DM204 – Spring 2011

Scheduling, Timetabling and Routing

Lecture 5

Flow Shop and Job Shop

Marco Chiarandini

Department of Mathematics & Computer Science
University of Southern Denmark

Outline

Course Overview

- ✔ Scheduling
  - ✔ Classification
  - ✔ Complexity issues
  - ✔ Single Machine
  - ✔ Parallel Machine
    - Flow Shop and Job Shop
    - Resource Constrained Project Scheduling Model
- Timetabling
  - Sport Timetabling
  - Reservations and Education
  - University Timetabling
  - Crew Scheduling
  - Public Transports
- Vechicle Routing
  - Capacited Models
  - Time Windows models
  - Rich Models

Outline

Flow Shop

General Shop Scheduling:

- $J = \{1, \ldots, N\}$ set of jobs; $M = \{1, 2, \ldots, m\}$ set of machines
- $J_j = \{O_{ij} \mid i = 1, \ldots, n_j\}$ set of operations for each job
- p_{ij} processing times of operations O_{ij}
- $\mu_{ij} \subseteq M$ machine eligibilities for each operation
- precedence constraints among the operations
- one job processed per machine at a time,
  one machine processing each job at a time
- C_j completion time of job j

➡ Find feasible schedule that minimize some regular function of C_j

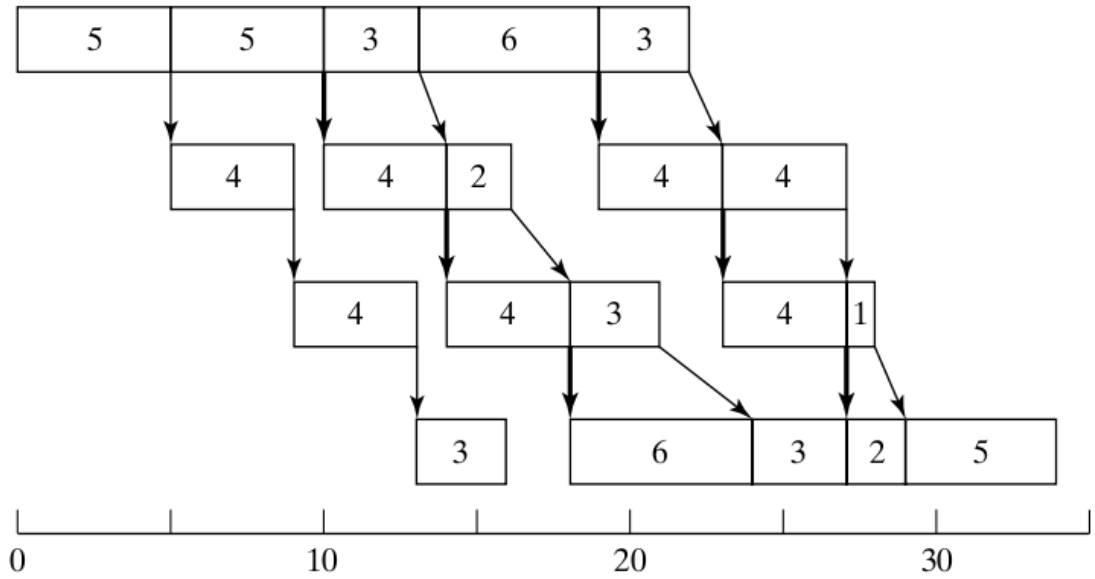
Flow Shop Scheduling:

- $\mu_{ij} = i$, $i = 1, 2, \ldots, m$
- precedence constraints: $O_{ij} \rightarrow O_{i+1,j}$, $i = 1, 2, \ldots, n$ for all jobs

Example

$jobs$	j_1	j_2	j_3	j_4	j_5
p_{1,j_k}	5	5	3	6	3
p_{2,j_k}	4	4	2	4	4
p_{3,j_k}	4	4	3	4	1
p_{4,j_k}	3	6	3	2	5

Gantt chart



schedule representation

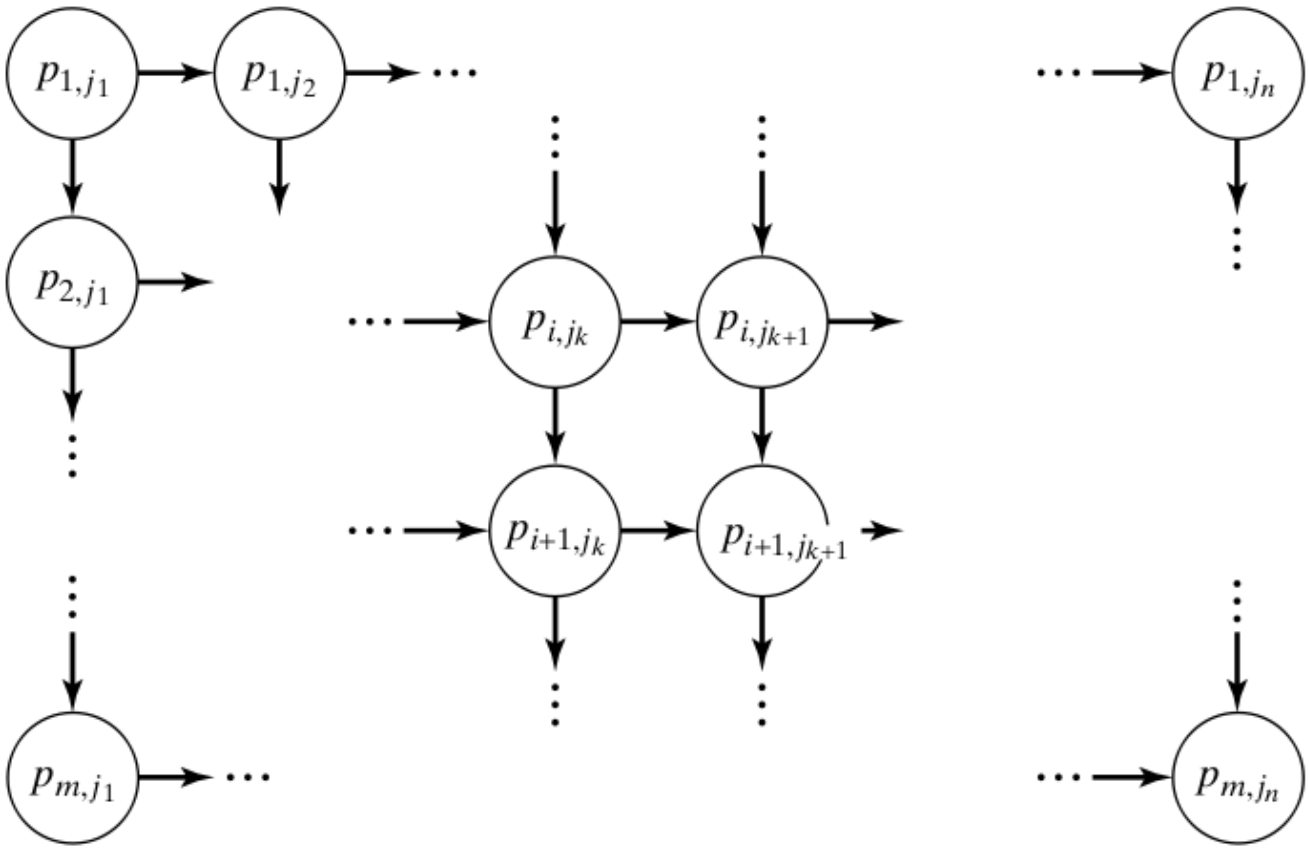
$\pi_1, \pi_2, \pi_3, \pi_4$:

$\pi_1 : O_{11}, O_{12}, O_{13}, O_{14}$
$\pi_2 : O_{21}, O_{22}, O_{23}, O_{24}$
$\pi_3 : O_{31}, O_{32}, O_{33}, O_{34}$
$\pi_4 : O_{41}, O_{42}, O_{43}, O_{44}$

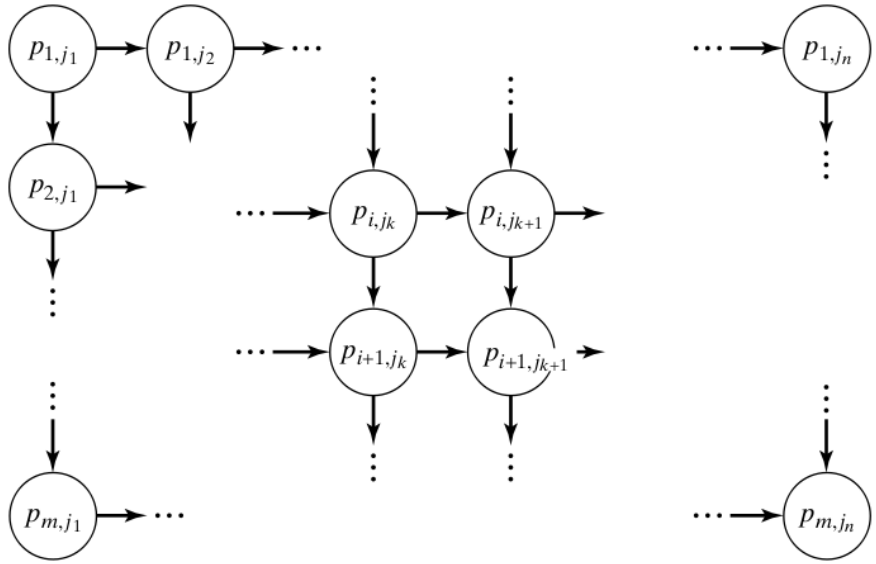
- we assume unlimited buffer
- if same job sequence on each machine ➡ permutation flow shop

Directed Graph Representation

Given a sequence: operation-on-node network,
jobs on columns, and machines on rows



Directed Graph Representation



Recursion for C_{max}

$$C_{i,\pi(1)} = \sum_{l=1}^{i} p_{l,\pi(1)}$$

$$C_{1,\pi(j)} = \sum_{l=1}^{j} p_{l,\pi(l)}$$

$$C_{i,\pi(j)} = \max\{C_{i-1,\pi(j)}, C_{i,\pi(j-1)}\} + p_{i,\pi(j)}$$

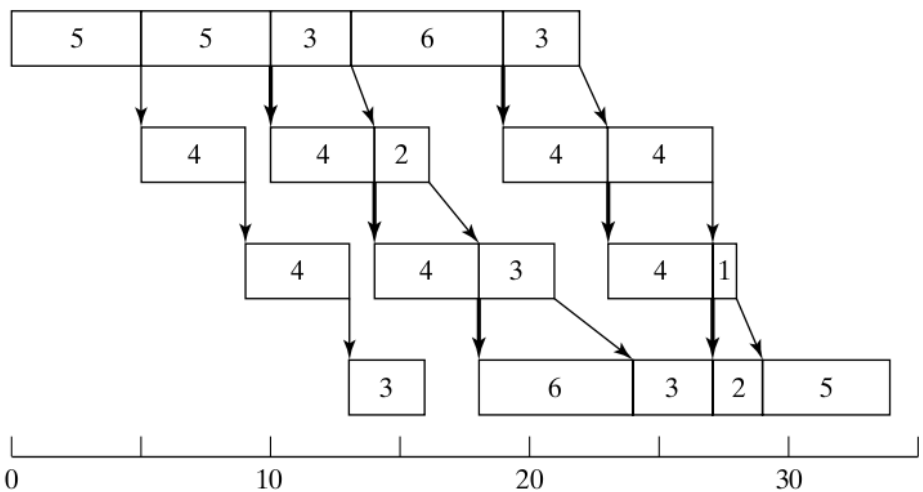
Computation cost?

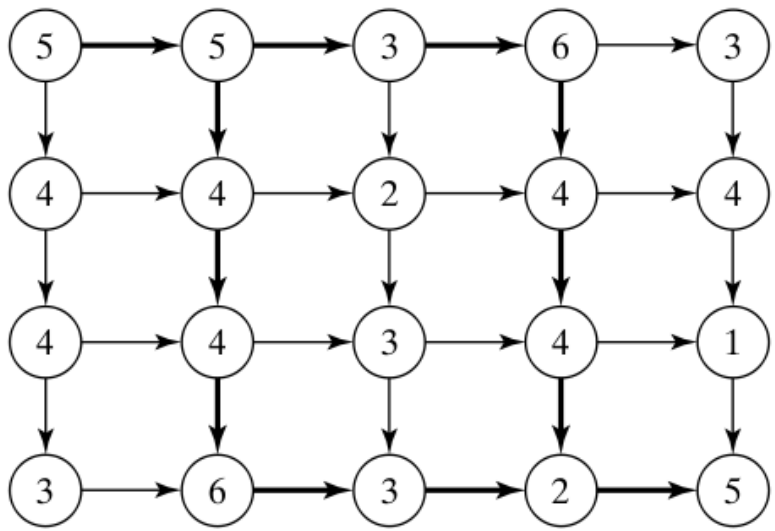
Example

$jobs$	j_1	j_2	j_3	j_4	j_5
p_{1,j_k}	5	5	3	6	3
p_{2,j_k}	4	4	2	4	4
p_{3,j_k}	4	4	3	4	1
p_{4,j_k}	3	6	3	2	5

$C_{max} = 34$

corresponds to longest path



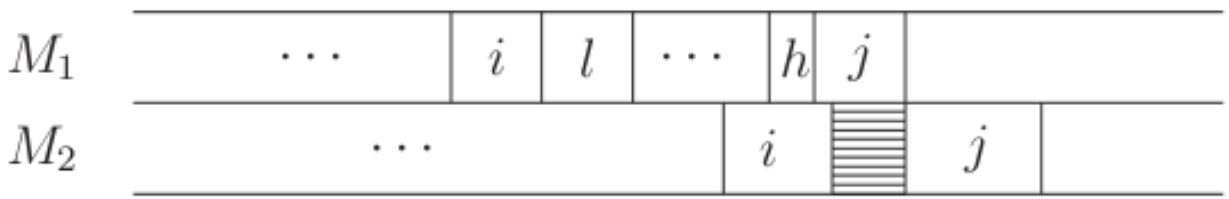


$Fm||C_{max}$

Theorem

There always exist an optimum sequence without change in the first two and last two machines.

Proof: By contradiction.



Corollary

$F2||C_{max}$ and $F3||C_{max}$ are permutation flow shop

Note: $F3||C_{max}$ is strongly NP-hard

$F2||C_{max}$

Intuition: give something short to process to 1 such that 2 becomes operative and give something long to process to 2 such that its buffer has time to fill.

Construct a sequence $T : T(1), \ldots, T(n)$ to process in the same order on both machines by concatenating two sequences:
a left sequence $L : L(1), \ldots, L(t)$, and a right sequence $R : R(t+1), \ldots, R(n)$, that is, $T = L \circ R$

[Selmer Johnson, 1954, Naval Research Logistic Quarterly]

Let J be the set of jobs to process

Let $T, L, R = \emptyset$

Step 1 Find (i^*, j^*) such that $p_{i^*,j^*} = \min\{p_{ij} \mid i \in 1, 2, j \in J\}$

Step 2 If $i^* = 1$ then $L = L \circ \{i^*\}$
else if $i^* = 2$ then $R = \{i^*\} \circ R$

Step 3 $J := J \setminus \{j^*\}$

Step 4 If $J \neq \emptyset$ go to Step 1 else $T = L \circ R$

Theorem

The sequence $T : T(1), , \ldots, T(n)$ is optimal.

Proof

- Assume at one iteration of the algorithm that job k has the min processing time on machine 1. Show that in this case job k has to go first on machine 1 than any other job selected later.
- By contradiction, show that if in a schedule S a job j precedes k on machine 1 and has larger processing time on 1, then S is a worse schedule than S'.
  There are three cases to consider.
- Iterate the proof for all jobs in L.
- Prove symmetrically for all jobs in R.

Construction Heuristics (1)

Fm | prmu | C_{max}

Slope heuristic

- schedule in decreasing order of $A_j = -\sum_{i=1}^{m}(m-(2i-1))p_{ij}$

Campbell, Dudek and Smith's heuristic (1970)

extension of Johnson's rule to when permutation is not dominant

- recursively create 2 machines 1 and $m-1$

$$p'_{ij} = \sum_{k=1}^{i} p_{kj} \qquad p''_{ij} = \sum_{k=m-i+1}^{m} p_{kj}$$

  and use Johnson's rule
- repeat for all $m-1$ possible pairings
- return the best for the overall m machine problem

Construction Heuristics (2)

$Fm \mid prmu \mid C_{max}$

Nawasz, Enscore, Ham's heuristic (1983)

Step 1: sort in decreasing order of $\sum_{i=1}^{m} p_{ij}$

Step 2: schedule the first 2 jobs at best

Step 3: insert all others in best position

Implementation in $O(n^2 m)$

[Framinan, Gupta, Leisten (2004)] examined 177 different arrangements of jobs in Step 1 and concluded that the NEH arrangement is the best one for C_{max}.

Iterated Greedy

$Fm \mid prmu \mid C_{max}$

Iterated Greedy [Ruiz, Stützle, 2007]

Destruction: remove d jobs at random

Construction: reinsert them with NEH heuristic in the order of removal

Local Search: insertion neighborhood
(first improvement, whole evaluation $O(n^2 m)$)

Acceptance Criterion: random walk, best, SA-like

Performance on up to $n = 500 \times m = 20$:

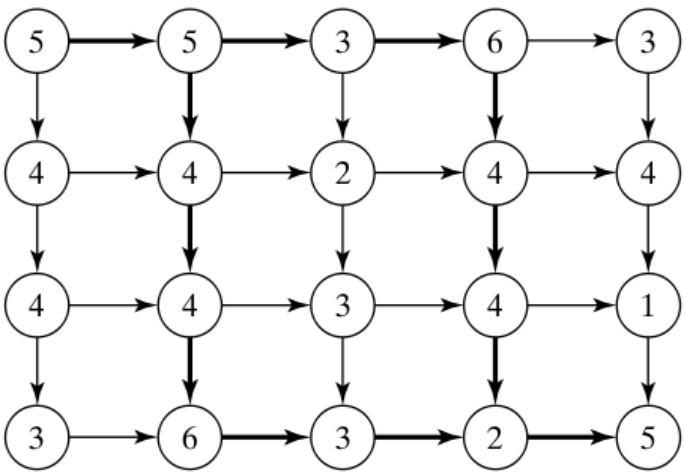
- NEH average gap 3.35% in less than 1 sec.
- IG average gap 0.44% in about 360 sec.

Efficient local search for $Fm|prmu|C_{max}$

Tabu search (TS) with insert neighborhood.

TS uses best strategy. ➡ need to search efficiently!

Neighborhood pruning [Novicki, Smutnicki, 1994, Grabowski, Wodecki, 2004]



A sequence $t = (t_1, t_2, \ldots, t_{m-1})$ defines a path in π:

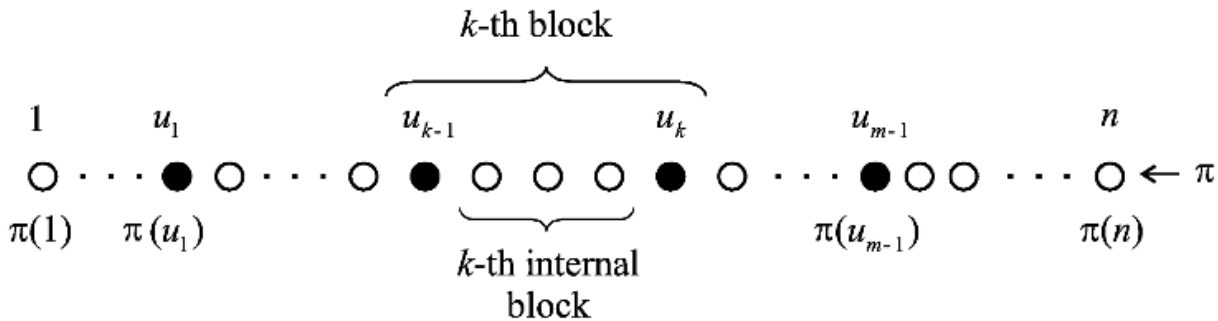
$$C(\pi, t) = \sum_{j=1}^{t_1} p_{\pi(j)1} + \sum_{j=t_1}^{t_2} p_{\pi(j)2} + \cdots + \sum_{j=t_{m-1}}^{n} p_{\pi(j)m}.$$

C_{max} expression through critical path:

$$C_{\max}(\pi) = \max_{1 \leqslant t_1 \leqslant t_2 \leqslant \cdots \leqslant t_{m-1} \leqslant n} \left(\sum_{j=1}^{t_1} p_{\pi(j)1} + \sum_{j=t_1}^{t_2} p_{\pi(j)2} + \cdots + \sum_{j=t_{m-1}}^{n} p_{\pi(j)m} \right)$$

critical path: $\vec{u} = (u_1, u_2, \ldots, u_m) : C_{max}(\pi) = C(\pi, u)$

Block B_k and Internal Block B_k^{Int}



Theorem (Werner, 1992)

Let $\pi, \pi' \in \Pi$, if π' has been obtained from π by a job insert so that $C_{max}(\pi') < C_{max}(\pi)$ then in π':

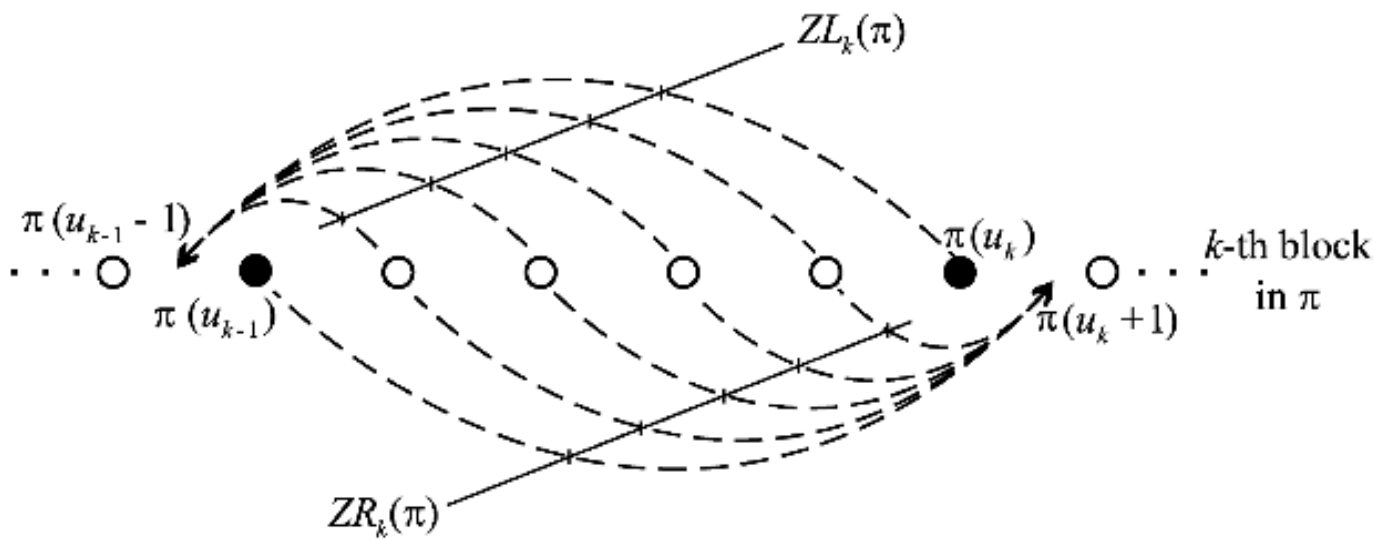
a) *at least one job $j \in B_k$ precedes job $\pi(u_{k-1})$, $k = 1, \ldots, m$, or*

b) *at least one job $j \in B_k$ succeeds job $\pi(u_k)$, $k = 1, \ldots, m$*

Corollary (Elimination Criterion)

If π' is obtained by π by an "internal block insertion" then $C_{max}(\pi') \geq C_{max}(\pi)$.

Hence we can restrict the search to where the good moves can be:



Further speedup: Use of lower bounds in delta evaluations:
Let δ^r_{x,u_k} indicate insertion of x after u_k (move of type $ZR_k(\pi)$)

$$\Delta(\delta^r_{x,u_k}) = \begin{cases} p_{\pi(x),k+1} - p_{\pi(u_k),k+1} & x \neq u_{k-1} \\ p_{\pi(x),k+1} - p_{\pi(u_k),k+1} + p_{\pi(u_{k-1}+1),k-1} - p_{\pi(x),k-1} & x = u_{k-1} \end{cases}$$

That is, add and remove from the adjacent blocks
It can be shown that:

$$C_{max}(\delta^r_{x,u_k}(\pi)) \geq C_{max}(\pi) + \Delta(\delta^r_{x,u_k})$$

Theorem (Nowicki and Smutnicki, 1996, EJOR)

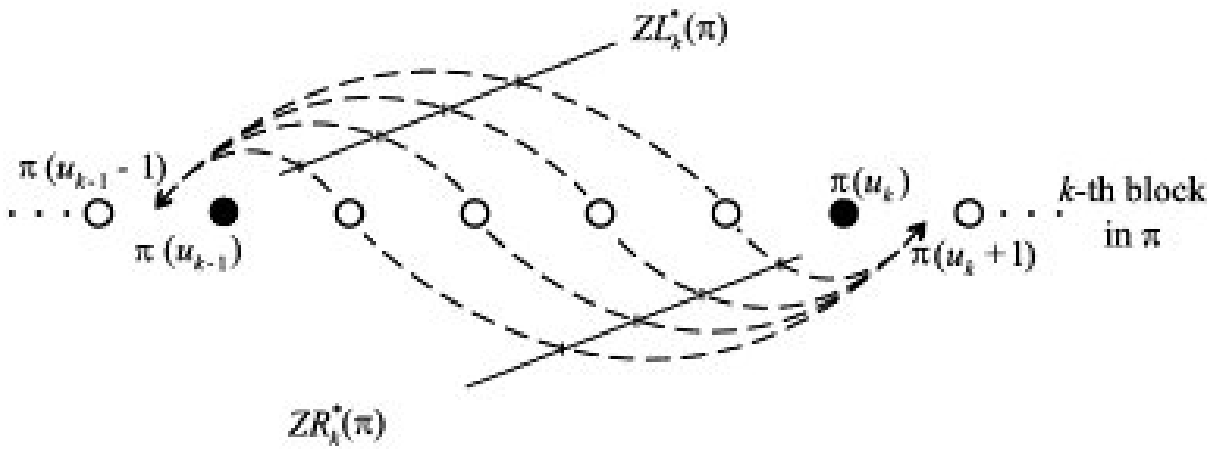
The neighborhood thus defined is connected.

Metaheuristic details:

Prohibition criterion:
an insertion δ_{x,u_k} is tabu if it restores the relative order of $\pi(x)$ and $\pi(x+1)$.

Tabu length: $TL = 6 + \left[\frac{n}{10m}\right]$

Perturbation



- perform all *inserts* among all the blocks that have $\Delta < 0$
- activated after `MaxIdleIter` idle iterations

Tabu Search: the final algorithm:

Initialization : $\pi = \pi_0$, $C^* = C_{max}(\pi)$, set iteration counter to zero.

Searching : Create UR_k and UL_k (set of non tabu moves)

Selection : Find the best move according to lower bound Δ.
Apply move. Compute true $C_{max}(\delta(\pi))$.
If improving compare with C^* and in case update.
Else increase number of idle iterations.

Perturbation : Apply perturbation if `MaxIdleIter` done.

Stop criterion : Exit if `MaxIter` iterations are done.

Outline

Job Shop

General Shop Scheduling:

- $J = \{1, \ldots, N\}$ set of jobs; $M = \{1, 2, \ldots, m\}$ set of machines
- $J_j = \{O_{ij} \mid i = 1, \ldots, n_j\}$ set of operations for each job
- p_{ij} processing times of operations O_{ij}
- $\mu_{ij} \subseteq M$ machine eligibilities for each operation
- precedence constraints among the operations
- one job processed per machine at a time,
  one machine processing each job at a time
- C_j completion time of job j

➨ Find feasible schedule that minimize some regular function of C_j

Job shop

- $\mu_{ij} = l, l = 1, \ldots, n_j$ and $\mu_{ij} \neq \mu_{i+1,j}$ (one machine per operation)
- $O_{1j} \to O_{2j} \to \ldots \to O_{n_j,j}$ precedences (without loss of generality)
- without repetition and with unlimited buffers

Task:

- Find a schedule $S = (S_{ij})$, indicating the starting times of O_{ij}, such that:

  it is feasible, that is,

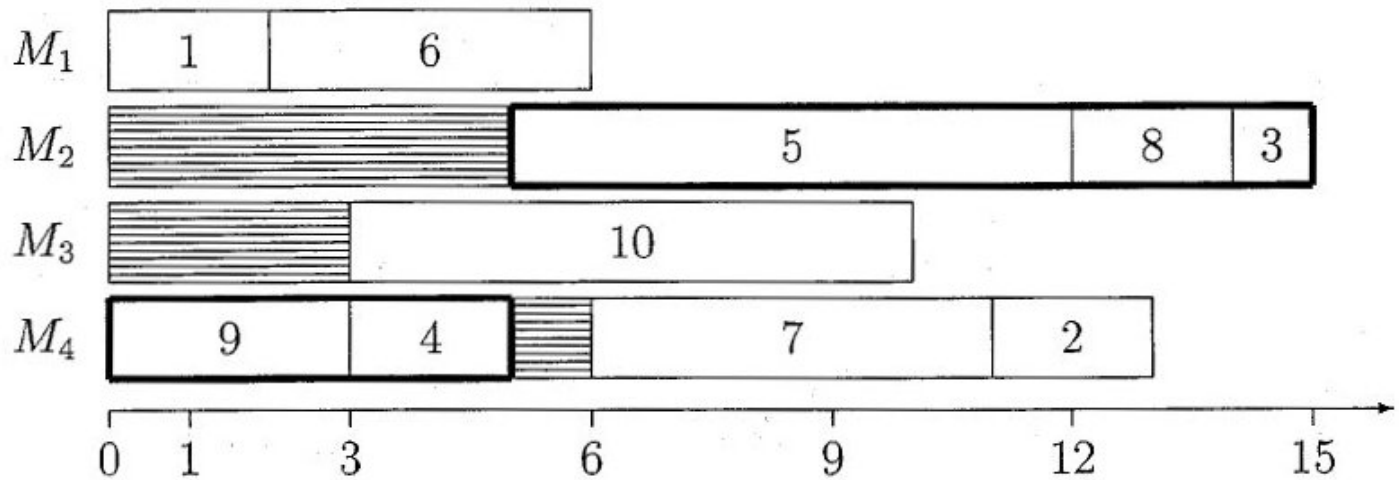
  - $S_{ij} + p_{ij} \leq S_{i+1,j}$ for all $O_{ij} \rightarrow O_{i+1,j}$
  - $S_{ij} + p_{ij} \leq S_{uv}$ or $S_{uv} + p_{uv} \leq S_{ij}$ for all operations with $\mu_{ij} = \mu_{uv}$.

  and has minimum makespan: $\min\{\max_{j \in J}(S_{n_j,j} + p_{n_j,j})\}$.

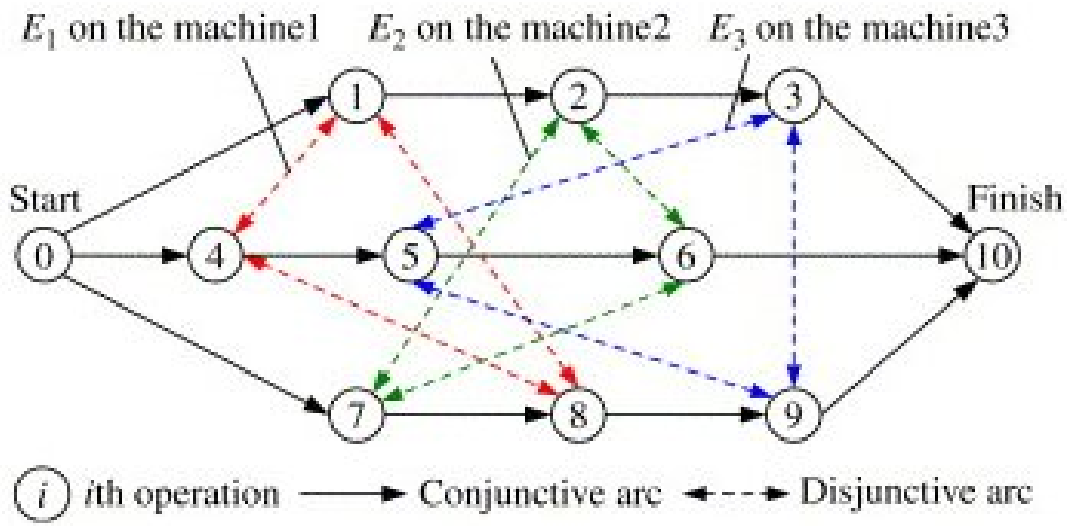
A schedule can also be represented by an m-tuple $\pi = (\pi^1, \pi^2, \ldots, \pi^m)$ where π^i defines the processing order on machine i.

There is always an optimal schedule that is semi-active.

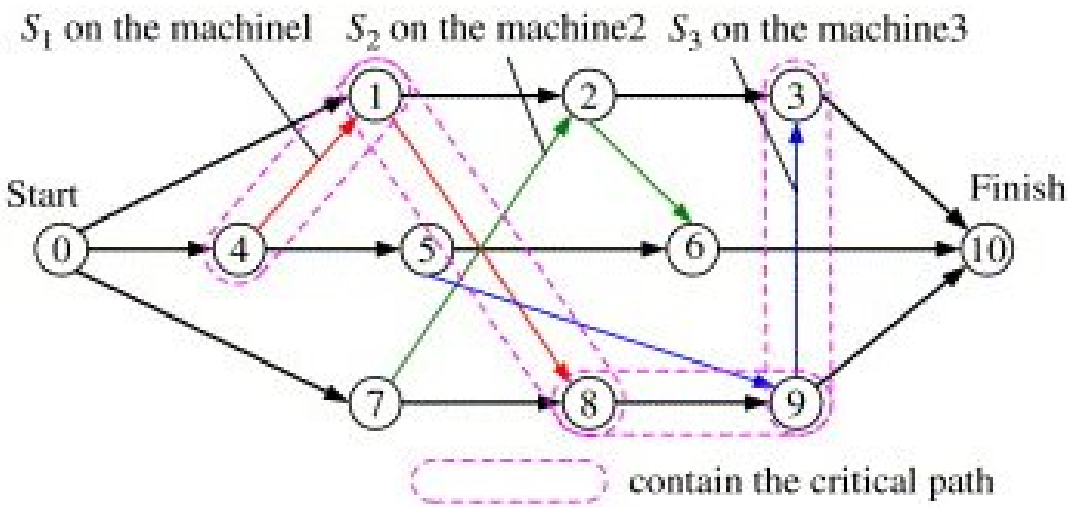
(semi-active schedule: for each machine, start each operation at the earliest feasible time.)


M_1
1
6
M_2
5
8
3
M_3
10
M_4
9
4
7
2
0
1
3
6
9
12
15

- Often simplified notation: $N = \{1, \ldots, n\}$ denotes the set of operations
- Disjunctive graph representation: $G = (N, A, E)$
  - vertices N: operations with two dummy operations 0 and $n+1$ denoting "start" and "finish".
  - directed arcs A, conjunctions
  - undirected arcs E, disjunctions
  - length of (i, j) in A is p_i



- A complete selection corresponds to choosing one direction for each arc of E.
- A complete selection that makes D acyclic corresponds to a feasible schedule and is called consistent.
- Complete, consistent selection $\Leftrightarrow$ semi-active schedule (feasible earliest start schedule).
- Length of longest path 0–$(n+1)$ in D corresponds to the makespan



Longest path computation

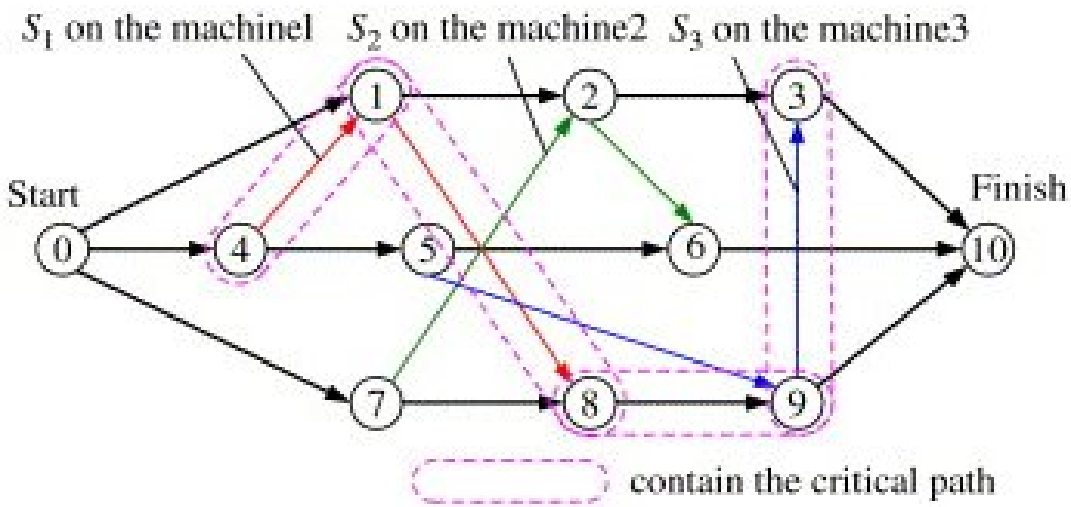
In an acyclic digraph:

- construct topological ordering ($i < j$ for all $i \to j \in A$)
- recursion:

$$r_0 = 0$$
$$r_l = \max_{\{j \mid j \to l \in A\}} \{r_j + p_j\} \qquad \text{for } l = 1, \ldots, n+1$$

- A block is a maximal sequence of adjacent critical operations processed on the same machine.

- In the Fig. below: $B_1 = \{4, 1, 8\}$ and $B_2 = \{9, 3\}$



- Any operation, u, has two immediate predecessors and successors:
  - its job predecessor $JP(u)$ and successor $JS(u)$
  - its machine predecessor $MP(u)$ and successor $MS(u)$

Exact methods

- Disjunctive programming

$$
\begin{array}{lll}
\min & C_{max} & \\
s.t. & x_{ij} + p_{ij} \leq C_{max} & \forall\, O_{ij} \in N \\
 & x_{ij} + p_{ij} \leq x_{lj} & \forall\, (O_{ij}, O_{lj}) \in A \\
 & x_{ij} + p_{ij} \leq x_{ik} \vee x_{ij} + p_{ij} \leq x_{ik} & \forall\, (O_{ij}, O_{ik}) \in E \\
 & x_{ij} \leq 0 & \forall\, i = 1, \ldots, m \;\; j = 1, \ldots, N
\end{array}
$$

- Constraint Programming
- Branch and Bound [Carlier and Pinson, 1983]

Typically unable to schedule optimally more than 10 jobs on 10 machines.
Best result is around 250 operations.

Branch and Bound [Carlier and Pinson, 1983] [B1, p. 179]

Let Ω contain the first operation of each job;

Let $r_{ij} = 0$ for all $O_{ij} \in \Omega$

Machine Selection Compute for the current partial schedule

$$t(\Omega) = \min_{ij \in \Omega} \{r_{ij} + p_{ij}\}$$

and let i^* denote the machine on which the minimum is achieved

Branching Let Ω' denote the set of all operations O_{i^*j} on machine i^* such that

$$r_{i^*j} < t(\Omega) \qquad \text{(i.e. eliminate } r_{i^*j} \geq t(\Omega))$$

For each operation in Ω', consider an (extended)partial schedule with that operation as the next one on machine i^*. For each such (extended) partial schedule, delete the operations from Ω, include its immediate follower in Ω and return to Machine Selection.

Lower Bounding:

- longest path in partially selected disjunctive digraph
- solve $1|r_{ij}|L_{max}$ on each machine i like if all other machines could process at the same time (see later shifting bottleneck heuristic) + longest path.

Shifting Bottleneck Heuristic

- A complete selection is made by the union of selections S_k for each clique E_k that corresponds to machines.
- **Idea**: use a priority rule for ordering the machines.
  choose each time the bottleneck machine and schedule jobs on that machine.
- Measure bottleneck quality of a machine k by finding optimal schedule to a certain single machine problem.
- Critical machine, if at least one of its arcs is on the critical path.

- $M_0 \subset M$ set of machines already sequenced.
- $k \in M \setminus M_0$
- $P(k, M_0)$ is problem $1 \mid r_j \mid L_{max}$ obtained by:
  - the selections in M_0
  - deleting each disjunctive arc in $p \in M \setminus M_0, p \neq k$
- $v(k, M_0)$ is the optimum of $P(k, M_0)$
- bottleneck $m = \arg \max_{k \in M \setminus M_0} \{v(k, M_0)\}$
- $M_0 = \emptyset$

Step 1: Identify bottleneck m among $k \in M \setminus M_0$ and sequence it optimally. Set $M_0 \leftarrow M_0 \cup \{m\}$

Step 2: Reoptimize the sequence of each critical machine $k \in M_0$ in turn: set $M'_o = M_0 - \{k\}$ and solve $P(k, M'_0)$.
Stop if $M_0 = M$ otherwise Step 1.

- Local Reoptimization Procedure

Construction of $P(k, M_0)$

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

- $r_j = L(0, j)$
- $d_j = L(0, n) - L(j, n) + p_j$

$L(i, j)$ length of longest path in G: Computable in $O(n)$

acyclic complete directed graph $\Longleftrightarrow$ transitive closure of its unique directed Hamiltonian path.

Hence, only predecessors and successor are to be checked.
The graph is not constructed explicitly, but by maintaining a list of jobs per machines and a list machines per jobs.

$1 \mid r_j \mid L_{max}$ can be solved optimally very efficiently.
Results reported up to 1000 jobs.

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

From one of the past lectures

[Maximum lateness with release dates]

- Strongly NP-hard (reduction from 3-partition)
- might have optimal schedule which is not non-delay
- Branch and bound algorithm (valid also for $1 \mid r_j, prec \mid L_{max}$)
  - **Branching**:
    schedule from the beginning (level k, $n!/(k-1)!$ nodes)
    elimination criterion: do not consider job j_k if:

$$r_j > \min_{l \in J} \{\max(t, r_l) + p_l\} \qquad J \text{ jobs to schedule, } t \text{ current time}$$

  - **Lower bounding**: relaxation to preemptive case for which EDD is optimal

Efficient local search for job shop

Solution representation:
m-tuple $\pi = (\pi^1, \pi^2, \ldots, \pi^m) \Longleftrightarrow$ oriented digraph $D_\pi = (N, A, E_\pi)$

Neighborhoods
Change the orientation of certain disjunctive arcs of the current complete selection

Issues:

1. Can it be decided easily if the new digraph $D_{\pi'}$ is acyclic?
2. Can the neighborhood selection S' improve the makespan?
3. Is the neighborhood connected?

Swap Neighborhood [Novicki, Smutnicki]

Reverse one oriented disjunctive arc (i, j) on some critical path.

Theorem

All neighbors are consistent selections.

Note: If the neighborhood is empty then there are no disjunctive arcs, nothing can be improved and the schedule is already optimal.

Theorem

The swap neighborhood is weakly optimal connected.

Insertion Neighborhood [Balas, Vazacopoulos, 1998]

For some nodes u, v in the critical path:

- move u right after v (forward insert)
- move v right before u (backward insert)

Theorem: If a critical path containing u and v also contains $JS(v)$ and

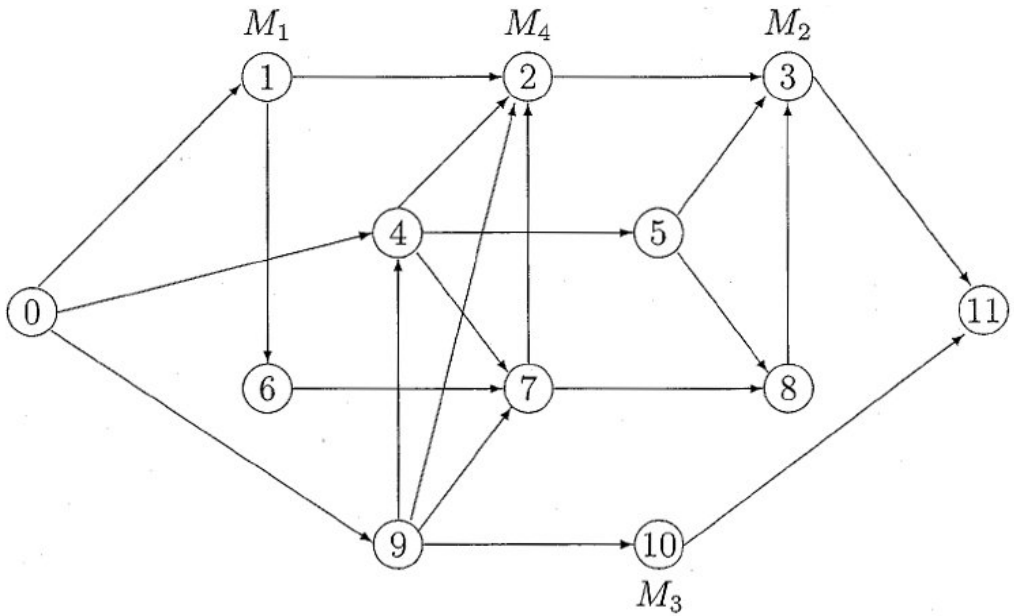
$$L(v, n) \geq L(JS(u), n)$$

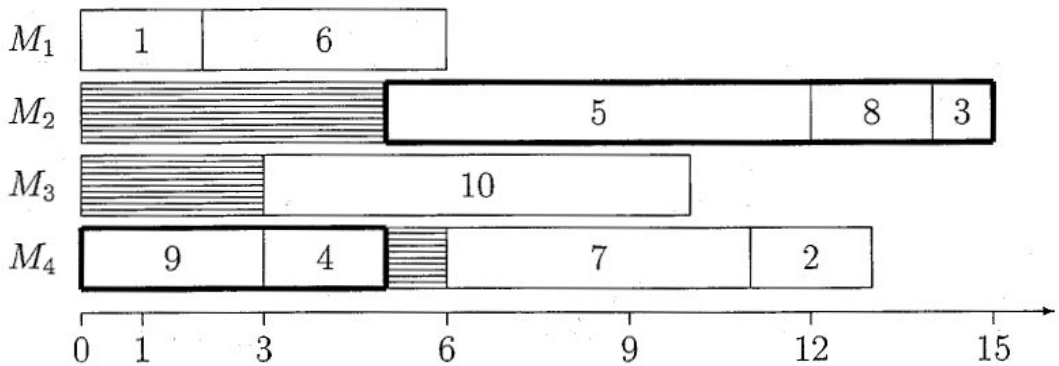
then a forward insert of u after v yields an acyclic complete selection.

Theorem: If a critical path containing u and v also contains $JS(v)$ and

$$L(0, u) + p_u \geq L(0, JP(v)) + p_{JP(v)}$$

then a backward insert of v before v yields an acyclic complete selection.


M_1
M_4
M_2
1
2
3
4
5
0
11
6
7
8
9
10
M_3


M_1
1
6
M_2
5
8
3
M_3
10
M_4
9
4
7
2
0
1
3
6
9
12
15

Theorem: (Elimination criterion) If $C_{max}(S') < C_{max}(S)$ then at least one operation of a machine block B on the critical path has to be processed before the first or after the last operation of B.

- Swap neighborhood can be restricted to first and last operations in the block
- Insert neighborhood can be restricted to moves similar to those saw for the flow shop. [Grabowski, Wodecki]

Tabu Search requires a best improvement strategy hence the neighborhood must be search very fast.

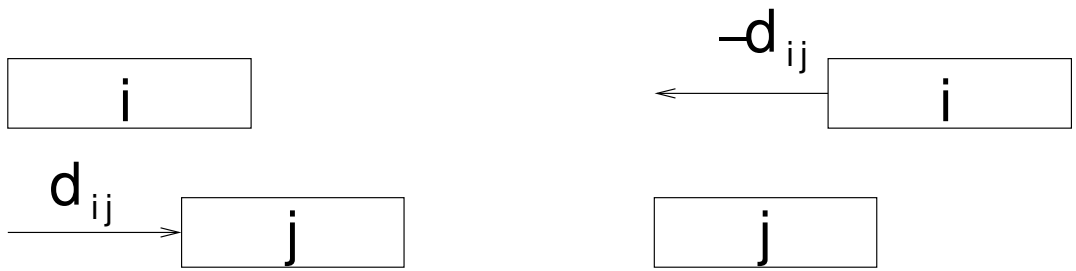
Neighbor evaluation:

- exact recomputation of the makespan $O(n)$
- approximate evaluation (rather involved procedure but much faster and effective in practice)

The implementation of Tabu Search follows the one saw for flow shop.

Outline

Generalizations: Time Lags



Generalized time constraints

They can be used to model:

- Release time:

$$S_0 + r_i \leq S_i \qquad \Longleftrightarrow \qquad d_{0i} = r_i$$

- Deadlines:

$$S_i + p_i - d_i \leq S_0 \qquad \Longleftrightarrow \qquad d_{i0} = p_i - d_i$$

- Modelling

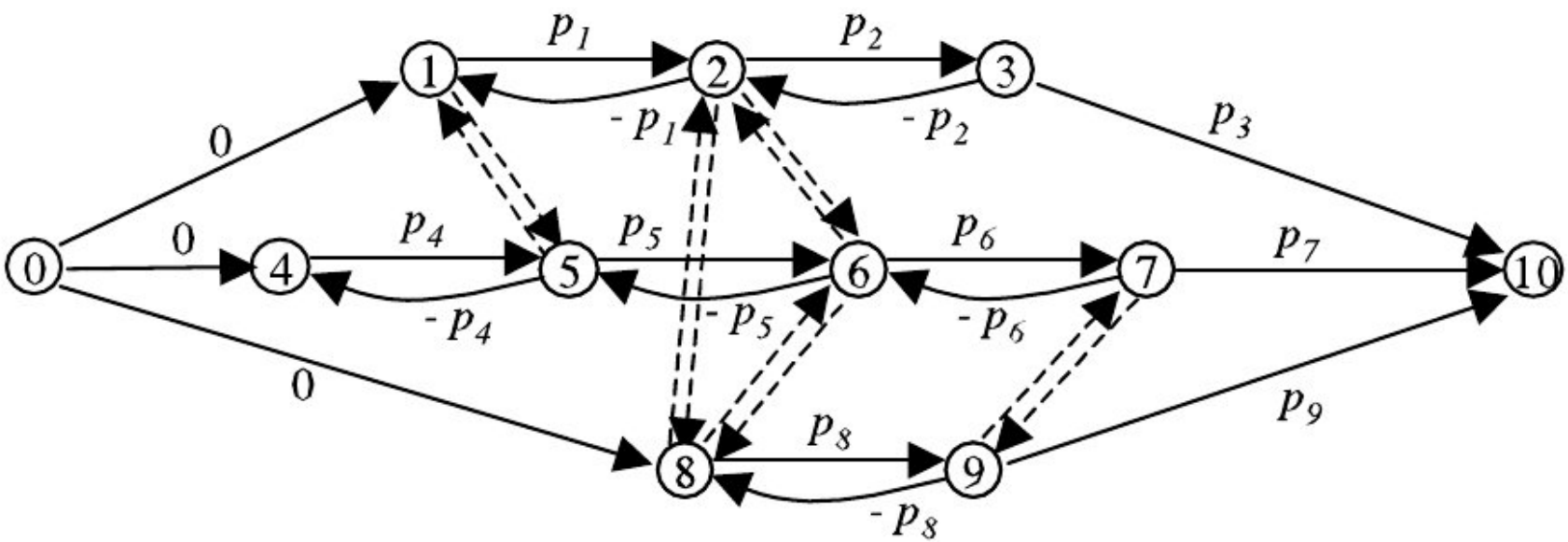
$$
\begin{array}{lll}
\min & C_{max} & \\
s.t. & x_{ij} + d_{ij} \leq C_{max} & \forall\, O_{ij} \in N \\
 & x_{ij} + d_{ij} \leq x_{lj} & \forall\, (O_{ij}, O_{lj}) \in A \\
 & x_{ij} + d_{ij} \leq x_{ik} \vee x_{ij} + d_{ij} \leq x_{ik} & \forall\, (O_{ij}, O_{ik}) \in E \\
 & x_{ij} \geq 0 & \forall\, i = 1, \ldots, m \;\; j = 1, \ldots, N
\end{array}
$$

- In the disjunctive graph, d_{ij} become the lengths of arcs

- Exact relative timing (perishability constraints):
  if operation j must start l_{ij} after operation i:

$$S_i + p_i + l_{ij} \leq S_j \qquad \text{and} \qquad S_j - (p_i + l_{ij}) \leq S_i$$

($l_{ij} = 0$ if no-wait constraint)



- Set up times:

$$S_i + p_i + s_{ij} \leq S_j \qquad \text{or} \qquad S_j + p_j + s_{ji} \leq S_i$$

- Machine unavailabilities:
  - Machine M_k unavailable in $[a_1, b_1],\ [a_2, b_2],\ \ldots\ ,\ [a_v, b_v]$
  - Introduce v artificial operations $\lambda = 1, \ldots, v$ with $\mu_\lambda = M_k$ and:
    $p_\lambda = b_\lambda - a_\lambda$
    $r_\lambda = a_\lambda$
    $d_\lambda = b_\lambda$

- Minimum lateness objectives:

$$L_{max} = \max_{j=1}^{N}\{C_j - d_j\} \qquad \Longleftrightarrow \qquad d_{n_j,n+1} = p_{n_j} - d_j$$

Blocking

Arises with limited buffers:
after processing, a job remains on the machine until the next machine is freed

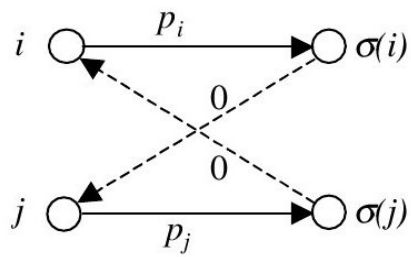
- Needed a generalization of the disjunctive graph model
  $\Longrightarrow$ Alternative graph model $G = (N, E, A)$ [Mascis, Pacciarelli, 2002]

1. two non-blocking operations to be processed on the same machine

$$S_i + p_i \leq S_j \qquad \text{or} \qquad S_j + p_j \leq S_i$$

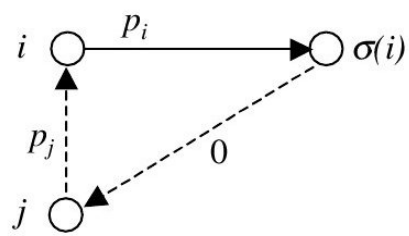
2. Two blocking operations i, j to be processed on the same machine $\mu(i) = \mu(j)$

$$S_{\sigma(j)} \leq S_i \qquad \text{or} \qquad S_{\sigma(i)} \leq S_j$$



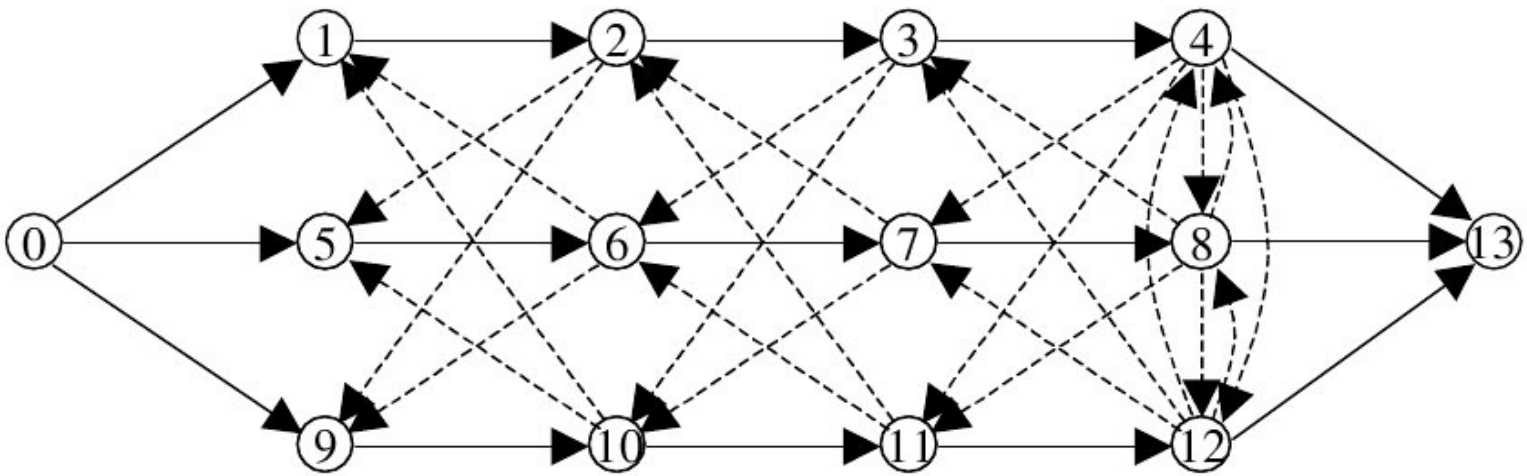
3. i is blocking, j is non-blocking (ideal) and i, j to be processed on the same machine $\mu(i) = \mu(j)$.

$$S_i + p_i \leq S_j \qquad \text{or} \qquad S_{\sigma(j)} \leq S_i$$

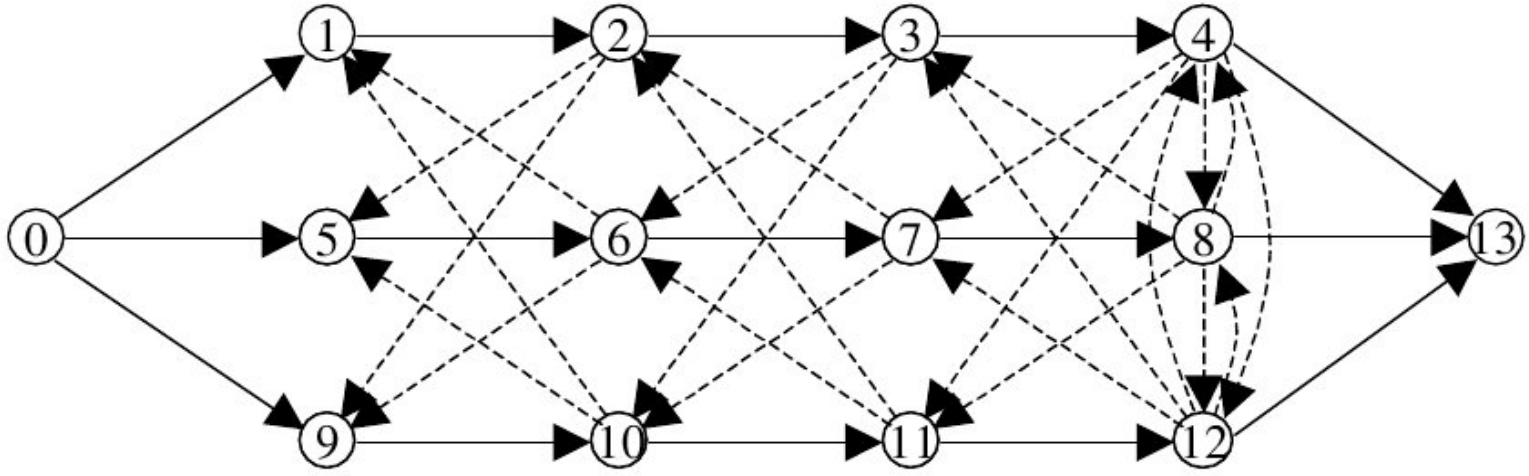


Example

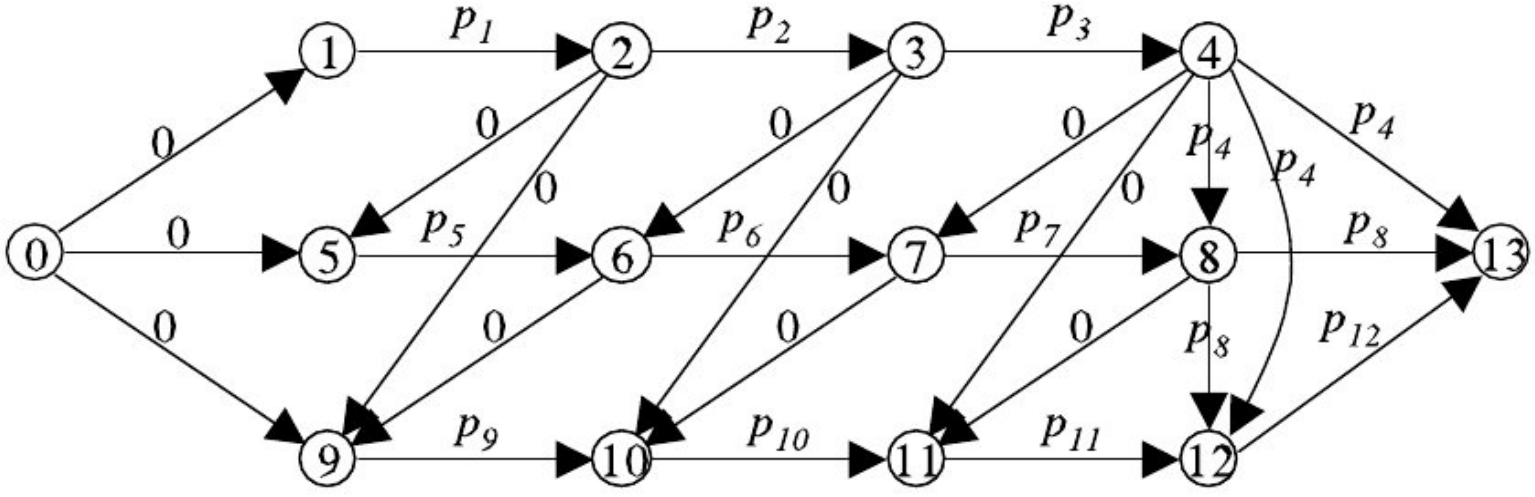
- $O_0, O_1, \ldots, O_{13}$
- $M(O_1) = M(O_5) = M(O_9)$
  $M(O_2) = M(O_6) = M(O_{10})$
  $M(O_3) = M(O_7) = M(O_{11})$



- Length of arcs can be negative
- Multiple occurrences possible: $((i,j),(u,v)) \in A$ and $((i,j),(h,k)) \in A$
- The last operation of a job j is always non-blocking.



- A complete selection S is consistent if it chooses alternatives from each pair such that the resulting graph does not contain positive cycles.



Example:

- $p_a = 4$
- $p_b = 2$
- $p_c = 1$

- b must start at least 9 days after a has started
- c must start at least 8 days after b is finished
- c must finish within 16 days after a has started

$$\begin{aligned} S_a + 9 &\leq S_b \\ S_b + 10 &\leq S_c \\ S_c - 15 &\leq S_a \end{aligned}$$

This leads to an absurd.
In the alternative graph the cycle is positive.

- The Makespan still corresponds to the longest path in the graph with the arc selection $G(S)$.
- Problem: now the digraph may contain cycles. Longest path with simple cyclic paths is NP-complete. However, here we have to care only of non-positive cycles.
- If there are no cycles of length strictly positive it can still be computed efficiently in $O(|N||E \cup A|)$ by Bellman-Ford (1958) algorithm.
- The algorithm iteratively considers all edges in a certain order and updates an array of longest path lengths for each vertex. It stops if a loop over all edges does not yield any update or after $|N|$ iterations over all edges (in which case we know there is a positive cycle).
- Possible to maintain incremental updates when changing the selection [Demetrescu, Frangioni, Marchetti-Spaccamela, Nanni, 2000].

Heuristic Methods

- The search space is highly constrained + detecting positive cycles is costly
- Hence local search methods not very successful
- Rely on the construction paradigm
- Rollout algorithm [Meloni, Pacciarelli, Pranzo, 2004]

Rollout

- Master process: grows a partial selection S^k:
  decides the next element to fix based on an heuristic function
  (selects the one with minimal value)

- Slave process: evaluates heuristically the alternative choices.
  Completes the selection by keeping fixed what passed by the master process and fixing one alternative at a time.

- Slave heuristics
  - *Avoid Maximum Current Completion time*
    find an arc (h, k) that if selected would increase most the length of the longest path in $G(S^k)$ and select its alternative

$$\max_{(uv)\in A}\{l(0,u) + a_{uv} + l(v,n)\}$$

  - *Select Most Critical Pair*
    find the pair that, in the worst case, would increase least the length of the longest path in $G(S^k)$ and select the best alternative

$$\max_{((ij),(hk))\in A} \min\{l(0,u) + a_{hk} + l(k,n), l(0,i) + a_{ij} + l(j,n)\}$$

  - *Select Max Sum Pair*
    find the pair with greatest potential effect on the length of the longest path in $G(S^k)$ and select the best alternative

$$\max_{((ij),(hk))\in A} |l(0,u) + a_{hk} + l(k,n) + l(0,i) + a_{ij} + l(j,n)|$$

Trade off quality *vs* keeping feasibility
Results depend on the characteristics of the instance.

Implementation details of the slave heuristics

- Once an arc is added we need to update all $L(0,u)$ and $L(u,n)$. Backward and forward visit $O(|F|+|A|)$
- When adding arc a_{ij}, we detect positive cycles if $L(i,j)+a_{ij}>0$. This happens only if we updated $L(0,i)$ or $L(j,n)$ in the previous point and hence it comes for free.
- Overall complexity $O(|A|(|F|+|A|))$

Speed up of Rollout:

- Stop if partial solution overtakes upper bound
- limit evaluation to say 20% of arcs in A