Scheduling

Zdenek Hanzalek
zdenek.hanzalek@cvut.cz

CTU in Prague

May 18, 2022

Table of Contents

Motivation Example: Lacquer Production Scheduling

Made-to-order lacquer production, where jobs are determined by type of lacquer, quantity and delivery date.

Goal:

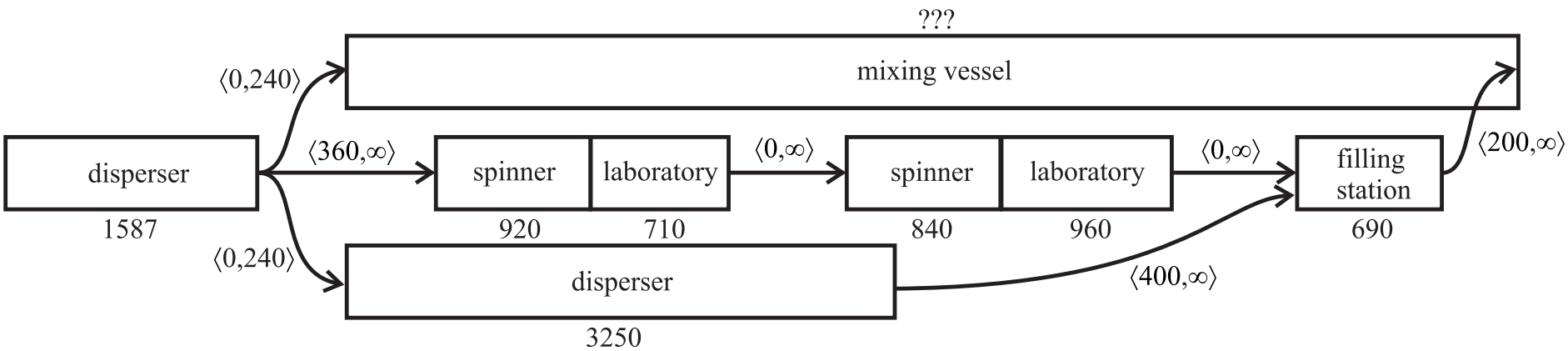
- minimize tardiness (delivery date overrun)
- minimize storage costs

Constraints:

- batch production of various kinds of lacquer
- varying production process/time for different kinds
- time constraints between start times and/or completion times of operations
- working hours (processing times of some operations exceed working hours)
- preparation (*set-up time*)

Motivation Example: Lacquer Production Scheduling - Formalization

Can be formalized as $PSm, 1 \,|temp, o_{ij}, tg| \sum w_j \cdot T_j$
There are temporal constraints on operations. We must consider:
(1) minimal delay between the end of one operation and the start of the next one (e.g. minimal delay needed to dissolve an ingredient)
(2) maximal delay between the end of one operation and the start of the next one (e.g. the lacquer can solidify).



Moreover, the processing time on some resource (e.g. reservoir) is give by the start and completion of different operations.

Motivation Example: Lacquer Production Scheduling

Example

- production of 29 jobs
- 3 types of lacquer
- 9 weeks time horizon

Scheduling - Basic Terminology

- **set of n tasks** $\mathcal{T} = \{T_1, T_2, \ldots, T_n\}$
- set of m types of resources (processors, machines, employees,...) with capacities R_k, $\mathcal{P} = \{P_1^1, \ldots, P_{R_1}^1, P_1^2, \ldots, P_{R_2}^2, \ldots\ldots, P_1^m, \ldots, P_{R_m}^m\}$
- **Scheduling is an assignment of a task to 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 $\left\langle r_i, \widetilde{d}_i \right\rangle$
- 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

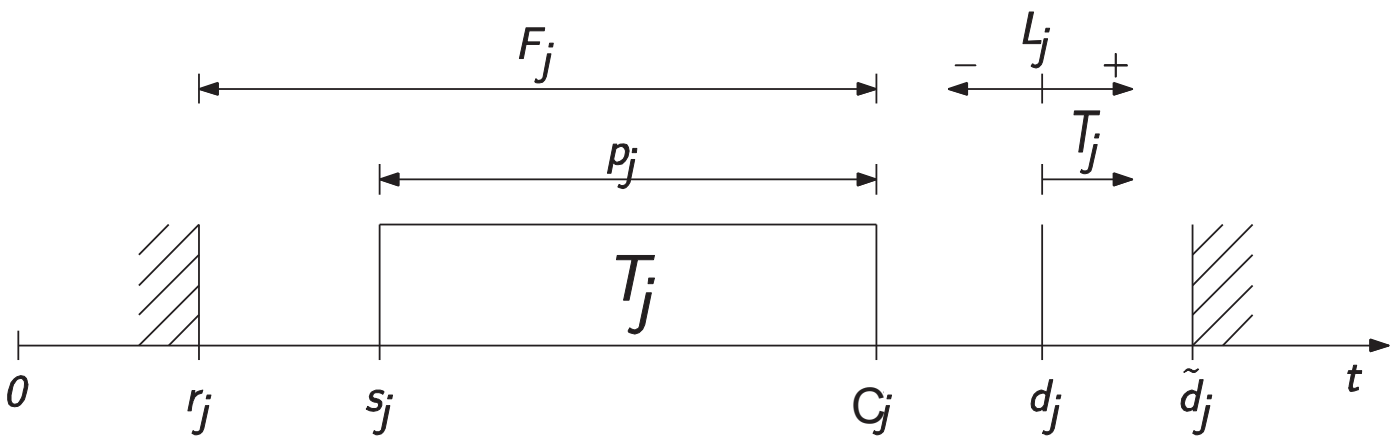
Task Parameters and Variables

Parameters

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

Variables

- **start time** s_j
- **completion time** C_j
- flow (lead) time $F_j = C_j - r_j$
- **lateness** $L_j = C_j - d_j$
- tardiness $T_j = \max\{C_j - d_j, 0\}$



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.

α - **resources**

- **Parallel resources** - a task can run on any resource (only one type of resource exists with capacity R, i.e. $\mathcal{P} = \left\{P^1, \ldots, P^R\right\}$).
- **Dedicated resources** - a task can run only on one resource (m resource types with unit capacity, i.e. $\mathcal{P} = \left\{P^1, P^2 \ldots, P^m\right\}$).
- **Project Scheduling** - m resource types, each with capacity R_k, i.e. $\mathcal{P} = \left\{P^1_1, \ldots, P^1_{R_1}, P^2_1, \ldots, P^2_{R_2}, \ldots\ldots, P^m_1, \ldots, P^m_{R_m}\right\}$.

Resources Characteristics α_1, α_2

α_1	=	1	single resource
		P	**parallel identical** resources
		Q	**parallel uniform** resources, computation time is inversely related to resource speed
		R	**parallel unrelated** resources, computation times are given as a matrix (resources x tasks)
		O	dedicated resources - **open-shop** - tasks are independent
		F	dedicated resources - **flow-shop** - tasks are grouped in the sequences (jobs) in the same order, each job visits each machine once
		J	dedicated resources - **job-shop** - order of tasks in jobs is arbitrary, resource can be used several times in a job
		PS	**Project Scheduling** - most general (several resource types with capacities, general precedence constraints)
α_2	=	$\emptyset$	arbitrary **number** of resources
		2	2 resources (or other specified number)
		m, R	m resource types with capacities R (Project Scheduling)

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

β_1	=	pmtn	**preemption is allowed**
		$\emptyset$	preemption is not allowed
β_2	=	prec	**precedence** constraints
		in-tree,out-tree	tree constraints
		chain	chain constraints
		tmpn	**temporal** constraints (for Project Sched.)
		$\emptyset$	independent tasks
β_3	=	r_j	release time
β_4	=	$p_j = k$	uniform processing time
		$p_L \leq p_j \leq p_U$	restricted processing time
		$\emptyset$	**arbitrary processing time**
β_5	=	$\widetilde{d}_j, d_j$	deadline, due-date
β_6	=	$n_j \leq k$	maximal number of tasks in a job
β_7	=	no-wait	buffers of zero capacity
β_8	=	set-up	time for resource **reconfiguration**

Optimality Criterion γ

γ	=		
		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: $P \,||\, C_{max}$ 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 (i.e. topological order), $C_{max} = \sum_{j=1}^{n} p_j$
- $1\,||\,C_{max}$ - 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)
- $1\,\left|\widetilde{d}_j\right|\,C_{max}$ - easy
  - the tasks are processed in a non-descending order of $\widetilde{d}_j$
  - can be solved by EDF - Earliest Deadline First
  - the feasible schedule doesn't have to exist
- $1\,\left|r_j, \widetilde{d}_j\right|\,C_{max}$ - NP-hard
  - NP-hardness proved by the pol. reduction from 3-Partition problem
  - for $p_j = 1$ there exists a polynomial algorithm

$1 \left| r_j, \widetilde{d}_j \right| C_{max}$ Problem is strongly NP-hard

We prove it by reduction from the 3-Partition problem, which is strongly NP-complete.

3-Partition decision problem instance, $I_{3P} = (A, B)$, is given as:

- a multiset A of $3m$ integers $a_1, a_2, \ldots, a_{3m}$ (sizes of **items**), and
- a positive integer B (size of **bins**) such that $\forall i \in \{1, 2, \ldots, 3m\} : \frac{B}{4} < a_i < \frac{B}{2}$ and $\sum_{i=1}^{3m} a_i = mB$.

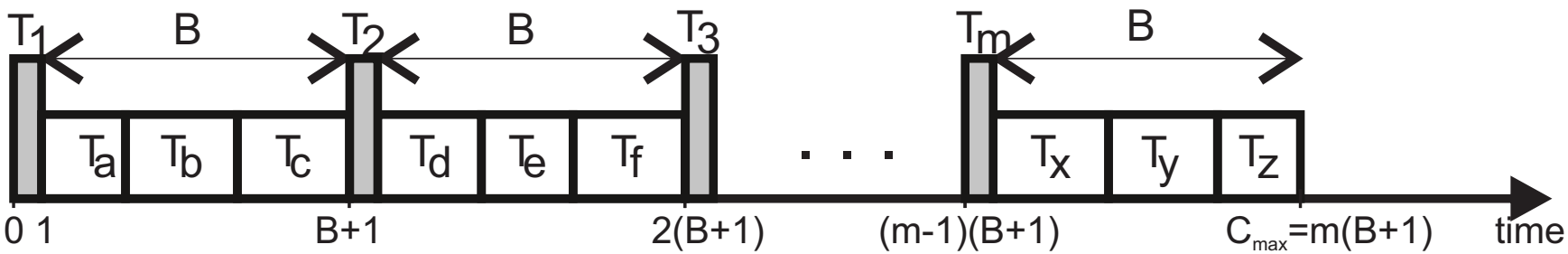
The problem is to determine whether A **can be partitioned** into m disjoint subsets $A_1, A_2, \ldots, A_m$ such that, $\forall j \in \{1, 2, \ldots, m\} : \sum_{a_i \in A_j} a_i = B$.

Note: if we show that there is a subset A_j which contains integers summing to B, then it must contain **three integers**. This follows from the assumption $\frac{B}{4} < a_i < \frac{B}{2}$ (try to sum-up 4 integers or 2 integers).
Example 1: $A = \{4, 5, 5, 5, 5, 6\}$, thus $m = 2$, $B = 15$, $3.75 < a_i < 7.5$. There is feasible 3-partition $A_1 = \{4, 5, 6\}$, $A_2 = \{5, 5, 5\}$.
Example 2: $A = \{4, 4, 4, 6, 6, 6\}$, thus $m = 2$, and $B = 15$. No solution.

Reduction from 3-Partition to $1 \left| r_j, \widetilde{d}_j \right| C_{max}$

From the given instance of the 3-Partition problem $I_{3P} = (A, B)$, we build $1 \left| r_j, \widetilde{d}_j \right| C_{max}$ scheduling problem instance I_{SCH} comprised of $4m$ tasks $T_j = (p_j, r_j, \widetilde{d}_j)$ as follows:

- $\forall j \in \{1, \ldots, m\} : T_j = (1, (B+1) \cdot (j-1), (B+1) \cdot (j-1)+1)$. These are "additional/artificial" tasks used to separate the subsets.
- $\forall j \in \{m+1, \ldots, 4m\} : T_j = (a_i, 0, \infty), i = j - m$.
  Each of these tasks T_j *corresponds to* the element a_i of I_{3P}.



It is easy to prove that the $I_{3P} = (A, B)$ has a solution if and only if the optimal solution of the related I_{SCH} has value of $C_{max} = m \cdot (B+1)$.

Position based ILP formulation for $1 \left| r_j, \widetilde{d}_j \right| C_{max}$

$x_{iq} = 1$ iff task i is at the q-th position in the sequence of tasks
t_q start time of task on the q-th position

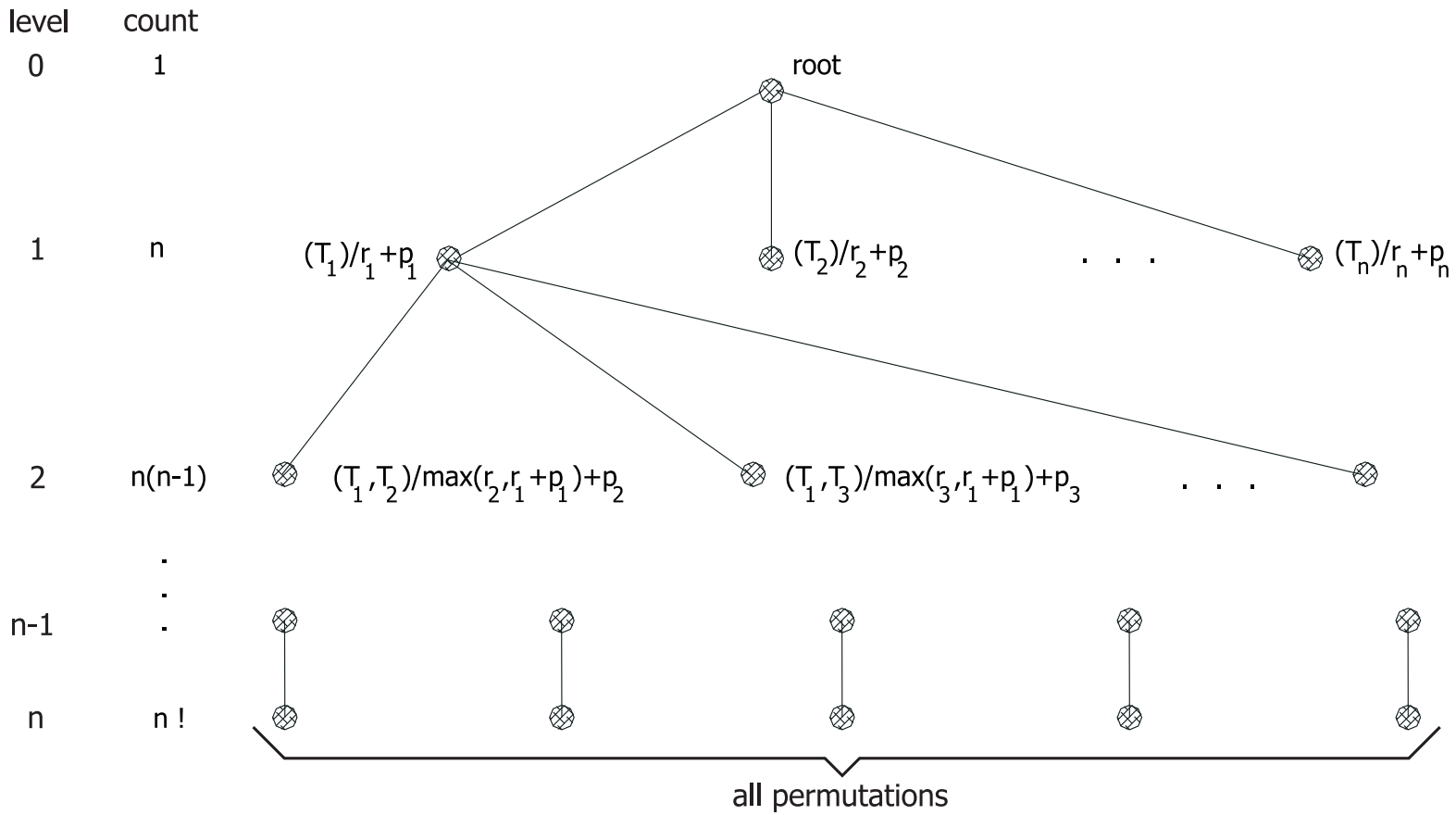
$$
\begin{aligned}
&\min C_{max} \\
&\text{subject to:} \\
&\textstyle\sum_{q=1}^{n} x_{iq} && = 1 && i = 1..n \\
&\textstyle\sum_{i=1}^{n} x_{iq} && = 1 && q = 1..n \\
&t_q && \geq \textstyle\sum_{i=1}^{n} r_i \cdot x_{iq} && q = 1..n \\
&t_q && \geq t_{q-1} + \textstyle\sum_{i=1}^{n} p_i \cdot x_{i,q-1} && q = 2..n \\
&t_q && \leq \textstyle\sum_{i=1}^{n} \widetilde{d}_i \cdot x_{i,q} - \sum_{i=1}^{n} p_i \cdot x_{i,q} && q = 1..n \\
&C_{max} && \geq t_n + \textstyle\sum_{i=1}^{n} p_i \cdot x_{in}
\end{aligned}
$$

variables: $x_{i \in 1..n, q \in 1..n} \in \{0, 1\}$, $C_{max} \in \langle 0, UB \rangle$, $t_{q \in 1..n} \in \langle 0, UB \rangle$

... more efficient than relative order ILP

Bratley's Algorithm for $1 \left| r_j, \widetilde{d}_j \right| C_{max}$

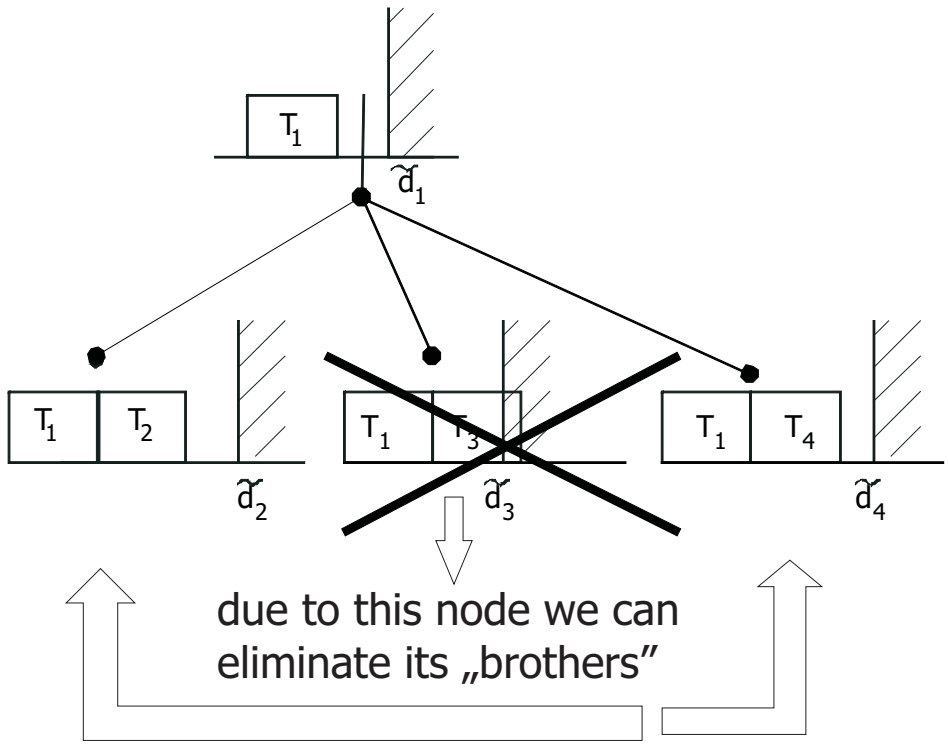
A **branch and bound** (B&B) algorithm.
Branching - without bounding it is an **enumerative method** that creates a solution tree (some of the nodes are infeasible). Every node is labeled by: (the order of tasks)/(completion time of the last task).



Reduction of the Tree - Bounding

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

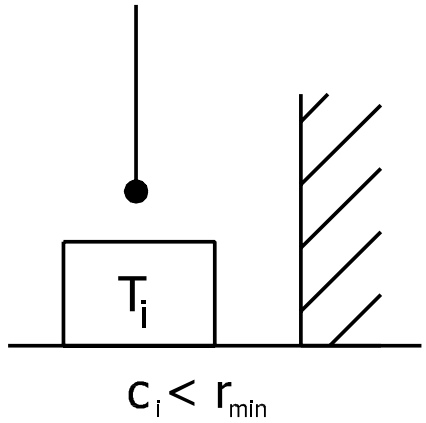
- If there is a node which exceeds any deadline, all its descendants should be eliminated
- Critical task (here T_3) will have to be scheduled anyway - therefore, all of its "brothers" should be eliminated as well

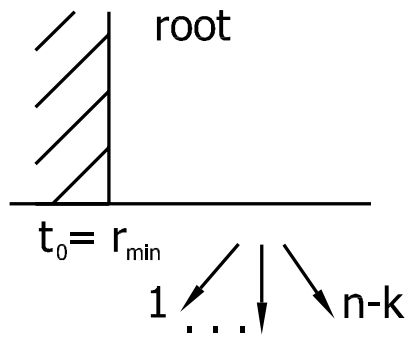


Tree Size Reduction - Decomposition

(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

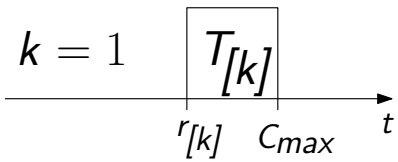
BRTP is a set of k tasks that satisfy:

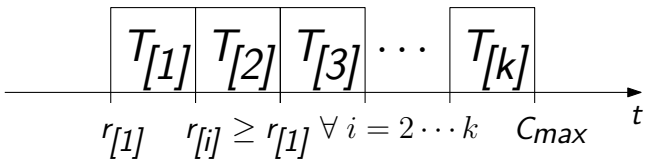
- first task $T_{[1]}$ starts at it's release time
- all k tasks till the end of the schedule run without "idle waiting"
- $r_{[1]} \leq r_{[i]}$ for all $i = 2 \ldots k$

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).



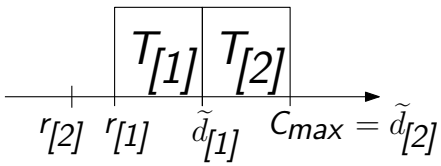


Proof:

- 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}

BRTP is not Necessary Condition of Optimality

Example:

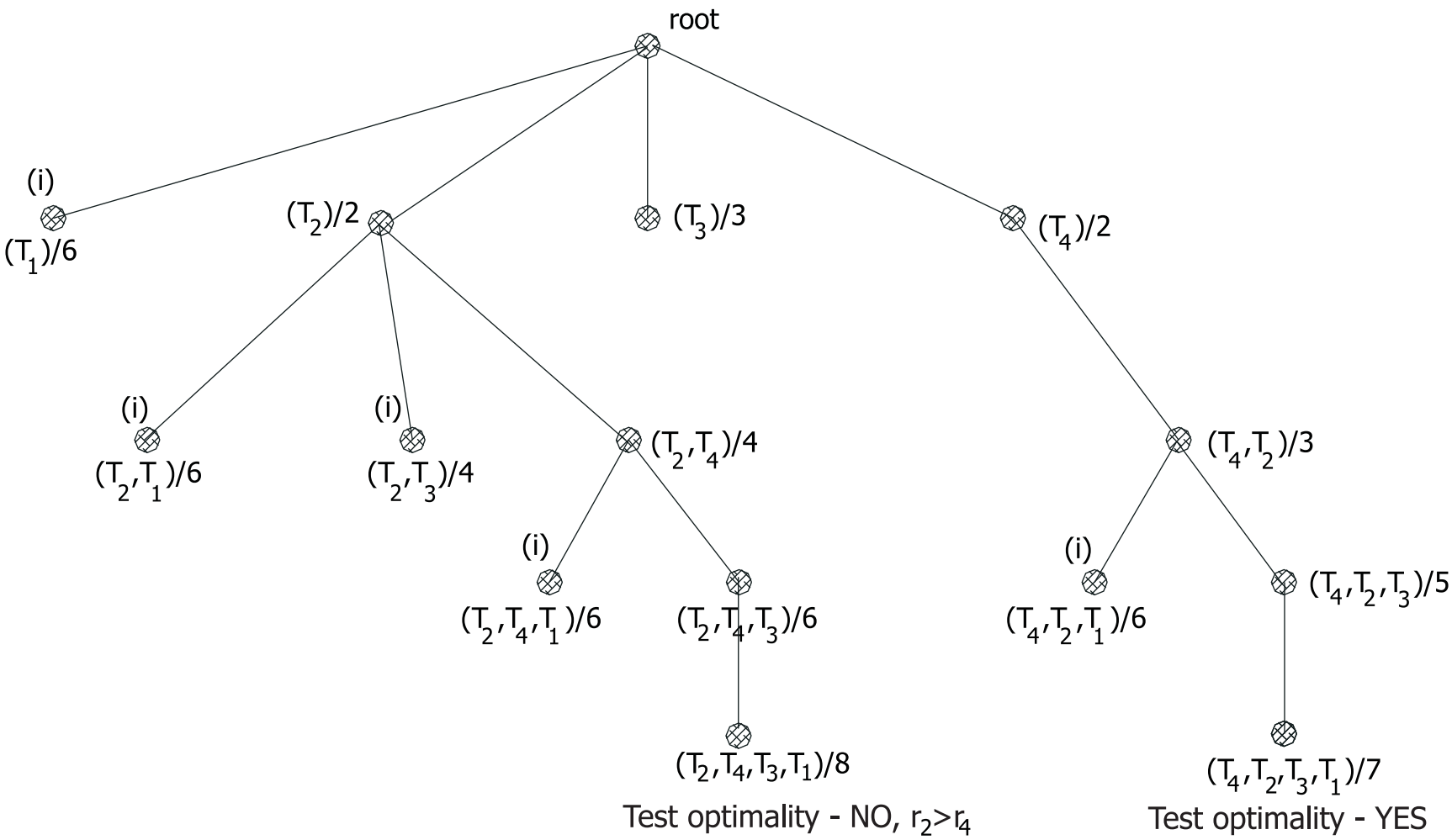


In this particular case, the schedule is optimal, but it does not have BRTP.

Tightening the bounds:
In general, C_{max} found without BRTP could be used for bounding further solutions while **setting all deadlines** to be at most $C_{max} - \epsilon$.
This ensures that if other feasible schedules exist, only those that are **better by** ϵ than the solution at hand, are generated.

Bratley's Algorithm - Example

r =(4,1,1,0), p =(2,1,2,2), $\widetilde{d}$ =(8,5,6,4)



Scheduling on One Resource
Minimizing $\sum w_j C_j$

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

Branch and Bound with LP for $1\,|\mathrm{prec}|\,\sum w_j C_j$

First, we formulate the problem as an 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:

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

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

ILP formulation for $1\,|\mathrm{prec}|\,\sum w_j C_j$

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

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

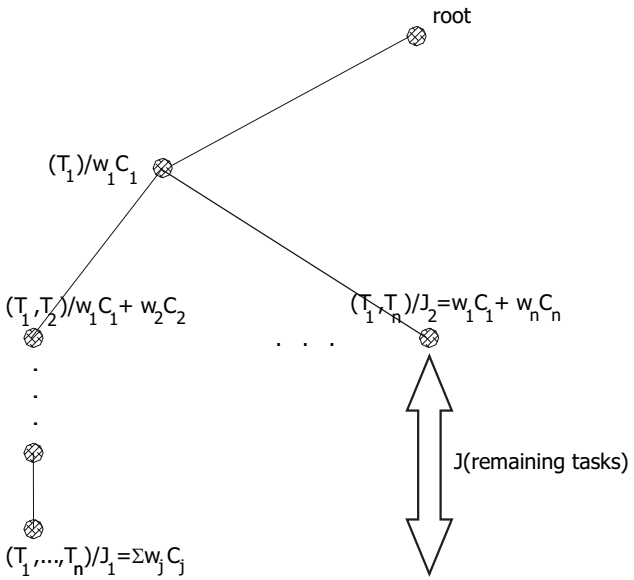
variables: $x_{i\in 1..n, j\in 1..n} \in \{0,1\}$

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 One Resource
Minimizing L_{max}

- $1 \,||\, L_{max}$ - solved by EDD (Earliest Due Date first) rule in polynomial time
- $1 \,|r_j|\, L_{max}$ - NP-hard
- $1 \,|r_j, p_j = 1|\, L_{max}$ - polynomial - iterating EDD
- $1 \,|\text{pmtn}, r_j|\, L_{max}$ - polynomial - iterating EDD by Horn
- $1 \left|\text{pmtn}, r_j, d_j = \tilde{d}_j\right| L_{max}$ - polynomial - the same Horn's algorithm often called EDF
- $1 \left|\text{pmtn}, \text{prec}, r_j, d_j = \widetilde{d}_j\right| L_{max}$ - polynomial - transformation to independent task set and then EDF

Important notes:

- minimization of L_{max} implies existence of due-date d_j (even if it is not in β notation)
- L_{max} may have negative value. In such case, minimization of lateness L_{max} may be interpreted as maximization of earliness.

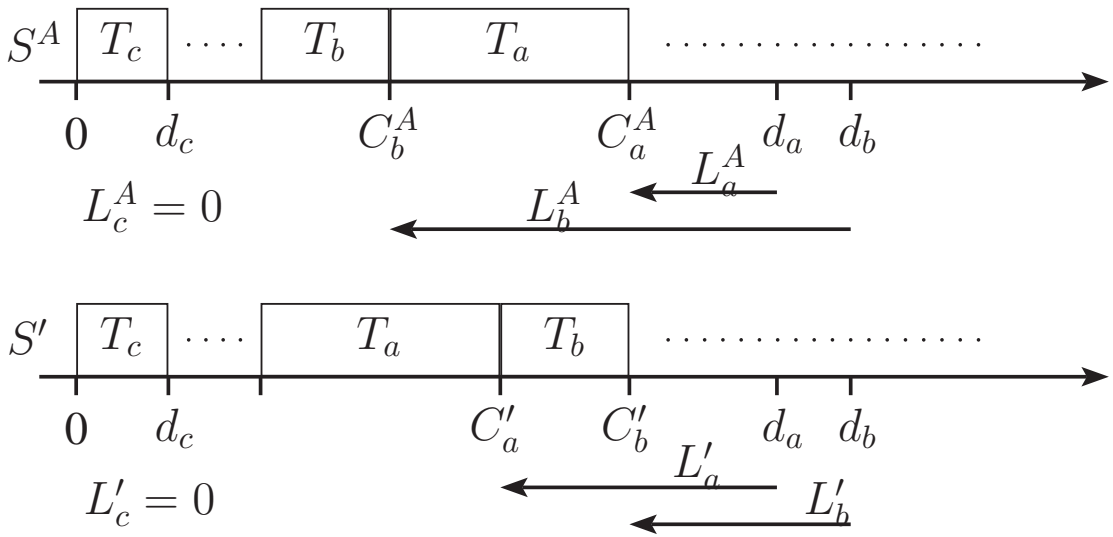
Problem 1 || L_{max} - EDD and its Optimality

- Can be solved by EDD (Earliest Due Date first), i.e. **tasks are scheduled in order of nondecreasing due dates**.
- Time complexity is $O(n \cdot \log n)$.

Optimality can be proven by simple swaps:

- Let S^A be an **optimal schedule** produced by an algorithm A.
- Let S^{EDD} be a schedule produced by EDD.
- If $S^A \neq S^{EDD}$, then there exist two tasks T_a and T_b with $d_a \leq d_b$, such that T_b immediately precedes T_a in S^A.
- Swap of T_a and T_b cannot increase the maximum lateness of the task set L_{max}.
- By finite number of swaps, S^A is changed to S^{EDD},
  $S^A \overset{swap}{\rightarrow} S' \overset{swap}{\rightarrow} \dots \overset{swap}{\rightarrow} S^{EDD}$

Problem 1 || L_{max} - Optimality of EDD - Illustration



$$L^A_{max}(\{T_a, T_b\}) = C^A_a - d_a$$

$$L'_{max}(\{T_a, T_b\}) = max\{L'_a, L'_b\}$$

Two cases must be considered when $p_a > 0$ and $p_b > 0$:
1) If $L'_a \geq L'_b$ then $L'_{max}(\{T_a, T_b\}) = C'_a - d_a < C^A_a - d_a$
2) If $L'_a \leq L'_b$ then $L'_{max}(\{T_a, T_b\}) = C'_b - d_b = C^A_a - d_b < C^A_a - d_a$

Since, in both cases, $L'_{max}(\{T_a, T_b\}) < L^A_{max}(\{T_a, T_b\})$ we can conclude that the swap of T_a and T_b decreases $L_{max}(\{T_a, T_b\})$ and thus it cannot increase $L_{max}(\mathcal{T})$, the maximum lateness of all tasks.

Problem 1 $|\text{pmtn}, r_j| L_{max}$ - Horn's Algorithm

Input: $\mathcal{T}$, set of n preemptive tasks. Processing times $(p_1, p_2, ..., p_n)$, release dates $(r_1, r_2, ..., r_n)$ and due-dates $(d_1, d_2, ..., d_n)$.

Output: Start times of preempted parts of tasks.

while $\mathcal{T} \neq \emptyset$ **do**

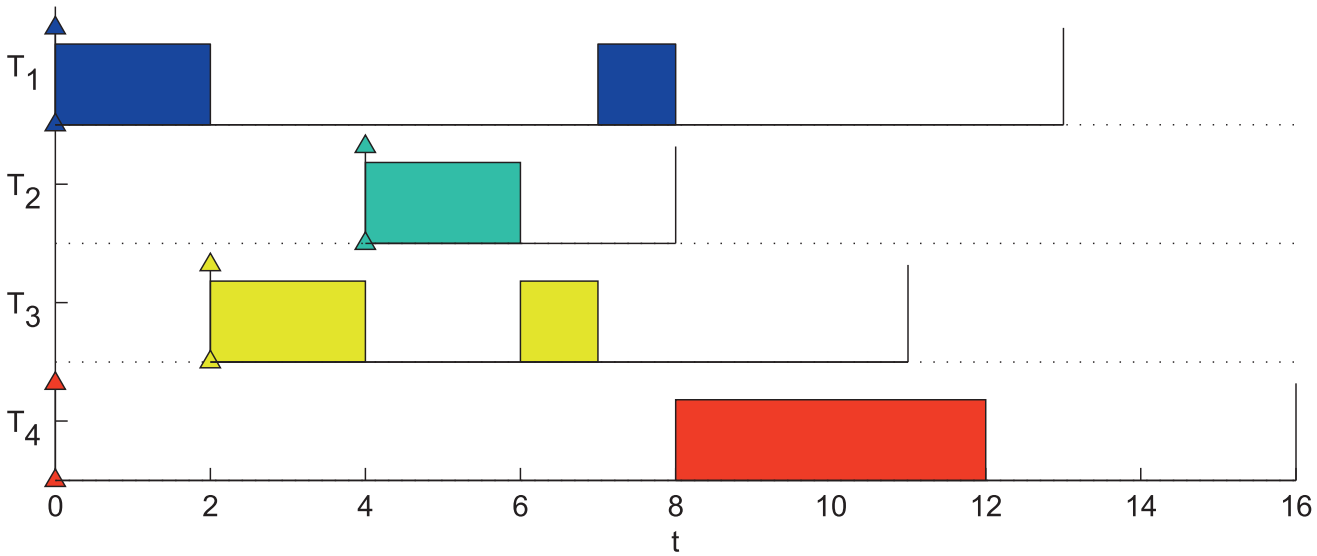
- $t_1 := \min_{T_j \in \mathcal{T}} \{r_j\}$; // t_1 is now
- **if** all tasks are ready at time t_1 **then** $t_2 = \infty$;
- **else** $t_2 = \min_{T_j \in \mathcal{T}} \{r_j | r_j > t_1\}$; // situation changes in t_2
- $\mathcal{T}' = \{T_j | T_j \in \mathcal{T}, r_j = t_1\}$; // set of ready tasks
- choose $T_k \in \mathcal{T}'$ with minimal d_j; // EDD in $\mathcal{T}'$
- $\delta := \min\{p_k, t_2 - t_1\}$;
- schedule T_k or its part in interval $\langle t_1, t_1 + \delta)$;
- **if** $\delta = p_k$ **then** $\mathcal{T} := \mathcal{T} \setminus \{T_k\}$;
- **else** $p_k := p_k - \delta$; // preemption
- **for** $T_j \in \mathcal{T}'$ **do** $r_j := t_1 + \delta$;

end

Problem 1 $|\text{pmtn}, r_j| L_{max}$

Example:

$$\mathcal{T} = \{T_1, T_2, T_3, T_4\}$$
$$p = (3, 2, 3, 4)$$
$$r = (0, 4, 2, 0)$$
$$d = (13, 8, 11, 16)$$



Optimality of Horn's Algorithm for $1|\text{pmtn}, r_j|L_{max}$ and its application to $1\left|\text{pmtn}, r_j, d_j = \widetilde{d}_j\right|L_{max}$

Theorem - Optimality of Horn's Algorithm

Given a set of n independent preemptive tasks with arbitrary release times, any algorithm that at any instant executes the task with earliest due date among all the ready tasks is optimal with respect to L_{max} minimization.

When we assume $1\left|\text{pmtn}, r_j, d_j = \widetilde{d}_j\right|L_{max}$ then the algorithm is often called **EDF (Earliest Deadline First)**. Such algorithm:

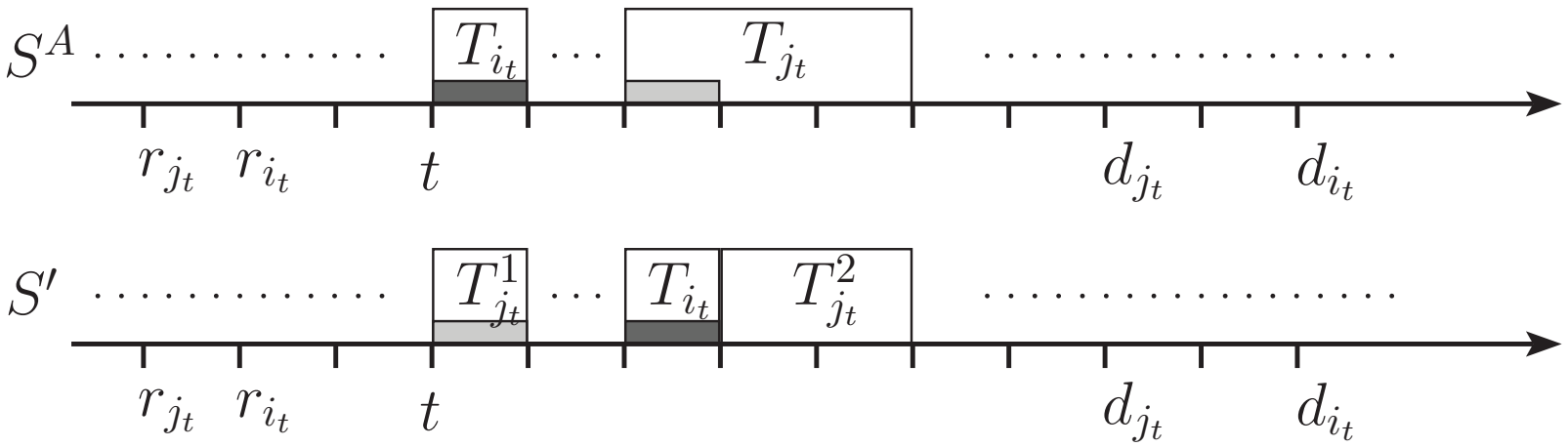
- **minimizes** L_{max}
- **decides schedulability** – if there exists a feasible schedule ($L_{max} \leq 0$) for the given instance, then the EDF is able to find it

Optimality of Horn's Algorithm

Assuming the input parameters to be nonnegative **integers**, the proof of the theorem is based on the following reasoning:

- Let S^A be an **optimal schedule** produced by algorithm A
- Let S^{EDF} be a schedule produced by EDF
- Let schedule S^A starts at time $t = 0$ and D is the latest due date.
- Without loss of generality S^A can be **divided into unit-time slices**.
- Let i_t is the id of the task executing slice t.
- Let j_t is the id of the ready task with earliest due date at time t.
- If $S^A \neq S^{EDF}$ then there is slice t such that $i_t \neq j_t$.
- Swap of slices of T_{i_t} and T_{j_t} cannot increase the maximum lateness of the task set L_{max}.
- S^{EDF} is obtained from S^A by at most D swaps.
  $S^A \overset{swap}{\rightarrow} S' \overset{swap}{\rightarrow} \dots \overset{swap}{\rightarrow} S^{EDF}$

Optimality of Horn's Algorithm - Illustration



Using the same argument adopted in the proof of EDD optimality for $1 \,||\, L_{max}$, it is easy to show that each swap cannot increase the maximum lateness of the task set L_{max}.

Problem 1 $|\text{pmtn, prec}, r_j, d_j = \widetilde{d}_j| L_{max}$

Chetto, Silly, Bouchentouf algorithm transforms set $\mathcal{T}$ of dependent tasks into $\mathcal{T}'$ **of independent tasks** by modification of timing parameters:

- modification of the **release dates**
  1. For any task without predecessors set $r_j^{'} = r_j$.
  2. Select task T_j such that its release date has not been modified but the release dates of all immediate predecessors T_h have been modified. If no such task exists, exit.
  3. Set $r_j^{'} = max\{r_j, max\{r_h^{'} + p_h | T_h \text{ is immediate predecessor of } T_j\}$ and jump to step 2.
- modification of **deadlines**
  1. For any task without successors set $\widetilde{d}_j^{'} = \widetilde{d}_j$.
  2. Select a task T_j such that its deadline has not been modified but the deadlines of all immediate successors T_k have been modified. If no such task exists, exit.
  3. Set $\widetilde{d}_j^{'} = min\{\widetilde{d}_j, min\{\widetilde{d}_k^{'} - p_k | T_k \text{ is immediate successor of } T_j\}$ and jump to step 2.
- EDF is executed on independent tasks in $\mathcal{T}'$

Problem 1 $\left|\text{pmtn, prec}, r_j, d_j = \widetilde{d}_j\right| L_{max}$

No problem constraints are violated by Chetto, Silly, Bouchentouf algorithm:

- $r_j^{'} \geq r_j$ and $\widetilde{d}_j^{'} \leq \widetilde{d}_j$, therefore, the **schedulability of** $\mathcal{T}'$ with respect to the timing constraints (release dates and deadlines) **implies also schedulability of** $\mathcal{T}$ with respect to the timing constraints
- due to the structure of the Horn's alg. (consider a set of released tasks and choose the one with smallest deadline) and modification of the timing constraints the scheduled **tasks of** $\mathcal{T}'$ **are ordered in the same way** as given by the precedence constraints, therefore, **precedence constraints of** $\mathcal{T}$ **are not violated**

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 $O(n)$
- $P \left|\text{pmtn}, r_j, \widetilde{d_j}\right| -$ - easy
  - decision version of **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 $P\,|\text{pmtn}|\,C_{max}$

Input: R, number of parallel identical resources, n, number of preemptive tasks and computation times $(p_1, p_2, ..., p_n)$.

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.

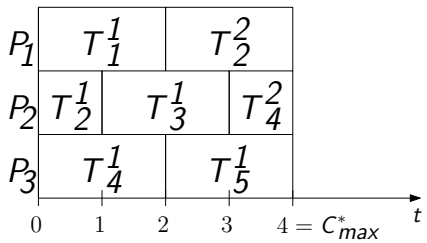
$s_i^1 = s_i^2 = z_i^1 = z_i^2 := 0$ for all $i \in 1 \ldots n$;
$t := 0; v := 1; i := 1;$
$C^*_{max} = \max\left\{\max_{i=1\ldots n}\{p_i\}, \frac{1}{R}\sum_1^n p_i\right\};$
while $i \leq n$ **do**
  if $t + p_i \leq C^*_{max}$ **then**
    $s_i^1 := t; z_i^1 := v;\ t := t + p_i; i := i + 1;$
  else
    $s_i^2 := t; z_i^2 := v;\ p_i := p_i - (C^*_{max} - t); t := 0; v := v + 1;$
  end
end

McNaughtnon's Algorithm for $P\,|\text{pmtn}|\,C_{max}$

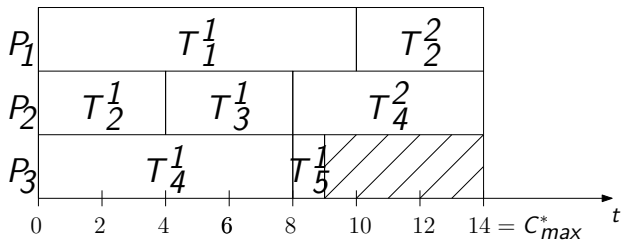
The term $C^*_{max} = \max\left\{\max_{i=1\ldots n}\{p_i\}, \frac{1}{R}\sum_1^n p_i\right\}$ should be interpreted as follows:

- component $\max_{i=1\ldots n}\{p_i\}$ 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 $\frac{1}{R}\sum_1^n p_i$ represents a situation when all resources work without idle waiting

Example 1:
$p = (2, 3, 2, 3, 2), R = 3$
compute $C^*_{max} = \max\left\{3, \frac{12}{3}\right\} = 4$



Example 2:
$p = (10, 8, 4, 14, 1), R = 3$
compute $C^*_{max} = \max\left\{14, \frac{37}{3}\right\} = 14$



List Scheduling - Approximation Alg. for $P|prec|C_{max}$

Input: R, number of parallel identical resources, n, number of non-preemptive tasks and their proc. times $(p_1, p_2, ..., p_n)$. DAG.
Output: start times $(s_1, s_2, ..., s_n)$ and resource IDs $(z_1, z_2, ..., z_n)$.

$t_v := 0$ for all $v \in 1 \ldots R$; // free time of resource
$s_i = z_i := 0$ for all $i \in 1 \ldots n$;
Sort tasks in list L;
for $i := 1$ **to** n **do** // for all tasks
  $k = arg\min_{v=1...R}\{t_v\}$; // choose res. with the lowest t_v
  Create S as a subset of tasks in L with smallest availability time a_i ;
  Choose $T_i \in S$ which is the first one in the list L;
  Remove T_i from the list L;
  $s_i = \max\{t_k, a_i\}; z_i = k;$ // assign T_i to P_k
  $t_k = s_i + p_i;$ // update free time of P_k
end

a_i, **availability time** of T_i, is equal to t_k if it has no predecessors, it is equal to ∞ if at least one of its predecessors is in L, and it is equal to $\max_{j \in Pred(T_i)}\{s_j + p_j\}$ if all of its predecessors have been removed from L.

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

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

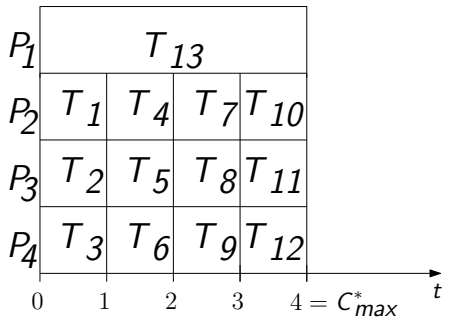
- We have a list (n-tuple) of tasks and when some resource is free, we assign available task from the list to this resource.
- The accuracy of LS depends on the criterion and sorting procedure.

Approximation factor of LS algorithm [Graham 1966]

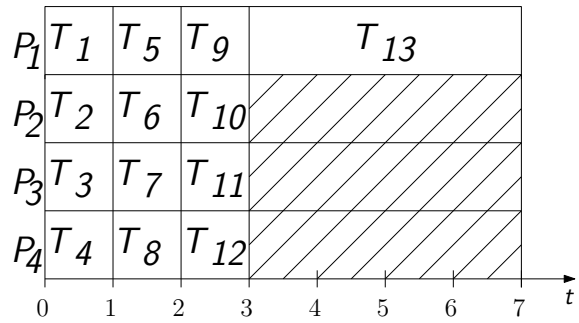
For $P\,|\text{prec}|\,C_{max}$ (and also for $P\,||\,C_{max}$) and arbitrary (unsorted) list L, List Scheduling is an approximation algorithm with factor $r_{LS} = 2 - \frac{1}{R}$

An example illustrating the case when the factor is attained:

$n = (R-1)\cdot R + 1$,
$p = (1, 1, \ldots, 1, R)$,
$\prec$ empty.
Illustration for $R = 4$
$r_{LS} = 2 - \frac{1}{4} = \frac{7}{4}$



$L = (T_n, T_1, \ldots, T_{n-1})$



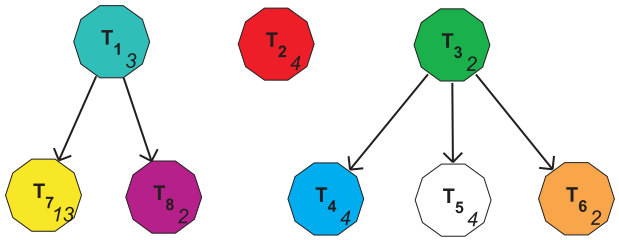
$L' = (T_1, T_2, \ldots, T_n)$

Anomalies of List Scheduling Algorithm

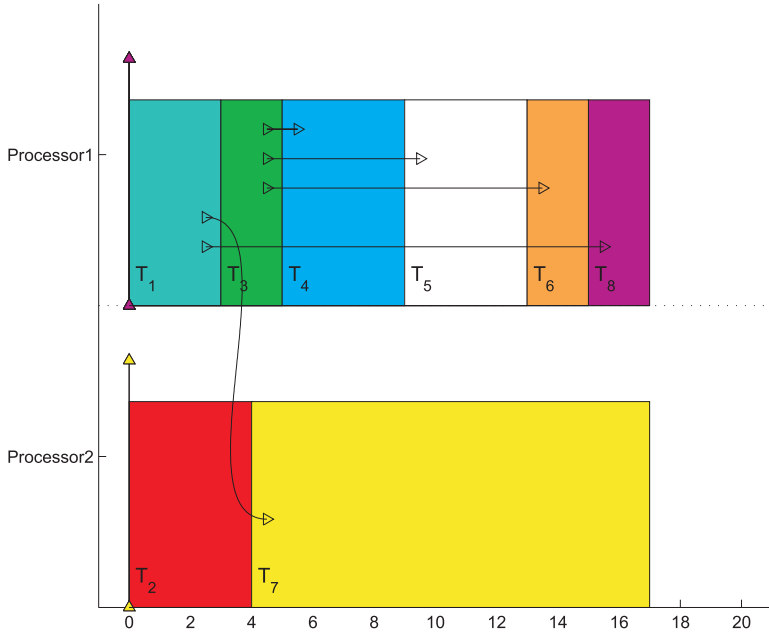
The LS algorithm depends not only on the **order of tasks in L**, but it exhibits **anomalies** (C_{max} surprisingly increases when relaxing some constraints/parameters) caused by:

- (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
$R = 2,\ n = 8,\ p = (3, 4, 2, 4, 4, 2, 13, 2)$

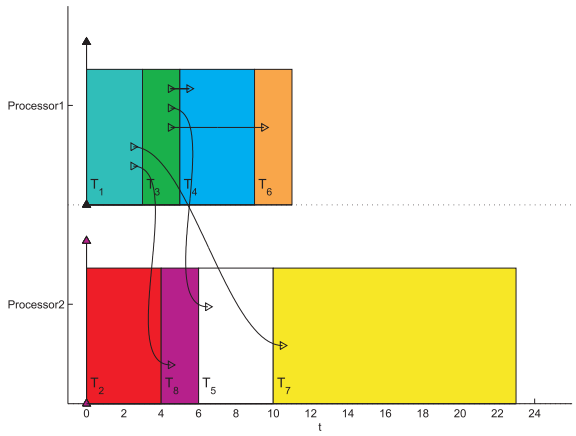


Using list $L = (T_1, T_2, T_3, T_4, T_5, T_6, T_7, T_8)$,
LS finds solution with $C^*_{max} = 17$.

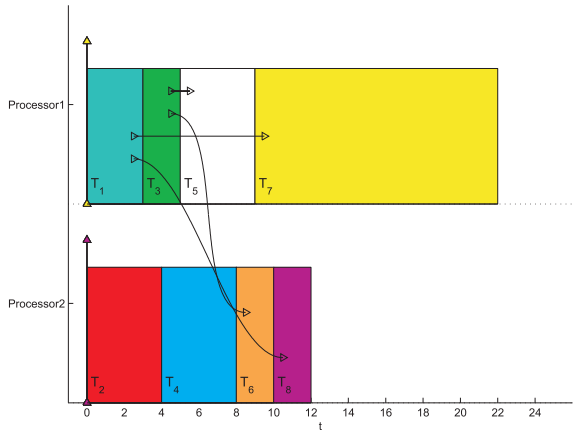


List Scheduling Anomalies - Prolongation of C_{max}

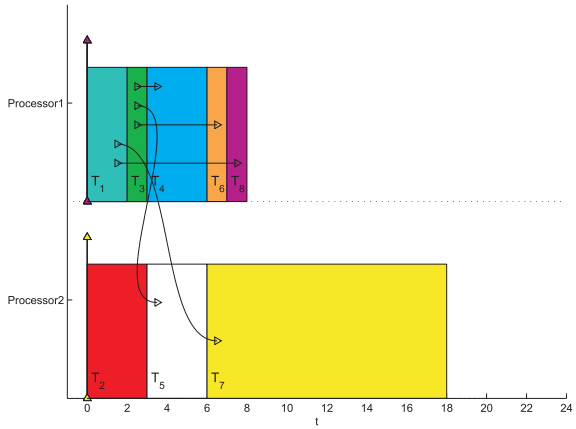
Exchange position of T_7 and T_8
$L = (T_1, T_2, T_3, T_4, T_5, T_6, T_8, T_7)$.



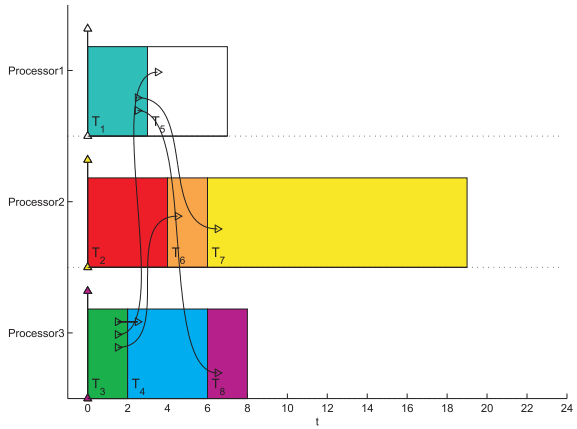
(2) Remove prec. constr. $T_3 \prec T_4$.



(1) Decrease p_i of all tasks by one.



(3) Add resource ($R = 3$).



LPT (Longest Processing Time First) - Approximation Algorithm for $P \,||\, C_{max}$

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 $P||C_{max}$ is an approximation algorithm with factor $r_{LPT} = \frac{4}{3} - \frac{1}{3R}$

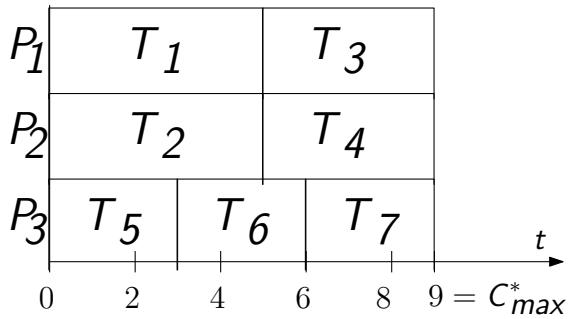
Time complexity of LPT algorithm is $O(n \cdot log(n))$ due to the sorting.

LPT (Longest Processing Time First) - Approximation Algorithm for $P \mid\mid C_{max}$

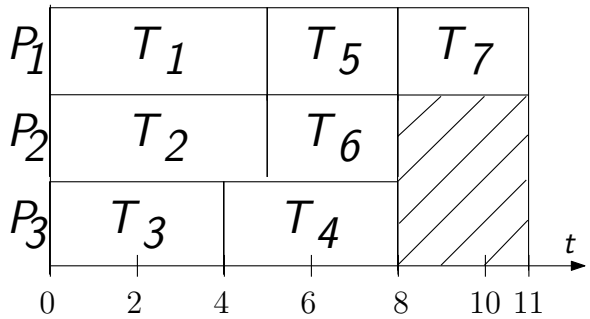
An example illustrating the case when the factor is attained:
$p = (2R-1, 2R-1, 2R-2, 2R-2, \ldots, R+1, R+1, R, R, R)$
$n = 2 \cdot R + 1, \prec$ empty,

Illustration for $R = 3$

optimum:



LPT:



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

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

Dynamic Programming for $P \,||\, C_{max}$ [Rothkopf]

For fixed R, the number of processors, there is a pseudopolynomial algorithm - input instance is restricted to bounded nonnegative integers: number of tasks and their processing times.

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

Dynamic Programming for $P \,||\, C_{max}$ [Rothkopf]

Input: R, the number of parallel identical resources, n, the number of nonpreemptive tasks and their processing time $(p_1, p_2, ..., p_n)$.

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

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

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

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

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

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

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

    `//` $x_{i-1}() = 1$ `‘‘smaller’’ by` p_i `in any direction`

  end

end

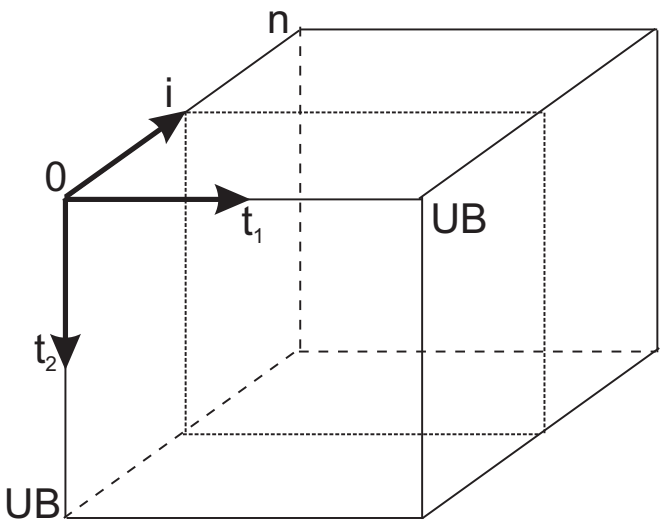
$C^*_{max} = \min_{x_n(t_1,t_2,\ldots,t_R)=1} \{\max_{v=1,2,\ldots R} \{t_v\}\}$;

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

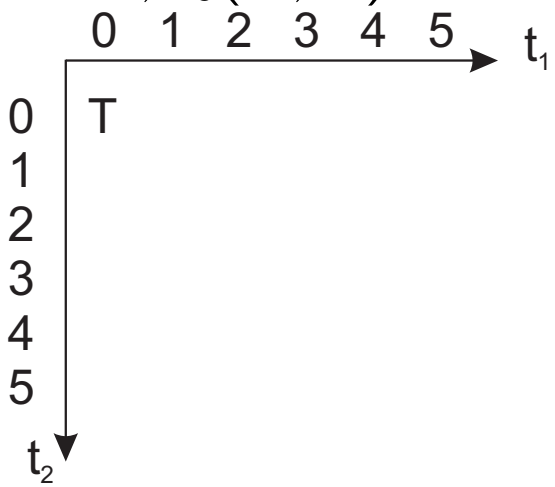
Time complexity is $O(n \cdot UB^R)$.

Example for $P \,||\, C_{max}$ [Rothkopf]

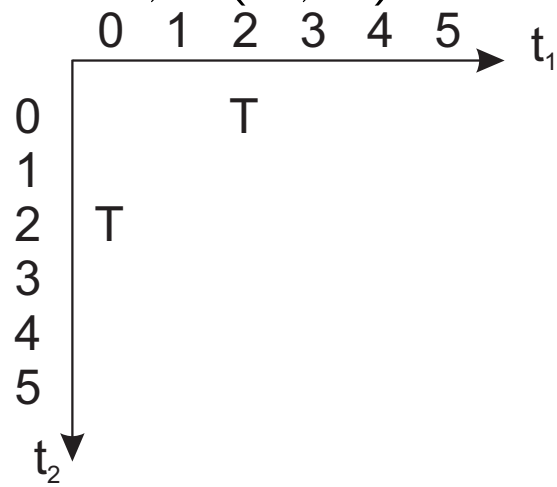
Example n=3, R=2, p=(2,1,2), UB=5.



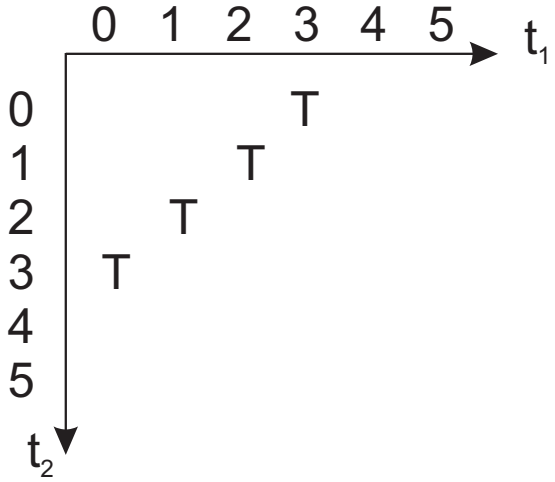
$i = 0, x_0(t_1, t_2)$



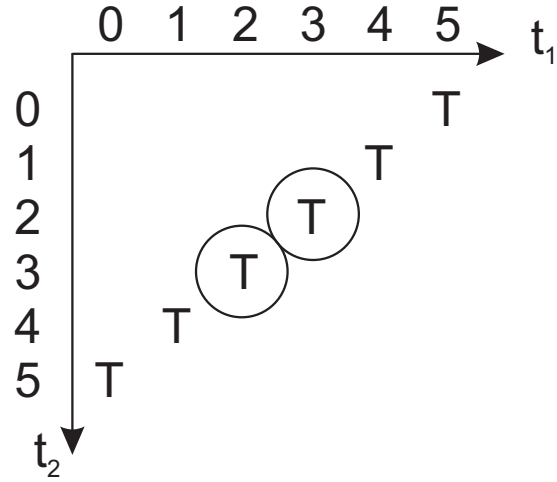
$i = 1, x_1(t_1, t_2)$



$i = 2, x_2(t_1, t_2)$



$i = 3, x_3(t_1, t_2)$



Complexity of $P \mid\mid C_{max}$ [Rothkopf]

- For polynomially bounded R, (i.e., $R = poly(n)$) the problem is strongly NP-hard.
  - Can be shown by reduction from 3-partition problem with $C_{max} = B, R = m, n = 3m, p_i = a_i$ (take care, in 3-partition the number 3 means number of items in the bin).
  - Here, the Rothkopf algorithm is not pseudopolynomial, since its complexity is $O(n \cdot UB^R) = O(n \cdot UB^{poly(n)})$, which is equal to $O(n \cdot UB^{n/3})$ for the instances reduced from 3-partition.
- For any constant R, (e.g., $R = 2$) the problem is weakly NP-hard
  - Here, the Rothkopf algorithm is pseudopolynomial, since its complexity is $O(n \cdot UB^2)$.
  - It can not be used to solve the instance reduced from 3-partition, but it can be used to solve the instances reduced from 2-partition problem, which is weakly NP-hard (take care, in 2-partition the number 2 means number of bins).

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

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** β
- the algorithm moves forward to time τ when **one of the tasks ends** or the task with a lower level would be processed by a bigger capacity β 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 $O(n^2)$.

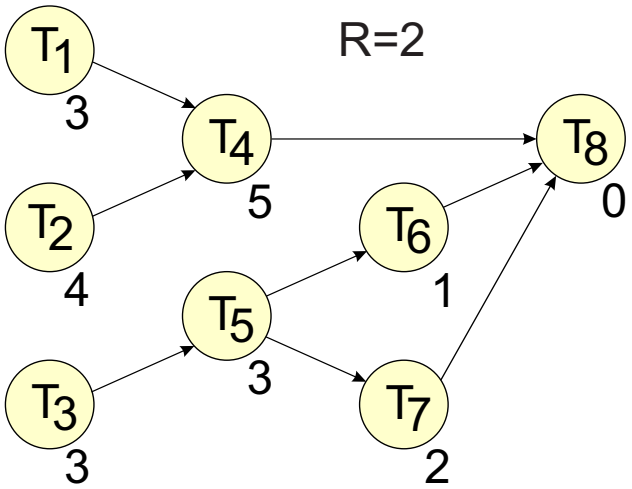
Input: R, the number of parallel identical resources, n, the number of preemptive tasks and proc. times $(p_1, p_2, ..., p_n)$. Prec. in DAG.
Output: n-elements vectors $s^1, ..., s^k, ...s^K$ and $z^1, ..., z^k, ...z^K$ where s_i^k is the start time of the k-th part of task T_i and z_i^k is corresponding resource ID.

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

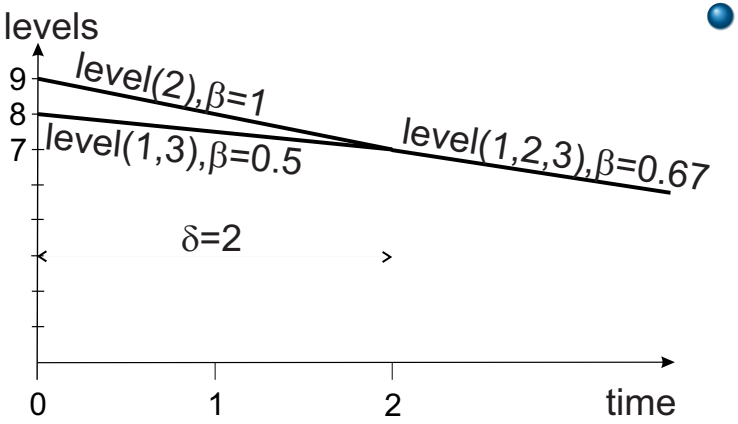
compute the level of all tasks ; $t := 0$; $h := R$; // h free res.
while *unfinished tasks exist* **do**
  construct $\mathcal{Z}$; // subset $\mathcal{T}$ of free tasks in time t
  while $h > 0$ *and* $|\mathcal{Z}| > 0$ **do** // free resources and free tasks
    construct $\mathcal{S}$; // subset $\mathcal{Z}$ 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 δ; // see explanation below
  decrease proc.times and levels by $(\delta) \cdot \beta$;// finished part of task
  $t := t + \delta$;$h := R$;
end
Use McNaughton's alg. to re-schedule parts with more tasks on less res.;

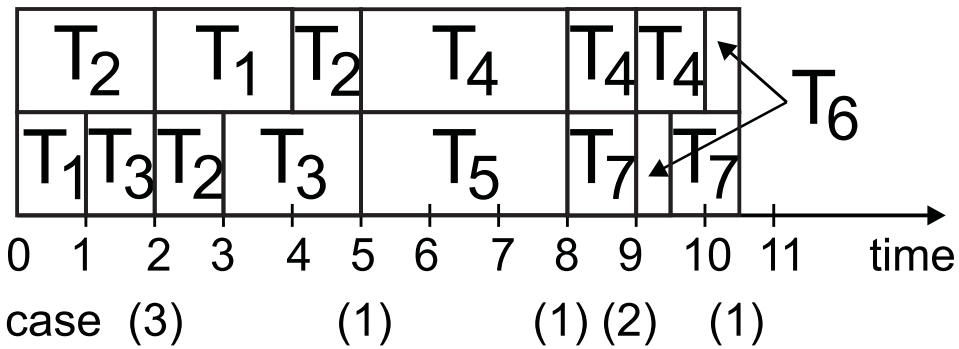
$t + \delta$ is time when (1) EITHER one **task is finished** (2) OR a current level of an assigned task becomes **lower than a level of an unassigned ready task** (3) OR a task executed at **faster rate** β starts to have current level **below** the current level of a task executed at **slower rate** β

Example - Muntz&Coffman's Alg. for $P\,|\text{pmtn, prec}|\,C_{max}$



- $t = 0, p = (3, 4, 3, 5, 3, 1, 2, 0)$
  $level = (8, 9, 8, 5, 5, 1, 2, 0), \mathcal{Z} = \{T_1, T_2, T_3\}$
  - $h = 2, \mathcal{S} = \{T_2\}, \beta = 1$
  - $h = 1, \mathcal{S} = \{T_1, T_3\}, \beta = 0.5$
  - $\delta = 2$ due to the case (3)
- $t = 2, p = (2, 2, 2, 5, 3, 1, 2, 0)$
  $level = (7, 7, 7, 5, 5, 1, 2, 0), \mathcal{Z} = \{T_1, T_2, T_3\}$
  - $h = 2, \mathcal{S} = \{T_1, T_2, T_3\}, \beta = 0.67$
  - $\delta = 3$ due to the case (1)
- ...





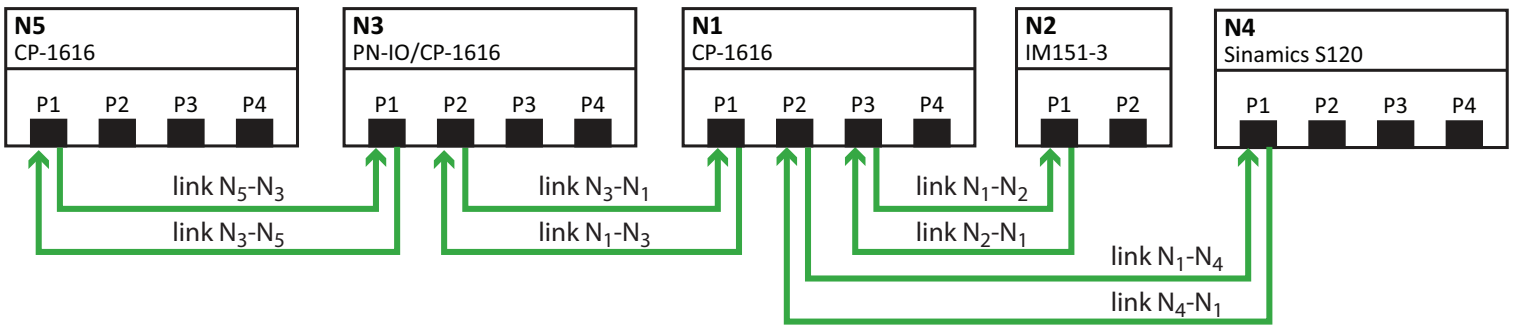
Project Scheduling - Minimizing C_{max}

- $PS1\,|\text{temp}|\,C_{max}$ - NP-hard
  - $PS1$ stands for single resource, temp stands for temporal constraints
  - Input: The number of non-preemptive tasks n and processing times $(p_1, p_2, ..., p_n)$. The temporal constraints defined by digraph G.
  - Output: n-element vector s, where s_i is the start time of T_i
- $PSm, 1\,|\text{temp}|\,C_{max}$ - NP-hard
  - $PSm, 1$ stands for m resource types, each of capacity 1
  - Input: The number of non-preemptive tasks n and processing times $(p_1, p_2, ..., p_n)$. The temporal constraints defined by digraph G. The number of dedicated resources m and the assignment of the tasks to the resources $(a_1, a_2, ..., a_n)$.
  - Output: n-element vector s, where s_i is the start time of T_i

Motivation Example: Message Scheduler for Profinet IO IRT - Specification

Profinet IO IRT is an Ethernet-based hard-real time communication protocol, which uses static schedules for time-critical data. Each node contains a special hardware switch that intentionally breaks the standard forwarding rules for a specified part of the period to ensure that no queuing delays occur for time-critical data.

Goal: Minimize the makespan (the schedule length) for time critical messages.



line	$N_1 \to N_3$	$N_1 \to N_4$	$N_1 \to N_2$	$N_2 \to N_1$	$N_3 \to N_1$	$N_4 \to N_1$	$N_3 \to N_5$	$N_5 \to N_3$
line delay [ns]	4875	5130	5862	3841	4875	4895	4875	4875

Motivation Example: Message Scheduler for Profinet IO IRT - Specification

Constraints:

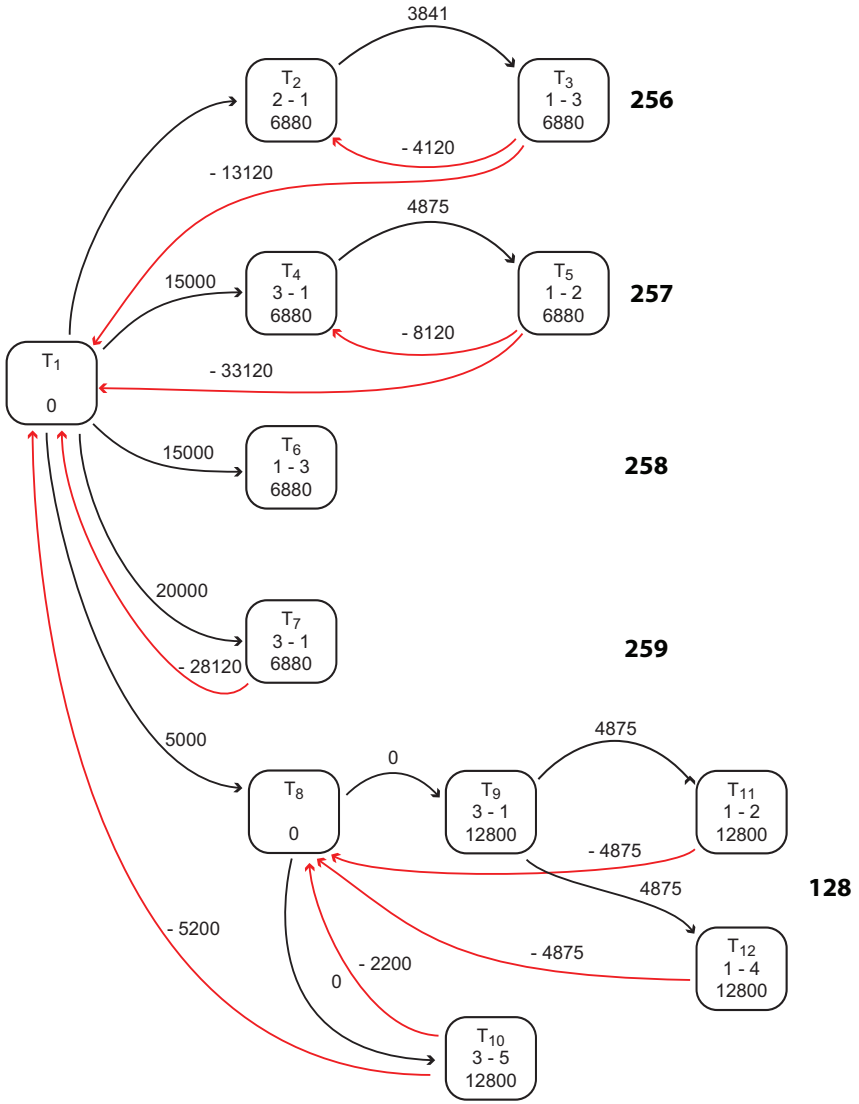
- tree topology $\Rightarrow$ fixed routing
- release date r - earliest time the message can be sent
- deadline $\widetilde{d}$ - latest time the message can be delivered
- maximal allowed end-to-end time delay

ID	source $\rightarrow$ target	length [ns]	r [ns]	$\widetilde{d}$ [ns]	end2end delay [ns]
256	$N_2 \rightarrow N_3$	5760	5000	20000	11000
257	$N_3 \rightarrow N_2$	5760	15000	40000	15000
258	$N_1 \rightarrow N_3$	5760	15000	–	–
259	$N_3 \rightarrow N_1$	5760	20000	35000	–
128	$N_3 \rightarrow \{N_1,N_2,N_4,N_5\}$	11680	5000	{–,–,–,18000}	{–,17675,17675,15000}

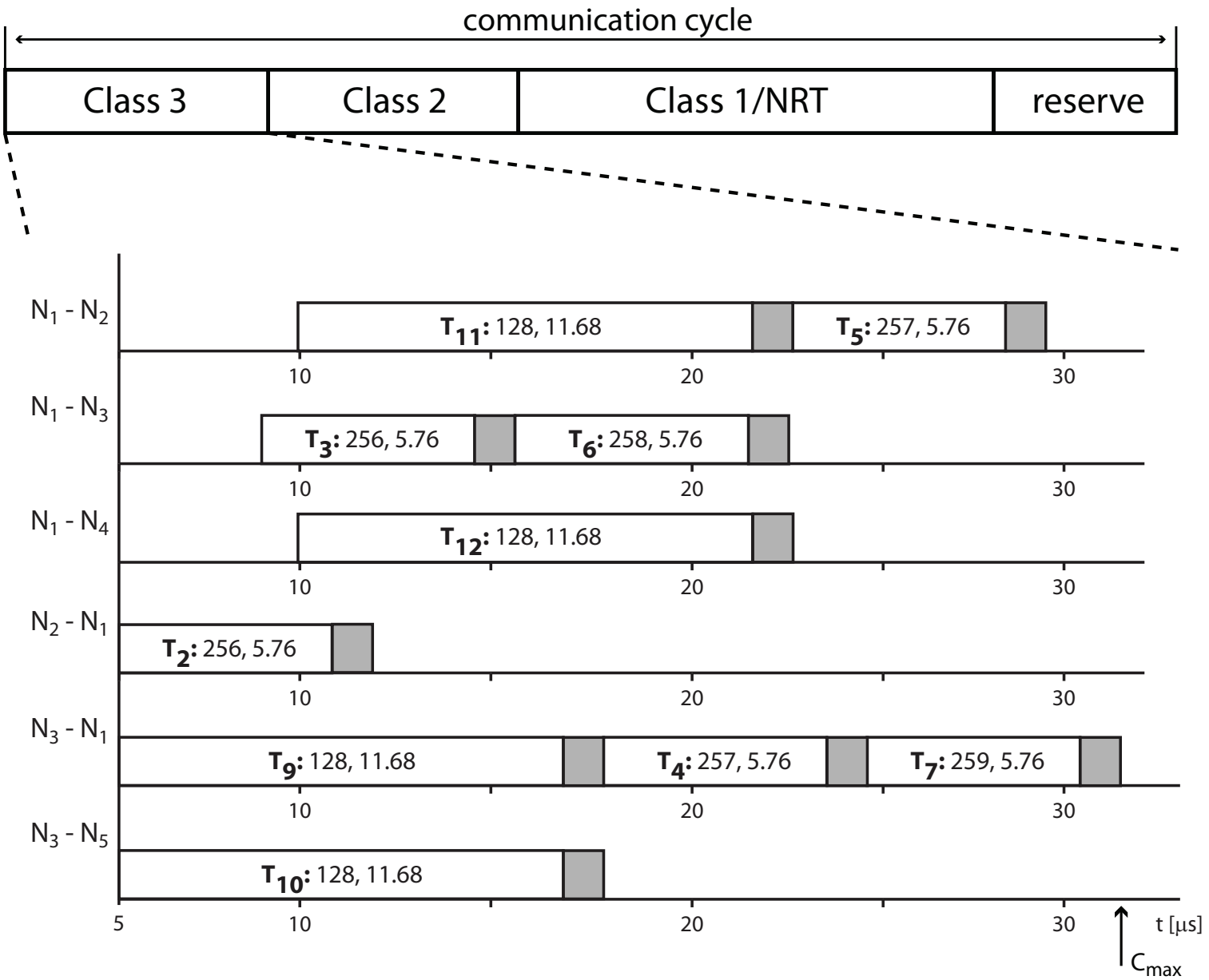
Motivation Example: Message Scheduler for Profinet IO IRT - Formalization

Can be formulated as $PSm, 1 \,|temp|\, C_{max}$ problem.

- task = message on a given line
- positive cost edge = r, precedence relations
- negative cost edge = $\widetilde{d}$, end-to-end delay
- unicast message = chain of tasks (assuming positive edges)
- multicast message = out-tree of tasks (assuming positive edges)

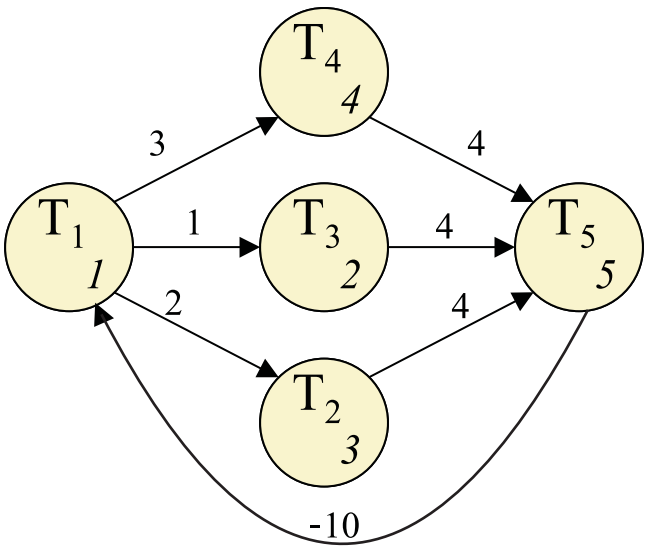


Motivation Ex.: Message Sch. for Profinet IO IRT - Result

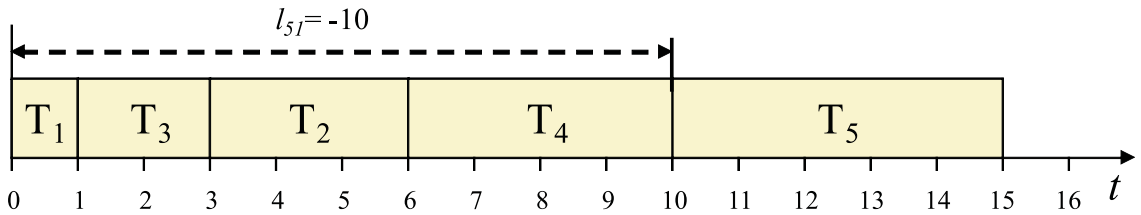


Temporal Constraints

- Set of non-preemptive tasks $\mathcal{T} = \{T_1, T_2, ..., T_n\}$ is represented by the nodes of the directed graph G (may include negative cycles).
- 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 $s_i + l_{ij} \leq s_j$.



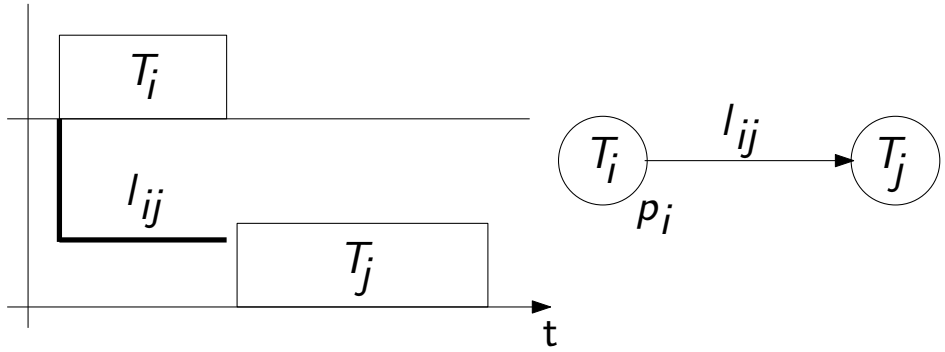
Temporal Constraints $s_i + l_{ij} \leq s_j$ with Positive l_{ij}

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

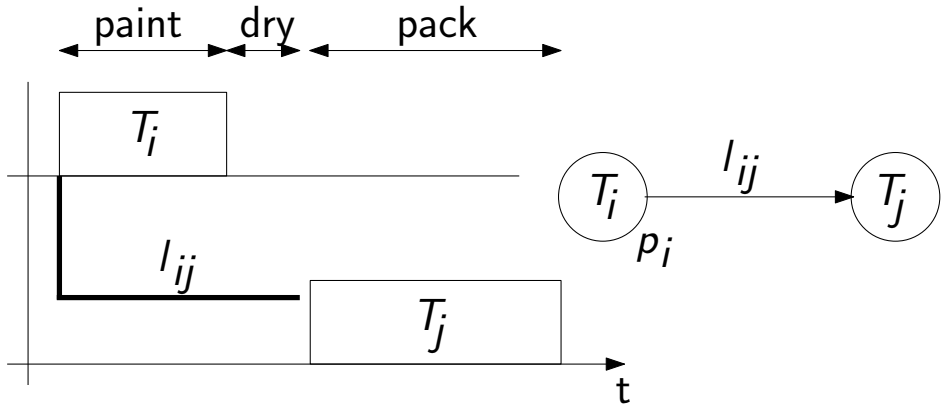
a) $l_{ij} = p_i$

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



b) $l_{ij} > p_i$

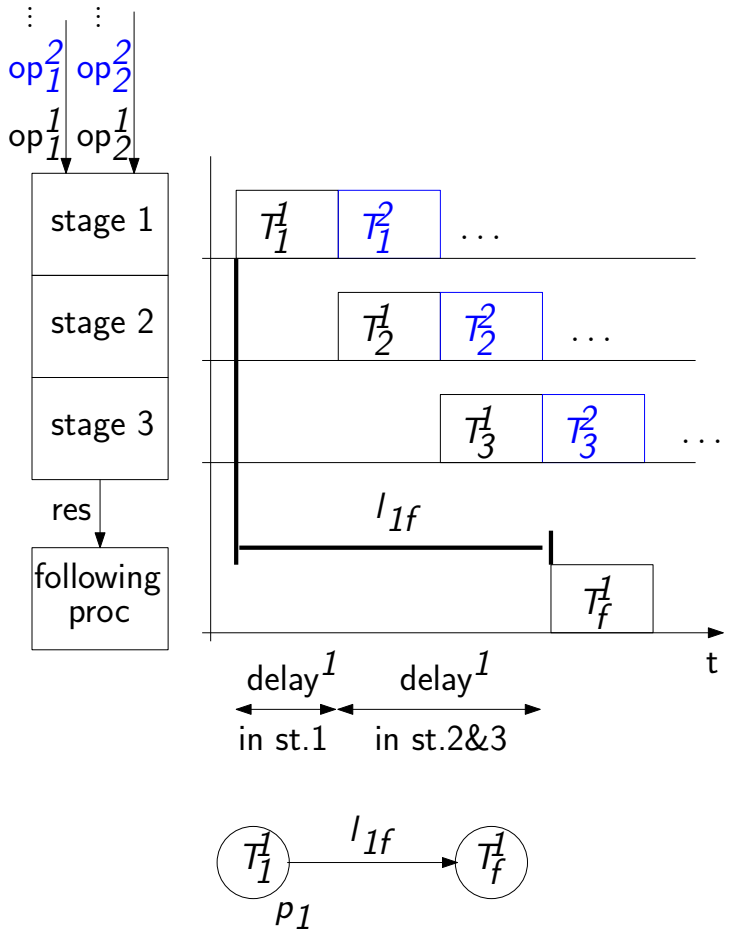
- the second task can start some time after the completion of previous task
- b.1) example of a dry operation performed in sufficiently large space



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

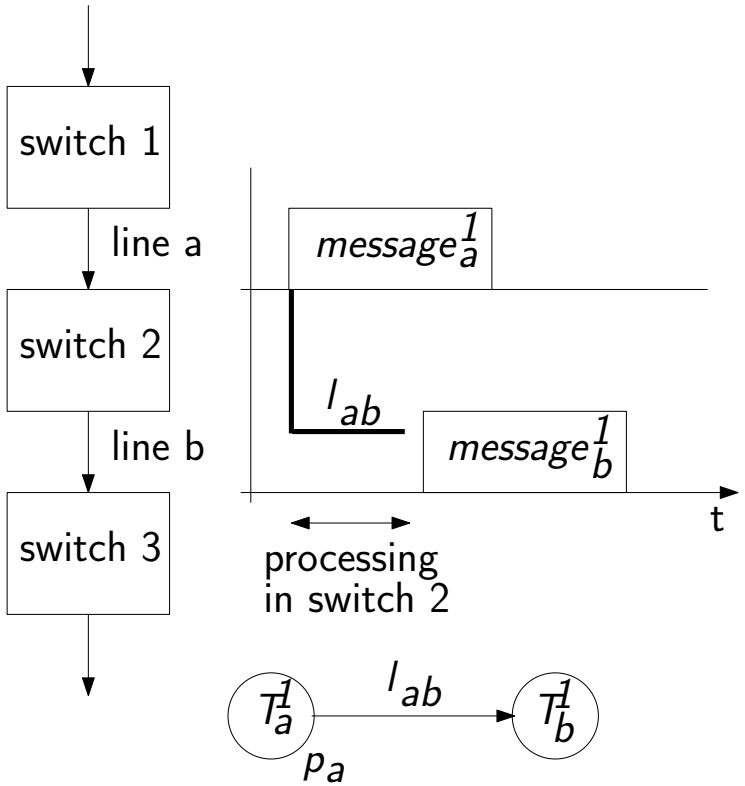


Temporal Constraints $s_i + l_{ij} \leq s_j$ with Positive l_{ij}

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.

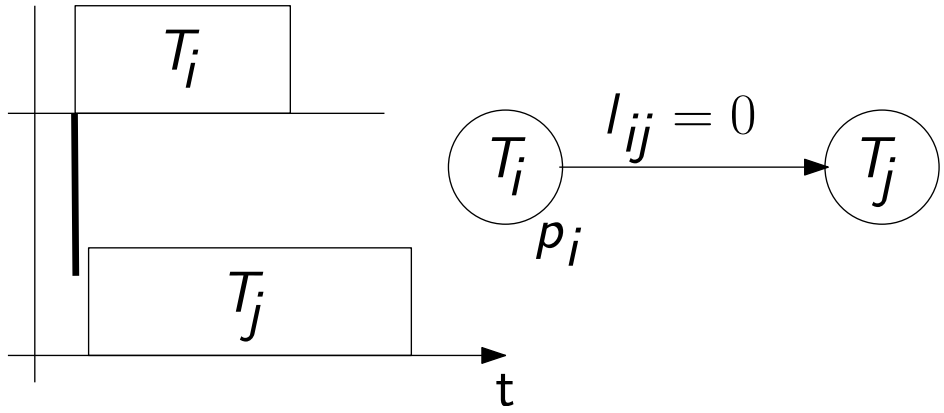
- time-triggered protocol
- **resources are communication links**
- l_{ab} represents the **delay in the switch**
- **different parts** of the same message are transmitted by several communication links at the same time



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

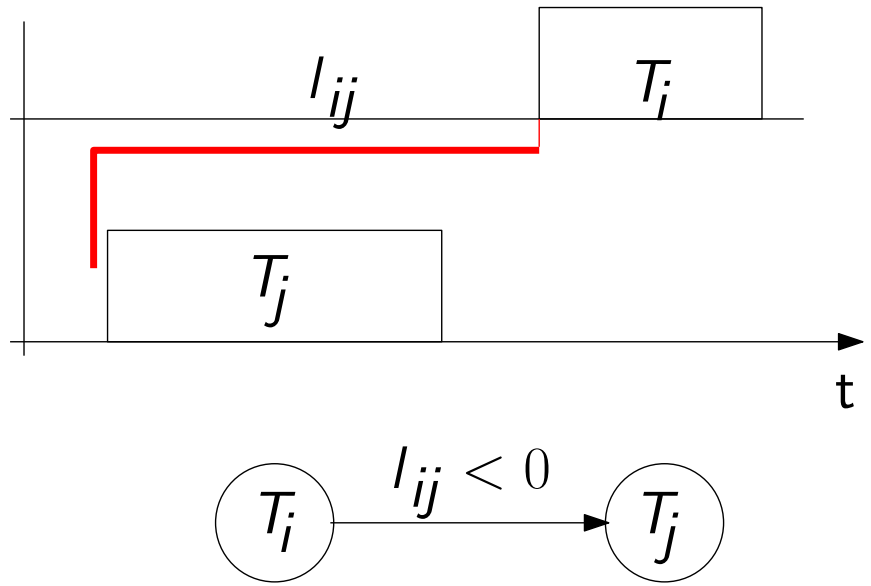
d) $l_{ij} = 0$

- 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



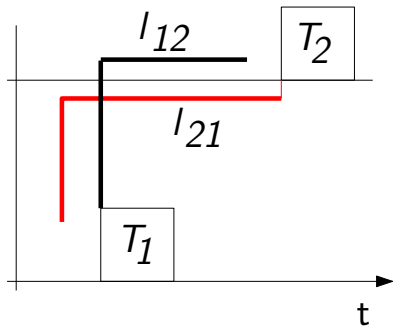
Cycles and Relative Time Windows

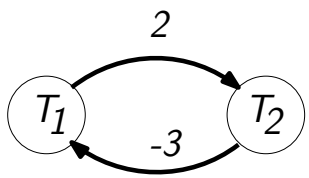
Absence of a positive cycle in graph G

- is a **necessary condition** for schedulability of $PS1\,|\text{temp}|\,C_{max}$
- is a **necessary and sufficient condition** for schedulability of the instance with **unlimited capacity of resources**. The schedule, which is restricted only by the temp. constraints, can be found in pol. time
  - by LP or
  - by the longest paths. For G we can create G'**, a complete digraph of longest paths**, where weight l_{ij} is the length of the longest directed path from T_i to T_j in G (if no directed path in G exists, the weight is $l_{ij} = -\infty$). A start time of T_j is lower bounded by the longest path from arbitrary node, i.e. $s_j \geq \max_{\forall i \in 1 \ldots n} l_{ij}$.

Example - relative time window, e.g. when applying a catalyst to the chemical process
If finite $l_{ij} \geq 0$ and $l_{ji} < 0$ do exist, 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





ILP formulation of $PS1\,|\text{temp}|\,C_{max}$

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 assumed to be 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 (tasks with zero processing time may be used to represent events).

Both models contain two types of constraints:

- temporal constraints
- resource constraints - prevent overlapping of tasks

Time-indexed Model for $PS1\,|\text{temp}|\,C_{max}$

$\min C_{max}$

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

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

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

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

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

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

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 $PS1\,|\text{temp}|\,C_{max}$

$\mathcal{T} = \{T_1, T_2, T_3\}$, $p = (1, 2, 1)$, $UB = 5$

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}

$\Sigma \leq 1$

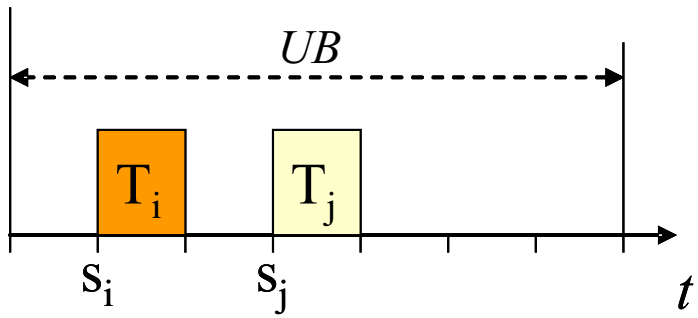
Relative-order Model for $PS1\,|\text{temp}|\,C_{max}$

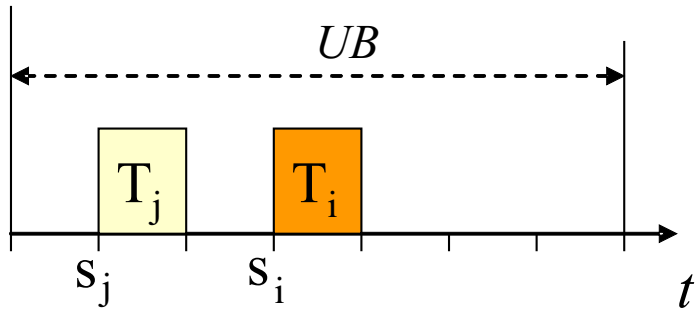
Resource constraint for couple of tasks:
$p_j \leq s_i - s_j + UB \cdot x_{ij} \leq UB - p_i$

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
$s_j + p_j \leq s_i$.





Relative-order Model for $PS1\,|\text{temp}|\,C_{max}$

$$\min C_{max}$$

$$s_i + l_{ij} \leq s_j \qquad \forall l_{ij} \neq -\infty \text{ and } i \neq j \text{ (temporal constraint)}$$

$$p_j \overset{violet}{\leq} s_i - s_j + UB \cdot x_{ij} \overset{green}{\leq} UB - p_i \qquad \forall i, j \in \{1, \ldots, n\} \text{ and } i < j \text{ (resource constraint)}$$

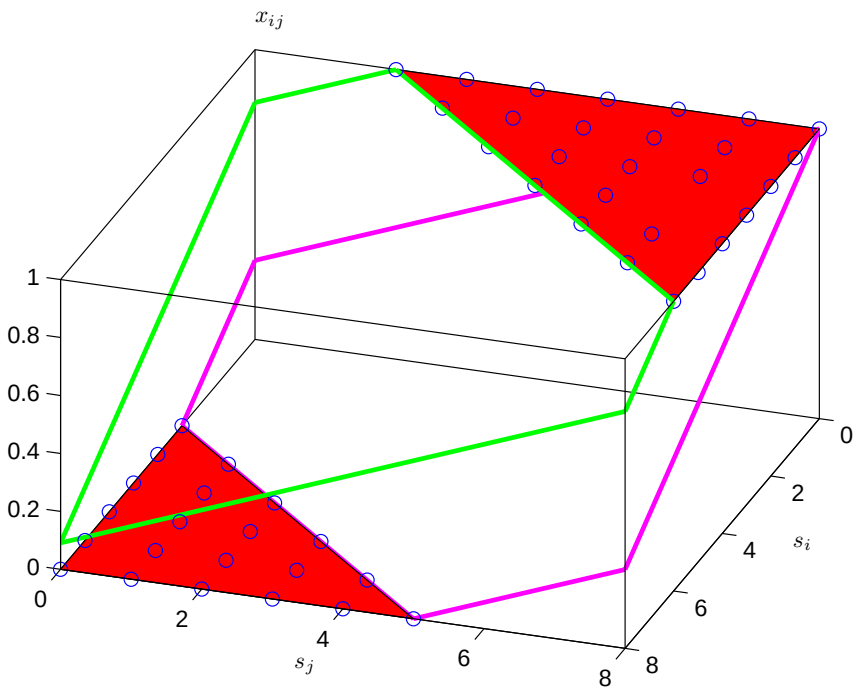
$$s_i + p_i \leq C_{max} \qquad \forall i \in \{1, \ldots, n\}$$

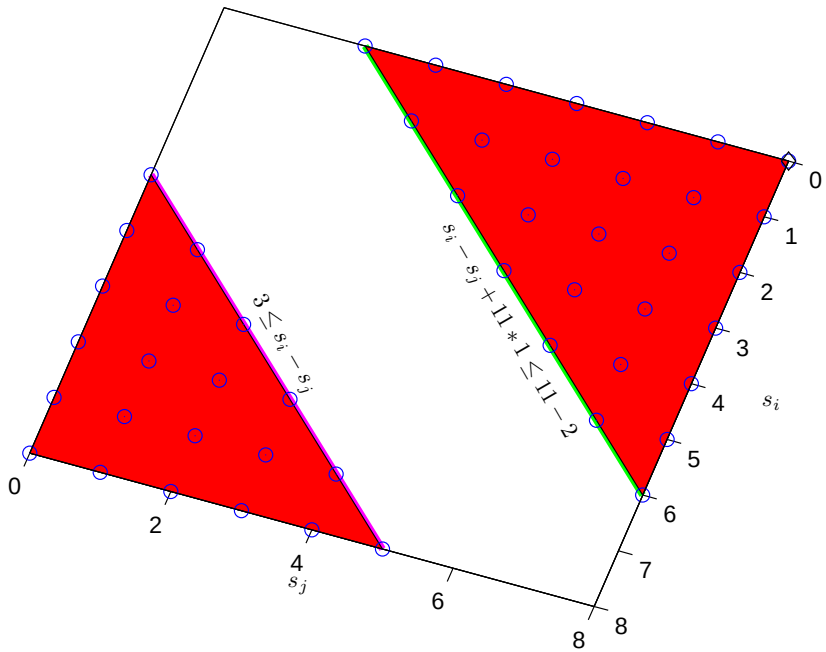
variables: $x_{ij} \in \{0, 1\}$, $C_{max} \in \langle 0, UB \rangle$, $s_i \in \langle 0, UB - p_i \rangle$

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

Relative-order Model for $PS1\,|\text{temp}|\,C_{max}$

Example: no temporal constraints, two tasks T_i, T_j with $p_i = 2$ and $p_j = 3$. We set $UB = 11$ and we study $s_i \in \langle 0, 8 \rangle$.
3D polytope (left) is determined by the resource constr. given by violet and green hyperplanes (see colors on the previous slide). Its projection to 2D space (right) shows both sequences of tasks. When we change UB, the hyperplanes in 3D decline.





Comparison of the Two Models

Each model is suitable for different types of tasks:

Time-indexed model:

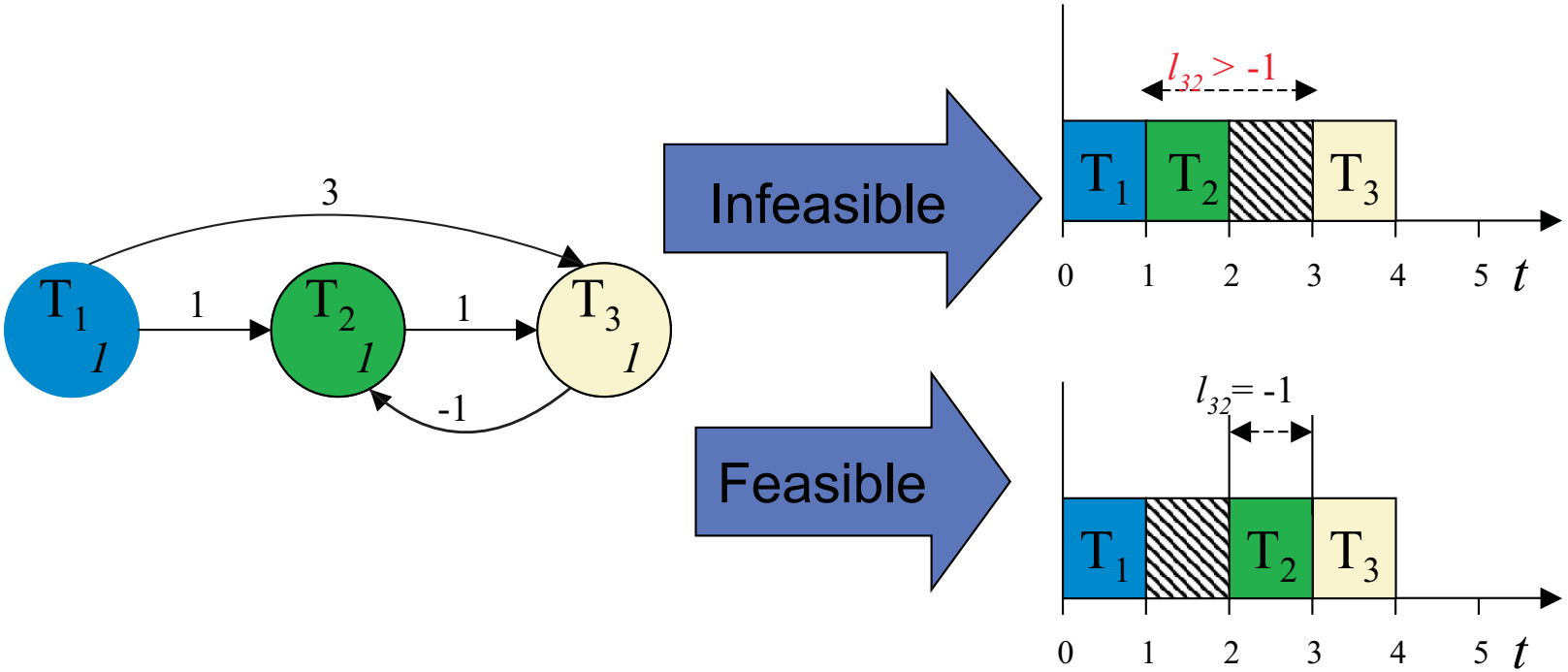
- (+) Can be easily extended for parallel identical processors.
- (+) ILP formulation does not need many constraints.
- (-) The size of the model grows with the size of UB.

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 of edges with positive weight, 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 Project Scheduling with Dedicated Resources of Unit Capacity $PSm, 1 \,|\text{temp}|\, C_{max}$

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

$$\min C_{max}$$

$$s_i + l_{ij} \leq s_j \qquad \forall l_{ij} \neq -\infty \text{ and } i \neq j$$
(temporal constraints)

$$p_j \leq s_i - s_j + UB \cdot x_{ij} \leq UB - p_i \qquad \forall i, j \in \{1, \ldots, n\},\ i < j \text{ and } \underline{a_i = a_j}$$
(on the same resource type)

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

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 $n + \left(n^2 - n\right)/2 + 1$ variables (exact number depends on the number of tasks scheduled on each resource type).

Modeling with Temporal Constraints

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

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

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

- scheduling of **multiprocessor task** - task needs more than one resource type at a given moment,

Reduction from 1 $\left|r_j, \widetilde{d}_j\right|$ C_{max} to $PS1\,|\text{temp}|\,C_{max}$

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

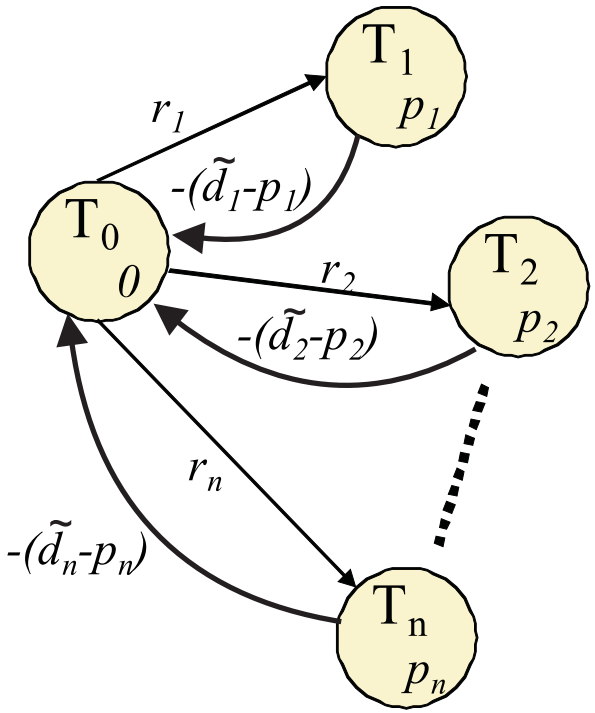
Instance 1 $\left|r_j, \widetilde{d}_j\right|$ C_{max}

$r = (r_1, r_2, \ldots, r_n)$

$p = (p_1, p_2, \ldots, p_n)$

$\widetilde{d} = (\widetilde{d}_1, \widetilde{d}_2, \ldots, \widetilde{d}_n)$

P-reducible



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

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 three steps:

- **Add dummy task** T_0 to have fixed point in time. Task T_0 has $p_0 = 0$ and precedes all tasks $\{T_1, \ldots, T_n\}$, i.e. $s_0 \leq s_i$. Due to the criterion $s_0 = 0$ (i.e. T_0 is fixed on the time axis).
- **Add new temporal constraints** to keep tasks $\{T_1, \ldots, T_n\}$ in their time-windows.
- **Transform the original temporal constraints** to $l'_{ij} = l_{ij} + (a_j - a_i) \cdot UB$. See its derivation on the next slide.
- **Add dummy task** T_{n+1} to propagate the criterion minimization in all time-windows. Task T_{n+1} has $p_{n+1} = 0$ and follows all task $T_i \in \mathcal{T}$, i.e. $C_i = s_i + p_i \leq s_{n+1}$.

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

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

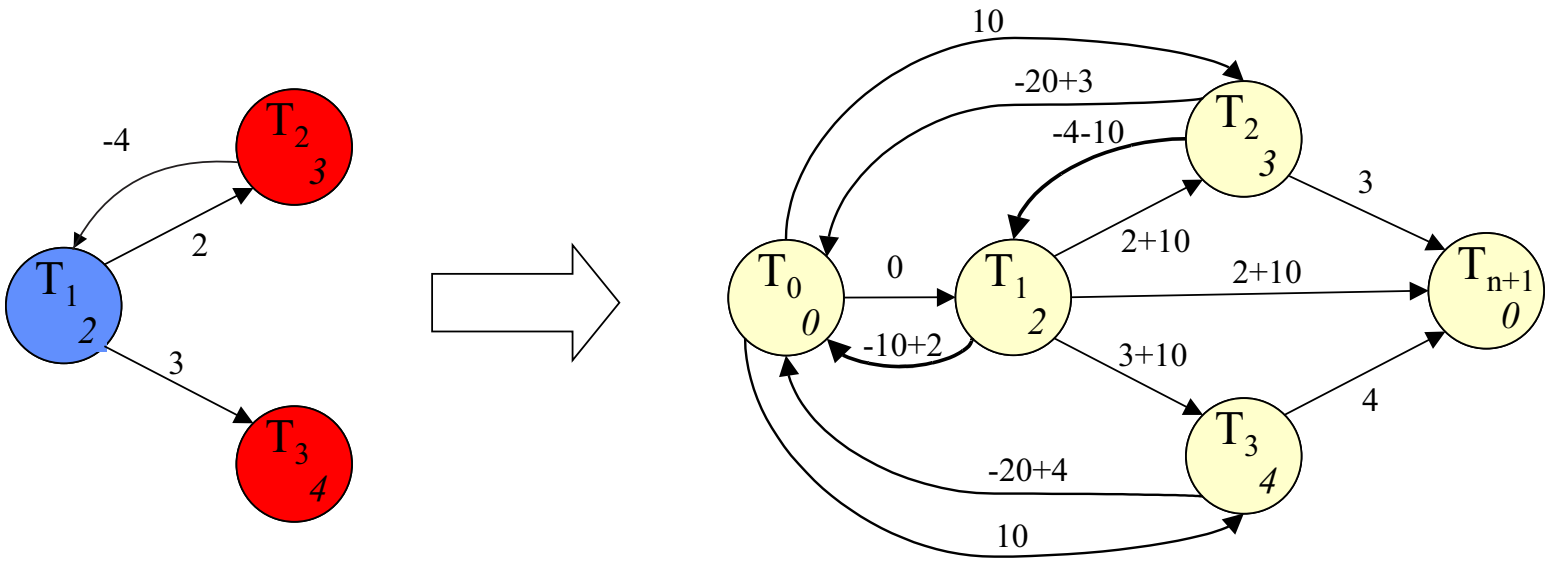
Temporal constraints $s_i + l_{ij} \leq s_j$ are transformed to:

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

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
T_1 on P^1 and T_2,T_3 on P^2

one resource

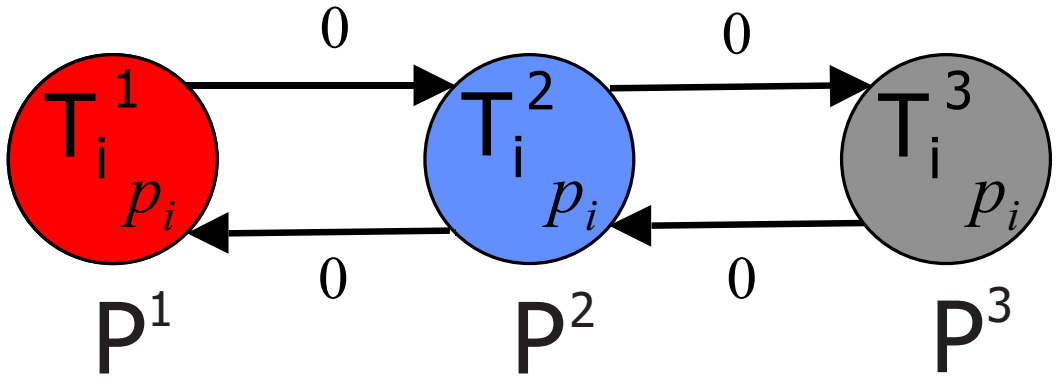
While minimizing the completion time of T_{n+1}, we push tasks T_1, T_2 and T_3 "to the left" due to the edges entering T_{n+1}

Multiprocessors Task

Transformation of multiprocessor task problem to $PSm, 1\,|\text{temp}|\,C_{max}$

- 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).



References


J. Błażewicz, K. Ecker, G. Schmidt, and J. Węglarz.
Scheduling Computer and Manufacturing Processes.
Springer, second edition, 2001.


Zdeněk Hanzálek, Michael Pinedo, and Guohua Wan.
Scheduling seminar.
www.schedulingseminar.com.
www.youtube.com/channel/UCUoCNnaAfw5NAntItILFn4A.


Klaus Neumann, Christoph Schwindt, and Jeurgen Zimmermann.
Project Scheduling with Time Windows and Scarce Resources.
Springer, 2003.


Sigrid Knust Peter Brucker.
Complexity results for scheduling problems.
www2.informatik.uni-osnabrueck.de/knust/class/.