



**OPPA European Social Fund  
Prague & EU: We invest in your future.**

---

# Combinatorial Optimization

Zdeněk Hanzálek  
hanzalek@fel.cvut.cz

CTU FEE Department of Control Engineering

May 3, 2013



European Social Fund Prague & EU: We invests in your future.

# Scheduling

Zdeněk Hanzálek, Přemysl Šůcha  
hanzalek@fel.cvut.cz

CTU FEE Department of Control Engineering

April 30, 2013

# Table of Contents

## 1 Basics notions

## 2 Scheduling on One Resource

- Minimizing  $C_{max}$ 
  - Bratley's Algorithm for  $1 |r_j, \tilde{d}_j| C_{max}$
- Minimizing  $\sum w_j C_j$ 
  - Branch&Bound with LP for  $1 |prec| \sum w_j C_j$

## 3 Scheduling on Parallel Identical Resources

- Minimizing  $C_{max}$

## 4 Project Scheduling

- Temporal constraints
- Minimizing  $C_{max}$ 
  - ILP Formulation for  $PS1 |temp| C_{max}$  - One Resource
  - ILP formulation for  $PSm, 1 |temp| C_{max}$  -  $m$  Dedicated Resources
- Modeling with Temporal Constraints

# Scheduling - Basic Terminology

- **set of  $n$  tasks**  $\mathcal{T} = \{T_1, T_2, \dots, T_n\}$
- set of  $m$  types of resources (processors, machines, employees,...) with capacities  $R_k$ ,  $\mathcal{P} = \{P_1^1, \dots, P_1^{R_1}, P_2^1, \dots, P_2^{R_2}, \dots, P_m^1, \dots, P_m^{R_m}\}$
- **Scheduling is an assignment of a task to a resources in time**
- Each task must be **completed**
  - this differs from planning which decides which task will be scheduled and processed
- Set of tasks is known when executing the scheduling algorithm (this is called **off-line scheduling**)
  - this differs from on-line scheduling - OS scheduler, for example, schedules new tasks using some policy (e.g. priority levels)
- A result is a schedule which determines which task is run on which resource and when. Often depicted as a **Gantt chart**.

# General and Specific Constraints

## General constraints:

- Each task is to be processed **by at most one resource** at a time (task is sequential)
- Each resource is capable of processing **at most one task** at a time

## Specific constraints:

- Task  $T_i$  has to be processed during time interval  $\langle r_i, \tilde{d}_i \rangle$
- When the precedence constraint is defined between  $T_i$  and  $T_j$ , i.e.  $T_i \prec T_j$ , then the processing of task  $T_j$  can't start before task  $T_i$  was completed
- If scheduling is non-preemptive, a task cannot be stopped and completed later
- If scheduling is preemptive, the number of preemptions must be finite

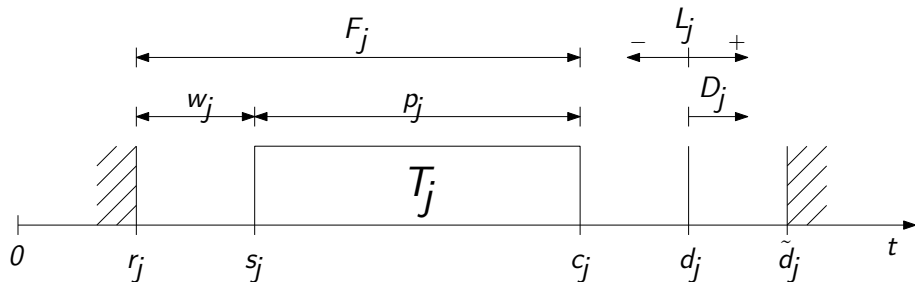
# Task Parameters and Variables

## Parameters

- **release time**  $r_j$
- **processing time**  $p_j$
- **due date**  $d_j$ , time in which task  $T_j$  should be completed
- **deadline**  $\tilde{d}_j$ , time in which task  $T_j$  has to be completed

## Variables

- **start time**  $s_j$
- **completion time**  $C_j$
- waiting time  $w_j = s_j - r_j$
- flow time  $F_j = C_j - r_j$
- **lateness**  $L_j = C_j - d_j$
- tardiness  $D_j = \max\{C_j - d_j, 0\}$



# Graham's Notation $\alpha | \beta | \gamma$

Classify scheduling problems by  
**resources** | **tasks** | **criterion**

Example:  $P2 | \text{pmtn} | C_{max}$  represents scheduling on two parallel identical resources, and preemption is allowed. The optimization criterion is the completion time of the last task.

$\alpha$  - **resources**

- **Parallel resources** - a task can run on any resource (only one type of resource exists with capacity  $R$ , i.e.  $\mathcal{P} = \{P^1, \dots, P^R\}$ ).
- **Dedicated resources** - a task can run only on one resource ( $m$  resource types with unit capacity, i.e.  $\mathcal{P} = \{P_1, P_2, \dots, P_m\}$ ).
- **Project Scheduling** -  $m$  resource types, each with capacity  $R_k$ , i.e.  $\mathcal{P} = \{P_1^1, \dots, P_1^{R_1}, P_2^1, \dots, P_2^{R_2}, \dots, P_m^1, \dots, P_m^{R_m}\}$ .



# Resources Characteristics $\alpha_1, \alpha_2$

$\alpha_1$	=	1	single resource
		P	<b>parallel identical</b> resources
		Q	<b>parallel uniform</b> resources, computation time is inversely related to resource speed
		R	<b>parallel unrelated</b> resources, computation times are given as a matrix (resources x tasks)
		O	dedicated resources - <b>open-shop</b> - tasks are independent
		F	dedicated resources - <b>flow-shop</b> - tasks are grouped in the sequences (jobs) in the same order, each job visits each machine once
		J	dedicated resources - <b>job-shop</b> - order of tasks in jobs is arbitrary, resource can be used several times in a job
		PS	<b>Project Scheduling</b> - most general (several resource types with capacities, general precedence constraints )
$\alpha_2$	=	$\emptyset$	arbitrary <b>number</b> of resources
		2	2 resources (or other specified number)
		$m, R$	$m$ resource types with capacities $R$ (Project Scheduling)

# Task Characteristics $\beta_1, \beta_2, \beta_3, \beta_4, \beta_5, \beta_6, \beta_7, \beta_8$

$\beta_1$	= pmtn $\emptyset$	<b>preemption is allowed</b> preemption is not allowed
$\beta_2$	= prec in-tree,out-tree chain tmpn $\emptyset$	<b>precedence</b> constraints tree constraints chain constraints <b>temporal</b> constraints (for Project Sched.) independent tasks
$\beta_3$	= $r_j$	release time
$\beta_4$	= $p_j = k$ $p_L \leq p_j \leq p_U$ $\emptyset$	uniform processing time restricted processing time <b>arbitrary processing time</b>
$\beta_5$	= $d_j, d_j$	deadline, due-date
$\beta_6$	= $n_j \leq k$	maximal number of tasks in a job
$\beta_7$	= no-wait	buffers of zero capacity
$\beta_8$	= set-up	time for resource <b>reconfiguration</b>

---

$\gamma = C_{max}$	minimize schedule length $C_{max} = \max \{C_j\}$ (makespan, i.e. completion time of the last task)
$\sum C_j$	minimize the sum of completion times
$\sum w_j C_j$	minimize weighted completion time
$L_{max}$	minimize max. lateness $L_{max} = \max \{C_j - d_j\}$
$\emptyset$	decision problem
...	

---

An Example:  $P \parallel C_{max}$  means:

Scheduling on an arbitrary number of parallel identical resources, no preemption, independent tasks (no precedence), tasks arrive to the system at time 0, processing times are arbitrary, objective is to minimize the schedule length.

# Scheduling on One Resource

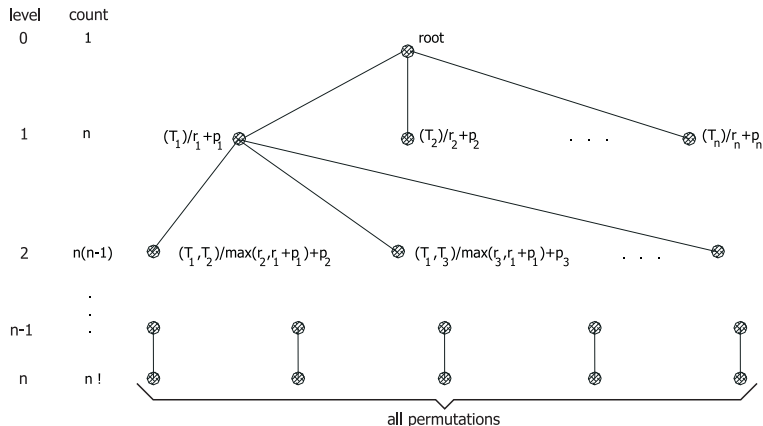
## Minimizing Makespan (i.e. schedule length $C_{max}$ )

- $1 |prec| C_{max}$  - easy
  - the tasks are processed in an arbitrary order that satisfies the precedence relation,  $C_{max} = \sum_{j=1}^n p_j$
- $1 || C_{max}$  - easy
- $1 |r_j| C_{max}$  - easy
  - the tasks are processed in a non-descending order of  $r_j$  ( $T_j$  with the lowest  $r_j$  first)
- $1 |\tilde{d}_j| C_{max}$  - easy
  - the tasks are processed in a non-descending order of  $\tilde{d}_j$
  - can be solved by EDF - Earliest Deadline First
  - the feasible schedule doesn't have to exist
- $1 |r_j, \tilde{d}_j| C_{max}$  - NP-hard
  - NP-hardness proved by the polynomial transformation from 3-Partition problem
  - for  $p_j = 1$  there exists a polynomial algorithm

# Bratley's Algorithm for $1 \mid r_j, \tilde{d}_j \mid C_{max}$

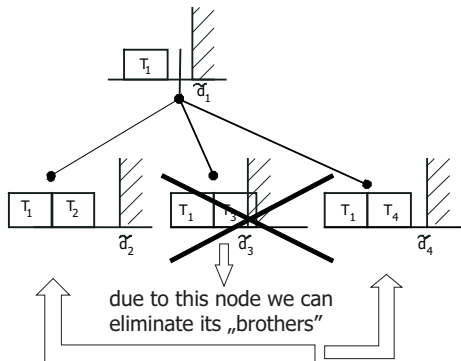
A **branch and bound** (B&B) algorithm.

Branching - without bounding it is an **enumerative method** that creates a solution tree (some of the nodes are infeasible). Every node is labeled by: (the order of tasks)/(completion time of the last task).



(i) **eliminate the node** exceeding the deadline (and all its “brothers”)

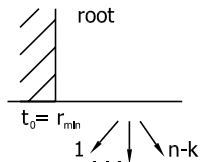
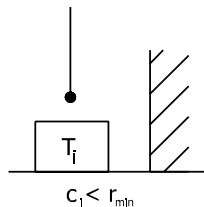
- If there is a node which exceeds any deadline, all its descendants should be eliminated
- Critical task (here  $T_3$ ) will have to be scheduled anyway - therefore, all of its “brothers” should be eliminated as well



# Tree Size Reduction - Decomposition

(ii) **problem decomposition** due to idle waiting - e.g. when the employee waits for the material, his work was optimal

- Consider node  $v$  on level  $k$ . If  $C_i$  of the last scheduled task is less than or equal to  $r_i$  of all unscheduled tasks, there is no need for backtrack above  $v$
- $v$  becomes a new root and there are  $n - k$  levels ( $n - k$  unscheduled tasks) to be scheduled



# Optimality Test - Termination of Bratley's Algorithm

## Definition: BRTP - Block with Release Time Property

BRTP is a set of  $k$  tasks that satisfy:

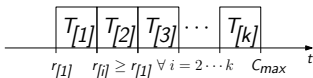
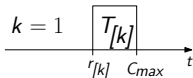
- first task  $T_{[1]}$  starts at its release time
- all tasks till the end of the schedule run without "idle waiting"
- $r_{[1]} \leq r_{[i]}$  for all  $i = 2 \dots k$

Note: "till the end of the schedule" implies there is at most one BRTP

## Lemma: sufficient condition of optimality

If BRTP exists, the schedule is optimal (the search is finished).

Proof:



- this schedule is optimal since the last task  $T_{[k]}$  can not be completed earlier
- order of prec. tasks is not important - see (ii)
- no task from BRTP can be done before  $r_{[1]}$
- there is no task after  $C_{max}$



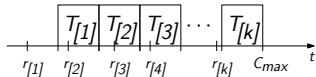
# B RTP is Sufficient and Necessary Condition of Optimality

## Proposition - necessary condition of optimality

If the schedule for  $1 \mid r_j, \tilde{d}_j \mid C_{max}$  is optimal, it contains B RTP.

Proof by contradiction: we show that the schedule without B RTP is not optimal. There are two cases of the schedule without B RTP:

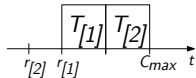
① the last block does not start at  $r_{[1]}$ :



- $r_{[1]} < s_{[1]}$
- $s_{[1]} < r_{[i]} < s_{[i]} \forall i = 2 \dots k$  (note that if  $i = 2 \dots k$  such that  $r_{[i]} = s_{[i]}$  exists, then B RTP exists from  $T_{[i]}$ )
- block can be moved left while maintaining actual order

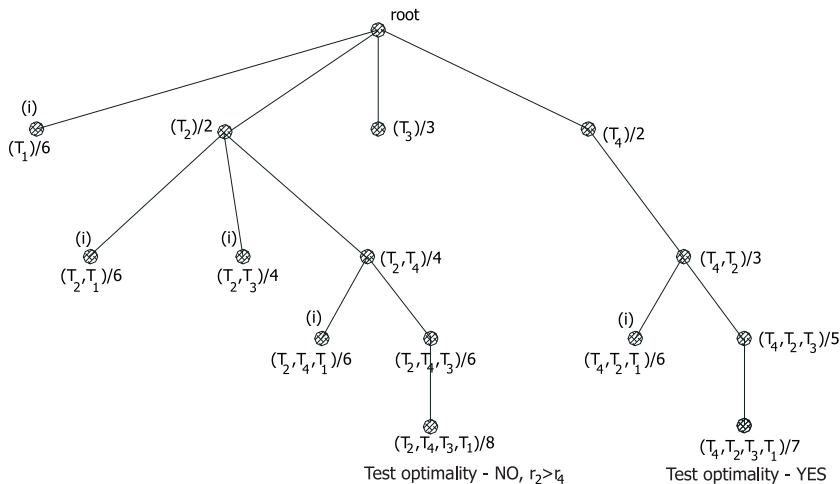
② some task can be placed before  $T_{[1]}$ , i.e. there is  $i = 2 \dots k$  such that  $r_{[i]} < s_{[1]}$  exists

- schedule can be improved while moving  $T_{[2]}$  before  $T_{[1]}$



# Bratley's Algorithm - Example

$$r = [4, 1, 1, 0], \quad p = [2, 1, 2, 2], \quad \tilde{d} = [8, 5, 6, 4]$$



# Scheduling on One Resource

## Minimizing $\sum w_j C_j$

- $1 \parallel \sum C_j$  - easy
  - SPT rule (Shortest Processing Time first) - schedule the tasks in a non-decreasing order of  $p_j$
- $1 \parallel \sum w_j C_j$  - easy
  - Weighted SPT - schedule the tasks in a non-decreasing order of  $\frac{p_j}{w_j}$
- $1 |r_j| \sum C_j$  - NP-hard
- $1 |pmtn, r_j| \sum C_j$  - can be solved by modified SPT
- $1 |pmtn, r_j| \sum w_j C_j$  - NP-hard
- $1 \left| \tilde{d}_j \right| \sum C_j$  - can be solved by modified SPT
- $1 \left| \tilde{d}_j \right| \sum w_j C_j$  - NP-hard
- $1 |prec| \sum C_j$  - NP-hard

First, we formulate the problem as a ILP:

- we use **variable**  $x_{ij} \in \{0, 1\}$  such that  $x_{ij} = 1$  iff  $T_i$  precedes  $T_j$  or  $i = j$
- we encode **precedence relations** into  $e_{ij} \in \{0, 1\}$  such that  $e_{ij} = 1$  iff there is a directed edge from  $T_i$  to  $T_j$  in the precedence graph  $G$  or  $i = j$
- **criterion** - completion time of task  $T_j$  consists of  $p_j$  and the processing time of its predecessors:

$$C_j = \sum_{i=1}^n p_i \cdot x_{ij}$$

$$w_j \cdot C_j = \sum_{i=1}^n p_i \cdot x_{ij} \cdot w_j$$

$$J = \sum_{j=1}^n w_j \cdot C_j = \sum_{j=1}^n \sum_{i=1}^n p_i \cdot x_{ij} \cdot w_j$$

from all feasible schedules  $x$  we look for the one that minimizes  $J(x)$ ,  
i.e.  $\min_x J(x)$

# ILP formulation for $1 \mid \text{prec} \mid \sum w_j C_j$

$$\begin{aligned} \min \quad & \sum_{j=1}^n \sum_{i=1}^n p_i \cdot x_{ij} \cdot w_j \\ \text{subject to:} \quad & x_{i,j} \geq e_{i,j} \quad i, j \in 1..n && \text{if } T_i \text{ precedes } T_j \text{ in } G, \\ & && \text{then it precedes } T_j \\ & && \text{in the schedule} \\ & x_{i,j} + x_{j,i} = 1 \quad i, j \in 1..n, i \neq j && \text{either } T_i \text{ precedes } T_j, \\ & && \text{or vice versa} \\ & 1 \leq x_{i,j} + x_{j,k} + x_{k,i} \leq 2 \quad i, j, k \in 1..n, && \text{no cycle exists in the} \\ & && \text{digraph of } x \\ & && i \neq j \neq k \\ & x_{i,i} = 1 \quad i \in 1..n \end{aligned}$$

parameters:  $p_{i \in 1..n} \in \mathbb{R}_0^+$   $e_{i \in 1..n, j \in 1..n} \in \{0, 1\}$

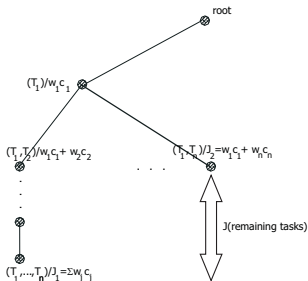
variables:  $x_{i \in 1..n, j \in 1..n} \in \{0, 1\}$

# Branch and Bound with LP Bounding

We relax on the integrality of variable  $x$ :

- $0 \leq x_{ij} \leq 1$  and  $x_{i \in 1..n, j \in 1..n} \in \mathbb{R}$
- This does not give us the right solution, however we can use the  $J^{LP}$ (remaining tasks) value of this LP formulation as a lower bound on the “amount of remaining work”

The Branch and Bound algorithm creates a similar tree as Bradley’s algorithm.



- Let  $J_1$  be the value of the best solution known up to now
- We discard the partial solution of value  $J_2$  not only when  $J_2 \geq J_1$ , but also when  $J_2 + J^{LP}(\text{remaining tasks}) \geq J_1$ . Since the solution space of ILP is a subspace of LP we know:  
 $J(\text{remaining tasks}) \geq J^{LP}(\text{remaining tasks})$ .

# Scheduling on Parallel Identical Resources

## Minimizing $C_{max}$

- $P2 || C_{max}$  - NP-hard
  - schedule  $n$  non-preemptive tasks on two parallel identical resources minimizing makespan, i.e. the completion time of the last task
  - the problem is NP-hard because the **2 partition problem** (see ILP lecture) can be reduced to  $P2 || C_{max}$  while comparing the optimal  $C_{max}$  with the threshold of  $0.5 * \sum_{i \in 1..n} p_i$ .
- $P | \text{pmtn} | C_{max}$  - easy
  - can be solved by the **McNaughton** algorithm in  $O(n)$
- $P | \text{pmtn}, r_j, \tilde{d}_j | C_{max}$  - easy
  - can be formulated as a **maximum flow problem** (see the lecture on Flows)
- $P | \text{prec} | C_{max}$  - NP-hard
  - **LS - approximation algorithm** with factor  $r_{LS} = 2 - \frac{1}{R}$ , where  $R$  is the number of parallel identical resources
- $P || C_{max}$  - NP-hard
  - **LPT - approximation algorithm** with factor  $r_{LPT} = \frac{4}{3} - \frac{1}{3R}$
  - **dynamic programming** - Rothkopf's pseudopolynomial algorithm
- $P | \text{pmtn}, \text{prec} | C_{max}$  - NP-hard
  - Muntz&Coffman's **level algorithm** with factor  $r_{MC} = 2 - \frac{2}{R}$

# McNaughton's Algorithm for $P |pmtn| C_{max}$

**Input:**  $R$ , number of parallel identical resources,  $n$ , number of preemptive tasks and computation times  $[p_1, p_2, \dots, p_n]$ .

**Output:**  $n$ -element vectors  $s^1, s^2, z^1, z^2$  where  $s_i^1$  (resp.  $s_i^2$ ) is start time of the first (resp. second) part of task  $T_i$  and  $z_i^1$  (resp.  $z_i^2$ ) is the resource ID on which the first (resp. second) part of task  $T_i$  will be executed.

$s_i^1 = s_i^2 = z_i^1 = z_i^2 := 0$  for all  $i \in 1 \dots n$ ;

$t := 0; v := 1; i := 1$ ;

$C_{max}^* = \max \{ \max_{i=1 \dots n} \{p_i\}, \frac{1}{R} \sum_{i=1}^n p_i \}$ ;

**while**  $i \leq n$  **do**

**if**  $t + p_i \leq C_{max}^*$  **then**

$s_i^1 := t; z_i^1 := v; t := t + p_i; i := i + 1$ ;

**else**

$s_i^2 := t; z_i^2 := v; p_i := p_i - (C_{max}^* - t); t := 0; v := v + 1$ ;

**end**

**end**



# McNaughton's Algorithm for $P | \text{pmtn} | C_{max}$

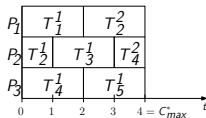
The term  $C_{max}^* = \max \left\{ \max_{i=1 \dots n} \{p_i\}, \frac{1}{R} \sum_{i=1}^n p_i \right\}$  should be interpreted as follows:

- component  $\max_{i=1 \dots n} \{p_i\}$  represents the sequential nature of each task - it's parts can be assigned to different resources, but these parts can not be run simultaneously. Note that each task can be divided into two parts at most.
- component  $\frac{1}{R} \sum_{i=1}^n p_i$  represents a situation when all resources work without idle waiting

Example 1:

$$p = [2, 3, 2, 3, 2], R = 3$$

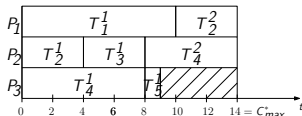
$$\text{compute } C_{max}^* = \max \left\{ 3, \frac{12}{3} \right\} = 4$$



Example 2:

$$p = [10, 8, 4, 14, 1], R = 3$$

$$\text{compute } C_{max}^* = \max \left\{ 14, \frac{37}{3} \right\} = 14$$



**Input:**  $R$ , number of parallel identical resources,  $n$ , number of non-preemptive tasks and computation times  $[p_1, p_2, \dots, p_n]$ .  $G$ , digraph of precedence constraints.

**Output:**  $n$ -element vectors  $s$  and  $z$  where  $s_i$  is the start time of  $T_i$  and  $z_i$  is the resource ID.

```

 $t_v := 0$  for all  $v \in 1 \dots R$ ;           // availability of resource
 $s_i = z_i := 0$  for all  $i \in 1 \dots n$ ;
Sort tasks in list L;
for  $count := 1$  to  $n$  do                // for all tasks
|    $k = \arg \min_{v=1 \dots R} \{t_v\}$ ; // choose res. with the lowest  $t_v$ 
|   Remove the first free task  $T_i$  from  $L$ ;
|    $s_i = \max\{t_k, \max_{j \in \text{Pred}(T_i)} \{s_j + p_j\}\}$ ;  $z_i = k$ ; // assign  $T_i$  to  $P_k$ 
|    $t_k = s_i + p_i$ ; // update availability time of  $P_k$ 
end

```

Task  $T_i$  is **free** if its predecessors have been completed.  $\text{Pred}(T_i)$  is a set of the task IDs that are **predecessors** of  $T_i$ . Complexity is  $O(n)$ .

# List Scheduling - Approximation algorithm for $P | \text{prec} | C_{max}$

List Scheduling (LS) is a general heuristic useful in many problems.

- We have a list ( $n$ -tuple) of tasks and when some resource is free, we assign the first free task from the list to this resource.
- The accuracy of LS depends on the criterion and sorting procedure.

## Approximation factor of LS algorithm [Graham 1966]

For  $P | \text{prec} | C_{max}$  (and also for  $P || C_{max}$ ) and arbitrary (unsorted) list  $L$ , List Scheduling is an approximation algorithm with factor  $r_{LS} = 2 - \frac{1}{R}$

An example illustrating the case when the factor is attained:

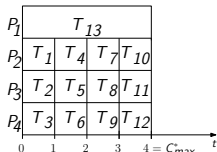
$$n = (R - 1) \cdot R + 1,$$

$$p = [1, 1, \dots, 1, R],$$

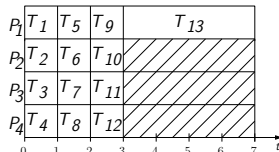
$\prec$  empty.

Illustration for  $R = 4$

$$r_{LS} = 2 - \frac{1}{4} = \frac{7}{4}$$



$$L = [T_n, T_1, \dots, T_{n-1}]$$



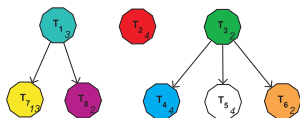
$$L' = [T_1, T_2, \dots, T_n]$$

# Anomalies of List Scheduling Algorithm

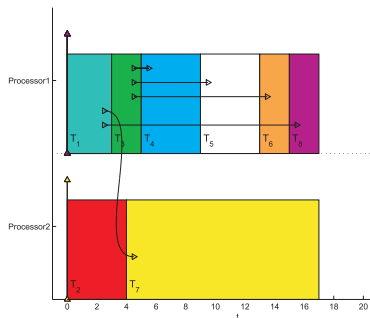
The LS algorithm depends not only on the **order of tasks in  $L$** , but it exhibits **anomalies** ( $C_{max}$  surprisingly increases when relaxing some constraints/parameters) caused by:

- 1 the decrease of processing time  $p_i$
- 2 the removal of some precedence constraints
- 3 the increase of the number of resources  $R$

Example illustrating different anomalies for  
 $R = 2$ ,  $n = 8$ ,  $p = [3, 4, 2, 4, 4, 2, 13, 2]$

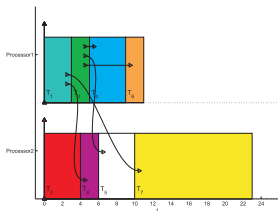


Using list  $L = [T_1, T_2, T_3, T_4, T_5, T_6, T_7, T_8]$ ,  
LS finds solution with  $C_{max}^* = 17$ .

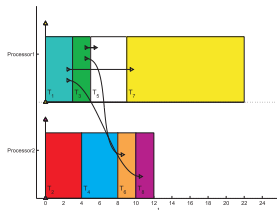


# List Scheduling Anomalies - Prolongation of $C_{max}$

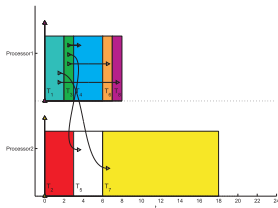
Exchange position of  $T_7$  and  $T_8$   
 $L = [T_1, T_2, T_3, T_4, T_5, T_6, T_8, T_7]$ .



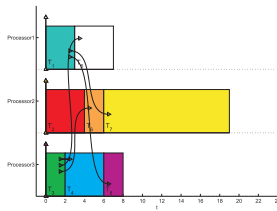
Remove prec. constraint  $T_3 \prec T_4$ .



Decrease  $p_i$  of all tasks by one.



Add resource ( $R = 3$ ).



# LPT (Longest Processing Time First)

- Approximation Algorithm for  $P \parallel C_{max}$

The approximation factor of the LS algorithm can be decreased using the **Longest Processing Time first** (LPT) strategy

- During initialization of LS, we sort list  $L$  in a non-increasing order of  $p_i$

Approximation factor of LPT algorithm [Graham 1966]

LPT algorithm for  $P \parallel C_{max}$  is an approximation algorithm with factor

$$r_{LPT} = \frac{4}{3} - \frac{1}{3R}$$

Time complexity of LPT algorithm is  $O(n \cdot \log(n))$  due to the sorting.

# LPT (Longest Processing Time First)

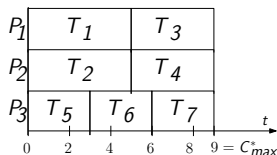
- Approximation Algorithm for  $P \parallel C_{max}$

An example illustrating the case when the factor is attained:

$$p = [2R - 1, 2R - 1, 2R - 2, 2R - 2, \dots, R + 1, R + 1, R, R, R]$$

$$n = 2 \cdot R + 1, \prec \text{ empty,}$$

optimum:



LPT:

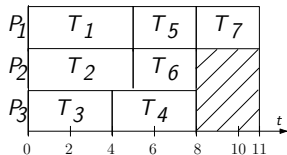


Illustration for  $R = 3$

$$r_{LPT} = \frac{4}{3} - \frac{1}{9} = \frac{11}{9}$$

## Factor of LPT algorithm

If the number of tasks is big, the factor can get better depending on  $k$  - the number of tasks assigned to the resource which finishes last:

$$r_{LPT} = 1 + \frac{1}{k} - \frac{1}{kR}$$

Pseudopolynomial algorithm - the range of discrete values is limited by the upper bound. In some special cases there exists a polynomial algorithm for such a restricted problem.

- we add a binary variable  $x_i(t_1, t_2, \dots, t_R)$  where
  - $i = 1, 2, \dots, n$  is the task index
  - $v = 1, 2, \dots, R$  is the index of the resource
  - $t_v = 0, 1, 2, \dots, UB$  is the time variable associated to the resource  $v$
  - $UB$  is upper bound on  $C_{max}$
- $x_i(t_1, t_2, \dots, t_R) = 1$  iff tasks  $T_1, T_2, \dots, T_i$  can be assigned to the resource such that  $P_v$  is occupied during the time interval  $\langle 0, t_v \rangle$ ;  $v = 1, 2, \dots, R$



**Input:**  $R$ , the number of parallel identical resources,  $n$ , the number of nonpreemptive tasks and their processing time  $[p_1, p_2, \dots, p_n]$ .

**Output:**  $n$ -elements vectors  $s$  and  $z$  where  $s_i$  is the start time and  $z_i$  is the resource ID.

**for**  $(t_1, t_2, \dots, t_R) \in \{1, 2, \dots, UB\}^R$  **do**  $x_0(t_1, t_2, \dots, t_R) := 0$ ;

$x_0(0, 0, \dots, 0) := 1$ ;

**for**  $i := 1$  **to**  $n$  **do** // for all tasks

**for**  $(t_1, t_2, \dots, t_R) \in \{0, 1, 2, \dots, UB\}^R$  **do** // in the whole space

$x_i(t_1, t_2, \dots, t_R) := \text{OR}_{v=1}^R x_{i-1}(t_1, t_2, \dots, t_v - p_i, \dots, t_R)$ ;

//  $x_i() = 1$  iff there existed

//  $x_{i-1}() = 1$  “smaller” by  $p_i$  in any direction

**end**

**end**

$C_{max}^* = \min_{x_n(t_1, t_2, \dots, t_R)=1} \{ \max_{v=1, 2, \dots, R} \{ t_v \} \}$ ;

Assign tasks  $T_n, T_{n-1}, \dots, T_1$  in the reverse direction;

Time complexity is  $O(n \cdot UB^R)$ . Example  $n=3, R=2, p=[2,1,2], C=5$ .

Principle:

- tasks are picked from **the list ordered by the level** of tasks
- **the level of task**  $T_j$  - sum of  $p_i$  (including  $p_j$ ) along the **longest path** from  $T_j$  to a terminal task (a task with no successor)
- when more tasks of the same level are assigned to less resources, each task gets **part of the resource capacity**  $\beta$
- the algorithm moves forward to time  $\tau$  when **one of the tasks ends** or the task with a lower level would be processed by a bigger capacity  $\beta$  than the tasks with a higher level

For  $P2 | \text{pmtn, prec} | C_{max}$  and  $P | \text{pmtn, forest} | C_{max}$ , the algorithm is **exact**.

For  $P | \text{pmtn, prec} | C_{max}$  **approximation** alg. with factor  $r_{MC} = 2 - \frac{2}{R}$ .

Time complexity is  $O(n^2)$ .

**Input:**  $R$ , the number of parallel identical resources,  $n$ , the number of preemptive tasks and proc. times  $[p_1, p_2, \dots, p_n]$ . Prec. graph  $G$ .

**Output:**  $n$ -elements vectors  $s$  and  $z$  where  $s_i$  is the start time and  $z_i$  is the resource ID.

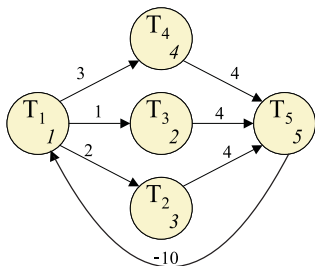
# Muntz&Coffman's Level Algorithm for $P | \text{pmtn, prec} | C_{max}$

```
compute the level of all tasks ; t:=0; h:=R; // h represents free res
while unfinished tasks exists do
  |
  | construct  $\mathcal{Z}$ ; // subset  $\mathcal{T}$  of free tasks in time t
  | while  $h > 0$  and  $|\mathcal{Z}| > 0$  do // free resources and free tasks
  | | construct  $\mathcal{S}$ ; // subset  $\mathcal{Z}$  of tasks of the highest level
  | | if  $|\mathcal{S}| > h$  then // more tasks than resources
  | | | assign part of capacity  $\beta := \frac{h}{|\mathcal{S}|}$  to tasks in  $\mathcal{S}$ ;  $h := 0$ ;
  | | | else
  | | | | assign one resource to each task in  $\mathcal{S}$ ;  $\beta := 1$ ;  $h := h - |\mathcal{S}|$ ;
  | | | end
  | | |  $\mathcal{Z} := \mathcal{Z} \setminus \mathcal{S}$ ;
  | end
  | compute  $\tau$ ; // time when one of the tasks is finished
  | decrease level of tasks by  $(\tau - t) \cdot \beta$ ; // finished part of task
  |  $t := \tau$ ;  $h := R$ ;
end
```

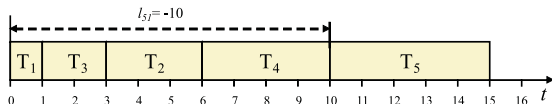
Use McNaughton's alg. to re-schedule parts with more tasks on less res.:

# Project Scheduling with Temporal Constraints

- Set of non-preemptive tasks  $\mathcal{T} = \{T_1, T_2, \dots, T_n\}$  is represented by the nodes of the directed graph  $G$ .
- Processing time  $p_i$  is assigned to each task.



- The edges represent temporal constraints. Each edge from  $T_i$  to  $T_j$  has the length  $l_{ij}$ .
- Each temporal constraint is characterized by one inequality  $s_i + l_{ij} \leq s_j$ .



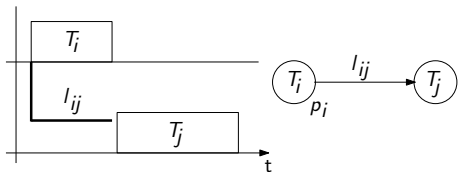
# Temporal Constraints $s_i + l_{ij} \leq s_j$ with Positive $l_{ij}$

Temporal Constraints (also called a **generalized precedence constraint** or a **positive-negative time lag**)

- the start time of one task depends on the start time of another task

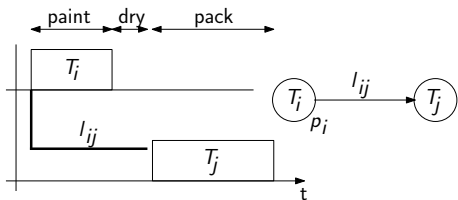
a)  $l_{ij} = p_i$

- “normal” precedence relation
- the second task can start when the previous task is finished



b)  $l_{ij} > p_i$

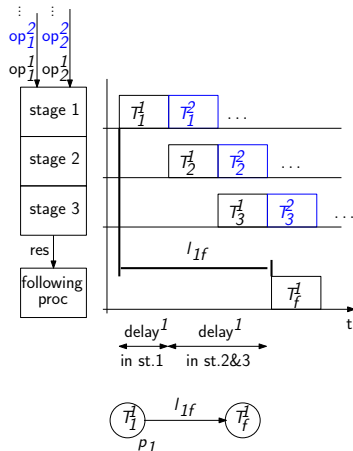
- the second task can start some time after the completion of previous task
- b.1) example of a dry operation performed in sufficiently large space



# Temporal Constraints $s_i + l_{ij} \leq s_j$ with Positive $l_{ij}$

b.2) another example with  $l_{ij} > p_i$  - pipe-lined ALU

- We assume the processing time to be equal in all stages
- **Result is available**  $l_{1f}$  tics after stage 1 reads operands
- **Stage 1 reads new operands** each  $p_1$  tics
- **Stages 2 and 3 are not modeled** since we have enough of these resources and they are synchronized with stage 1



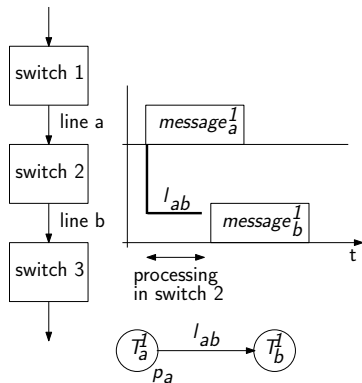
# Temporal Constraints $s_i + l_{ij} \leq s_j$ with Positive $l_{ij}$

c)  $0 < l_{ij} < p_i$

**Partial results of the previous task** may be used to start the execution of the following task.

E.g. the **cut-through** mechanism, where the switch starts transmission on the output port earlier than it receives the complete message on the input port.

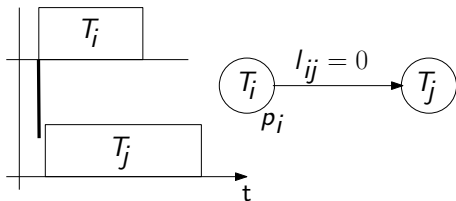
- time-triggered protocol
- resources are communication links
- $l_{ab}$  represents the **processing** (of one bit) in the switch
- **different parts** of the same message are transmitted by several communication links at the same time



# Temporal Constraints $s_i + l_{ij} \leq s_j$ with Zero or Negative $l_{ij}$

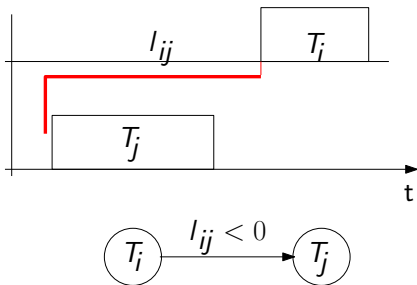
d)  $l_{ij} = 0$

- Task  $T_i$  has to start earlier or at the same time as  $T_j$



e)  $l_{ij} < 0$

- Task  $T_i$  has to start earlier or **at most**  $|l_{ij}|$  **later** than  $T_j$
- It loses the sense of “normal” precedence relation, since  $T_i$  does not have to precede  $T_j$
- It represents the **relative deadline** of  $T_i$  related to the start-time of  $T_j$





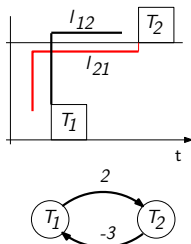
# Cycles and Relative Time Windows

- Absence of a positive cycle in graph  $G$ 
  - it is a necessary condition for schedulability
  - it is a necessary and sufficient condition for schedulability of the instance with unlimited resources capacity (the schedule is restricted only by the temporal constraints - can be computed easily by LP)
- For  $G$  we can create a complete digraph  $G'$  where weight  $l_{ij}$  is the length of the longest oriented path from  $T_i$  to  $T_j$  in  $G$  (if no oriented edge exists in  $G$  or  $G'$ , the weight is  $l_{ij} = -\infty$ ). In the following text, we think of  $l_{ij}$  as an edge in complete graph  $G'$  of the longest paths.
  - $s_j \geq \max_{\forall i \in 1 \dots n} l_{ij}$ , - start time of  $T_j$  is lower bounded by the longest path from arbitrary node.

Example - relative time window

If  $l_{ij} \geq 0$  and  $l_{ji} < 0$  exists, tasks  $T_i$  and  $T_j$  are constrained by the relative time window.

- the length of the negative cycle determines the “clearance” of the time window
- e.g. applying a catalyst to the chemical process



# Project Scheduling

## Minimizing $C_{max}$

- $PS1 \mid \text{temp} \mid C_{max}$  - NP-hard
  - Input: The number of non-preemptive tasks  $n$  and processing times  $[p_1, p_2, \dots, p_n]$ . The temporal constraints defined by digraph  $G$ .
  - Output:  $n$ -element vector  $s$ , where  $s_i$  is the start time of  $T_i$
  - We will show Time-indexed and Relative-order ILP formulations
- $PSm, 1 \mid \text{temp} \mid C_{max}$  - NP-hard
  - Input: The number of non-preemptive tasks  $n$  and processing times  $[p_1, p_2, \dots, p_n]$ . The temporal constraints defined by digraph  $G$ . The number of dedicated resources  $m$  and the assignment of the tasks to the resources  $[a_1, a_2, \dots, a_n]$ , where  $a_i$  is the index of the resource on which task  $T_i$  will be executed.
  - Output:  $n$ -element vector  $s$ , where  $s_i$  is the start time of  $T_i$
  - We show the Relative-order ILP formulation

Task can be represented in two ways:

- **Time-indexed** - ILP model is based on variable  $x_{it}$ , which is equal to 1 iff  $s_i = t$ . Otherwise, it is equal to zero. Processing times are positive integers.
- **Relative-order** - ILP model is based on the relative order of tasks given by variable  $x_{ij}$ , which is equal to 1 iff task  $T_i$  precedes task  $T_j$ . Otherwise, it is equal to zero. The processing times are nonnegative real numbers.

Both models contain two types of constraints:

- precedence constraints
- resource constraints - prevent overlapping of tasks

min  $C_{max}$

$$\sum_{t=0}^{UB-1} (t \cdot x_{it}) + l_{ij} \leq \sum_{t=0}^{UB-1} (t \cdot x_{jt}) \quad \forall l_{ij} \neq -\infty \text{ a } i \neq j \text{ (prec. const.)}$$

$$\sum_{i=1}^n \left( \sum_{k=\max(0, t-p_i+1)}^t x_{ik} \right) \leq 1 \quad \forall t \in \{0, \dots, UB-1\} \text{ (resource)}$$

$$\sum_{t=0}^{UB-1} x_{it} = 1 \quad \forall i \in \{1, \dots, n\} \text{ ( } T_i \text{ is scheduled)}$$

$$\sum_{t=0}^{UB-1} (t \cdot x_{it}) + p_i \leq C_{max} \quad \forall i \in \{1, \dots, n\}$$

variables:  $x_{it} \in \{0, 1\}$ ,  $C_{max} \in \{0, \dots, UB\}$

$UB$  - upper bound of  $C_{max}$  (e.g.  $UB = \sum_{i=1}^n \max \{p_i, \max_{i,j \in \{1, \dots, n\}} l_{ij}\}$ ).

Start time of  $T_i$  is  $s_i = \sum_{t=0}^{UB-1} (t \cdot x_{it})$ .

Model contains  $n \cdot UB + 1$  variables and  $|E| + UB + 2n$  constraints.

Constant  $|E|$  represents the number of temporal constraints (edges in  $G$ ).

# Time-indexed Model for $PS1 \mid \text{temp} \mid C_{max}$

$$\mathcal{T} = \{T_1, T_2, T_3\}, \quad p = [1, 2, 1], \quad UB = 5$$

$T_1$  is scheduled:

	0	1	2	3	4
$T_1$	$x_{10}$	$x_{11}$	$x_{12}$	$x_{13}$	$x_{14}$
$T_2$	$x_{20}$	$x_{21}$	$x_{22}$	$x_{23}$	$x_{24}$
$T_3$	$x_{30}$	$x_{31}$	$x_{32}$	$x_{33}$	$x_{34}$

$\Sigma = 1$

Resource constr. at time 2:

	0	1	2	3	4
$T_1$	$x_{10}$	$x_{11}$	$x_{12}$	$x_{13}$	$x_{14}$
$T_2$	$x_{20}$	$x_{21}$	$x_{22}$	$x_{23}$	$x_{24}$
$T_3$	$x_{30}$	$x_{31}$	$x_{32}$	$x_{33}$	$x_{34}$

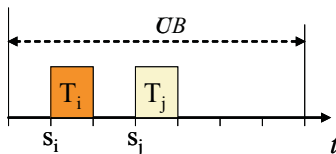
$\Sigma \leq 1$

**Resource constraint** for couple of tasks:

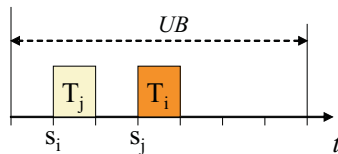
$$p_j \leq s_i - s_j + UB \cdot x_{ij} \leq UB - p_i$$

The constraint uses “big M” (here  $UB$  - upper bound on  $C_{max}$ ).

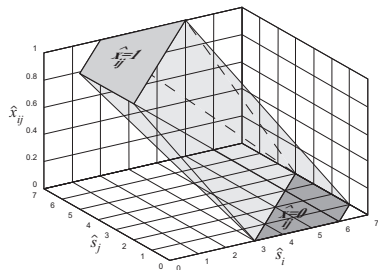
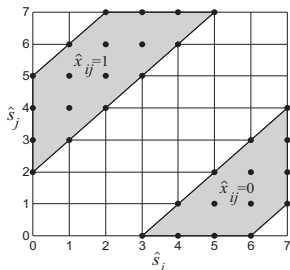
If  $x_{ij} = 1$ ,  $T_i$  **precedes** task  $T_j$  and the constraint is formulated as  $s_i + p_i \leq s_j$ .



If  $x_{ij} = 0$ ,  $T_i$  **follows** task  $T_j$  and the constraint is formulated as  $s_j + p_j \leq s_i$ .



An example of a polytope which is determined by the resource constraint for a pair of tasks  $T_i$  and  $T_j$  with  $p_i = 2$  and  $p_j = 3$ . There are no precedence constraints among the tasks and  $UB = 8$ .



$$\min C_{max}$$

$$s_i + l_{ij} \leq s_j \quad \forall l_{ij} \neq -\infty \text{ a } i \neq j$$

(temporal constraint)

$$p_j \leq s_i - s_j + UB \cdot x_{ij} \leq UB - p_i \quad \forall i, j \in \{1, \dots, n\} \text{ a } i < j$$

(resource constraint)

$$s_i + p_i \leq C_{max} \quad \forall i \in \{1, \dots, n\}$$

variables:  $x_{ij} \in \{0, 1\}$ ,  $C_{max} \in \langle 0, UB \rangle$ ,  $s_i \in \langle 0, UB \rangle$

The model contains  $n + (n^2 - n) / 2 + 1$  variables and  $|E| + (n^2 - n) + n$  constraints.  $|E|$  is a number of temporal constraints (edges in  $G$ ).



# Comparison of the Two Models

Each model is suitable for different types of tasks:

Time-indexed model:

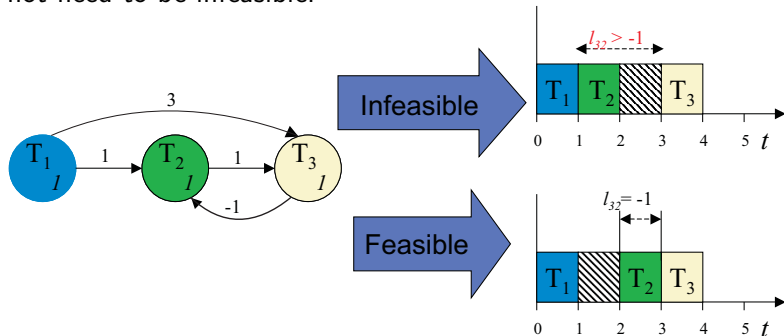
- (+) Can be easily extended for parallel identical processors.
- (+) ILP formulation does not need many constraints.
- (-) The size of the model grows with the size of  $UB$ .

Relative-order model:

- (+) The size of ILP model does not depend on  $UB$ .
- (-) Requires a big number of constraints.

# Feasibility Test for Heuristic Algorithms

If the partial schedule (found for example by a greedy algorithm which inserts tasks in a topological order, or the partial result during the Branch and Bound algorithm) violates some time constraints, the order of tasks does not need to be infeasible.



When the optimal order of the tasks in the schedule is known (variables  $x_{ij}$  are constants), it is easy to find the start time of the tasks (for example by LP formulation involving time constraints only).

## Relative-order Model for $PSm, 1 | \text{temp} | C_{max}$

Part of the input parameters are the number of resources  $m$  and **assignment of the tasks to the resources**  $[a_1, \dots, a_i, \dots, a_n]$ , where  $a_i$  is index of the resource on which task  $T_i$  will be running.

$$\min C_{max}$$

$$s_i + l_{ij} \leq s_j$$

$$\forall l_{ij} \neq -\infty \text{ and } i \neq j$$

(temporal constraints)

$$p_j \leq s_i - s_j + UB \cdot x_{ij} \leq UB - p_i$$

$$\forall i, j \in \{1, \dots, n\}, i < j \text{ and } \underline{a_i = a_j}$$

(independent on each resource)

$$s_i + p_i \leq C_{max}$$

$$\forall i \in \{1, \dots, n\}$$

variables:  $x_{ij} \in \{0, 1\}$ ,  $C_{max} \in \langle 0, UB \rangle$ ,  $s_i \in \langle 0, UB \rangle$

Model consists of less than  $n + (n^2 - n) / 2 + 1$  variables (exact number depends on the number of tasks scheduled on each resource).

Using  $PS1 | \text{temp} | C_{max}$  we will model:

- $1 | r_j, \tilde{d}_j | C_{max}$
- scheduling on **dedicated resources**  $PSm, 1 | \text{temp} | C_{max}$

Using  $PSm, 1 | \text{temp} | C_{max}$  we will model:

- scheduling of **multiprocessor tasks** - task needs more than one resource type at a given moment
- scheduling with **setup times** - two subsequent tasks executed on one resource need to be separated by idle waiting, for example to change the tool.

# Reduction from $1 \mid r_j, \tilde{d}_j \mid C_{max}$ to $PS1 \mid \text{temp} \mid C_{max}$

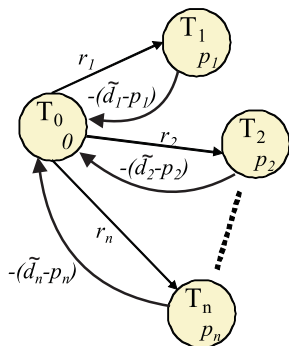
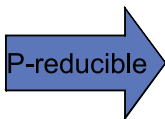
This polynomial reduction proves that  $PS1 \mid \text{temp} \mid C_{max}$  is NP-hard, since Bratley's problem is NP-hard.

Instance  $1 \mid r_j, \tilde{d}_j \mid C_{max}$

$$r = [r_1, r_2, \dots, r_n]$$

$$p = [p_1, p_2, \dots, p_n]$$

$$d = [d_1, d_2, \dots, d_n]$$



Reduction from  $PSm, 1 | \text{temp} | C_{max}$  to  $PS1 | \text{temp} | C_{max}$  is based on the projection of **each resource to the independent time window**. In other words, the schedule of tasks on  $P_j$  is projected into interval  $\langle (j-1) \cdot UB, j \cdot UB \rangle$

Transformation consists of two steps:

- **Add dummy tasks**  $T_0$  and  $T_{n+1}$  with  $p_0 = p_{n+1} = 0$ .
  - Task  $T_0$ , processed on  $P_1$ , precedes all tasks  $T_i \in \mathcal{T}$ , ie.  $s_0 \leq s_i$ .
  - Task  $T_{n+1}$ , processed on  $P_m$ , follows all task  $T_i \in \mathcal{T}$ , tj.  $s_i + p_i \leq s_{n+1}$ .
- **Transform the original temporal constraints** to  $l'_{ij} = l_{ij} + (a_j - a_i) \cdot UB$ .

The new start time  $s'_i$  of each task on processor  $a_i$  is:  
 $s'_i = s_i + (a_i - 1) \cdot UB$ .

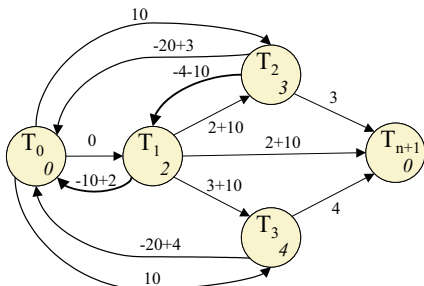
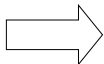
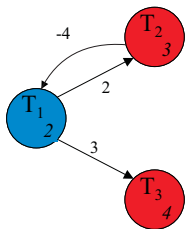
Temporal constraints  $s_i + l_{ij} \leq s_j$  are transformed to:

$$\begin{aligned} s'_i - (a_i - 1) \cdot UB + l_{ij} &\leq s'_j - (a_j - 1) \cdot UB \\ s'_i + l_{ij} + (a_j - a_i) \cdot UB &\leq s'_j \end{aligned}$$

The transformed temporal constraint will look like  $s'_i + l'_{ij} \leq s'_j$ , where:

$$l'_{ij} = l_{ij} + (a_j - a_i) \cdot UB$$

# Reduction from $PSm, 1 | \text{temp} | C_{max}$ to $PS1 | \text{temp} | C_{max}$



two dedicated resources

one resource

While minimizing the completion time of  $T_{n+1}$ , we push tasks  $T_1$ ,  $T_2$  and  $T_3$  “to the left” due to the edges entering  $T_{n+1}$

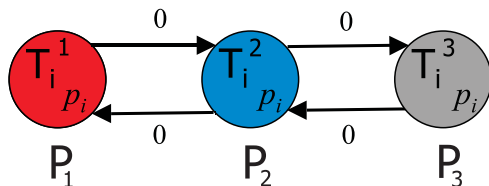


# Multiprocessors Tasks

Transformation of multiprocessor tasks to  $PSm, 1 |temp| C_{max}$

- create as many virtual tasks as there are processors needed to execute the physical tasks
- ensure that the virtual tasks of the given physical task start at the same time - this is done by two edges with weight  $l_{ij} = l_{ji} = 0$ . Consequently  $s_i \leq s_j$  and  $s_j \leq s_i$ .

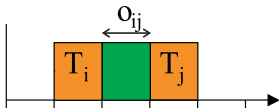
Example: Task  $T_i$  needs resources  $[P_1, P_2, P_3]$ .



# Changeover Time (i.e. Sequence Dependent Set-up Time)

The set-up time  $o_{ij}$  is a time needed to separate task  $T_i$  from  $T_j$ . It is used for example to change the tool in the machine.

Since the order of tasks is unknown in advance, we can not determine which set-up time will be used.



Reduction of the scheduling problem with the set-up time to

$PSm, 1 | \text{temp} | C_{max}$

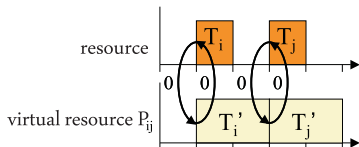
- for each pair of set-up constrained tasks add the virtual resource and a pair of extended virtual tasks
- ensure that the virtual task and the physical task start at the same time




# Changeover Time (i.e. Sequence Dependent Set-up Time)

For each pair of tasks such that the set up time  $o_{ij} > 0$  or  $o_{ji} > 0$ , the virtual resource  $P_{ij}$  and the corresponding virtual tasks  $T'_i$  and  $T'_j$  are added.

- Task  $T'_i$  has  $p'_i = p_i + o_{ij}$  and task  $T'_j$  has  $p'_j = p_j + o_{ji}$ .
- Both tasks run on one virtual resource  $P_{ij}$ .
- Task  $T'_i$  (resp.  $T'_j$ ) is synchronized with the original task by:

$$s_i \leq s'_i \quad s'_i \leq s_i \quad \text{resp.} \quad s_j \leq s'_j \quad s'_j \leq s_j$$



-  J. Błażewicz, K. Ecker, G. Schmidt, and J. Węglarz.  
*Scheduling Computer and Manufacturing Processes*.  
Springer, second edition, 2001.
-  Klaus Neumann, Christoph Schwindt, and Jürgen Zimmermann.  
*Project Scheduling with Time Windows and Scarce Resources*.  
Springer, 2003.
-  Sigrid Knust Peter Brucker.  
Complexity results for scheduling problems.  
<http://www.ict.kth.se/courses/ID2204/index.html>.



**OPPA European Social Fund  
Prague & EU: We invest in your future.**

---