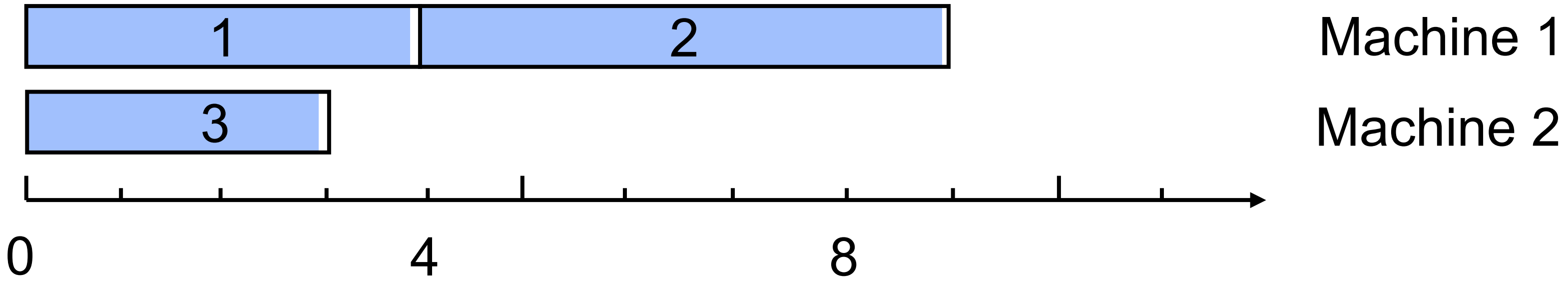
$$\sum_{j=1}^{n} x_{ikj} \le 1 \qquad i = 1,\ldots, m \text{ and } k = 1,\ldots, n$$

$$x_{ikj} \in \{0,1\} \qquad i = 1,\ldots, m,\ k = 1,\ldots, n, \text{ and } j = 1,\ldots, n$$

Example $R_m||\sum C_j$

jobs	1	2	3
P_{1j}	4	5	3
p_{2j}	8	9	3

- 2 machines and 3 jobs
- The optimal solution corresponds to $x_{121}=x_{112}=x_{213}=1$. All other x_{ikj} are 0. The optimal schedule is not nondelay.



$\sum C_j$ with Preemptions (1)

The nonpreemptive SPT rule is also optimal for $P_m|prmp|\sum C_j$.

$Q_m|prmp|\sum C_j$ can be solved by the

Shortest Remaining Processing Time on the Fastest Machine (SRPT-FM) rule.

- Useful lemma: There is an optimal schedule with $C_j \le C_k$ when $p_j \le p_k$ for all j and k. (Proof by pairwise exchange)
- Under SRPT-FM, we have $C_n \le C_{n-1} \le \ldots \le C_1$.
- Assumption: There are n machines.
  - If there are more jobs than machines, then machines with speed 0 are added.
  - If there are more machines than jobs, then the slowest machines are not used.

$\sum C_j$ with Preemptions (2)

$$v_1 C_n = p_n$$

$$v_2 C_n + v_1(C_{n-1} - C_n) = p_{n-1}$$

$$v_3 C_n + v_2(C_{n-1} - C_n) + v_1(C_{n-2} - C_{n-1}) = p_{n-2}$$

$$\vdots$$

$$v_n C_n + v_{n-1}(C_{n-1} - C_n) + v_1(C_1 - C_2) = p_1$$

Adding these equations yields

$$v_1 C_n = p_n$$

$$v_2 C_n + v_1 C_{n-1} = p_n + p_{n-1}$$

$$v_3 C_n + v_2 C_{n-1} + v_1 C_{n-2} = p_n + p_{n-1} + p_{n-2}$$

$$\vdots$$

$$v_n C_n + v_{n-1} C_{n-1} + \ldots + v_1 C_1 = p_n + p_{n-1} + \ldots + p_1$$

$\sum C_j$ with Preemptions (3)

Let S' be an optimal schedule with $C'_n \leq C'_{n-1} \leq ... \leq C'_1$ (see the lemma).
Then we have $C'_n \geq p_n/v_1 \Rightarrow v_1C'_n \geq p_n$.

The amount of processing on jobs n and n –1 is upper bounded by
$(v_1 + v_2)C'_n + v_1(C'_{n-1} - C'_n)$. $\Rightarrow v_2C'_n + v_1C'_{n-1} \geq p_n + p_{n-1}$
Similarly, we obtain

$$v_kC'_n + v_{k-1}C'_{n-1} + ... + v_1C'_{n-k+1} \geq p_n + p_{n-1} + ... + p_{n-k+1}$$

This yields

$$v_1C'_n \geq v_1C_n$$
$$v_2C'_n + v_1C'_{n-1} \geq v_2C_n + v_1C_{n-1}$$
$$\vdots$$
$$v_nC'_n + v_{n-1}C'_{n-1} + ... + v_1C'_1 \geq v_nC_n + v_{n-1}C_{n-1} + ... + v_1C_1$$

$\sum C_j$ with Preemptions (4)

We want to transform this system of inequalities into a new system such that

- inequality i is multiplied by $\alpha_i \geq 0$ and
- the sum of all those transformed inequalities yields $\sum C'_j \geq \sum C_j$.
- The proof is complete, if those α_i exists.
- α_i must satisfy

$$\begin{array}{rcl}
v_1\alpha_1 + v_2\alpha_2 + \ldots + v_n\alpha_n & = & 1 \\
v_1\alpha_2 + v_2\alpha_3 + \ldots + v_{n-1}\alpha_n & = & 1 \\
\vdots & & \\
v_1\alpha_n & = & 1
\end{array}$$

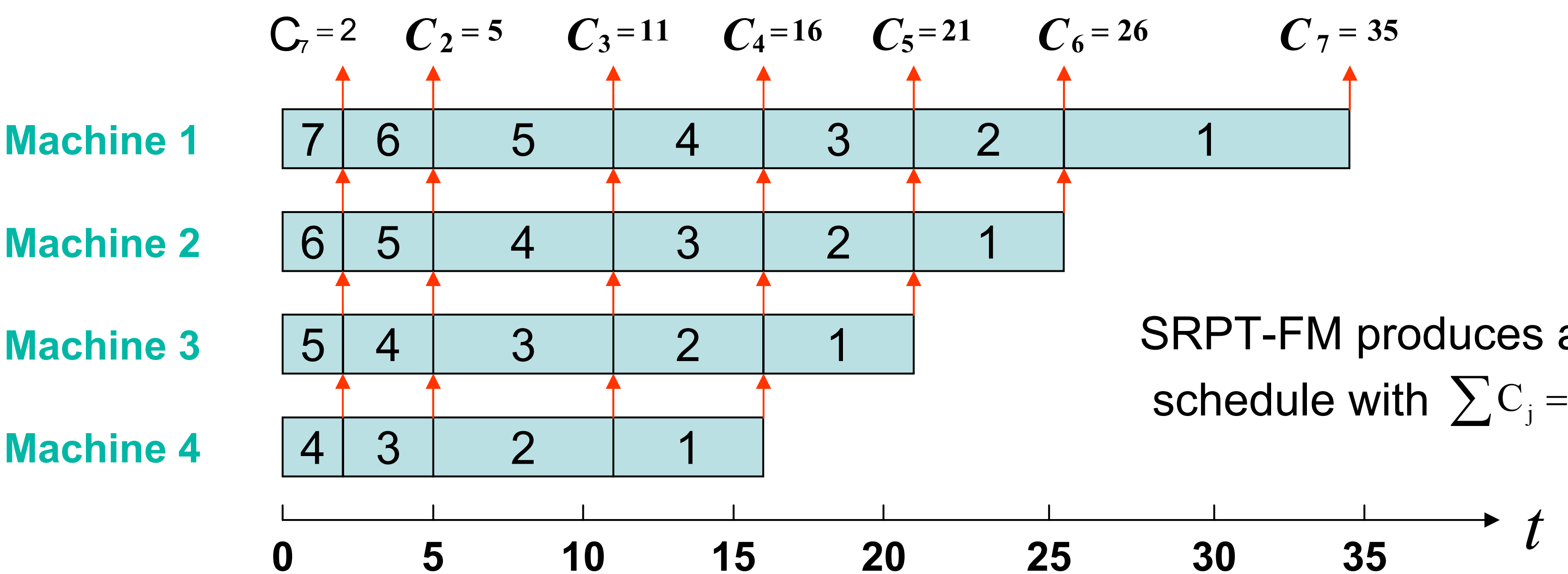
Those α_i exists as $v_1 \geq v_2 \geq \ldots \geq v_n$ holds.

Application of the SRPT-FM Rule

machines	1	2	3	4
v_i	4	2	2	1

jobs	1	2	3	4	5	6	7
p_i	61	46	45	40	34	16	8

Preemptions are only allowed at integer points in time.



SRPT-FM produces an optimal schedule with $\sum C_j = 116$

Due – Date Related Objectives

$P_m \,||\, C_{max} \propto P_m \,||\, L_{max}$ (all due dates 0)

- ➡ The problem is NP-hard.

$Q_m \mid prmp \mid L_{max}$

Assume $L_{max} = z$

- ➡ $C_j \leq d_j + z$
- ➡ set $\bar{d}_j = d_j + z$ (hard deadline)
- ➡ Hard deadlines are release dates in the reversed problem.
- ➡ Finding a schedule for this problem is equivalent to solving $Q_m \mid r_j, prmp \mid C_{max}$
- ➡ If all jobs in the reverse problem “finish” at a time not smaller than 0, then there exists a schedule for $Q_m \mid prmp \mid L_{max}$ with $L_{max} \leq z$.
- ➡ The minimum value for z can be found by a simple search.

Example P2 | prmp | L_{max}

jobs	1	2	3	4
d_j	9	8	5	4
p_j	8	3	3	3

Is there a feasible schedule with $L_{max} = 0$? ($\bar{d}_j = d_j$)

jobs	1	2	3	4
r_j	0	1	4	5
p_j	8	3	3	3

Is there a feasible schedule with $C_{max} \leq 9$?

Flow Shops

- Each job must follow the same route.
  - There is a sequence of machines.
- There may be limited buffer space between neighboring machines.
  - The job must sometimes remain in the previous machine: **Blocking**.
- The main objective in flow shop scheduling is the makespan.
  - It is related to utilization of the machines.
- If the First-come-first-served principle is in effect, then jobs cannot pass each other.
  - **Permutation flow shop**

Unlimited Intermediate Storage

- Permutation Schedule $j_1, j_2, \ldots, j_n$

$$C_{i,j_1} = \sum_{l=1}^{i} p_{l,j_1} \qquad i = 1, \ldots, m$$

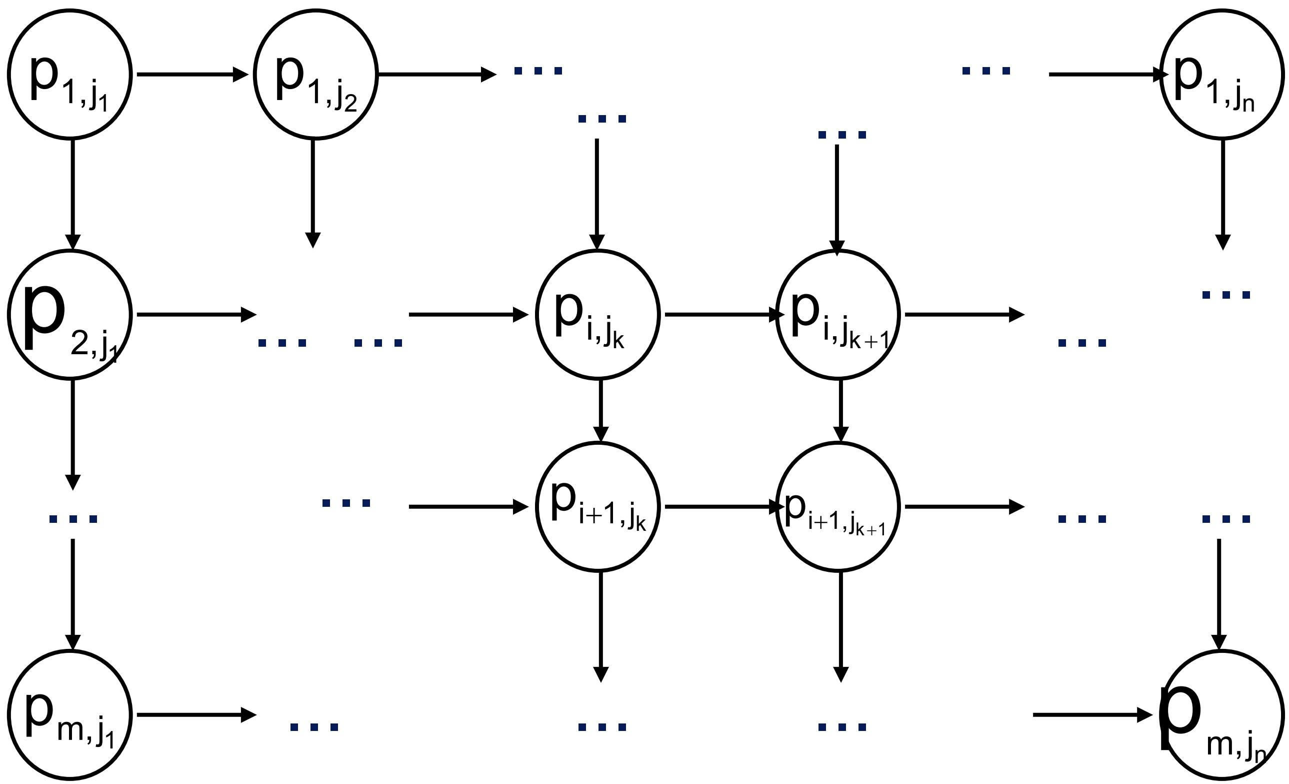
$$C_{1,j_k} = \sum_{l=1}^{k} p_{1,j_l} \qquad k = 1, \ldots, n$$

$$C_{i,j_k} = \max(C_{i-1,j_k}, C_{i,j_{k-1}}) + p_{i,j_k}$$

$$i = 2, \ldots, m \quad k = 2, \ldots, n$$

- There is always an optimal schedule without job sequence changes in the first two and last two machines.
  - ➡ F2|| C_{max} and F3|| C_{max} do not require a job sequence change in some optimal schedule.

Directed Graph for $F_m|prmu|C_{max}$

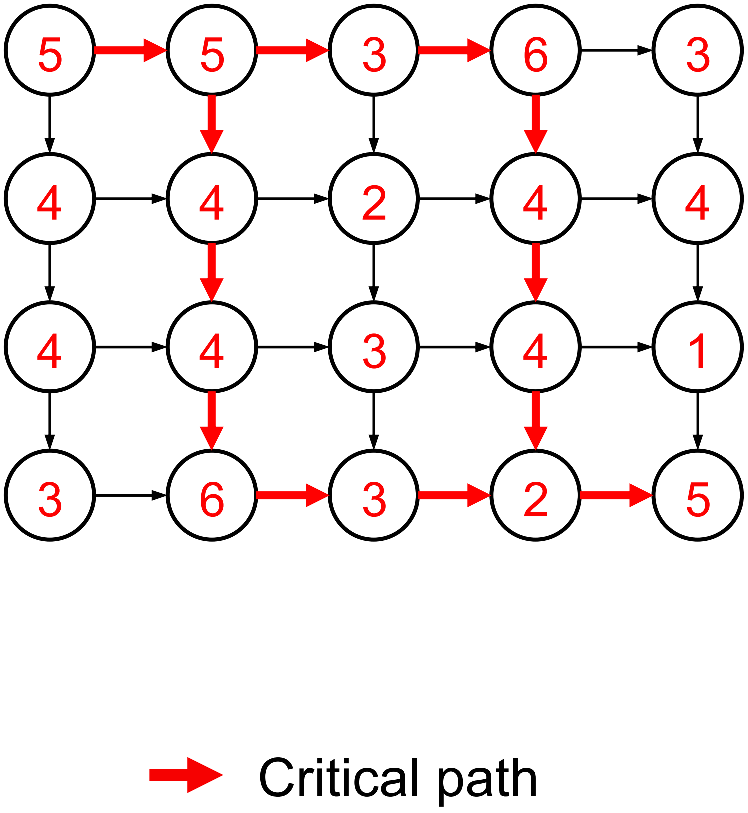


Example $F4|prmu|C_{max}$

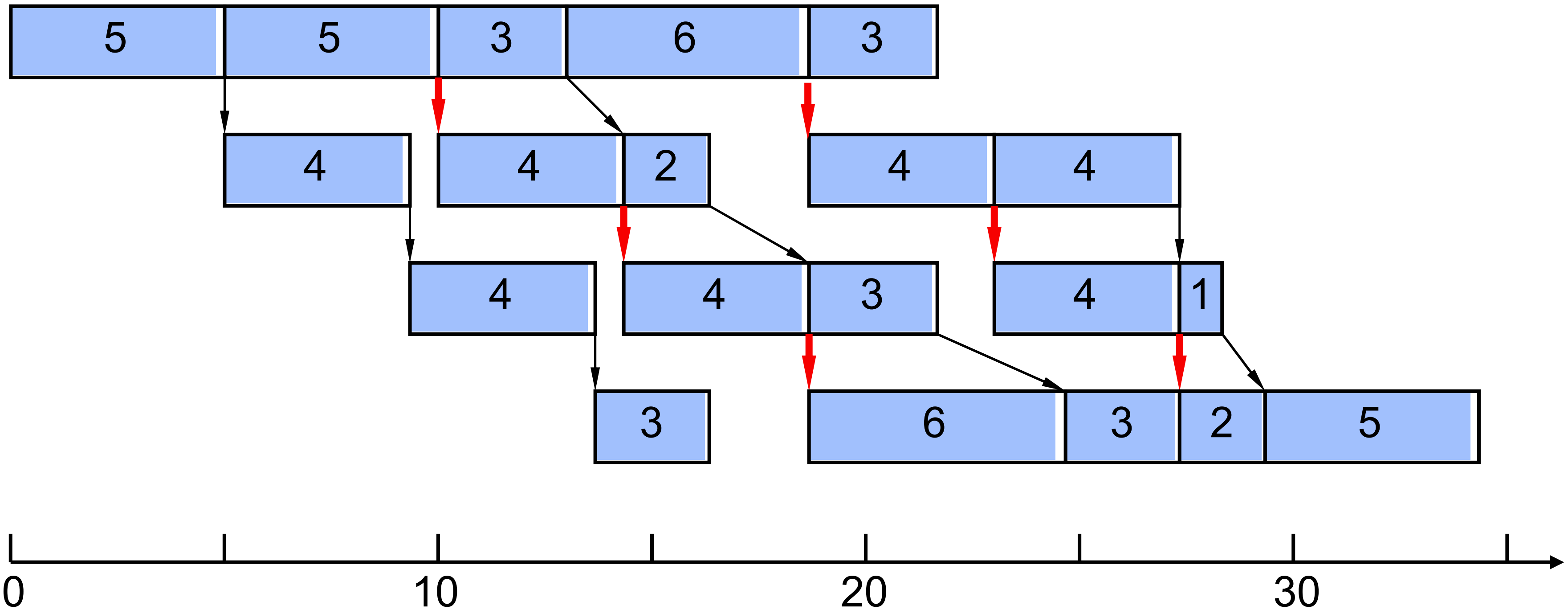
5 jobs on 4 machines with the following processing times

jobs	j_1	j_2	j_3	j_4	j_5
p_{1,j_k}	5	5	3	6	3
p_{2,j_k}	4	4	2	4	4
p_{3,j_k}	4	4	3	4	1
p_{4,j_k}	3	6	3	2	5

Directed Graph in the Example



Gantt Chart in the Example



Reversibility

Two *m* machine permutation flow shops with *n* jobs are considered with $p_{ij}^{(1)} = p^{(2)}_{m+1-i,j}$.

- $p_{ij}^{(1)}$ and $p_{ij}^{(2)}$ denote the processing times of job j in the first and the second flow shop, respectively.

Sequencing the jobs according to permutation $j_1, \ldots, j_n$ in the first flow shop produces the same makespan as permutation $j_n, \ldots, j_1$ in the second flow shop.

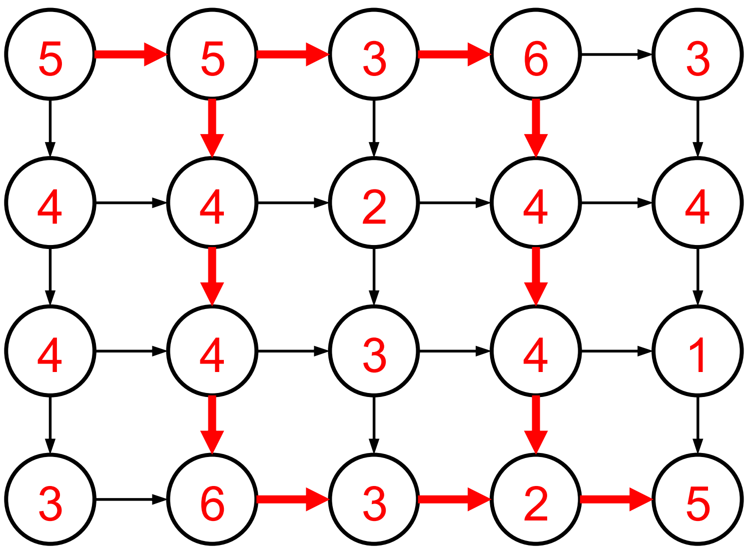
- The makespan does not change if the jobs traverse the flow shop in the opposite direction in reverse order (**Reversibility**).

Example Reversibility (1)

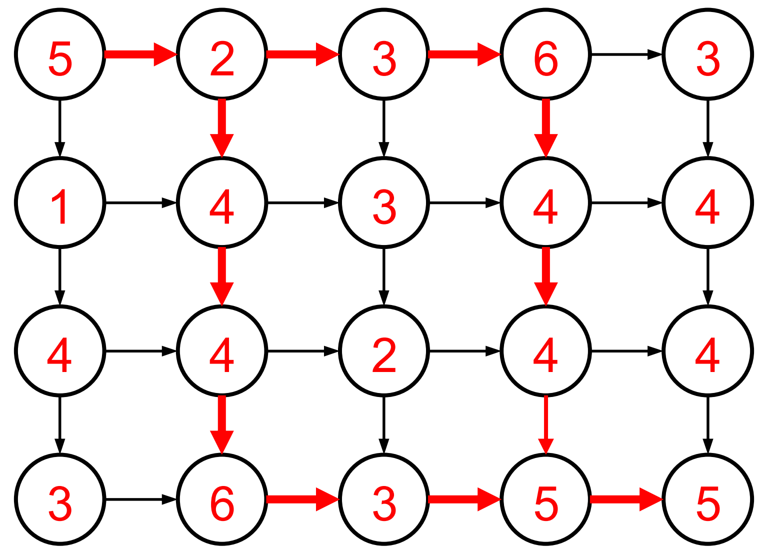
5 jobs on 4 machines with the following processing times (original processing times in parentheses)

jobs	j_1	j_2	j_3	j_4	j_5
p_{1,j_k}	3 (5)	6 (5)	3 (3)	2 (6)	5 (3)
p_{2,j_k}	4 (4)	4 (4)	3 (2)	4 (4)	1 (4)
p_{3,j_k}	4 (4)	4 (4)	2 (3)	4 (4)	4 (1)
p_{4,j_k}	5 (3)	5 (6)	3 (3)	6 (2)	3 (5)

Example Reversibility (2)

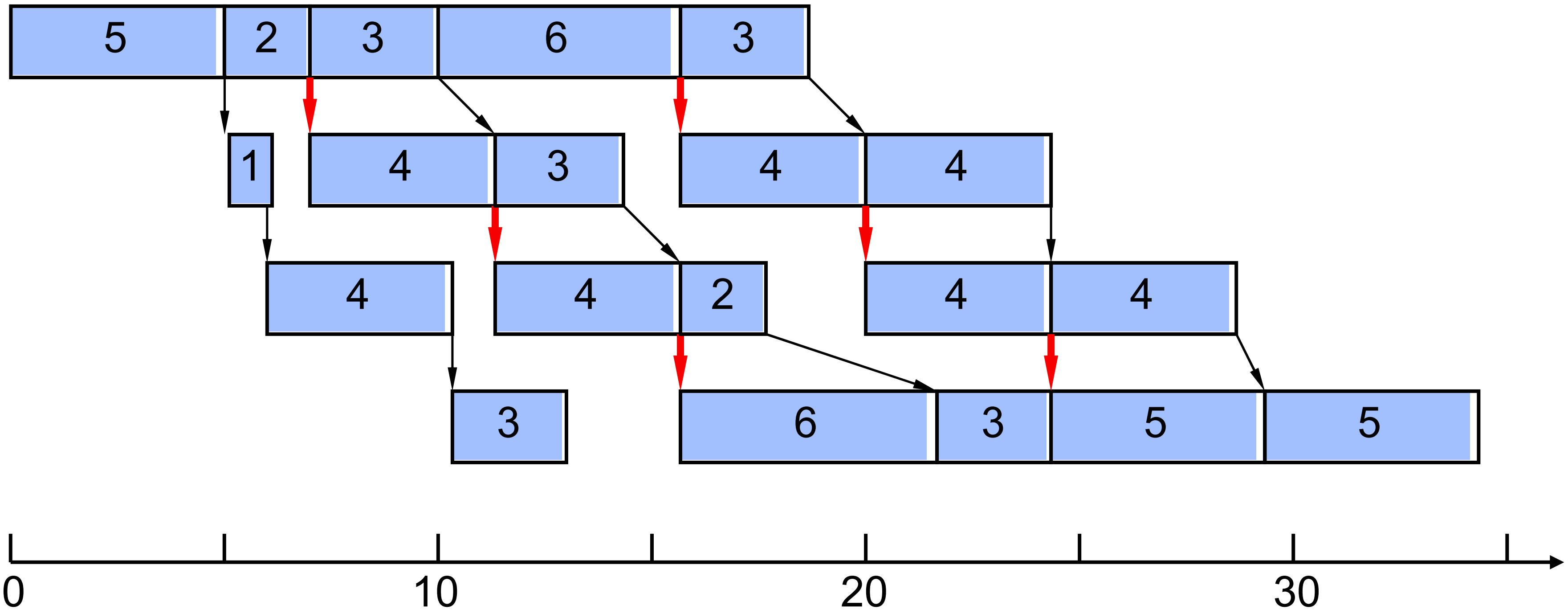


Original problem



Reverse problem

Example Reversibility (3)



$F2||C_{max}$

$F2||C_{max}$ with unlimited storage in between the two machines

- The optimal solution is always a permutation.

- Johnson's rule produces an optimal schedule.

  - The job set is partitioned into 2 sets.

    Set I : all jobs with $p_{1j} \leq p_{2j}$

    Set II : all jobs with $p_{2j} < p_{1j}$

  SPT (1) – LPT(2) schedule:

  - All jobs of Set I are scheduled first in increasing order of p_{1j} (SPT).
  - All jobs of Set II are scheduled afterwards in decreasing order of p_{2j} (LPT).

- There are many other optimal schedules besides SPT(1) – LPT(2) schedules.

  - The SPT(1) - LPT(2) schedule structure cannot be generalized to yield optimal schedules for flow shops with more than two machines.

Proof of Johnson's Rule (1)

Contradiction: Assume that another schedule S is optimal.

- There is a pair of adjacent jobs *j* followed by *k* such that one of the following conditions hold:
  - Job *j* belongs to Set II and job *k* to Set I; (Case 1)
  - Jobs *j* and *k* belong to Set I and $p_{1j} > p_{1k}$; (Case 2)
  - Jobs *j* and *k* belong to Set II and $p_{2j} < p_{2k}$; (Case 3)

- Sequence in schedule S: job $l \Rightarrow$ job $j \Rightarrow$ job $k \Rightarrow$ job h

  C_{ij} : completion time of job *j* on machine *i* in schedule S

  C'_{ij} : completion time of job *j* on machine i in the new schedule.

Proof of Johnson's Rule (2)

- Interchange of j and k
  - ➡ Starting time ($C_{1l} + p_{1j} + p_{1k}$) of job *h* on machine 1 is not affected
    Starting time of job *h* on machine 2:
    $$C_{2k} = \max(\max(C_{2l}, C_{1l} + p_{1j}) + p_{2j}, C_{1l} + p_{1j} + p_{1k}) + p_{2k}$$
    $$= \max(C_{2l} + p_{2j} + p_{2k}, C_{1l} + p_{1j} + p_{2j} + p_{2k}, C_{1l} + p_{1j} + p_{1k} + p_{2k})$$
    $$C'_{2j} = \max(C_{2l} + p_{2k} + p_{2j}, C_{1l} + p_{1k} + p_{2k} + p_{2j}, C_{1l} + p_{1k} + p_{1j} + p_{2j})$$

- Case 1 : $p_{1j} > p_{2j}$ and $p_{1k} \leq p_{2k}$
  - ➡ $C_{1l} + p_{1k} + p_{2k} + p_{2j} < C_{1l} + p_{1j} + p_{1k} + p_{2k}$
    $C_{1l} + p_{1k} + p_{1j} + p_{2j} \leq C_{1l} + p_{1j} + p_{2j} + p_{2k}$
  - ➡ $C'_{2j} \leq C_{2k}$

- Case 2 : $p_{1j} \leq p_{2j}$, $p_{1k} \leq p_{2k}$, and $p_{1j} > p_{1k}$

$$\left.\begin{array}{l} C_{1l} + p_{1k} + p_{2k} + p_{2j} \\ \\ C_{1l} + p_{1k} + p_{1j} + p_{2j} \end{array}\right\} \leq C_{1l} + p_{1j} + p_{2j} + p_{2k}$$

- Case 3 is similar to Case 2 (reversibility property).

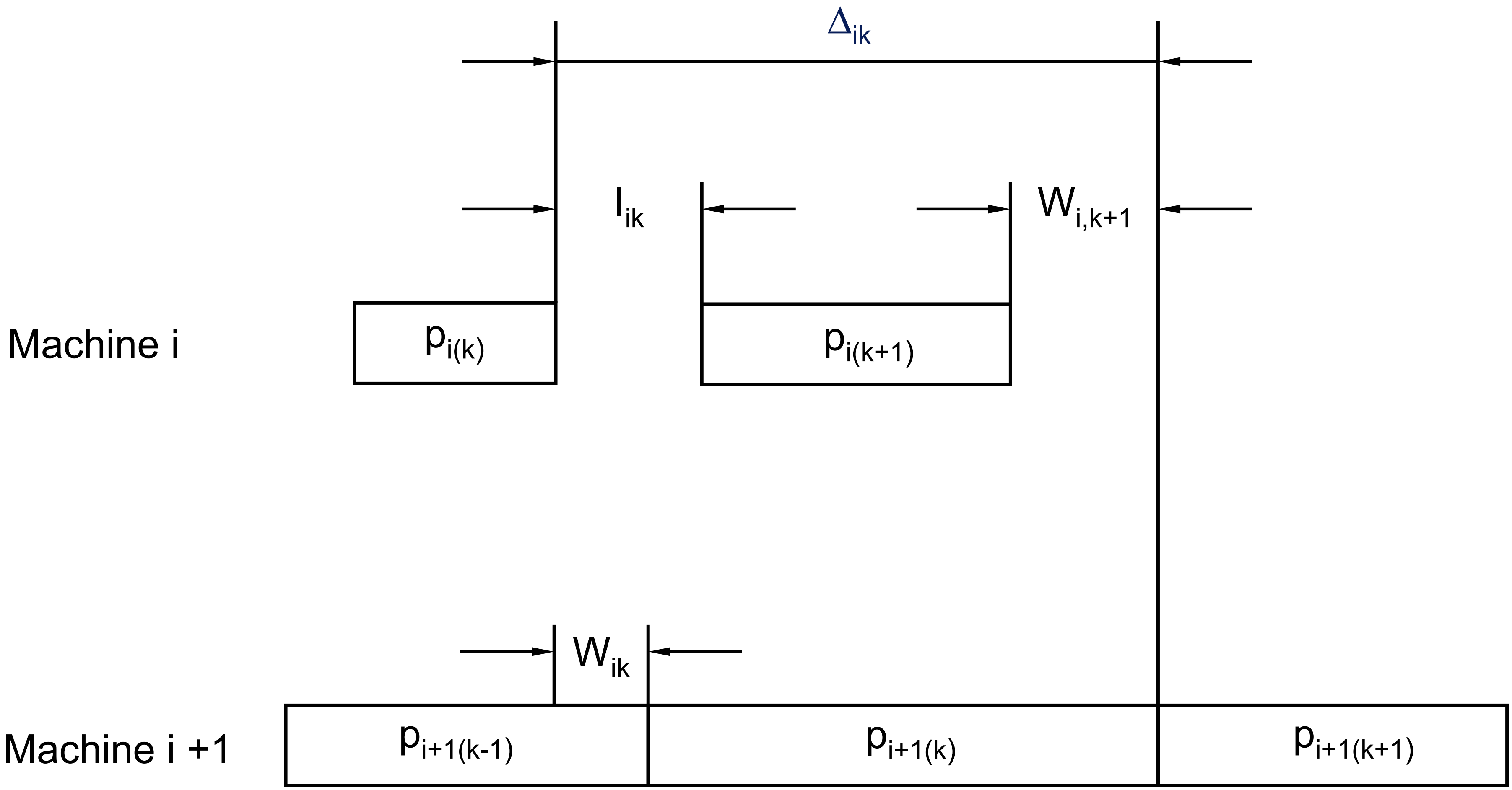
$F_m \mid prmu \mid C_{max}$

Formulation as a Mixed Integer Program (MIP)

- Decision variable x_{jk} = 1, if job *j* is the *k*th job in the sequence.
- I_{ik} : amount of idle time on machine *i* between the processing of jobs in position *k* and *k*+1
- W_{ik}: amount of waiting time of job in position *k* between machines *i* and *i*+1
- Δ_{ik}: difference between start time of the job in position *k*+1 on machine *i*+1 and completion time of the job in position *k* on machine *I*
- $p_{i(k)}$: processing time of the job in position *k* on machine *I*

➡ $\Delta_{ik} = I_{ik} + p_{i(k+1)} + W_{i,k+1} = W_{ik} + p_{i+1(k)} + I_{i+1,k}$

Graphical Description of Δ_{ik}



$W_{ik} > 0$ and $I_{i+1,\,k} = 0$

MIP for $F_m \mid prmu \mid C_{max}$ (1)

- Minimizing the makespan ≡ Minimizing the idle time on machine *m*

$$\sum_{i=1}^{m-1} p_{i(1)} + \sum_{j=1}^{n-1} I_{mj}$$

earliest start time of job in position 1 at machine k

intermediate idle time

- Remember : $p_{i(k)} = \sum_{j=1}^{n} x_{jk} p_{ij}$

  ➡ there is only one job at position k!

MIP for $F_m \mid prmu \mid C_{max}$ (2)

$$\min\left(\sum_{i=1}^{m-1}\sum_{j=1}^{n} x_{j1} p_{ij}\right) + \sum_{j=1}^{n-1} I_{mj}$$

subject to

$$\sum_{j=1}^{n} x_{jk} = 1 \qquad k = 1, \dots, n$$

$$\sum_{k=1}^{n} x_{jk} = 1 \qquad j = 1, \dots, n$$

$$I_{ik} + \sum_{j=1}^{n} x_{j,k+1} p_{ij} + W_{i,k+1} - W_{ik} - \sum_{j=1}^{n} x_{jk} p_{i+1,j} - I_{i+1,k} = 0$$

for k = 1, ..., n-1; i = 1, ..., m-1

$W_{i1} = 0$ i = 1, ..., m-1 $x_{jk} \in \{0,1\}$ j=1, ...,n

$I_{1k} = 0$ k = 1, ..., n-1 k=1, ...,m

$W_{ik} \geq 0$ i = 1, ..., m-1; k = 1, ..., n

$I_{ik} \geq 0$ i = 1, ..., m; k = 1, ..., n-1

$F3||C_{max}$

- $F3 \mid\mid C_{max}$ is strongly NP-hard.
  - Proof by reduction from 3 – Partition
- An optimal solution for $F3 \mid\mid C_{max}$ does not require sequence changes.
  - $F_m \mid prmu \mid C_{max}$ is strongly NP – hard.
- $F_m \mid prmu, p_{ij} = p_j \mid C_{max}$: proportionate permutation flow shop
  - The processing of job j is the same on each machine.
- $C_{max} = \sum_{j=1}^{n} p_j + (m-1)\max(p_1, ..., p_n)$ for

  $F_m \mid prmu, p_{ij} = p_j \mid C_{max}$ (independent of the sequence)

  This is also true for $F_m \mid p_{rj} = p_j \mid C_{max}$.

Proportionate Flow Shop

Similarities between the single machine and the proportionate (permutation) flow shop environments

1. SPT is optimal for $1 \mid\mid \sum C_j$ and $F_m \mid prmu, p_{ij} = p_j \mid \sum C_j$.
2. The algorithm that produces an optimal schedule for $1 \mid\mid \sum U_j$ also results in an optimal schedule for $F_m \mid prmu, p_{ij} = p_j \mid \sum U_j$.
3. The algorithm that produces an optimal schedule for $1 \mid\mid h_{max}$ also results in an optimal schedule for $F_m \mid prmu, p_{ij} = p_j \mid h_{max}$.
4. The pseudo-polynomial dynamic programming algorithm $1 \mid\mid \sum T_j$ is also applicable to $F_m \mid prmu, p_{ij} = p_j \mid \sum T_j$.
5. The elimination criteria that hold for $1 \mid\mid \sum w_jT$ also hold for $F_m \mid prmu, p_{ij} = p_j \mid \sum w_jT_j$.

$F2 \mid\mid \sum C_j$

- $F2 \mid\mid \sum C_j$ is strongly NP – hard
  - ➔ $Fm \mid prmu \mid \sum C_j$ is strongly NP – hard

    as sequence changes are not required in the optimal schedule for 2 machines

Slope Heuristic

- Slope index A_j for job j

$$A_j = -\sum_{i=1}^{m} (m-(2i-1))p_{ij}$$

- Sequencing of jobs in decreasing order of the slope index
- Consider 5 jobs on 4 machines with the following processing times

jobs	j_1	j_2	j_3	j_4	j_5
p_{1,j_k}	5	5	3	6	3
p_{2,j_k}	4	4	2	4	4
p_{3,j_k}	4	4	3	4	1
p_{4,j_k}	3	6	3	2	5

A_1 = -(3 x 5) – (1 x 4) + (1 x 4) + (3 x 3) = -6

A_2 = -(3 x 5) – (1 x 4) + (1 x 4) + (3 x 6) = +3

A_3 = -(3 x 3) – (1 x 2) + (1 x 3) + (3 x 3) = +1

A_4 = -(3 x 6) – (1 x 4) + (1 x 4) + (3 x 2) = -12

A_5 = -(3 x 3) – (1 x 4) + (1 x 1) + (3 x 5) = +3

Sequences 2,5,3,1,4 and 5,2,3,1,4 are optimal and the makespan is 32.

Flow Shops with Limited Intermediate Storage (1)

- Assumption: No intermediate storage, otherwise one storage place is modeled as machine on which all jobs have 0 processing time
- $Fm \mid block \mid C_{max}$
- D_{ij} : time when job j leaves machine i, $D_{ij} \geq C_{ij}$
- For sequence $j_1, \ldots, j_n$ the following equations hold

$$D_{i,j_1} = \sum_{l=1}^{i} p_{l,j_1}$$

$$D_{i,j_k} = \max(D_{i-1,j_k} + p_{i,j_k}, D_{i+1,j_{k-1}})$$

$$D_{m,j_k} = D_{m-1,j_k} + p_{m,j_k}$$

➡ Critical path in a directed graph

Weight of node (i, j_k) specifies the departure time of job j_k from machine i

Edges have weights 0 or a processing time

Flow Shops with Limited Intermediate Storage (2)

- The reversibility result holds as well:
- If $p_{ij}^{(1)} = p^{(2)}_{m+1-I,j}$ then sequence $j_1, \ldots, j_n$ in the first flow shop has the same makespan as sequence $j_n, \ldots., j_1$ in the second flow shop
- $F2 \mid block \mid C_{max}$ is equivalent to a Traveling Salesman problem with n+1 cities
- When a job starts its processing on machine 1 then the proceeding job starts its processing on machine 2
  - ➡ time for job j_k on machine 1

    $\max(p_{1,j_k}, p_{2,j_{k-1}})$

Exception: The first job j^* in the sequence spends time p_{1,j^*} on machine 1

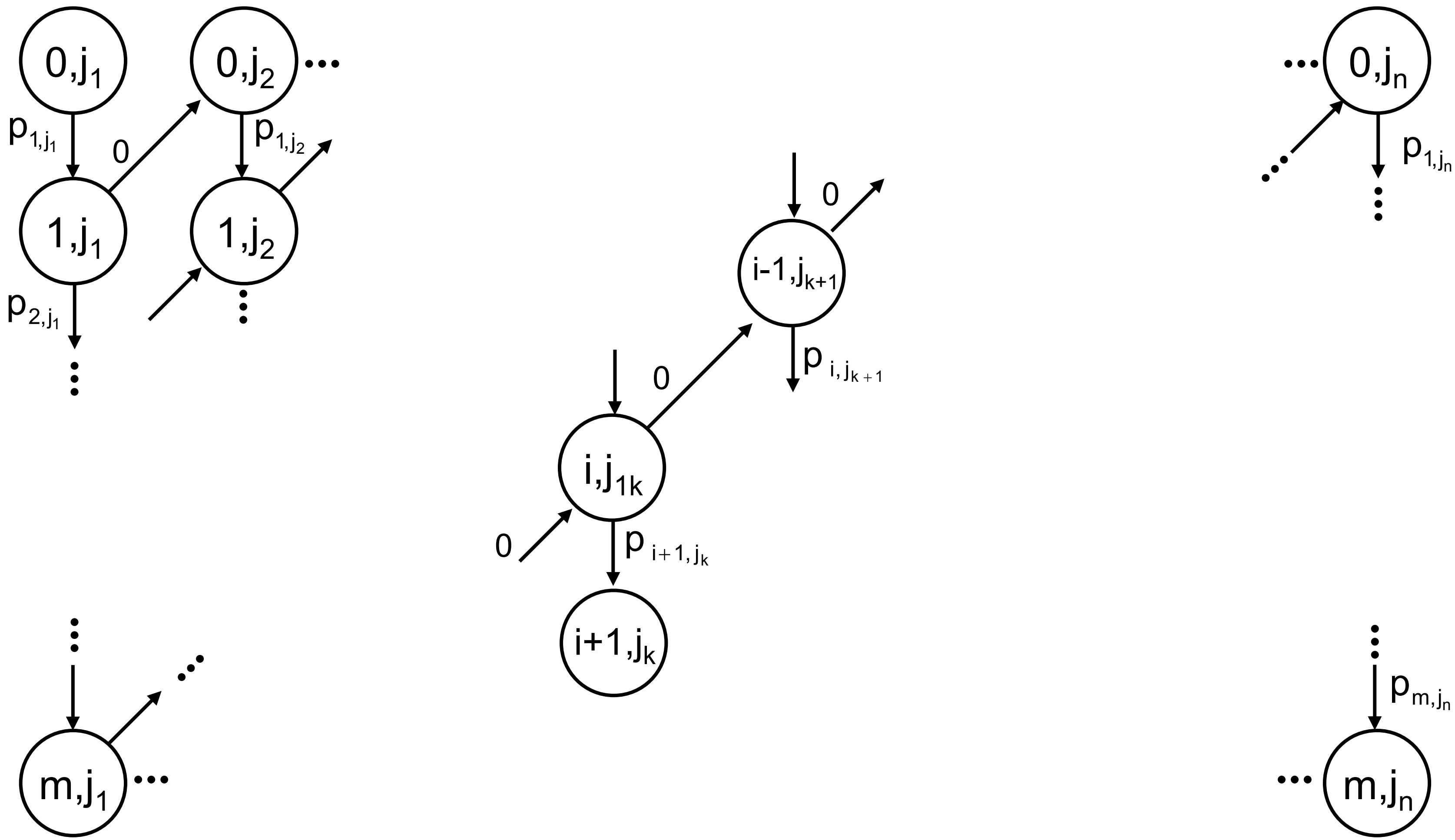
Distance from city j to city k

$$d_{0k} = p_{1k}$$

$$d_{j0} = p_{2j}$$

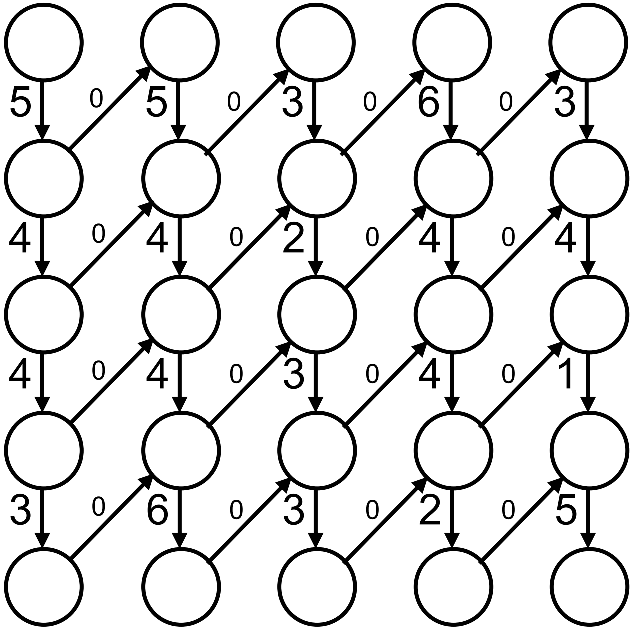
$$d_{jk} = \max(p_{2j}, p_{1k})$$

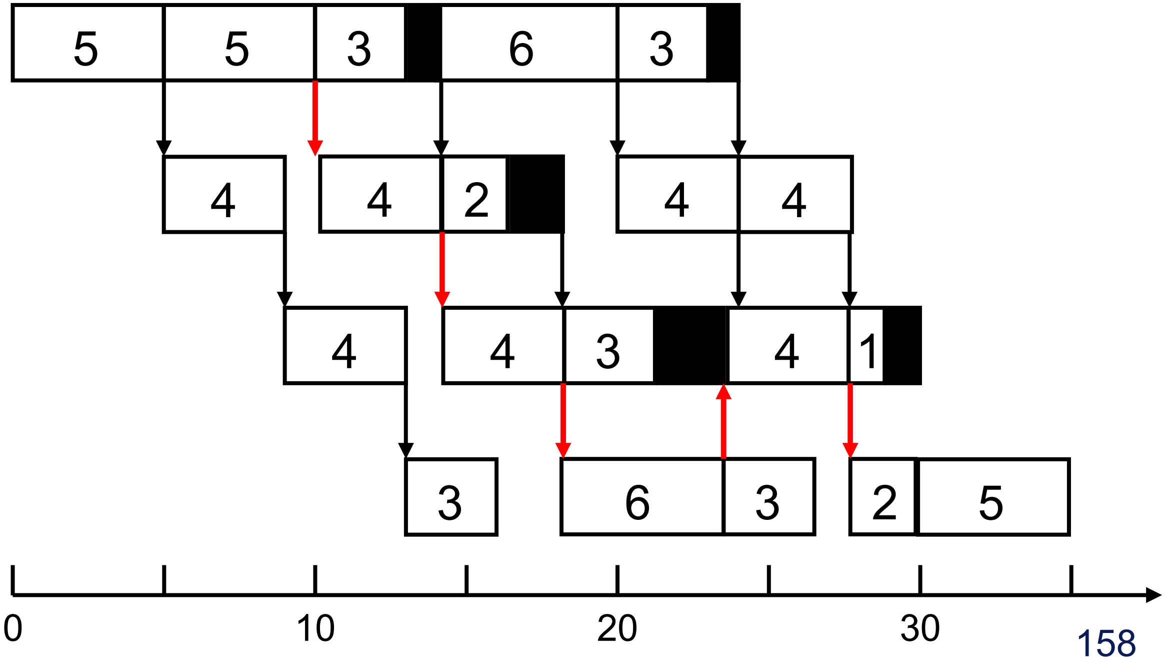
Directed Graph for the Computation of the Makespan



Graph Representation of a Flow Shop with Blocking

jobs	j_1	j_2	j_3	j_4	j_5
p_{1,j_k}	5	5	3	6	3
p_{2,j_k}	4	4	2	4	4
p_{3,j_k}	4	4	3	4	1
p_{4,j_k}	3	6	3	2	5





Example: A Two Machine Flow Shop with Blocking and the TSP (1)

- Consider 4 job instance with processing times

jobs	1	2	3	4
$P_{1,j}$	2	3	3	9
$P_{2,j}$	8	4	6	2

- Translates into a TSP with 5 cities

cities	0	1	2	3	4
$b_{,j}$	0	2	3	3	9
$a_{,j}$	0	8	4	6	2

- There are two optimal schedules
- 1, 4, 2, 3 $\Rightarrow$ $0 \rightarrow 1 \rightarrow 4 \rightarrow 2 \rightarrow 3 \rightarrow 0$ and
- 1, 4, 3, 2

Example: A Two Machine Flow Shop with Blocking and the TSP (2)

- Comparison SPT(1) – LPT(2) schedules for unlimited buffers:

  1, 3, 4, 2;   1, 2, 3, 4   and   1, 3, 2, 4

- F3 | block | C_{max} is strongly NP – hard and cannot be described as a traveling salesman problem

Special Cases of Fm | block | C_{max}

- Special cases of Fm | block | C_{max}
  - Proportionate case: Fm | block, $p_{ij} = p_j$ | C_{max}
- A schedule is optimal for Fm | block, $p_{ij} = p_j$ | C_{max} if and only if it is an SPT- LPT schedule
- Proof :

$$\underbrace{C_{max} \geq \sum_{j=1}^{n} p_j + (m-1)\max(p_1, \ldots, p_n)}_{\text{optimal makespan with unlimited buffers}}$$

- Proof – concept:
  - Any SPT-LPT schedule matches the lower bound
  - Any other schedule is strictly greater than the lower bound

SPT- LPT Schedule

- SPT – part: A job is never blocked
- LPT – part: No machine must ever wait for a job
  - ➡ The makespan of an SPT – LPT schedule is identical to an SPT – LPT schedule for unlimited buffers.
- Second part of the proof by contradiction

  The job j_k with longest processing time contributes m times its processing time to the makespan
- If the schedule is no SPT- LPT schedule
  - ➡ a job j_h is positioned between two jobs with a longer processing time
  - ➡ this job is either blocked in the SPT part or the following jobs cannot be processed on machine m without idle time in between

Profile Fitting (PF)

- Heuristic for Fm | block | C_{max}
  - Local optimization
  - Selection of a first job (e.g. smallest sum of processing time)
  - Pick the first job as next that wastes the minimal time on all m machines.
  - Using weights to weight the idle times on the machines depending the degree of congestion

Application of the PF Heuristic

jobs	j_1	j_2	j_3	j_4	j_5
p_{1,j_k}	5	5	3	6	3
p_{2,j_k}	4	4	2	4	4
p_{3,j_k}	4	4	3	4	1
p_{4,j_k}	3	6	3	2	5

- First job: job 3 (shortest total processing time)
- Second job : job 1 2 4 5
  idle time 11 11 15 3
  - ➡ job 5
  - ➡ Sequence: 3 5 1 2 4 makespan 32

    makespan for unlimited storage
  - ➡ optimal makespan
- First job: job 2 (largest total processing time)
  - ➡ Sequence: 2 1 3 5 4 makespan 35

$$F2 \mid block \mid C_{max} = F2 \mid nwt \mid C_{max}$$

but $$Fm \mid block \mid C_{max} \neq Fm \mid nwt \mid C_{max}$$

Flexible Flow Shop with Unlimited Intermediate Storage (1)

- Proportionate case

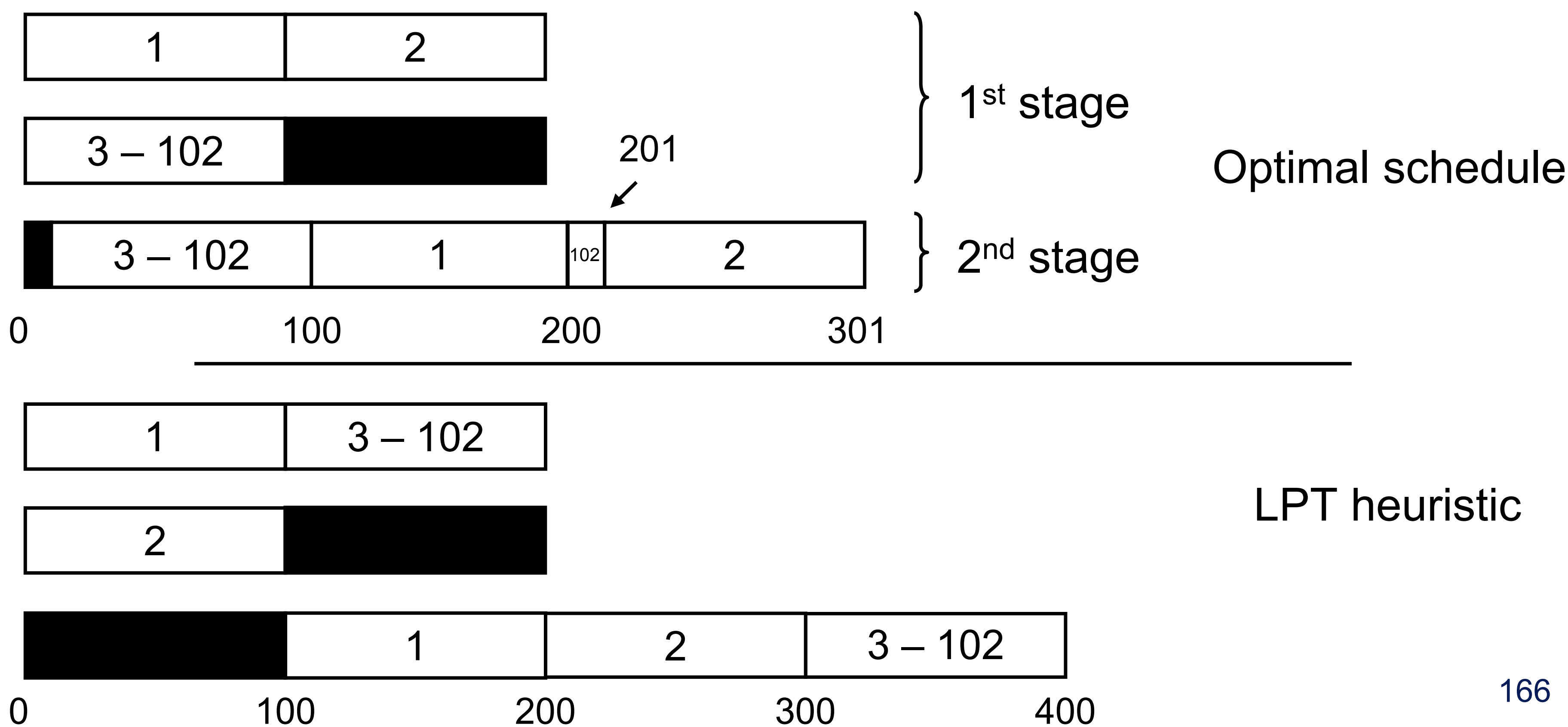
  $FF_c \mid p_{ij} = p_j \mid C_{max}$

non preemptive	preemptive
LPT heuristic	LRPT heuristic
➡ NP hard	optimal for a single stage

Example: Minimizing Makespan with LPT

- $p_1 = p_2 = 100$ $\quad\quad$ $p_3 = p_4 = \ldots = p_{102} = 1$
- 2 stages: 2 machines at first stage

  1 machine at second stage



Flexible Flow Shop with Unlimited Intermediate Storage (2)

- $FF_c \mid p_{ij} = p_j \mid \sum C_j$
- SPT is optimal for a single stage and for any numbers of stage with a single machine at each stage
- SPT rule is optimal for $FF_c \mid p_{ij} = p_j \mid \sum C_j$ if each stage has at least as many machines as the preceding stage
- Proof:

  Single stage SPT minimizes $\sum C_j$ and the sum of the starting times $\sum (C_j - p_j)$

  c stages: C_j occurs not earlier than cp_j time units after its starting time at the first stage

  Same number of machines at each stage:

  SPT: each need not wait for processing at the next stage

$$\Rightarrow \quad \sum_{j=1}^{n} C_j = \text{sum of the starting times} + \sum_{j=1}^{n} cp_j$$

Job Shops

- **The route of every job is fixed but not all jobs follow the same route**
- $J2 \mid\mid C_{max}$
- $J_{1,2}$: set of all jobs that have to be processed first on machine 1
- $J_{2,1}$: set of all jobs that have to be processed first on machine 2
- Observation: If a job from $J_{1,2}$ has completed its processing on machine 1 the postponing of its processing on machine 2 does not matter as long as machine 2 is not idle.
- A similar observation hold for $J_{2,1}$
  - ➡ a job from $J_{1,2}$ has a higher priority on machine 1 than any job form $J_{2,1}$ and vice versa
- Determining the sequence of jobs from $J_{1,2}$
  - ➡ $F2 \mid\mid C_{max}$: SPT(1) – LPT(2) sequence
  - ➡ machine 1 will always be busy
- $J2 \mid\mid C_{max}$ can be reduced to two $F2 \mid\mid C_{max}$ problems

Representation as a Disjunctive Graph G

- Jm || C_{max} is strongly NP hard
- Representation as a disjunctive graph G

Set of nodes N :

Each node corresponds to an operation (i, j) of job j on machine i

Set of conjunctive edges A:

An edge from (i, j) to (k, j) denotes that job j must be processed on machine k immediately after it is processed on machine i

Set of disjunctive edges B:

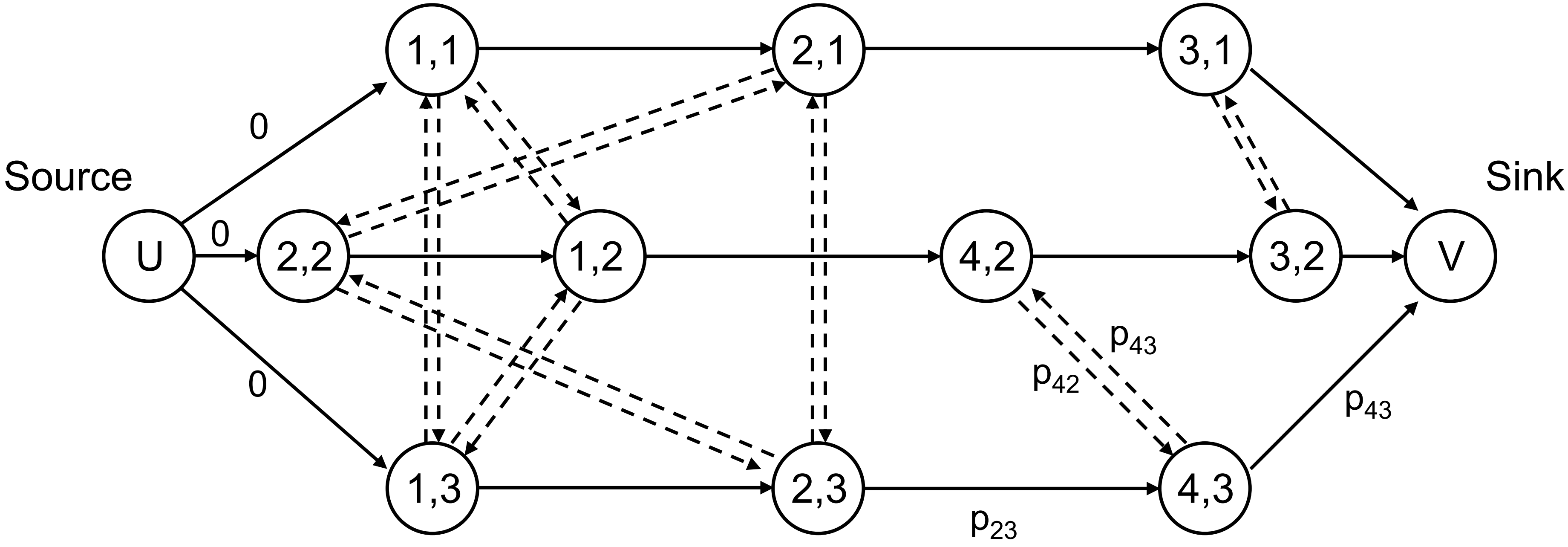
There is a disjunctive edge from any operation (i, j) to any operation (i, h), that is, between any two operations that are executed on the same machine

➡ All disjunctive edges of a machine form a cliques of double arcs

Each edge has the processing time of its origin node as weight

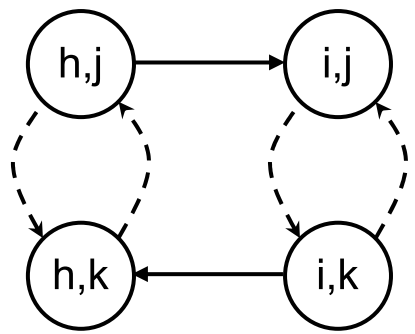
Directed Graph for Job Shop

- There is a dummy source node U connected to the first operation of each job. The edges leaving U have the weight 0.
- There is a dummy sink node V, that is the target of the last operation of each job.



Feasible Schedule

- Feasible schedule: Selection of one disjunctive edge from each pair of disjunctive edges between two nodes such that the resulting graph is acyclic
- Example



- D: set of selective disjunctive edges
- G(D): Graph including D and all conjunctive edges
- Makespan of a feasible schedule: Longest path from U to V in G(D)
  - 1. Selection of the disjunctive edges D
  - 2. Determination of the critical path

Disjunctive Programming Formulation

- y_{ij}: starting time of operation (i,j)
- Minimize C_{max} subject to

$y_{kj} \geq y_{ij} + p_{ij}$ if $(i,j) \rightarrow (k,j)$ is a conjunctive edge

$C_{max} \geq y_{ij} + p_{ij}$ for all operations (i,j)

$y_{ij} \geq y_{il} + p_{il}$ or

$y_{il} \geq y_{ij} + p_{ij}$ for all (i,l) and (i,j) with i = 1, …, m

$y_{ij} \geq 0$ for all operations (i,j)

Example: Disjunctive Programming Formulation

- 4 machines , 3 jobs

jobs	machine sequence	processing times
1	1, 2, 3	$p_{11} = 10$, $p_{21} = 8$, $p_{31} = 4$
2	2, 1, 4, 3	$p_{22} = 8$, $p_{12} = 3$, $p_{42} = 5$, $p_{32} = 6$
3	1, 2, 4	$p_{13} = 4$, $p_{23} = 7$, $p_{43} = 3$

- $y_{21} \geq y_{11} + p_{11} = y_{11} + 10$
- $C_{max} \geq y_{11} + p_{11} = y_{11} + 10$
- $y_{11} \geq y_{12} + p_{12} = y_{12} + 3$ or $y_{12} \geq y_{11} + p_{11} = y_{11} + 10$

Branch and Bound Method to Determine all Active Schedules

- Ω :set of all schedulable operations (predecessors of these operations are already scheduled),
- $r_{i,j}$:earliest possible starting time of operation
- $(i,j) \in \Omega$
- $\Omega' \subseteq \Omega$
- t(Ω) smallest starting time of a operation

Generation of all Active Schedules

- Step 1: (Initial Conditions) Let Ω contain the first operation of each job; Let $r_{ij} = 0$, for all $(i,j) \in \Omega$

- Step 2: (machine selection) compute for the current partial schedule $t(\Omega) = \min_{(i,j)\in\Omega}\{r_{ij} + p_{ij}\}$

  and let i* denote the machine on which the minimum is achieved.

- Step 3: (Branching) Let Ω' denote the set of all operations (i*,j) on machine i* such that $r_{i^*j} < t(\Omega)$

  For each operation in Ω', consider an (extended) partial schedule with that operation as the next one on machine i*.

  For each such (extended) partial schedule, delete the operation from Ω, include its immediate follower in Ω, and return to Step 2.

Generation of all Active Schedules

- Result: Tree with each active schedule being a leaf

- A node v in this tree: partial schedule

➡ Selection of disjunctive edges to describe the order of all operations that are predecessors of Ω

- An outgoing edge of v: Selection of an operation $(i^*, j) \in \Omega'$
  as the next job on machine i*

➡ The number of edges leaving node v = number of operations in Ω'

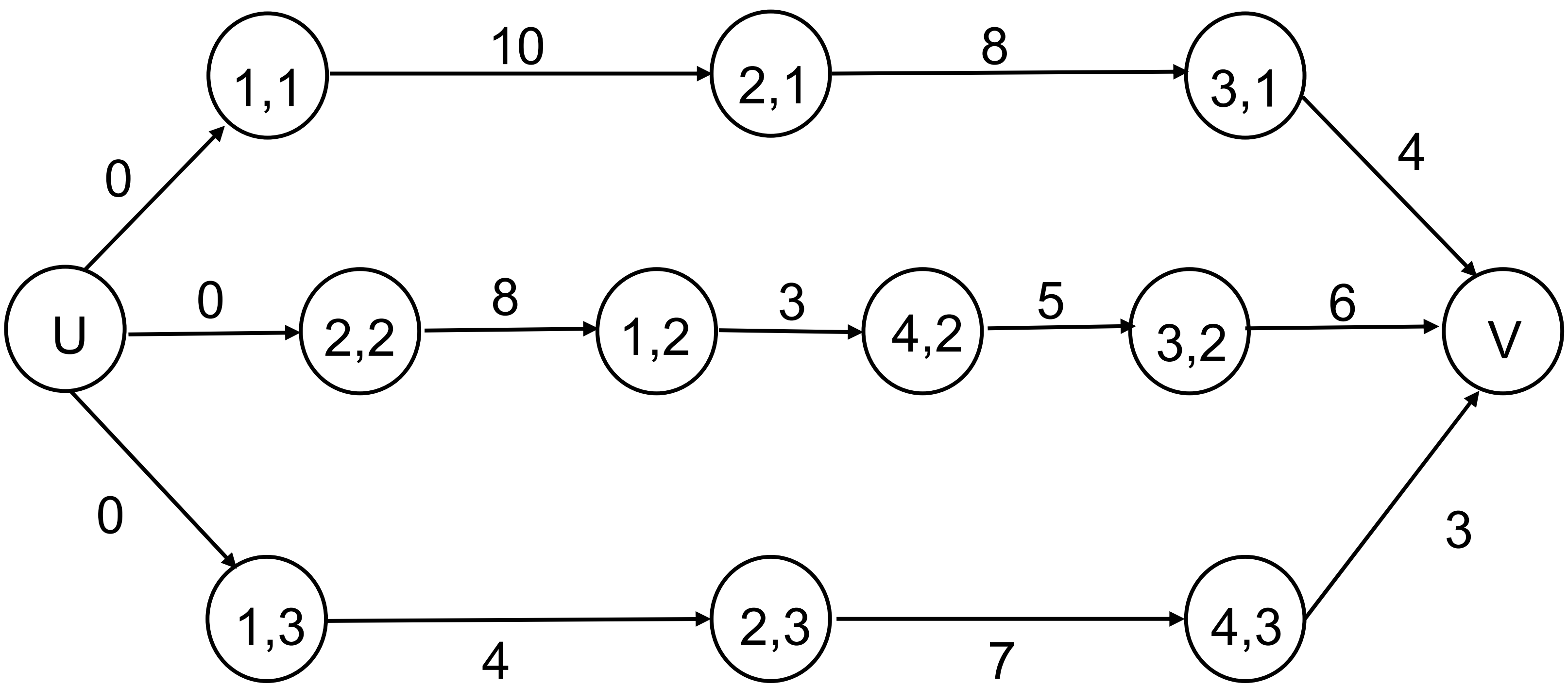
- v': successor of v

➡ Set D' of the selected disjunctive edges at v' $\rightarrow$ G(D')

Lower Bound for Makespan at v'

- simple lower bound: critical path in graph G(D')
- complex lower bound:
  - critical path from the source to any unscheduled operation: release date of this operation
  - critical path form any unscheduled operation to the sink: due date of this operation
  - Sequencing of all unscheduled operations on the appropriate machine for each machine separately
    - $1 \mid r_j \mid L_{max}$ for each machine (strongly NP-hard)
      - Reasonable performance in practice

Application of Branch and Bound



Application of Branch and Bound Level 1

- Initial graph: only conjunctive edges
- Makespan: 22

- Level 1:

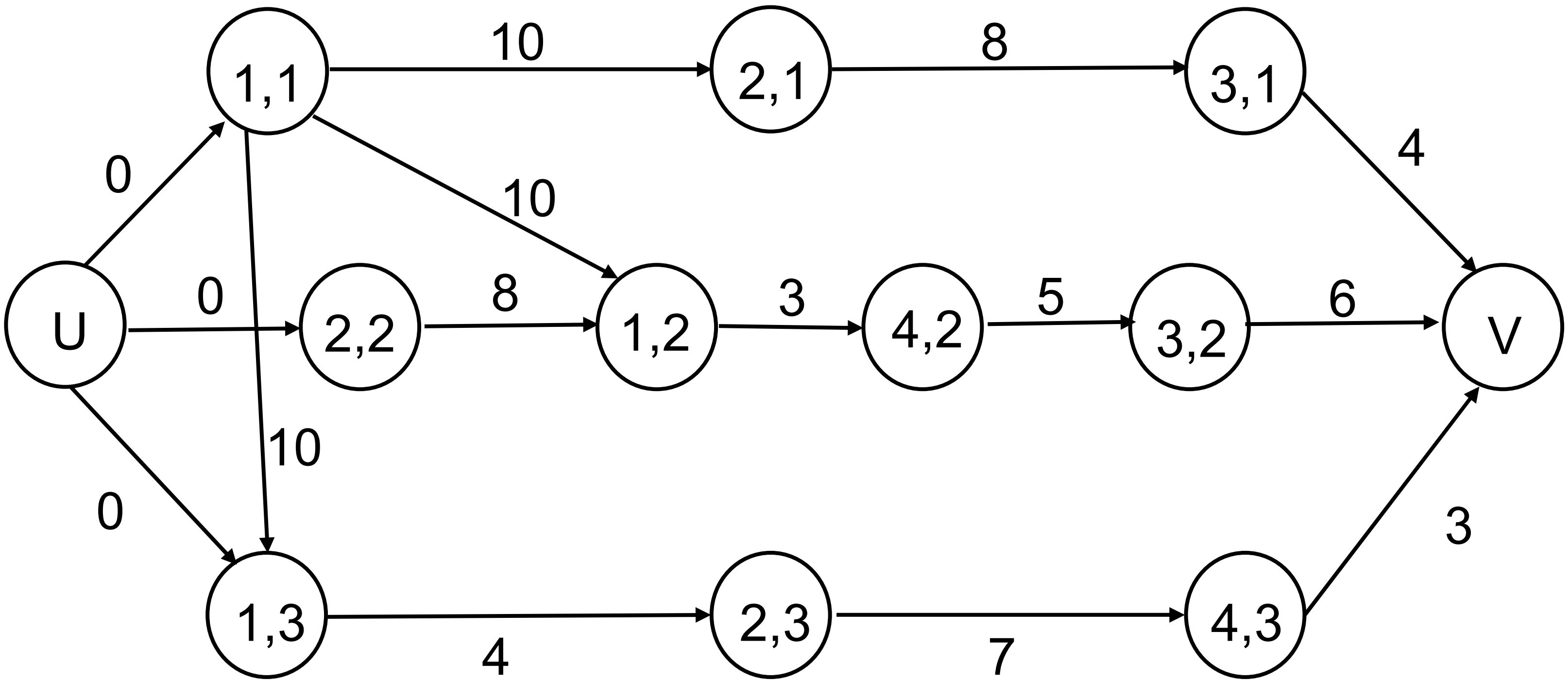
$$\Omega = \{(1,1),(2,2),(1,3)\}$$

$$t(\Omega) = \min\{0+10, 0+8, 0+4\} = 4$$

$$i^* = 1$$

$$\Omega' = \{(1,1),\ (1,3)\}$$

Schedule Operation (1,1) first



Schedule Operation (1,1) first

- 2 disjunctive edges are added
- (1,1) → (1,2)
- (1,1) → (1,3)
- ➡ Makespan: 24

Schedule Operation (1,1) first

- Improvements of lower bound by generating an instance of $1 \mid r_j \mid L_{max}$ for machine 1

jobs	1	2	3
p_{ij}	10	3	4
r_{ij}	0	10	10
d_{ij}	12	13	14

- L_{max} =3 with sequence 1,2,3
- Makespan: 24+3=27

Schedule Operation (1,1) first

- Instance of 1 | r_j | L_{max} for machine 2

jobs	1	2	3
p_{ij}	8	8	7
r_{ij}	10	0	14
d_{ij}	20	10	21

- L_{max} = 4 with sequence 2,1,3
- Makespan: 24+4 = 28

Schedule Operation (1,3) first

- 2 disjunctive edges are added → Makespan: 26
- $1 \mid r_j \mid L_{max}$ for machine 1
- $L_{max} = 2$ with sequence 3, 1, 2
- ➡ Makespan: 26+2=28

Application of Branch an Bound
Level 2

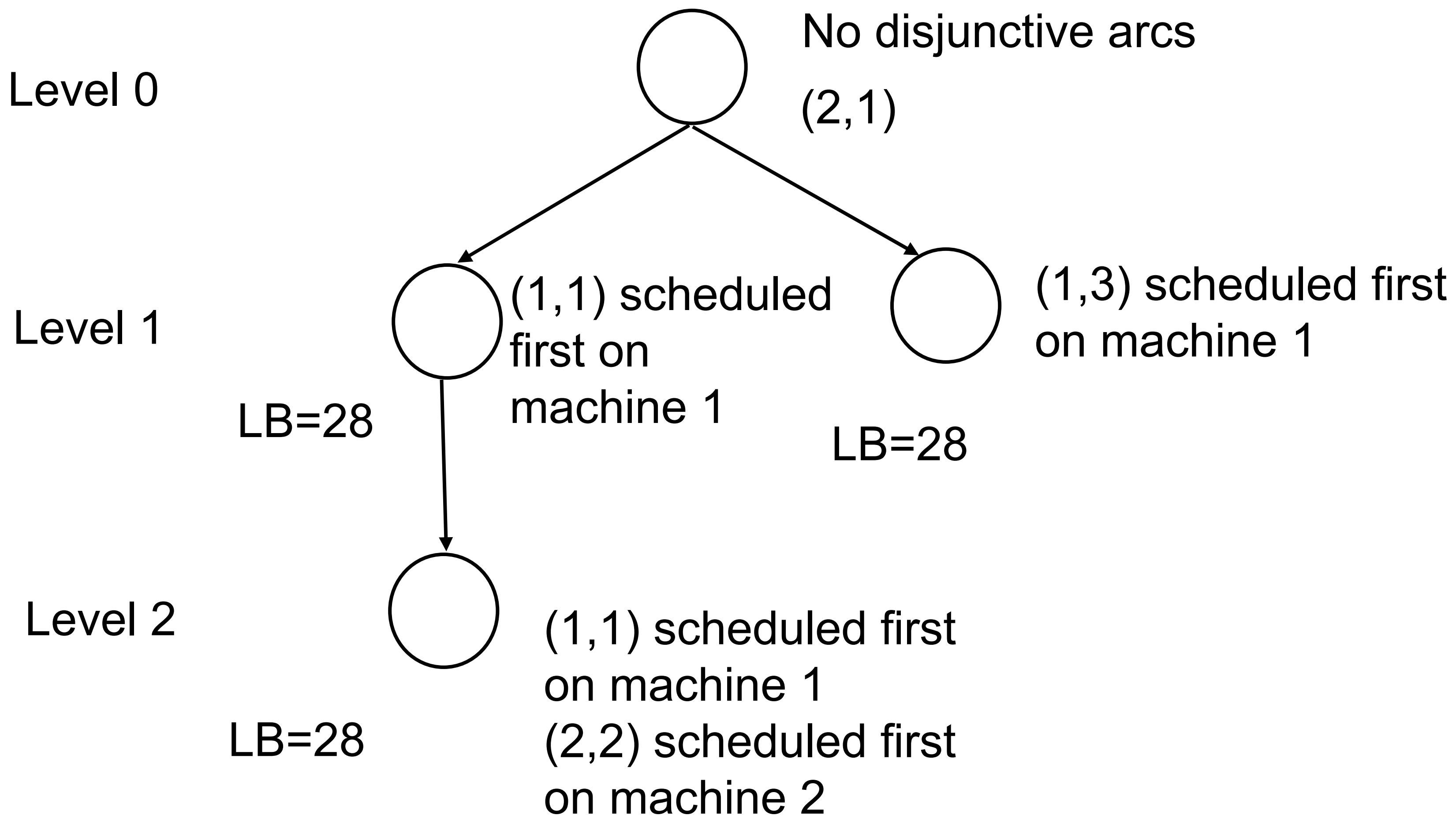
- Level 2: Branch from node (1,1)

$$\Omega = \{(2,2),(2,1),(1,3)\}$$
$$t(\Omega) = \min(0+8,10+8,10+4) = 8$$
$$\Omega' = \{(2,2)\}$$
$$i^* = 2$$

- There is only one choice
  (2,2) → (2,1); (2,2) → (2,3)

- Two disjunctive edges are added

Branching Tree

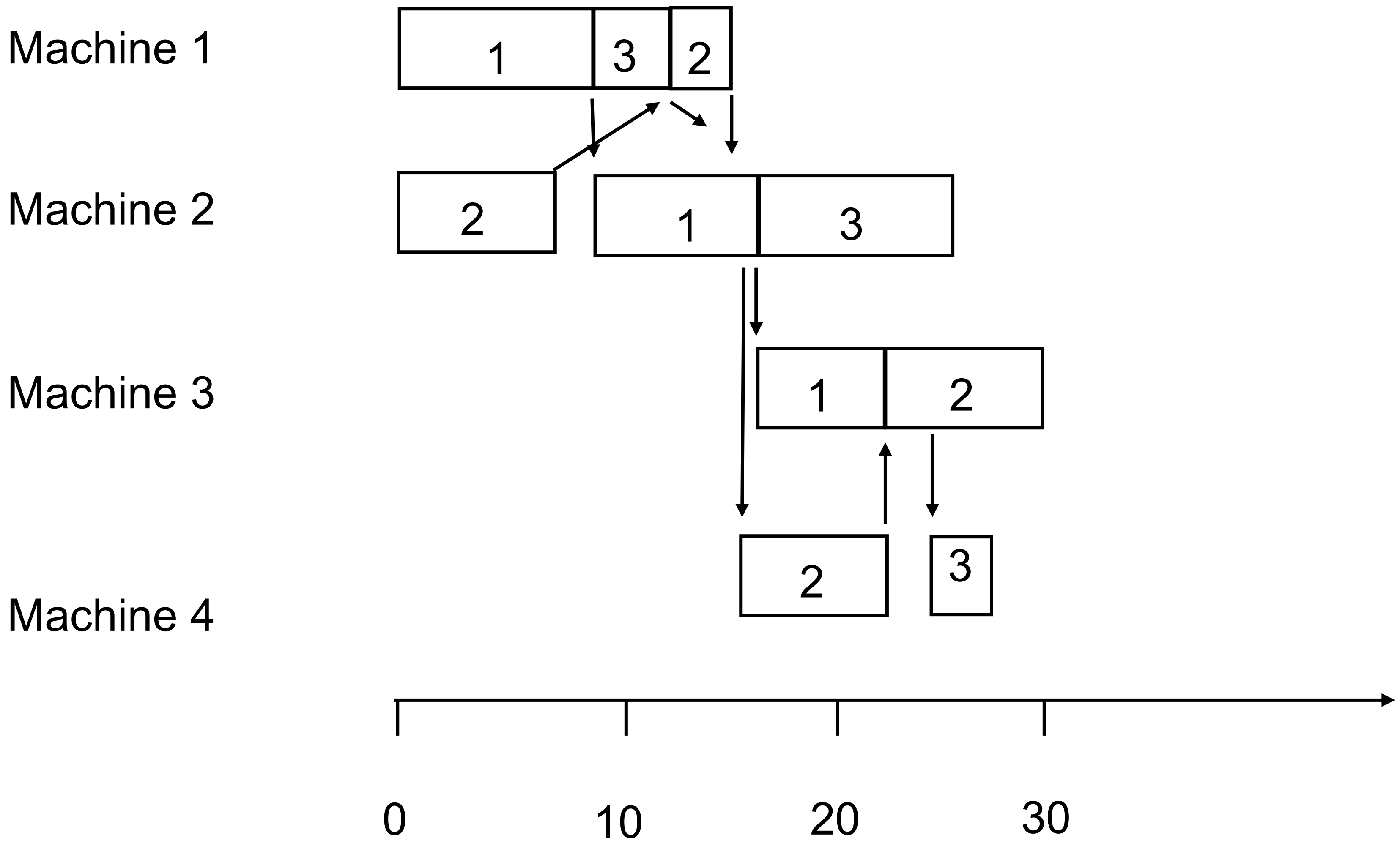


Continuation of the Procedure yields

machine	job sequence
1	1 3 2 (or 1 2 3)
2	2 1 3
3	1 2
4	2 3

Makespan: 28

Gantt Chart for J4 || C_{max}



Shifting Bottleneck Heuristic

- A sequence of operations has been determined for a subset M_0 of all m machines.
  - disjunctive edges are fixed for those machines
- Another machine must be selected to be included in M_0: Cause of severest disruption (bottleneck)
- All disjunctive edges for machines not in M_0 are deleted $\rightarrow$ Graph G'

  Makespan of G' : C_{max} (M_0)
  - for each operation (i, j) with $i \notin M_0$ determine release date and due date
  - allowed time window
- Each machine not in M_0 produces a separate $1 \mid r_j \mid L_{max}$ problem
  - L_{max}(i): minimum L_{max} of machine i
- Machine k with the largest L_{max}(i) value is the bottleneck
  - Determination of the optimal sequence for this machine $\rightarrow$ Introduction of disjunctive edges
  - Makespan increase from M_0 to $M_0 \cup \{k\}$ by at least L_{max}(k)

Shifting Bottleneck Heuristic

- Resequencing of the operation of all machines in M_0

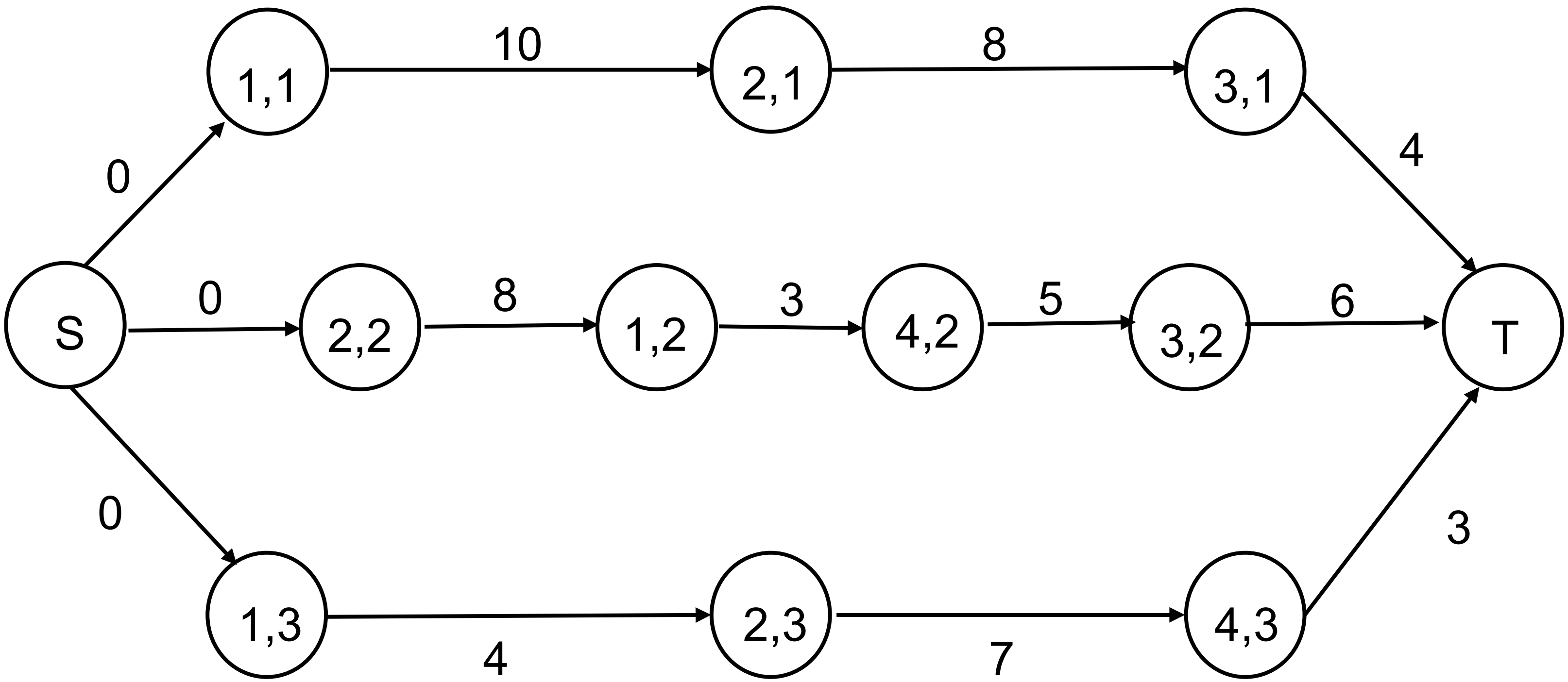
jobs	machine sequence	processing times
1	1, 2, 3	$p_{11} = 10, p_{21} = 8, p_{31} = 4$
2	2, 1, 4, 3	$p_{22} = 8, p_{12} = 3, p_{42} = 5, p_{32} = 6$
3	1, 2, 4	$p_{13} = 4, p_{23} = 7, p_{43} = 3$

- Iteration 1 : $M_0 = \varnothing$ G' contains only conjunctive edges
  - ➡ Makespan (total processing time for any job) : 22
- $1 \mid r_j \mid L_{max}$ problem for machine 1:
  optimal sequence 1, 2, 3 $\rightarrow L_{max}(1)=5$
- $1 \mid r_j \mid L_{max}$ problem for machine 2:
  optimal sequence 2, 3, 1 $\rightarrow L_{max}(2)=5$
- Similar $L_{max}(3)=4$, $L_{max}(4)=0$
  - ➡ Machine 1 or machine 2 are the bottleneck

Shifting Bottleneck Heuristic

- Machine 1 is selected $\rightarrow$ disjunctive edges are added : graph G''

$C_{max}(\{1\}) = C_{max}(\varnothing) + L_{max}(1) = 22 + 5 = 27$



Shifting Bottleneck Heuristic

- Iteration 2
- $1 \mid r_j \mid L_{max}$ problem for machine 2
  optimal sequence 2, 1, 3 $\rightarrow L_{max}(2)=1$
- $1 \mid r_j \mid L_{max}$ problem for machine 3
  optimal sequences 1, 2 and 2, 1 $\rightarrow L_{max}(3) =1$
  Similar L_{max} (4) = 0
- Machine 2 is selected : $M_0 = \{1, 2\}$
  C_{max} ({1,2}) = C_{max} ({1}) + L_{max} (2) = 27 + 1 = 28
  Disjunctive edges are added to include machine 2
  Resequencing for machine 1 does not yield any improvement
- Iteration 3
  No further bottleneck is encountered
  $L_{max}(3)=0$, $L_{max}(4)=0$
  ➡ Overall makespan 28

machines	1	2	3	4
sequences	1, 2, 3	2, 1, 3	2, 1	2, 3

Open Shops

- O2 || C_{max} $\qquad C_{max} \geq \max\left(\sum_{j=1}^{n} p_{1j}, \sum_{j=1}^{n} p_{2j}\right)$
- In which cases is C_{max} strictly greater than the right hand side of the inequality?
- Non delay schedules
  - Idle period only iff one job remains to be processed and this job is executed on the other machine: at most on one of the two machines
- Longest Alternate Processing Time first (LAPT)

  Whenever a machine is free start processing the job that has the longest processing time on the other machine
- The LAPT rule yields an optimal schedule for O2 || C_{max} with makespan

$$C_{max} = \max\left(\max_{j \in \{1,\dots,n\}}(p_{1j} + p_{2j}), \sum_{j=1}^{n} p_{1j}, \sum_{j=1}^{n} p_{2j}\right)$$

Open Shops

- Assumption

  $p_{1j} \leq p_{1k}$; $p_{2j} \leq p_{1k}$

  ➔ longest processing time belongs to operation (1, k)

  LAPT: Job k is started on machine 2 at time 0

  ➔ Job k has lowest priority on machine 1
- It is only executed on machine 1 if no other job is available for processing on machine 1

  a) k is the last job to be processed on machine 1

  b) k is the second to last job to be processed in machine 1 and the last job is not available due to processing on machine 2
- Generalization: The 2(n-1) remaining operations can be processed in any order without unforced idleness.
- No idle period in any machine $\rightarrow$ optimal schedule

Open Shops

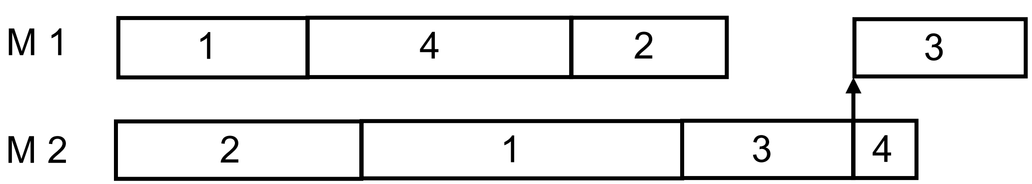
- Case 1: Idle period on machine 2
  - job2 needs processing on machine 2 (last job on machine 2) <u>and</u> job l is still processed on machine 1
  - job l starts on machine 2 at the same time when job k starts on machine 1 p1k ≥ p2l → machine 1 determines makespan and there is no idle time on machine 1 → optimal schedule
- Case 2: Idle period on machine 1
  - all operations are executed on machine 1 except (1, k) and job k is still processed on machine 2
  - makespan is determined by machine 2 → optimal schedule without idle periods
  - makespan is determined by machine 1 → makespan $p_{2k} + p_{1k}$, optimal schedule

General Heuristic Rule

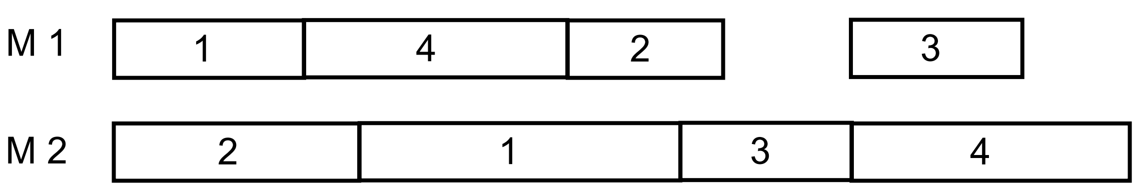
- Longest Total Remaining Processing on Other Machines first rule
  but Om || C_{max} is NP hard for m ≥ 3
  (LAPT is also optimal for O2 | prmp | C_{max})
- Lower bound

$$C_{max} \geq \max\left(\max_{j \in \{1,\ldots,n\}} \sum_{i=1}^{m} p_{ij}, \max_{j \in \{1,\ldots,m\}} \sum_{j=1}^{n} p_{ij} \right)$$

➡ The optimal schedule matches the lower bound
The problem O2 || L_{max} is strongly NP hard (Reduction of 3 Partitions)



unnecessary increase in makespan

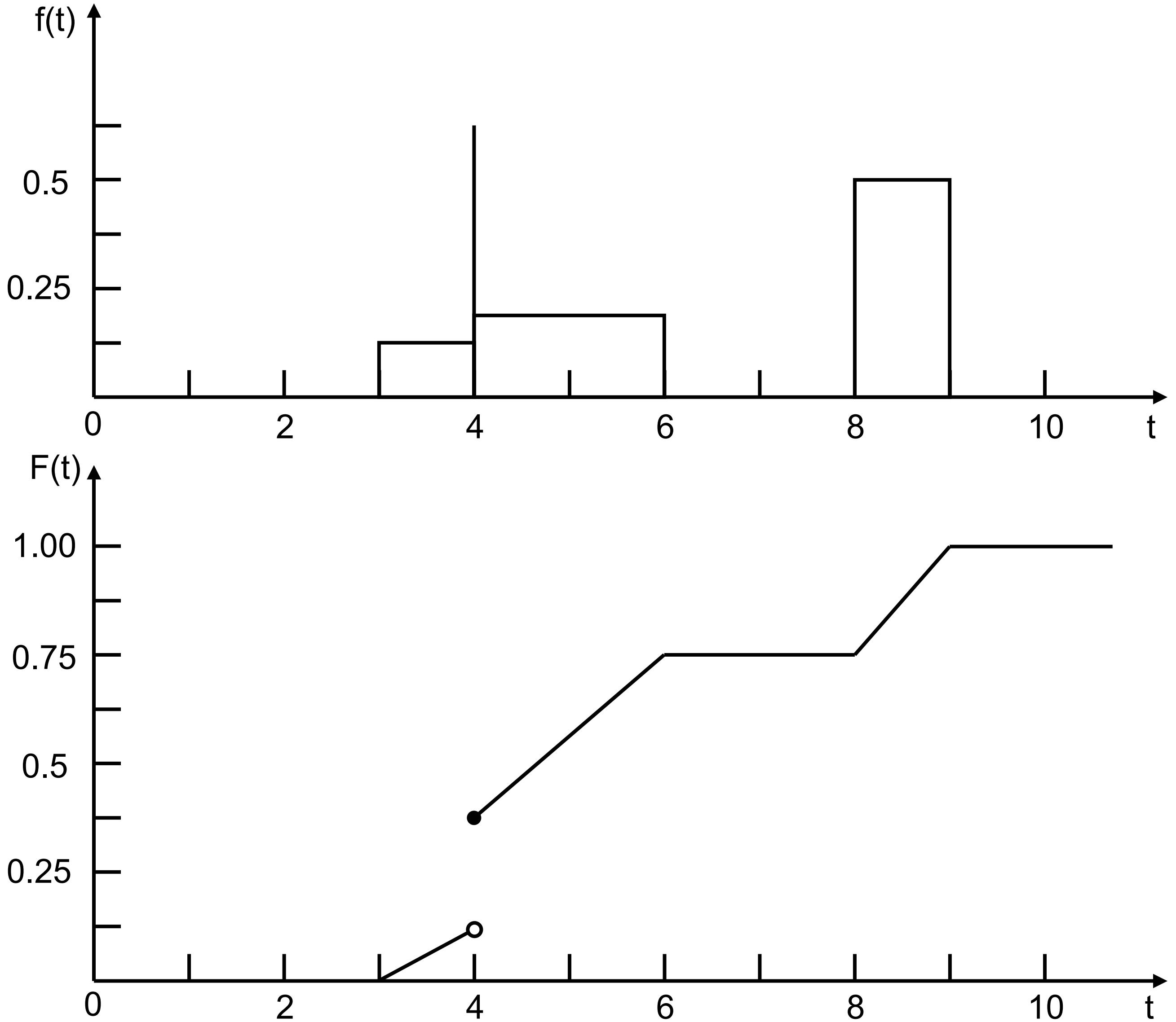


no unnecessary increase in makespan

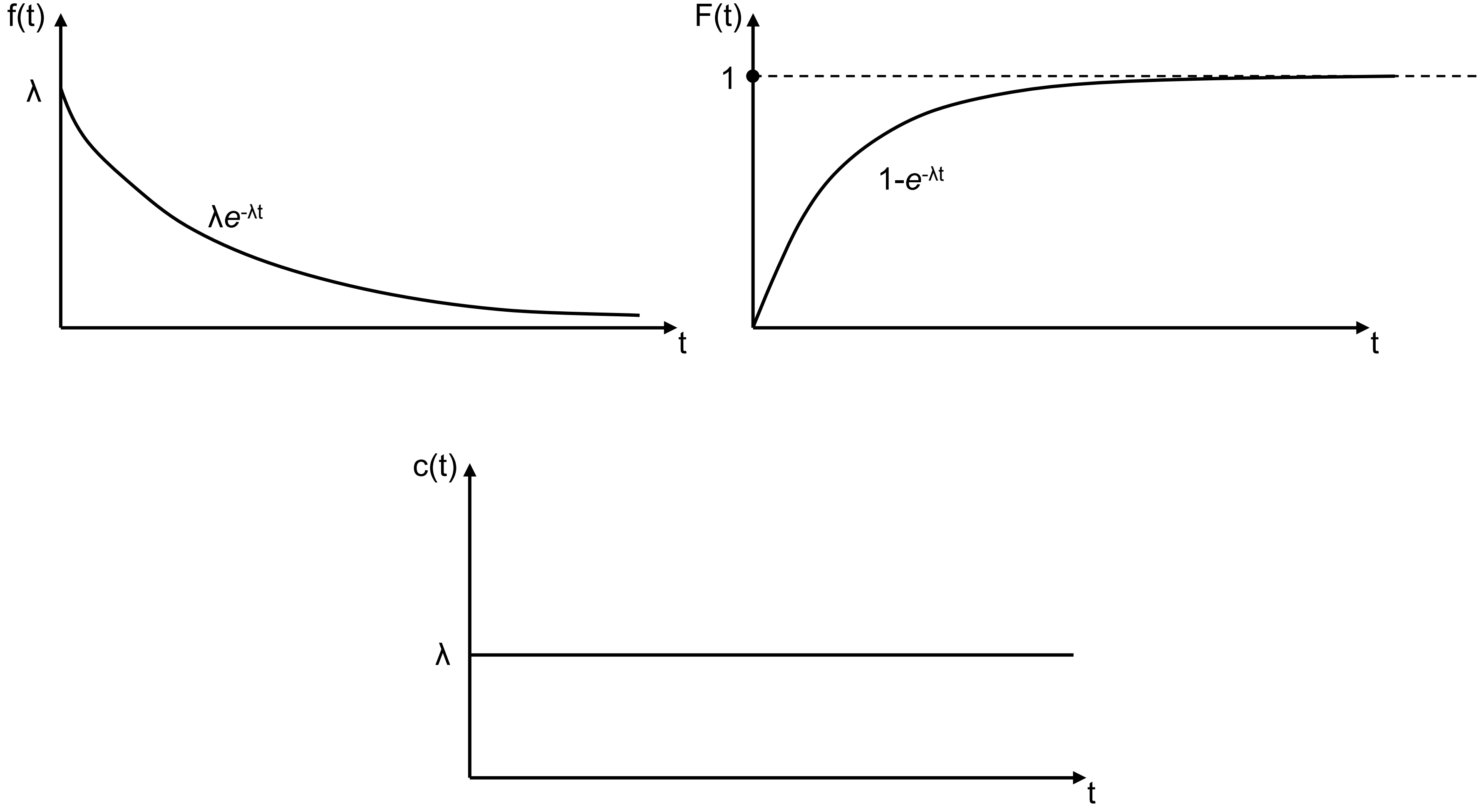
Stochastic Models: Notation

- X_{ij} = the random processing time of job j on machine i
- $1/\lambda_{ij}$ = the mean or expected value of the random variable X_{ij}
- R_j = the random release date of job j
- D_j = the random due date of job j
- w_j = the weight (or importance factor) of job j

Density and Distribution Function Example



The Exponentional Distribution



Stochastic Dominance

- X_1 is said to be larger in expectation than X_2 if $E(X_1) \geq E(X_2)$.
- X_1 is said to be stochastically larger than X_2 if

  $P(X_1 > t) \geq P(X_2 > t)$ or $1-F_1(t) \geq 1-F_2(t)$ for all t.

  - Notation: $X_1 \geq_{st} X_2$

- Likelihood ratio sense
  - Continuous case: X_1 is larger than X_2 in the likelihood ratio sense if $f_1(t)/f_2(t)$ is nondecreasing in t, $t \geq 0$.
  - Discrete case: X_1 is larger than X_2 in the likelihood ratio sense if $P(X_1=t)/P(X_2=t)$ is nondecreasing in t, $t = 0, 1, 2, \ldots$
  - Notation: $X_1 \geq_{lr} X_2$
- X_1 is almost surely larger than or equal X_2 if $P(X_1 \geq X_2) = 1$.
  - Implies that f_1 and f_2 may overlap at most on one point
  - Notation: $X_1 \geq_{a.s.} X_2$

Stochastic Dominance

Chain of implications

$$\begin{tikzcd} \boxed{\text{Almost surely larger}} \arrow[r] & \boxed{\text{Larger in likelihood ratio sense}} \arrow[d] \\ \boxed{\text{Larger in expectation}} & \boxed{\text{Stochastically larger}} \arrow[l] \end{tikzcd}$$

Stochastic Dominance based on Variance

- X_1 is said to be larger than X_2 in the variance sense if

$$\text{var}(X_1) > \text{var}(X_2)$$

- X_1 is said to be more variable than X_2 if

$$\int_0^\infty h(t)dF_1(t) \geq \int_0^\infty h(t)dF_2(t) \qquad \text{continuous case}$$

$$\sum_{t=0}^\infty h(t)P(X_1 = t) \geq \sum_{t=0}^\infty h(t)P(X_2 = t) \qquad \text{discrete case}$$

for all convex functions h.

  - Notation: $X_1 \geq_{cx} X_2$

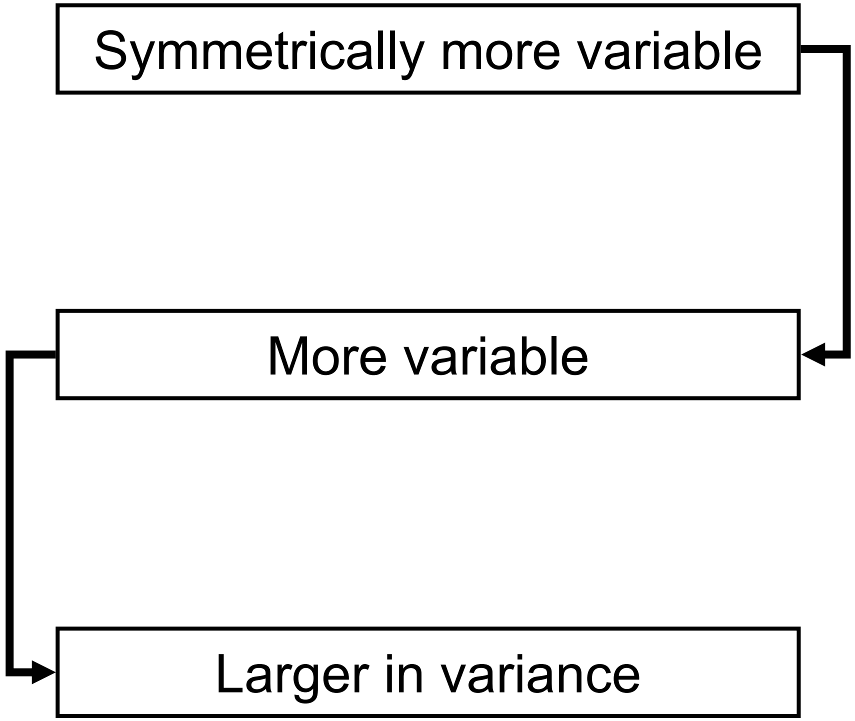
- X_1 is said to be symmetrically more variable than X_2 if $f_1(t)$ and $f_2(t)$ are symmetric around the same mean $1/\lambda$ and

$$F_1(t) \geq F_2(t) \quad \text{for} \quad 0 \leq t \leq 1/\lambda \quad \text{and}$$

$$F_1(t) \leq F_2(t) \quad \text{for} \quad 1/\lambda \leq t \leq 2/\lambda$$

Stochastic Dominance based on Variance

Chain of implications



Increasing Convex Ordering

- X_1 is said to larger than X_2 in the increasing convex sense if

$$\int_0^\infty h(t)dF_1(t) \geq \int_0^\infty h(t)dF_2(t) \qquad \text{continuous case}$$

$$\sum_{t=0}^\infty h(t)P(X_1 = t) \geq \sum_{t=0}^\infty h(t)P(X_2 = t) \qquad \text{discrete case}$$

for all increasing convex functions h.

  - Notation: $X_1 \geq_{icx} X_2$

Stochastically larger → Larger in the increasing convex sense

More variable → Larger in the increasing convex sense

Lemma: Increasing Convex Ordering

- Two vectors of independent random variables

  $X_1^{(1)}, \ldots, X_n^{(1)}$ and $X_1^{(2)}, \ldots, X_n^{(2)}$

  All 2n variables are independent
- Let

  $$Z_1 = g(X_1^{(1)}, \ldots, X_n^{(1)})$$

  and

  $$Z_2 = g(X_1^{(2)}, \ldots, X_n^{(2)})$$

  where g is increasing convex in each one of the n arguments.

- If $X_j^{(1)} \geq_{icx} X_j^{(2)}$, $j = 1, \ldots, n$, then $Z_1 \geq_{icx} Z_2$
  - Proof by induction

Classes of Policies

- Nonpreemptive Static List Policy

  The decision maker orders the jobs at time zero according to a priority list which does not change during the evolution of the process. Every time a machine is freed the next job on the list is selected for processing.

- Preemptive Static List Policy

  The decision maker orders the jobs at time zero according to a priority list which includes jobs with nonzero release dates. At any point in time the job at the top of the list of available jobs is the one to be processed on the machines.

Classes of Policies

- Nonpreemptive Dynamic Policy

  Every time a machine is freed, the decision maker is allowed to determine which job goes next.

  - Decision may depend on available information like current time, number of waiting jobs, number of currently processed jobs…
  - Preemption is not allowed. Every job that is started has to be executed without interruption.

- Preemptive Dynamic Policy

  Every time a machine is freed, the decision maker is allowed to determine which job goes next.

  - Preemption is allowed