Advanced Algorithms **University of Bremen**
Summer 2025, Lecture 9 Faculty 3: Math/CS
Lecturer: Nicole Megow May 28, 2025

Scheduling problems

Scheduling problems are concerned with the efficient allocation of tasks to scarce resources over time. They arise in a wide range of settings where resourcessuch as machines, processors, or workersmust be assigned to jobs that come with varying constraints and objectives.

Gernally speaking, we are given a set of n jobs $J = \{1, 2, \dots, n\} =: [n]$ and m machines $\{1, \dots, m\} =: [m]$. Each job has a certain processing requirement. A *schedule* is an assignment of jobs to machines such that at any given time, a machine executes at most 1 job and each job is being processed by at most 1 machine. In *preemptive* scheduling, jobs can be interrupted at any time and resume processing at any time later at no additional cost. In *non-preemptive* scheduling, jobs must run till completion once they have started.

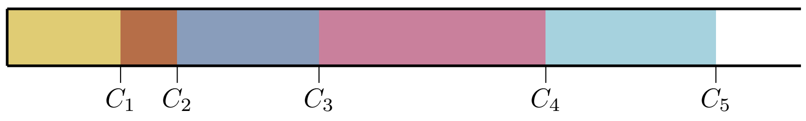
In the following, we study two standard scheduling objectives: minimizing the sum of all job completion times and minimizing the **makespan** (also called **load balancing**), which is the latest completion time over all jobs.

1 Minimizing the total completion time

1.1 Single machine

Given a set of jobs J, where each job $j \in J$ has a processing time $p_j \geq 0$ and a weight $w_j \geq 0$, find a permutation of all jobs such that the total sum of completion times, $\sum_{j \in J} w_j C_j$, is minimized. Here, C_j denotes the completion time of job j, that is, the earliest point in time when the job has been processed for p_j units of time. This problem is denoted as $1 \mid\mid \sum w_j C_j$.

We visualize schedules typically using a *Gantt chart*, where time runs horizontally and each job is represented by (possibly multiple) rectangle(s) on a timeline.

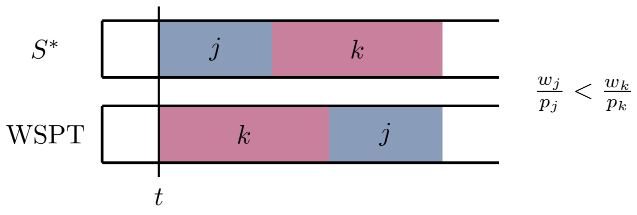


Algorithm: Sequence jobs in non-increasing order of their ratios $\frac{w_j}{p_j}$. This algorithm is called *Smith's rule* or *Weighted Shortest Processing Time (WSPT) rule.*

We show in the following that this algorithm computes an optimal schedule for the single-machine scheduling problem. In fact, we show the stronger result that any optimal schedule must follow Smith's rule.

Theorem 1 (Smith 1956)**.** *In an optimal schedule for $1 \mid\mid \sum w_j C_j$, jobs must be scheduled in WSPT order.*

Proof. We prove this by a simple interchange argument. Let S^* be an optimal non-WSPT schedule. Then there is a pair of adjacent jobs j, k in S^* that is not in WSPT order. That means, j is scheduled before k whereas $\frac{w_j}{p_j} < \frac{w_k}{p_k}$. Consider the earliest such pair in S^* with job j starting at time t.



Now, consider the schedule that is obtained from S^* by swapping j and k. The change in the objective value is

$$\begin{aligned}
&-w_jC_j - w_kC_k + w_k(t+p_k) + w_j(t+p_k+p_j)\\
&\qquad\qquad = -w_j(t+p_j) - w_k(t+p_j+p_k) + w_k(t+p_k) + w_j(t+p_k+p_j)\\
&\qquad\qquad = -w_kp_j + w_jp_k\\
&\qquad\qquad < 0.
\end{aligned}$$

The first equality uses the facts that $C_j = t + p_j$ and $C_k = t + p_j + p_k$. The final inequality is due to the assumption on the non-WSPT order of the j, k in S^*.

From this negative change in the objective value after swapping the jobs follows that the schedule S^* cannot have been an optimal one which is a contradiction. Hence, all jobs must be in WSPT order in an optimal schedule. □

1.2 Unrelated parallel machines

Consider the unrelated machine scheduling problem: given a set of jobs J, where each job $j \in J$ has a processing time of $p_{ij} \geq 0$ when assigned to machine i, the task is to find an assignment and schedule for the jobs that minimizes the total sum of completion times, $\sum_{j\in J} w_jC_j$. This problem is denoted as $R \,||\, \sum C_j$.

Theorem 2 (Horn 1973)**.** *The scheduling problem $R \,||\, \sum C_j$ can be solved in polynomimal time.*

Proof. We model this scheduling problem as a minimum-cost matching problem in a bipartite graph (aka assignment problem), which can be solved in polynomial time.

Consider an arbitrary machine i and an assignment of jobs $j \to i$. Then the optimal schedule and its objective value on i can be precisely characterized. As shown earlier, the optimal schedule on each machine orders jobs according to the WSPT rule. Let $\pi_i(k)$ denote the job that is scheduled at the k-th last position on machine i. Obviously, this is the k-th longest job that has been assigned to i. Such a job contributes $k \cdot p_{i\pi_i(k)}$ to the total completion time on machine i. Hence, the total completion time of jobs assigned to machine i can be expressed as

$$\sum_{j:j\to i} C_j = \sum_{k=1}^{n} k \cdot p_{i\pi_i(k)}\,,$$

when defining $\pi_i(k) := 0$ for positions k which are not filled by any job, i.e., for any k that exceeds the number of jobs assigned to i.

Now we construct a bipartite edge-weighted graph $(A \cup B, E)$ with the property that an assignment of jobs to machines and scheduling positions with total completion time z corresponds to a matching of the same cost z. To balance the graph, we add dummy jobs with $p_{ij} = 0$ such that the number of jobs is $n \cdot m$. (We aim for a perfect matching.)

Let $A = J$, that is, each job $j \in J$ corresponds to a vertex in A, and each machine-position pair (i, k), for $i \in [m]$ and $k \in [n]$ corresponds to a vertex in B. We add edges between any j and (i, k) with weight $k \cdot \pi_i(j)$. A perfect matching of minimum cost corresponds to a schedule of minimum total completion time. □

2 Makespan minimization

We consider now a another well-studied objective function, the *makespan* $C_{max} := max_{j\in[n]} C_j$. In the preemptive makespan minimization problem on identical parallel machines, we are given a set of jobs J, each with processing time $p_j \geq 0$, and m identical parallel machines to execute these jobs, possibly using preemption, such that the makespan is minimized. The problem is denoted as $P \mid \text{pmtn} \mid C_{\max}$

We first observe two lower bounds on the optimal makespan $C^*_{\max}$. Since a job cannot run simultaneously on multiple machines, we have

$$C^*_{\max} \geq \max_{j\in J} p_j.$$

Further, an optimal schedule must process all jobs with their processing requirements on m machines and may, at best, achieve a perfect load balancing. Hence, it has a makespan

$$C^*_{\max} \geq \frac{1}{m} \cdot \sum_{j\in J} p_j.$$

We will see an algorithm which achieves a makespan matching the maximum of the two lower bounds and is, thus, optimal.

Algorithm: McNaughtons Wrap Around Rule schedules jobs on m identical machines by placing them sequentially filling machine by machine up to time $q := \max\left\{\max_{j\in J} p_j, \frac{1}{m} \cdot \sum_{j\in J} p_j\right\}$, wrapping to a new machine whenever a job would overflow. That means, if a job does not fit entirely on the current machine, it is preempted and the remainder is assigned to the beginning of the next machine.

Theorem 3 (McNaughton 1959). *The Wrap Around Rule solves the problem $P \mid \text{pmtn} \mid C_{\max}$ optimally in polynomial time.*

Proof. The algorithm finishes all processing volume by the makespan q, since $\sum_{j\in J} p_j \leq m \cdot q$. Further, it obtains a feasible schedule since no job runs simultaneously on multiple machines. To see that consider the critical situation when a job j is preempted and "wrapped around"; since $q \geq p_j$, the two execution intervals on the different machines cannot overlap. Since the algorithm achieves a makespan q, which is a lower bound on the minimum makespan, it is optimal. □

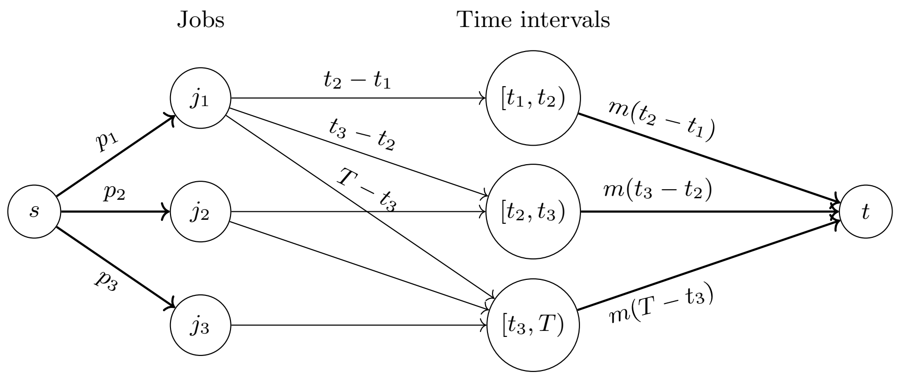
In a slightly more general case, jobs may have different release dates $r_j \geq 0$ that indicate the earliest possible starting time of a job. This problem can also be solved in polynomial time.

Theorem 4 (Horn 1974). *The problem $P \mid r_j, \text{pmtn} \mid C_{\max}$ can be solved optimally in polynomial time.*

Proof. We reduce the problem to a network flow problem and binary search to assign processing volume to time slots and then use McNaughton's Wrap Around Rule to solve the actual problem.

Suppose we new the optimal makespan T. Consider all distinct release times $t_1, t_2, \ldots, t_k$ and T, and notice that in the time interval between any adjacent such time points, $[t_i, t_{i+1})$, the set of available jobs does not change. We want to assign processing volume of jobs to such time intervals such that (i) the total processing capacity bound on m machines, $m(t_{i+1}, t_i)$, is not violated and (ii) no job is assigned with more volume than it could sequentially finish, that is, $t_{i+1} - t_i$.

Construct an s-t-network over the bipartite graph $(A \cup B, E)$ with $A = J$ and B with one node for each of the relevant time intervals $[t_1, t_2), [t_2, t_3), \ldots, [t_k, T)$. For every job j, there is an edge from s to j with capacity p_j, and for every time interval $[t_i, t_{i+1})$, there is an edge to t with capacity $m(t_{i+1} - t_i))$. Further, there is an edge from a node j to every feasible time interval node $[t_i, t_{i+1})$ with $t_i \geq r_j$ with edge weight $t_{i+1} - t_i$.



A feasible flow corresponds to an allocation of processing volume to time intervals with the desired properties (i) and (ii). Given these properties, we can consider any time interval separately and find a feasible schedule using McNaughton's Wrap Around Rule. This is a feasible schedule and it meets the lower bound on the optimal makespan.

To find the optimal makespan T, we use standard binary search in the interval between 1 and $2 \cdot \max_{j \in J} r_j + \frac{1}{m} \sum_{j \in J} p_j$. $\square$