Scheduling: flow shops

Logistics

Giovanni Righini

Università degli Studi
di Milano

Flow shops

Flow shop models arise when jobs must undergo a given sequence of operations in the same order.

- Flow shops with unlimited intermediate storage
- Flow shops with limited intermediate storage

Flexible flow shops arise when a set of parallel machines are available for each stage.

Most studies concentrate on makespan minimization, which is computationally difficult.

The other classical objectives are even harder.

Permutation flow shops

$Fm||C_{max}$.

There exists an optimal solution in which the jobs are processed in the same order on the first two machines and on the last two machines.

When sequence changes are allowed and there are 4 or more machines, then it may be optimal to change the order of the jobs.

Problems in which it is explicitly forbidden to change the sequence are called permutation flow shops.

Non-permutation flow shops are significantly harder.

Makespan minimization

Given a permutation $1, 2, \ldots, n$ of the jobs for an m machine flow shop, completion times can be computed recursively:

$$C_{i1} = \sum_{l=1}^{i} p_{l1}$$

$$C_{1j} = \sum_{k=1}^{j} p_{1k}$$

$$C_{ij} = \max\{C_{i-1\,j}, C_{i\,j-1}\} + p_{ij}$$

The makespan is the cost of the critical path from node $(1,1)$ to node (m,n) on a grid graph with

- a node for each (i,j) pair,
- horizontal arcs from $(i,j-1)$ to (i,j),
- vertical arcs from $(i-1,j)$ to (i,j)
- weights p_{ij} associated with the nodes.

Permutation $n, n-1, \ldots, 2, 1$ yields the same makespan as $1, 2, \ldots, n-1, n$.

Two machines: Johnson's rule

Consider $F2||C_{max}$.

Johnson's rule. Partition the jobs into two sets:

- Set I : jobs with $p_{1j} < p_{2j}$
- Set II: jobs with $p_{1j} > p_{2j}$

Jobs with $p_{1j} = p_{2j}$ can be put in either set, arbitarily.

Schedule the jobs in Set I first, according to the SPT rule. Then schedule the jobs in Set II, according to the LPT rule. Ties can be broken arbitrarily.

The resulting SPT(1)-LPT(2) schedule is optimal (the rule is sufficient but not necessary).

More than 2 machines: MIP formulation

$Fm|prmu|C_{max}$ is hard from $m \geq 3$. It can be formulated as a MIP model.

Variables. A binary variable x_{jk} indicates whether job j is in position k in the permutation.

A variable I_{ik} represents the idle time on machine i between the job in position k and the next one.

A variable W_{ik} represents the waiting time of the job in position k between machine i and the next one.

Objective. The makespan is minimized by minimizing the total idle time on the last machine:

$$minimize \;\; C_{max} = \sum_{i=1}^{m-1} \sum_{j=1}^{n} p_{ij} x_{j1} + \sum_{k=1}^{n-1} I_{mk} + \sum_{j=1}^{n} p_{mj}.$$

The first sum indicates the time needed to start the first job on the last machine; the second sum is the total idle time on the last machine after processing the first job; the third term is constant.

More than 2 machines: MIP formulation

Constraints. Assignment constraints for each job and position:

$$\sum_{j=1}^{n} x_{jk} = 1 \quad \forall k = 1, \ldots, n$$

$$\sum_{k=1}^{n} x_{jk} = 1 \quad \forall j = 1, \ldots, n$$

Consistency of idle times and waiting times:

$$I_{ik} + \sum_{j=1}^{n} p_{ij} x_{jk+1} + W_{ik+1} =$$

$$W_{ik} + \sum_{j=1}^{n} p_{i+1\ j} x_{jk} + I_{i+1\ k} \qquad \forall 1 \leq i \leq m-1, \forall 1 \leq k \leq n-1$$

The first job does never wait and the first machine is never idle:

$$W_{i1} = 0 \quad \forall 1 \leq i \leq m-1$$

$$I_{1k} = 0 \quad \forall 1 \leq k \leq n-1.$$

Special cases

Proportionate permutation flow shop $Fm|prmu, p_{ij} = p_j|C_{max}$ is the special case in which the processing times do not depend on the machine.

For $Fm|prmu, p_{ij} = p_j|C_{max}$ the makespan equals

$$C_{max} = \sum_{j=1}^{n} p_j + (m-1) \max_j \{p_j\}$$

and it is independent of the schedule.

Permutation schedules are optimal even for $Fm|p_{ij} = p_j|C_{max}$.

Special cases

Proportionate flow shops have several common characteristics with single-machine scheduling problems:

- the SPT rule is optimal for $1||\sum C_j$ as well as for $Fm|prmu, p_{ij} = p_j|\sum C_j$;
- the algorithm that optimizes $1||\sum U_j$ also optimizes $Fm|prmu, p_{ij} = p_j|\sum U_j$;
- the algorithm that optimizes $1||h_{max}$ also optimizes $Fm|prmu, p_{ij} = p_j|h_{max}$;
- the pseudo-polynomial D.P. algorithm that optimizes $1||\sum T_j$ also optimizes $Fm|prmu, p_{ij} = p_j|\sum T_j$;
- the elimination criteria that are valid for $1||\sum w_j T_j$ are also valid for $Fm|prmu, p_{ij} = p_j|\sum w_j T_j$.

Machines with different speed

On a machine with speed v_i each job j spends $p_{ij} = p_j / v_i$ time.

The bottleneck machine is the machine with minimum speed.

In $Fm|prmu, p_{ij} = p_j| \sum C_{max}$ with machines at different speeds, the LPT and SPT rules are optimal.

Heuristics

Since the $Fm|prmu|C_{max}$ is difficult, several heuristics have been developed to approximate it.

Slope heuristic. A slope index is computed for each job:

$$A_j = -\sum_{i=1}^{m} (m - (2i - 1))\, p_{ij}\,.$$

Then jobs are scheduled in non-increasing order of the slope index.

Jobs with small processing time on the first machines and large processing times on the last machines should better be placed in the initial positions of the schedule.

Total completion time

$F2||\sum C_j$ is already strongly NP-hard.

$Fm|p_{ij} = p_j|\sum C_j$ is optimized by the SPT rule.

$Fm|p_{ij} = p_j|\sum w_j C_j$ is not optimized by the WSPT rule, but it is polynomially solvable.