**Hardness of Open-shop scheduling**

Theorem: The problem O3|| $C_{max}$ (Minimize makespan of an open-shop schedule on 3 machines) is NP-hard.

Proof: Reduction from *Partition*. The input for *Partition* is a set S of n integers $a_1,\ldots,a_n$ whose total sum is 2B. The problem is to find a subset of these integers whose total sum is exactly B. Given an instance S for *Partition* build the following instance for O3|| $C_{max}$ :
There are 3n+1 jobs, for the first n jobs (that is, for j=1,…,n) $p_{j,1}=a_j$, $p_{j,2}=p_{j,3}=0$. For the next n jobs (that is, for j=n+1,…,2n) $p_{j,2}=a_j$, $p_{j,1}=p_{j,3}=0$, for the next n jobs (that is, for j=2n+1,…,3n) $p_{j,3}=a_j$, $p_{j,1}=p_{j,2}=0$, and for the last job $p_{3n+1,1}= p_{3n+1,2} = p_{3n+1,3} =B$.

Claim: $C_{max}$ = 3B if and only if the set S has a partition.

The proof has two parts – one for each direction of the 'if and only if'
1. Assume there is a schedule with $C_{max}$ = 3B, it must be that the last job $J_{3n+1}$ is processed on the machines one after the other with no idle. Assume w.l.o.g that $M_2$ is the second machine to process this job. Therefore, $J_{3n+1}$ is processed on $M_2$ during the time interval [B,2B]. As a result the processing of the n jobs j=n+1,…,2n with $p_{j,2}=a_j$, $p_{j,1}=p_{j,3}=0$ on $M_2$ splits between the intervals [0,B] and [2B,3B], inducing a partition.

2. Assume the set S has a partition into two sets U and W each with total size B. A valid schedule with $C_{max}$ = 3B is the following: On $M_1$: $J_{3n+1}$ followed by the first n jobs. On $M_2$: the jobs of the second set of jobs (j=n+1,…,2n) originated from U, then $J_{3n+1}$ , then the jobs of the second set of jobs originated from W. On $M_3$: the jobs 2n+1…,3n and then $J_{3n+1}$. Since a partition exists, the second machine is processed with no idle and the jobs from U and W exactly fill the B-segments before and after the processing of job $J_{3n+1}$.

**<u>Facility Location</u>**

Theorem: The network covering problem is NP-hard.

Proof: Reduction from Dominating Set.
A dominating set in an undirected graph is a collection S of vertices with the property that every vertex v in G is either in S, or there is an edge between a vertex in S and v.
Example: o—x—o—o—x

Given a graph G and an integer k, it is NP-hard to determine whether G has a DS of size k.
Given an instance of DS we build an instance of the covering problem: Same network, same k, all edges have length 1, all nodes have covering demand 1.
It is easy to see that a DS is exactly a cover.

**<u>Covering a tree</u>**

**<u>Theorem:</u>** The algorithm (slides 15-23) is optimal, that is, it uses the minimal possible number of centers.
**<u>Proof (draft):</u>** Let k be the number of centers determined by the algorithm. We show that there exists a set H of k nodes such that for every two nodes v1,v2 in H it holds that d(v1,v2)> s_v1+s_v2. (*)
Given that such a set exists, at least k centers are required, since otherwise, by the pigeonhole principle, there exists a center that covers more than one node in the set, which is impossible.
The set H is defined as follows: For every center c set by the algorithm, add to H the node for which d(v,c)=s_v and is the first node processed by the algorithm among the nodes covered by c.

Proof of the property (*): Consider a pair of vertices v1,v2 in H. W.l.o.g, the center covering v1 was set first. Since G is a tree, there is only one path between v1 and v2. By the way the algorithm proceeds, the center covering v1 is located along this path (this requires more arguments from graph theory that we skip). Since it does not cover v2, it must be that that d(v1,v2)> s_v1+s_v2.