DM204, 2010
SCHEDULING, TIMETABLING AND ROUTING

Lecture 19
Flow Shop

Marco Chiarandini

Department of Mathematics & Computer Science
University of Southern Denmark

Outline

Course Overview

- ✔ Problem Introduction
  - ✔ Scheduling classification
  - ✔ Scheduling complexity
  - ✔ RCPSP
- ✔ General Methods
  - ✔ Integer Programming
  - ✔ Constraint Programming
  - ✔ Heuristics
  - ✔ Dynamic Programming
  - Branch and Bound

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

Outline

1. Flow Shop
   - Introduction
   - Makespan calculation
   - Johnson's algorithm
   - Construction heuristics
   - Iterated Greedy
   - Efficient Local Search and Tabu Search

2. Job Shop
   - Modelling
   - Exact Methods
   - Local Search Methods

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

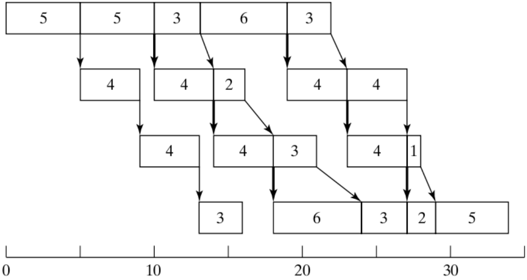
Flow Shop Scheduling:

- $\mu_{ij} = l$, $l = 1, 2, \ldots, m$
- 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

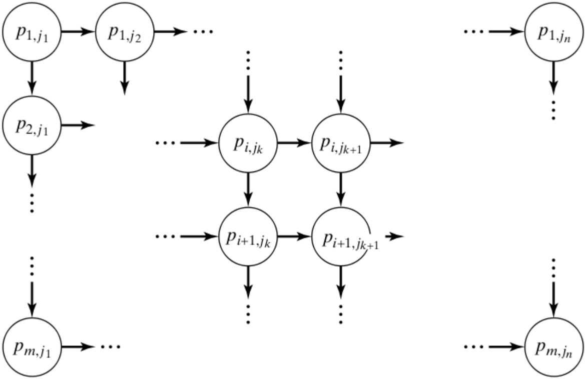
$\pi_1, \pi_2, \pi_3, \pi_4$:

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

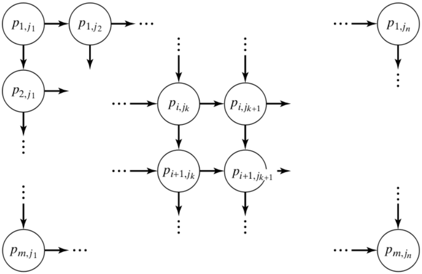
- we assume unlimited buffer
- if same job sequence on each machine ➨ permutation flow shop

Directed Graph Representation

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



Directed Graph Representation



Recursion for C_{max}

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

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

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

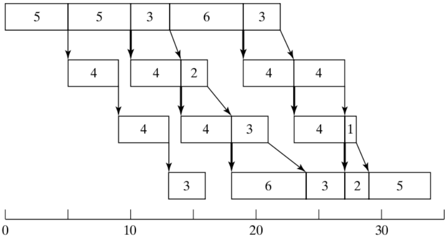
Computation cost?

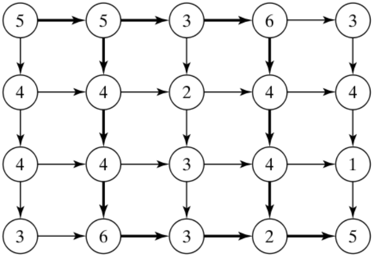
Example

$jobs$	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



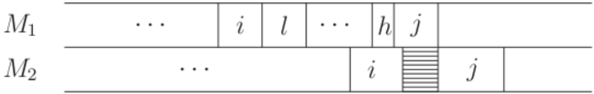


$Fm \mid \mid C_{max}$

Theorem

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

Proof: By contradiction.



Corollary

$F2 \mid \mid C_{max}$ *and* $F3 \mid \mid C_{max}$ *are permutation flow shop*

Note: $F3 \mid \mid C_{max}$ 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 $p_{i^*,j^*} = \min\{p_{ij} \,|\, i \in 1, 2, j \in J\}$
Step 2 If $i^* = 1$ then $L = L \circ \{i^*\}$
else if $i^* = 2$ then $R = R \circ \{i^*\}$
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* | *prmu* | *C_{max}

Nawasz, Enscore, Ham's heuristic (1983)

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

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 $O(n^2m)$)

Acceptance Criterion: random walk, best, SA-like

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

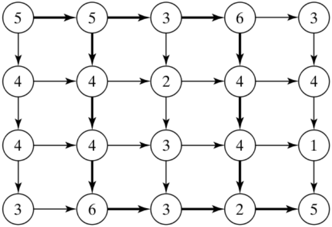
- NEH average gap 3.35% in less than 1 sec.
- IG average gap 0.44% in about 360 sec.

Efficient local search for $Fm \mid prmu \mid C_{max}$

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:

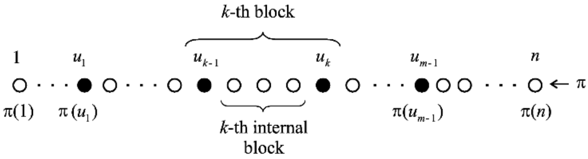
$$C(\pi, t) = \sum_{j=1}^{t_1} p_{\pi(j)1} + \sum_{j=t_1}^{t_2} p_{\pi(j)2} + \cdots + \sum_{j=t_{m-1}}^{n} p_{\pi(j)m}$$

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':

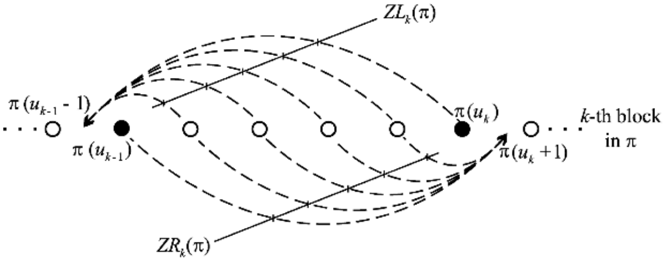
a) *at least one job $j \in B_k$ precedes job $\pi(u_{k-1})$, $k = 1, \ldots, m$, or*

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)

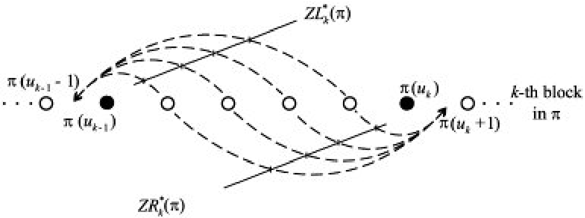
The neighborhood thus defined is connected.

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.