§4 Scheduling with scarce resources: Introduction and Complexity

The model

$V = \{1, \dots, n\}$ set of jobs

$G =$ graph (V, E) of precedence constraints

$\mathcal{F} = \{F_1, \dots, F_k\}$ system of forbidden sets (resource constraints)

$K =$ cost function

$x = (x_1, \dots, x_n)$ / $X = (X_1, \dots, X_n)$ deterministic or stochastic processing times

deterministic case:

find a schedule S that respects $G, x, \mathcal{F}$ (feasible schedule)

such $K(S, x) \overset{\text{Def}}{=} K(C_1, \dots, C_n)$ is minimum or "good"

$\uparrow$

$C_j = S_j + x_j$

m-machine problems

m identical machines/processors

every job needs one, every machine can process only one job at a time

$\Rightarrow \mathcal{F} = \{ F \subseteq V \mid |F| = m+1, F \text{ is an antichain of } G \}$

4.1 THEOREM: Let $m = 1$ and $E_G = \emptyset$ (no precedence constraints). Then

(1) idle time does not pay for any K

└ processor is not busy, but there is a job that could be processed

(2) For $K = \sum_j w_j C_j$ Smith's rule constructs an optimal schedule

- sort the jobs s.t. $\frac{x_{i_1}}{w_{i_1}} \leq \frac{x_{i_2}}{w_{i_2}} \leq \dots$
- schedule the jobs in that order

(proof is a simple exchange argument)

(3) For $K = L_{max}$, Jackson's rule constructs an optimal schedule

- sort jobs s.t. $d_{i_1} \leq d_{i_2} \leq \dots \leq d_{i_n}$ (d_j = due date of job j)
- schedule them in this order

(4) Problem $\kappa = L_{max}$, but jobs have release dates r_j (i.e. $S_j \geq r_j$) is NP-hard

(5) $\sum_j w_j T_j$ is NP-hard

Proof: (1) obvious

(2) exchange argument for 2 adjacent jobs

(3) homework

(4)(5) without proof □

Theorem shows: small changes turn a problem from polynomially solvable to NP-hard

Typical for scheduling

1975 - 1985 many scheduling problems have been classified (about 8000) about 80% of them are NP-hard

(Lawler, Lenstra, Rinooy Kan)

webpage: http://www.informatik.uni-osnabrueck.de/knust/class

Other example

m-machine problem with $m = 2$, arbitrary precedence constraints G

$x_j = 1$ for all j $\kappa = C_{max}$

↑ models jobs in a parallel processor environment

$x_j = 1$ ≙ time slots for processing

every long job is subdivided into a chain of pieces with unit length

Case $m = 2$ is polynomially solvable

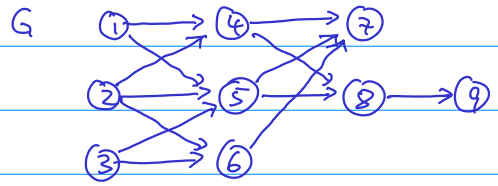
4.2 THEOREM: $(C^G_{max})^{OPT}(\mathbb{I})$ = maximum size of a matching in $Incomp(G)$ + # of unmatched jobs, and one can construct

the schedule from the matching in poly time [Fujii et al '67, '71]

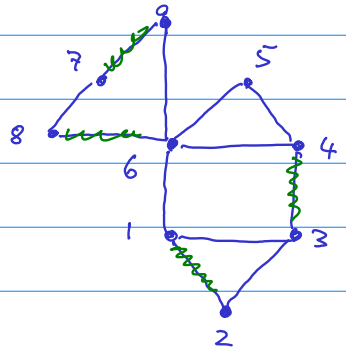
$I_{comp}(G)$ has vertex set V (the same as G)

$\{i,j\}$ is an edge of $I_{comp}(G) :\Leftrightarrow i \neq j$ and $i \parallel_G j$

Example:



$I_{comp}(G)$

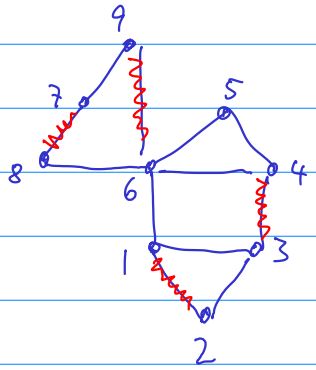


an optimal schedule:

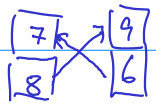
1	3	5	6	7
2	4	///	8	9

see that every time slot with 2 jobs in parallel defines a matched edge in $I_{comp}(G)$ (the green matching in the example)

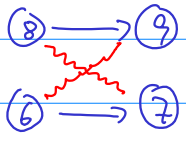
Now: from the matching to a schedule



⇐ this is a max matching, but it cannot be turned in to a schedule (s.t. matched jobs are scheduled in the same time slot)

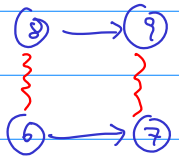


contradiction, there is no ordering of the time slots respecting the precedence constraints!



⇐ call this a crossing (matching edges cross w.r.t. precedence constraints)

⇓ "uncrossing" of the matching

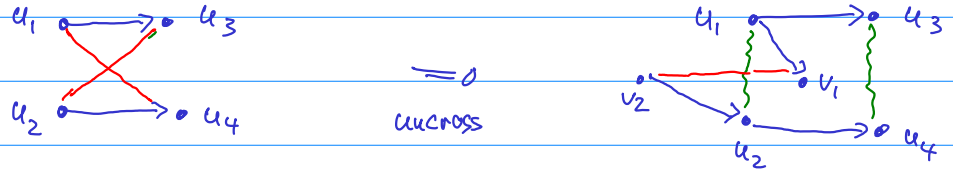


Claim: For every matching M, there is a non-crossing matching of the same size that can be obtained by uncrossing crossings in any order

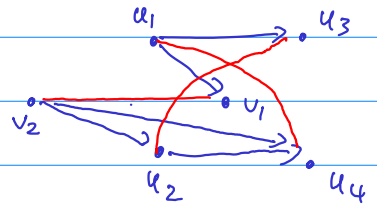
So every uncrossing reduces the number of crossings

this is what we will show

suppose this is not the case $\Rightarrow$ uncrossing creates a new crossing with u_1, v_1, v_2, u_2

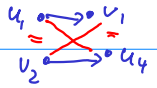


$\Rightarrow$ $u_1 < v_1$, $v_2 < u_2$ $(v_1, v_2) \in M$



before uncrossing

$\Rightarrow$ (v_1, v_2) crosses with (u_1, u_4) before the uncrossing



$\Rightarrow$ that crossing no longer exists after uncrossing

$\Rightarrow$ for every new crossing, another old crossing involving the same edges is also uncrossed

$\Rightarrow$ number of crossing drops $\square$

Now assume that M is non-crossing

want to construct a schedule from M

order matching edges $\begin{matrix} a & & c \\ b & < & d \end{matrix}$ $:\Leftrightarrow$ $a < c$ or $a < d$ or $b < c$ or $b < d$

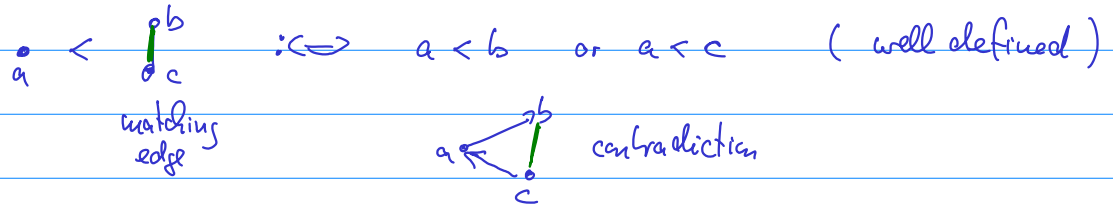
$a < d$ is $i_4 G$

ordering of matching edges defines a unique ordering because there are no crossings

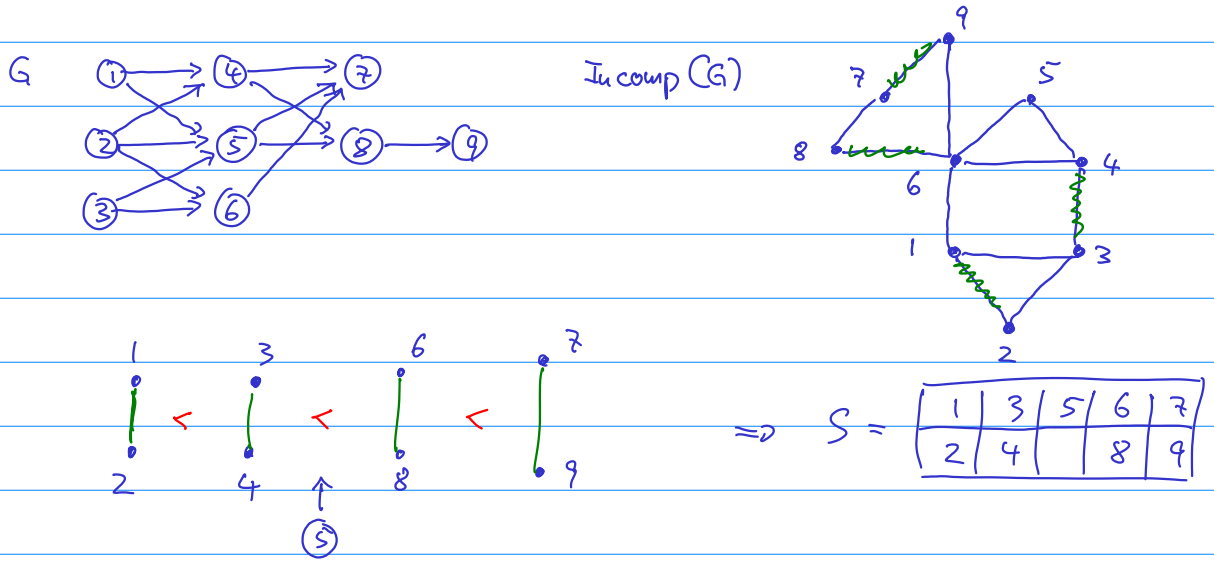
take a linear extension of that ordering of matching edges

linear extension is a linear order preserving the ordering relation between matching edges

insert unmatched jobs into this linear order by observing



the linear order of matching edges and unmatched jobs defines a schedule



□

m - machine problem is open for $m = 3$ (complexity not known)

↑ famous 3- machine problem

if m is part of the input, then

4.3 THEOREM: The following problem is NP- complete

m - machine problem

Given: G, m, time t, $x = 1$

Question: is there a feasible schedule with makespan $\leq t$

Proof: CLIQUE reducible to m-machine scheduling

Given: Graph $G = (V, E)$, number $k \in \mathbb{N}$

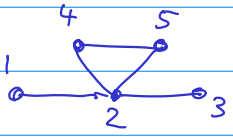
Question: does G have a clique of size k

Construct from this instance an instance of m-machine problem

Graph G defines vertex jobs and edge jobs

every vertex job is before the edge jobs containing that vertex

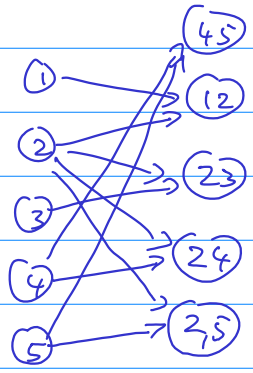
Example



G

$k = 3$

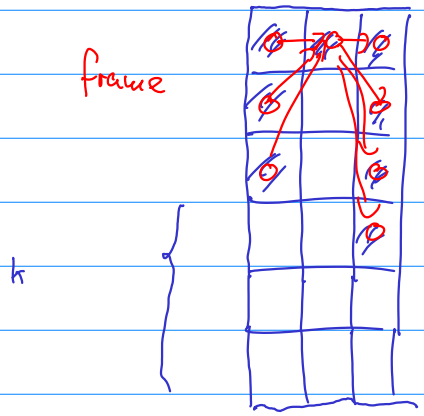
$\Rightarrow$



real jobs

in addition there are dummy jobs that define a "frame" in which the "real" jobs must be scheduled, together with the bound $t = 3$ for the makespan

$t = 3 \Rightarrow$ 3 time slots



$n - k + \binom{k}{2}$ remaining job

Claim: Graph has a clique of size k $\Leftrightarrow$ $\exists$ feasible schedule of length 3

"$\Rightarrow$" schedule k jobs from the clique in time slot 1

schedule edge jobs from the clique + remaining vertex jobs

remaining jobs in time slot 3

"$\Leftarrow$" graph has no clique of size k

$\Rightarrow$ one slot remains empty at time 2

$\Rightarrow$ schedule has length > 3 $\square$