



Planning & Scheduling

Roman Barták

Department of Theoretical Computer Science and Mathematical Logic

Classical Scheduling

From planning to scheduling

- **Planning** deals with **causal relations** between actions and solves the problem which actions are necessary to reach a goal.
- **Scheduling** focuses on **allocation** of actions to **time** and **space** (resources).

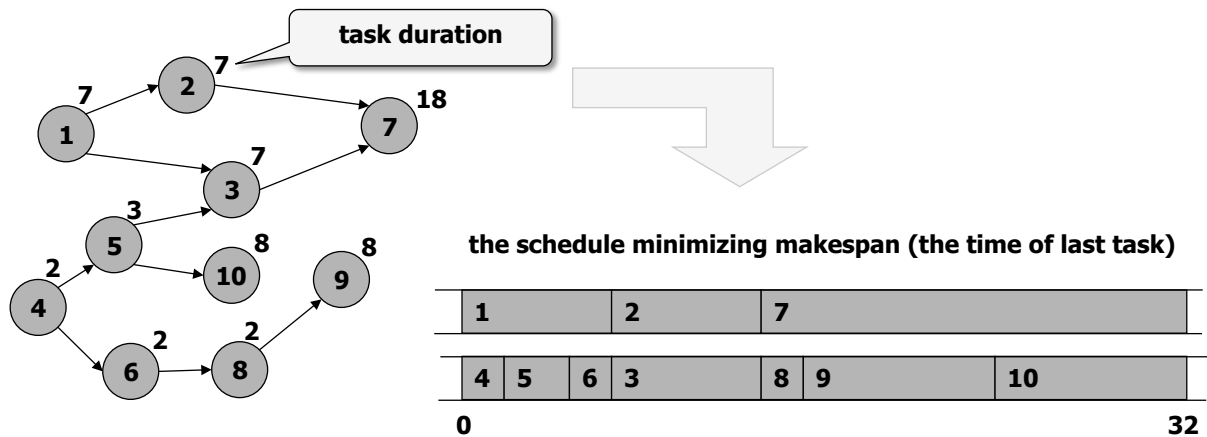


- Sometimes both tasks are better to be solved together.
 - For example, when there are many plans but only a few of them can be scheduled.

Scheduling deals with optimal allocation of a given set of actions to time and resources.

Example (construction of a bicycle by two workers):

- tasks have a fixed duration and are non-interruptible
- there are precedence constraints between the tasks



There exists a widely accepted classification of scheduling problems*, so called **Graham notation**.

* unary resources assumed

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

resource description

- Describes resource allocation
- unique resource
 - alternative resources
 - identical
 - uniform
 - arbitrary
 - multi-operation mode
 - job-shop,
 - open-shop
 - flow-shop

task description

- Describes restrictions on tasks
- pre-emption
 - precedence relations
 - release time
 - deadline
 - batch processing

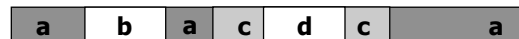
objectives

Describes criteria to be optimized



Usually we look for time allocation of tasks:

- **start time $s(a)$**
 - there can be an earliest possible start time (**release date**)
- **end time $e(a)$**
 - there can be a latest possible end time (**deadline**)
 - there can be a recommended end time $\delta(a)$ (**due date**)
- **duration $p(a)$**
 - it can be constant or dependent on a resource (if resource allocation is part of the problem)
- Let us assume that at most one task can be processed by a resource at any time (**unary/disjunctive resource**)
 - the task can run without interruption (**non-preemptive tasks**)
 - $p(a) = e(a) - s(a)$
 - or the task can be interrupted by other tasks (**preemptive tasks**)
 - $p(a) = \sum_i p_i(a) \leq e(a) - s(a)$



The scheduling problem is usually an optimisation problem.

Typical **objectives** are based on the following notions:

– makespan C	$\max\{e(a) \mid a \in A\}$
– lateness L	$e(a) - \delta(a)$
– earliness E	$\max(0, \delta(a) - e(a))$
– tardiness τ	$\max(0, e(a) - \delta(a))$
– absolute deviation D	$ e(a) - \delta(a) $
– square deviation S	$(e(a) - \delta(a))^2$
– late tasks U	0, for $e(a) \leq \delta(a)$ 1, otherwise

Other typical objectives:

- minimization of resource reconfiguration
- minimization of the number of used resources
- minimization of maximal resource load
- maximization of the number of scheduled tasks



<p>$Qm \parallel r_i \mid C_{max}$ Lawler et al. [133]</p> <p>$Qm \parallel \sum w_i C_i$ Lawler et al. [133]</p> <p>$Qm \parallel \sum w_i U_i$ Lawler et al. [133]</p> <p>Table 5.2: Pseudopolynomially solvable parallel machine scheduling problems without preemption.</p>	<p>$P \mid pmtn \mid C_{max}$ McNaughton [152]</p> <p>$P \mid outtree; pmtn; r_i \mid C_{max}$ Lawler [127]</p> <p>$P \mid tree; pmtn \mid C_{max}$ Gonzalez & Johnson [92]</p> <p>$Pm \mid prec \mid C_{max}$ [121]</p> <p>$Q2 \mid prec; pmtn; \mid L_{max}$ Lawler [127]</p> <p>$Q2 \mid prec; pmtn; r_i \mid L_{max}$ Lawler [127]</p> <p>$P \mid pmtn \mid L_{max}$ Baptiste [16]</p> <p>$Q \mid pmtn \mid L_{max}$ [94]</p> <p>$Q \mid pmtn, r_i; d_i \mid -$ [94]</p> <p>$R \mid pmtn; r_i \mid L_{max}$ Lawler & Labetoulle [130]</p>	<p>$J2 \mid n_i \leq 2 \mid C_i$ [121]</p> <p>$Jm \mid \mid C_{max}$</p> <p>$J2 \mid p_{ij} = 1 \mid C_i$ Brucker [28]</p> <p>$J2 \mid p_{ij} = 1; r_i$ Timkovsky [183]</p> <p>$J2 \mid n = k \mid C_{max}$ Brucker & Krämer [42]</p> <p>$J \mid prec; p_{ij} = 1; r_i; n = k \mid f_{max}$ Kubiak & Timkovsky [119]</p> <p>$J2 \mid p_{ij} = 1 \mid \sum C_i$ Kravchenko [114]</p> <p>$J \mid p_{ij} = 1; r_i; n = k \mid f$ Brucker et al. [44]</p> <p>$J \mid prec; r_i; n = 2; pmtn \mid f$ Sotskov [174]</p> <p>$J \mid prec; p_{ij} = 1; r_i; n = k \mid \sum f_i$ Brucker & Krämer [39]</p>
<p>$1 \mid r_j \mid L_{max}$</p> <p>* $P \parallel C_{max}$ Garey & Johnson [86]</p> <p>* $P \mid p_i = 1;intree; r_i \mid C_{max}$ Brucker et al. [29]</p> <p>* $P \mid p_i = 1;prec \mid C_{max}$ Ullman [187]</p> <p>* $P2 \mid chains \mid C_{max}$ Du et al. [74]</p> <p>* $Q \mid p_i = 1;chains \mid C_{max}$ Kubiak [117]</p> <p>* $P \mid p_i = 1;outtree$</p> <p>* $P \mid p_i = 1;intree$</p> <p>* $P \mid p_i = 1;prec \mid \sum C_i$</p> <p>* $P2 \mid chains \mid \sum C_i$ Du et al. [74]</p> <p>* $P2 \mid r_i \mid \sum C_i$ Single-machine problem</p> <p>* $P2 \parallel \sum w_i C_i$ Bruno et al. [48]</p> <p>* $P \parallel \sum w_i C_i$ Lenstra [138]</p> <p>* $P2 \mid p_i = 1;chains \mid \sum w_i C_i$ Timkovsky [185]</p> <p>* $P2 \mid p_i = 1;chains \mid \sum U_i$ Single-machine problem</p> <p>* $P2 \mid p_i = 1;chains \mid \sum T_i$ Single-machine problem</p> <p>Table 5.3: NP-hard parallel machine problems without preemption.</p>	<p>$Pm \mid preempt \mid C_{max}$</p> <p>$P \mid pmtn \mid \sum C_i$ Labetoulle et al. [121]</p> <p>$P \mid p_i = p; pmtn \mid \sum w_i C_i$ McNaughton [152]</p> <p>$Qm \mid pmtn \mid \sum U_i$ Lawler [126]</p> <p>$Pm \mid p_i = p; pmtn \mid \sum w_i U_i$ Baptiste [17], Baptiste [10]</p> <p>Table 5.4: Polynomially solvable preemptive parallel machine scheduling problems.</p>	<p>$F2 \mid \mid C_{max}$</p> <p>$J2 \mid n_i \leq 2 \mid C_{max}$</p> <p>Table 6.6: Polynomially solvable job shop problems.</p> <p>Table 6.7: Pseudopolynomially solvable job shop problems.</p>

Peter Brucker: **Scheduling Algorithms**, Third Edition, Springer Verlag, 2001

$1 \mid r_j \mid L_{max}$



Single resource scheduling problem with **release dates** and **due dates**. The objective is to minimize **lateness**.

In general, this is an **NP-hard problem**.

Adding further constraints (such as identical release dates or identical due dates) or by relaxing some constraints (allowing pre-emption) give a **tractable problem** (solvable in polynomial time)

- all tasks have **identical release date** ($r_j = r$)
 - we will use the **earliest due date (EDD) rule**
 - tasks are ordered in ascending order by due dates
- all tasks have **identical due date** ($\delta_j = \delta$)
 - tasks are ordered in ascending order by release dates
- tasks are **pre-emptible**
 - we will use again the EDD rule in the following way
 - select tasks with the earliest release date and if there are more such tasks then select the task with earliest due date
 - if the scheduled task finishes or some other task becomes available (we reach its release date) then continue with the task that can run at that time and has earliest due date

Example

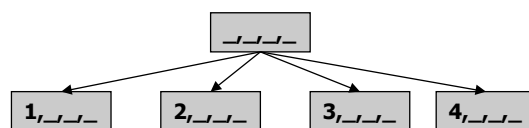
task	p_j	r_j	δ_j
1 	2	0	8
2 	6	0	7
3 	5	1	6



$1 | r_j | L_{max}$ is an NP-hard problem

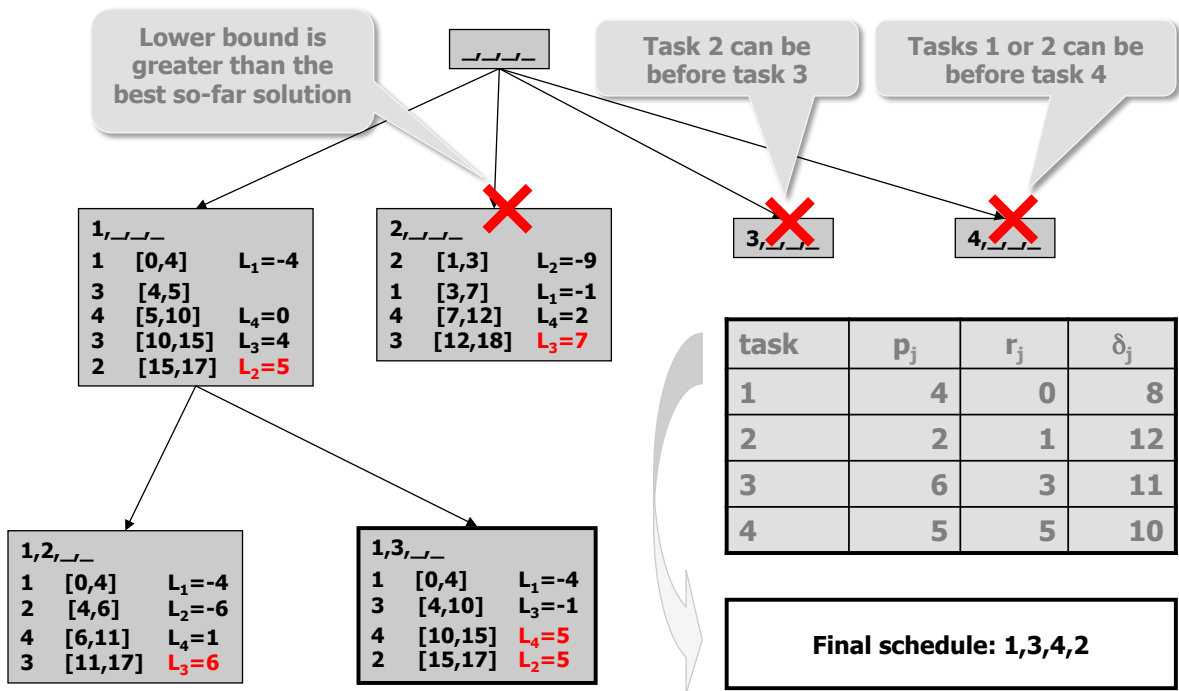
We will exploit the **branch-and-bound** method .

- in general, the time complexity is **exponential**
- the schedule is built from **left to right** by exploring tasks
 - **branching** corresponds to tasks selected to be scheduled first



Only tasks j are assumed such that there is no task i that can be completed before j without shifting the start time of task j .

- we will exploit a **lower bound** for lateness in each sub-tree
 - if the lower bound is greater than the best-so-far solution (bound) then we do not need to explore the sub-tree (no better solution is there)
 - the lower bound is obtained via **relaxation** to $1 | r_j, \text{preempt} | L_{max}$
 - » the optimal preemptive schedule will never have worse lateness than the optimal non-preemptive schedule
 - if we obtain a solution to the problem $1 | r_j, \text{preempt} | L_{max}$ that is not preemptive then we can use this solution (no need to continue in search) as the best non-preemptive schedule



We do not need to continue because:

- optimal pre-emptive solution does not use pre-emption
- optimal value equals the lower bound from the parent node

Parallel resources

Frequently, there may be alternative resources to process tasks:

- **identical resources**
 - task duration does not depend on the resource allocated to the task
- **uniform resources**
 - base task duration is multiplied by resource-speed co-efficient to get task duration for the resource
 - for example, if A is 3-times longer than B on machine R then A will be 3-times longer on any other machine S, even if the duration of A on R may be different from the duration of A on S
- **general resources**
 - task duration depends on the resource arbitrarily
 - it may happen that duration of A is smaller than duration of B on machine R, while B is shorter than A on machine S

Note:

- If there are **alternative resources for preemptive tasks** then during preemption the task can migrate from one resource to another resource (the only restriction is that parts of the task on different resource cannot overlap in time).

Minimize makespan for preemptive tasks running on m identical resources.

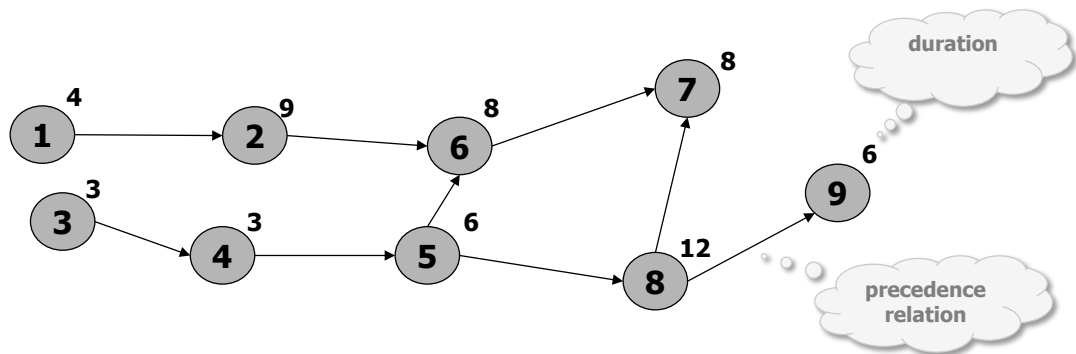
Lower bound for makespan:

– $LB = \max\{\max_i p_i, (\sum_i p_i)/m\}$

The schedule with makespan LB can be constructed in time **$O(n)$** , where n is the number of tasks:

- sequence tasks in any order on the first (any) resource
- when reaching time LB, then split the last task (that exceeds LB) and put it to the next empty resource
- thanks to $p_i \leq LB$ the two parts of the split task will not overlap in time

Minimize makespan for n non-preemptive tasks to be allocated to m identical alternative resources. There are precedence relations between the tasks.



- $P_m | prec | C_{max}$ $n \leq m$ use a critical-path method
- $P_m | prec | C_{max}$ $2 \leq m < n$ NP-hard problem

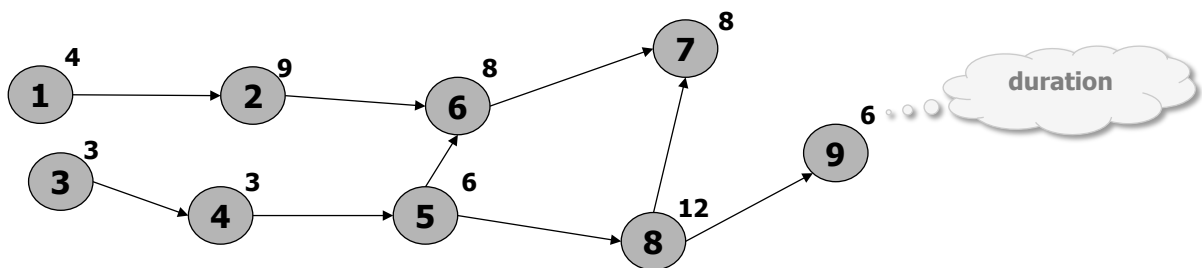
Some terminology:

- **free task** – a task that can be delayed (a bit) without delaying makespan
- **critical task** – a task that cannot be delayed without delaying makespan
- **critical path** – a set of critical tasks

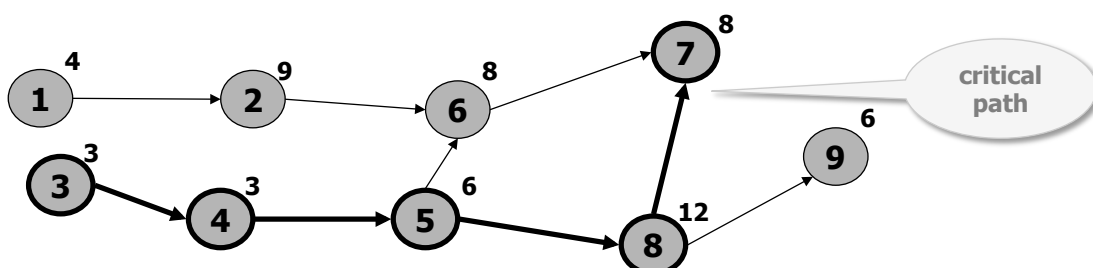
Solution approach

- **forward stage** finds the earliest start times (est) and earliest completion times (ect) for all tasks and the value of makespan C_{max}
 - start with tasks that have no predecessor: $est_i=0, ect_i=p_i$
 - process tasks such that all their predecessors have been processed:
 $est_i=\max_{j<i} ect_j, ect_i=est_i+p_i$
 - $C_{max} = \max_i ect_i$
- **backward stage** finds the latest start times (lst) and latest completion times (lct) for all tasks
 - start with tasks that have no successor: $lct_i=C_{max}, lst_i=C_{max}-p_i$
 - process tasks such that all their successors have been processed:
 $lct_i=\min_{i<j} lst_j, lst_i=lct_i-p_i$
 - verify $0 = \min_i lst_i$
- All tasks such that $est_i=lst_i$ are **critical tasks**.
- The **critical path** is a task sequence starting at time 0 and finishing at time C_{max} .

Example: Pm | prec | C_{max}



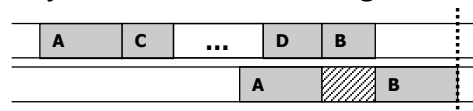
task	1	2	3	4	5	6	7	8	9
est	0	4	0	3	6	13	24	12	24
ect	4	13	3	6	12	21	32	24	30
lst	3	7	0	3	6	16	24	12	26
lct	7	16	3	6	12	24	32	24	32



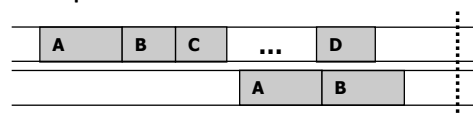
- Job frequently consists of smaller operations. This is called a **shop problem**.
 - there can be precedence relations between the operations
 - resource is pre-allocated to each operation
- **Job-shop**
 - operations within a job are totally ordered
 - different jobs may have different operations with different orders
 - usually, each resource is visited at most once by each job
- **Flow-shop**
 - a special case of job-shop
 - all jobs have identical operations in the same order
 - for example assembly line production
- **Open-shop**
 - similar to flow-shop, but no precedence relations between operations
 - for example product configuration problems

When minimizing makespan for the flowshop problem, there exists an optimal schedule such that the order of operations on the first (last) two resources is identical.

- let k be the number of jobs such that the order of their operations is identical on the first two resources
- let us take an optimal schedule with maximal k
- if k is smaller than the number of jobs then the following situation appears:



- then we can put the operation of job B right after the operation of A on the first resource and shift the operation of B on the second resource to earlier time



- this way the makespan will not enlarge so we still have an optimal schedule, but with a larger value of k – this is a contradiction as k was maximal (hence k equals the number of jobs)

Flow-shop problem with two resources and minimization of makespan.

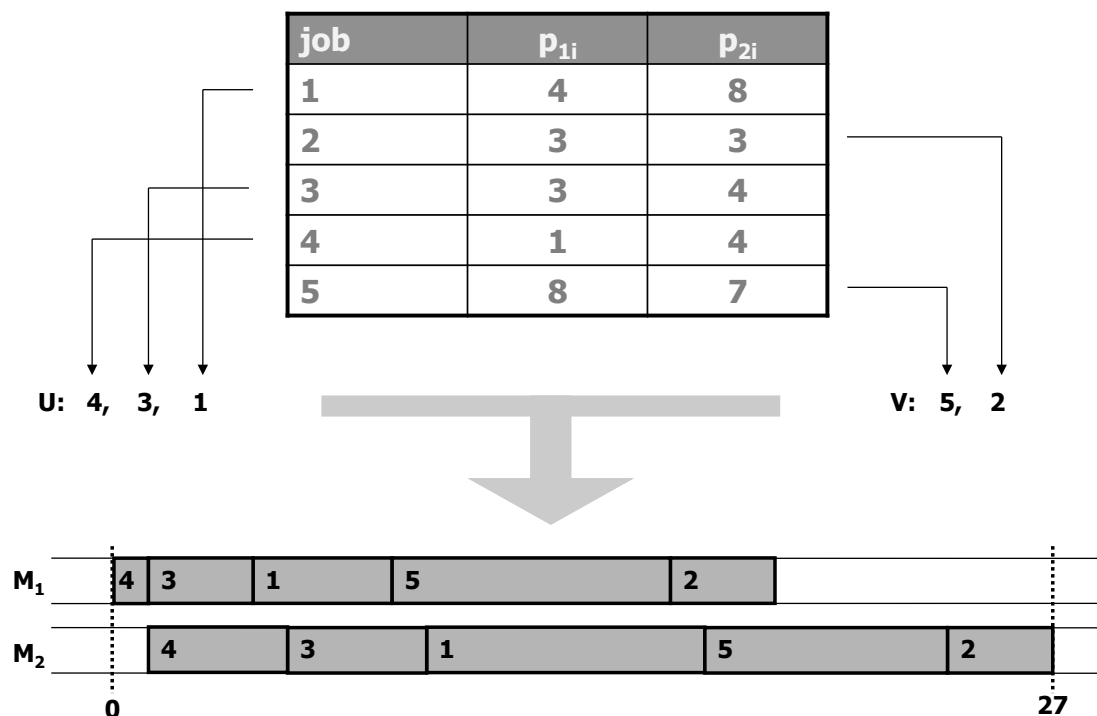
- each job consists of two operations, the first operation runs on the first resource, the second operation runs on the second resource
- the goal is to minimize the end time of the last task (makespan)

Based on the previous theorem, it is enough to **order the jobs on the first resource** (as the order of jobs on the second resource will be identical).

Solution approach:

1. distribute the jobs into two sets:
 - jobs with operation on resource 1 shorter than the operation on resource 2
 $U = \{j \mid p_{1j} < p_{2j}\}$
 - jobs with operation on resource 1 not shorter than the operation on resource 2
 $V = \{j \mid p_{1j} \geq p_{2j}\}$
2. order jobs in **U** in **ascending order by p_{1j}**
3. order jobs in **V** in **descending order by p_{2j}**
4. add jobs from **V** after **U**, which fully determines the order of jobs
5. the schedule is constructed from left-to-right by allocating operations to earliest possible start times in the above order

Example: F2 | | C_{max}



Job-shop problem consists of:

- a set of **m resources**
- a set of **n jobs**
 - each job consists of a **sequence of operations** (different jobs may have different number and different order of operations)
 - resource and duration is given for each operation
 - when processing the job, a resource can be visited once or more times (re-circulation)
- **objective function**

Most job-shop scheduling problems belong among **NP-hard problems**. Job-shop scheduling problems are the most widely used classical scheduling problems.

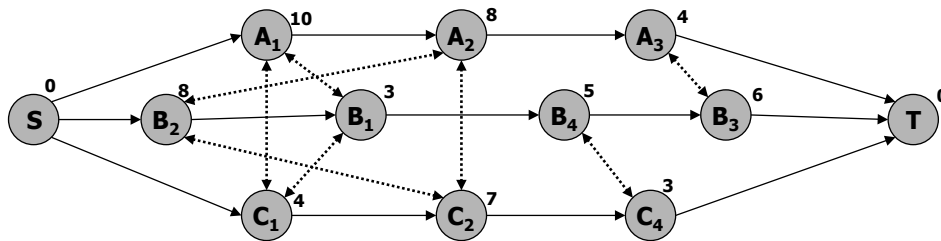
Job-shop problem with two resources and at most two operations in each job with the goal to minimize makespan.

- This is a tractable problem that can be reduced to the **flow-shop problems with two resources**:
 1. jobs are distributed into four sets:
 - I_1 jobs with a single operation running on M_1
 - I_2 jobs with a single operation running on M_2
 - $I_{1,2}$ jobs with the first operation running on M_1 and second operation on M_2
 - $I_{2,1}$ jobs with the first operation running on M_2 and second operation on M_1
 2. find an optimal sequences of jobs $R_{1,2}$ and $R_{2,1}$ for flow-shop problems with job sets $I_{1,2}$ and $I_{2,1}$
 3. the final schedule is obtained as follows:
 - allocate jobs $I_{1,2}$ on resource M_1 according to schedule $R_{1,2}$, then allocate jobs I_1 in any order, and finally allocate jobs $I_{2,1}$ according to schedule $R_{2,1}$
 - allocate jobs $I_{2,1}$ on resource M_2 according to schedule $R_{2,1}$, then allocate jobs I_2 in any order, and finally allocate jobs $I_{1,2}$ according to schedule $R_{1,2}$
- Is it an optimal schedule?**
- YES, because at least one of the resources will run without any interruption and schedules $R_{1,2}$ and $R_{2,1}$ are optimal

How to solve job-shop problems $J_m \parallel C_{max}$ in general?

The problem and its solution can be encoded as a **disjunctive graph**:

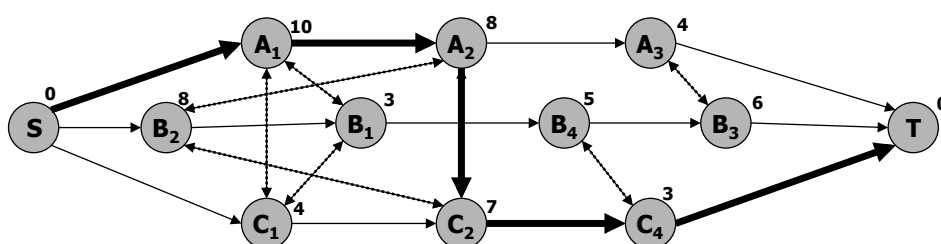
- **nodes** describe operations
 - each node is annotated by a weight equal to operation duration
 - two additional nodes S and T with weights 0
- **directed arcs**
 - **conjunctive arcs** describe precedence relations between operations in jobs
 - there are arcs from S to operations having no predecessor
 - there are arcs to T from operations without successors
 - **disjunctive arcs** connecting operations of different jobs that are allocated to the same resource (the arcs go in both directions)
 - disjunctive arcs form cliques in the graph
 - one arc from each disjunctive pair is to be selected to order operations so they do not overlap on the machine



Scheduling with disjunctive graphs

How to find a feasible schedule?

- **one arc is selected from each pair of corresponding disjunctive arcs** to remove cycles from the graph
 - a cycle in the schedule means infeasible schedule
- **makespan** is equal to the longest path in the graph from S to T (the path length equals the sum of weights of nodes in the path)
 - the goal is to select the arcs to **minimize the length of the longest path** (the critical path)



Job-shop $J_m \mid \mid C_{\max}$ is an **NP-complete problem!**

To solve the general job-shop problem we will exploit the idea of **non-linear programming**:

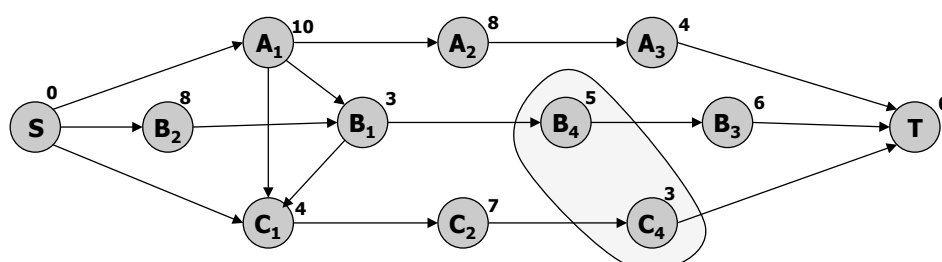
- All but one variables are kept fixed and optimisation is done via tuning the last variable (this is done for all the variables)
 - for job-shop it means optimizing each resource separately
- to increase chances for a good solution the resources are scheduled in the decreasing order of importance

Answers to two questions are critical for the algorithm:

- Which resource should be scheduled first (second, etc.)?
- How to find a schedule for a given resource (compatible with other resources)?

- Let us start with a partial schedule – a disjunctive graph with some selected disjunctive arcs.
- For each not-yet scheduled resource X , solve the problem $1 \mid r_j \mid L_{\max}$ with operations allocated to X , where:
 - duration p_i of operations is given by the original problem
 - release dates r_i are equal to the length of the longest path from S to operation I
 - due dates of operations are defined as follows

$$\delta_i = \text{length_of_longest_path}(S,T) - \text{length_of_longest_path}(i,T) + p_i$$
- Resource with the largest lateness L_{\max} is called a **bottleneck** and the schedule for this resource is added to the partial schedule.



	B ₄	C ₄
p	5	3
r	13	24
δ	21	27

The quality of obtained schedule can be further improved by **re-scheduling (re-ordering) already scheduled resources**.

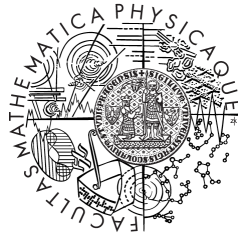
- let us take any already-scheduled resource X (except the last one, just scheduled) and remove its selected disjunctive arcs from the partial schedule
- solve again the problem $\mathbf{1} \mid r_j \mid L_{\max}$ with the operations of resource X
- add the new resource schedule for X to the partial schedule

Algorithm Shifting Bottleneck at a glance

1. start with a graph with conjunctive arcs only (the initial partial solution)
2. analyze non-yet scheduled resources
 - by building and solving problems $\mathbf{1} \mid r_j \mid L_{\max}$
3. select the bottleneck resource and add its schedule (selection of disjunctive arcs) to the graph
4. try to reorder the already scheduled resources
5. repeat the process until all resources are scheduled

Note:

- This is an **(greedy) heuristic algorithm** that does not guarantee optimality!



© 2014 Roman Barták

Department of Theoretical Computer Science and Mathematical Logic
bartak@ktiml.mff.cuni.cz