# Lecture 19,20,21: Shop Scheduling Problems

July 14th,16th, 21st 2009

---

In the next three lectures, we will look at shop scheduling problems. We will be interested in minimizing the makespan. To recall, in a shop scheduling problem, we have $n$ jobs, but each job has $m$ operations corresponding to the $m$ machines. Different operations take different times. A job is said to have been completed iff all its operations have been completed. The various shop scheduling problems puts restrictions on how these operations can be scheduled on the various machines. There are three main kinds:

- **Flow Shop**: In this, the operations of each job must be processed in the same order on the machines. The order is assumed to be $(1, 2, \ldots, n)$. The problem is denoted as $(F||C_{max})$.

  Furthermore, if there is a restriction that every machine must also process the jobs in the same order, we call the restriction *permutation* flow shop denoting the problem as $(F|prmu|C_{max})$.

- **Job Shop**: This is a generalization of flow shop in which every jobs comes with its own order in which it needs to be processed on the machines.

- **Open Shop**: In this, there is no restriction on the order in which the operations need to be processed on the machines.

## 1 Flow shop with 2 machines

All the shop scheduling problems are solvable if there are 2 machines. We only saw the flow shop version in class. The other two could be a nice question in the finals.

We first start with the following claim.

**Claim 1.1.** *In the case of* 2 *machines, the optimal makespan for the flow shop problem is same with or without the permutation constraint.*

*Proof.* Call a pair of jobs $(j, k)$ bad if they are not processed by both machines in the same order. Let $S$ be an optimal schedule with the minimum number of bad pairs. If we show that there are no bad pairs, then we are done. Suppose for a contradiction that there is a bad pair $(j, k)$. Let job $k$ immediately precede job $j$ on machine 1 such that job $k$ is processed after job $j$ on machine 2.

Let $S'$ be the schedule obtained from $S$ by swapping jobs $j$ and $k$ on machine 1 and having all other jobs remain in the same order. Note that $S'$ is a feasible schedule: job $j$ is processed earlier on machine 1, so is still completed before job $j$ starts on machine 2; job $k$ does not start on machine 2 until after both jobs $j$ and $k$ are completed on machine 1. Swapping the order of jobs $j$ and $k$ means that job $k$ still does not start until after job $k$ ends.

Note that the completion times of the jobs on machine 2 do not change, and so the makespan does not change. Thus, we get another optimal schedule with less number of bad pairs contradicting our choice. □

Therefore, we need to find a permutation of the jobs which minimized makespan. The algorithm is as follows, and is called $SPT(1)$-$LPT(2)$.

1. Let $J_1 := \{j \in J : p_{1j} \leq p_{2j}\}$ be the set of jobs which have no more processing time on machine 1 than on machine 2. Let $J_2 := J \setminus J_1$, be the remaining jobs.

2. The permutation in which the jobs are to be scheduled is as follows. Schedule jobs of $J_1$ in SPT of their $p_{1j}$'s, and then schedule the jobs of $J_2$ in LPT of their $p_{2j}$'s.

**Theorem 1.2.** *$SPT(1)$-$LPT(2)$ produces an optimal schedule for $F2||C_{\max}$.*

*Proof.* (Sketch of what was done in class)

Let $S$ be a permutation schedule for $F2||C_{\max}$. Renumber the jobs according to their order (ie. they are scheduled in the order $1, 2, \ldots, n$).

We may assume that there is no idle time on machine 1 (because scheduling jobs as early as possible on machine 1 does not violate feasibility). Let $t$ be any idle time on machine 2. We may assume that the job $k$ starting at time $t$ on machine 2 starts immediately after job $k$ completes on machine 1. Otherwise we could've started job $k$ earlier on machine 2. We will say job $k$ is responsible for this idle time $t$. Let $\ell$ be the last job responsible for any idle time.

The makespan of the schedule is determined by the sum of the processing times of the first $\ell$ jobs on machine 1 and the sum of the processing times of the last $n - \ell + 1$ jobs on machine 2: $C_{\max} = \sum_{j \leq \ell} p_{j,1} + \sum_{j \geq \ell} p_{j,2}$. If we reduce *all* the processing times by a fixed amount $\Delta$, then *any* sum of $n + 1$ processing times decreases by $(n + 1)\Delta$. Therefore, if we show that given these decreased processing times, and a fixed order of jobs, then the jobs responsible for the idle times in the old processing times are the same as that in the new ones, then we have shown that the optimum remains the same with these new processing times. Assume this for the time being, we come to it a bit later.

Recall that if a job $j$ has $p_{j,1} = 0$, then it is scheduled first on machine 1: it does not delay any jobs on machine 1 and only buys waiting time on machine 2. Likewise, if a job $j$ has $p_{j,2} = 0$, then it can be scheduled last on machine 2.

We can construct an optimal schedule by repeatedly finding the job $j$ that has minimum processing time on either job (ie. $\Delta = \arg\min_j \{p_{j,1}, p_{j,2}\}$, subtracting the that value from that of all the processing times and then scheduling the job with zero processing time according to the above rule and repeating. The schedule produced is the same as that given by $SPT(1)$-$LPT(2)$ rule.

Coming back to why given a schedule with new processing times, the pattern of idle times remain the same. Take a idle time $[t_1, t_2]$ in the old processing times. Assume $t_1 > 0$, that is, we are not at the first idle time for which job 1 is responsible. The reason why idle times behave similarly is that with the new processing times, there will be a corresponding idle time at time $[t_1', t_2']$ where $t_2' = t_2 - k\Delta$ for some integer $k$, and $t_1' = t_1 - k\Delta$, for the same $k$. Draw a picture to convince yourself of this. We are not being rigorous here. □

# 2 Job Shop Scheduling

## 2.1 Disjunctive Graph of the Instance

Given a job shop instance, we construct the disjunctive graph as follows. In this graph, there is a vertex $(i, j)$ for every operation on each job. There is also a source $s$ and a sink $t$.

There are two kinds of edges between these vertices. There are fixed arcs and disjunctive edges. For any job $j$, there is arc from $(i, j)$ to $(i', j)$ if $j$ needs to be processed on machine $i$ before machine $i'$. The source $s$ has an arc to all nodes $(i, j)$ which do not have any incoming fixed arcs. The sink $t$ has an arc from all vertices $(i, j)$ with no out-going edges.

For any machine $i$, there is an *undirected* edge from $(i, j)$ and $(i, j')$, for any two jobs $j$ and $j'$. These edges are called disjunctive edges. This completes the description of the graph.

The semantic of the graph is as follows. Suppose we fix an *acyclic* orientation of the disjunctive edges. So, every disjunctive edge of the form $(i, j) - (i, j')$ is either oriented as $(i, j) \to (i, j')$ or $(i, j') \to (i, j)$. Once given this acyclic directed graph, we think of the arcs as precedence constraints which must be satisfied by the schedule. This leads to a valid job shop schedule as follows.

Given an orientation, let us first describe the weights on the arcs. An arc (fixed or the oriented disjunctive) coming out of a vertex $(i, j)$ has weight $p_{i,j}$. Given these weights, define $t_{i,j}$ for any vertex $i, j$ as the *longest* path from $s$ to $(i, j)$. This is the time when job $j$ is scheduled on machine $i$, and thus an orientation of the disjunctive edges correspond to a schedule for the job shop scheduling problem.

In the next section we see how we can use the above disjunctive graph to get an integer program formulation for the job-shop scheduling problem. Finally, we will see a heuristic to get a good orientation for the above problem.

## 2.2 Integer Programming Formulation

We have a variable $C$ for the makespan, and $C_{ij}$ for the completion time of job $j$'s operation on machine $i$. Therefore, we have the following constraints corresponding to fixed arcs:

$$C_{i'j} \geq C_{ij} + p_{i'j} \qquad \forall \text{ fixed } (i, j) \to (i', j) \tag{1}$$

For the disjunctive edge $(i, j) - (i, j')$, we have to decide which way to orient it. If we knew the orientation, then we could write a linear constraint on the completion times as follows

$$\begin{aligned} C_{ij'} &\geq C_{ij} + p_{ij'} \qquad \text{if } (i, j) \to (i, j') \\ &\text{or} \\ C_{ij} &\geq C_{ij'} + p_{ij} \qquad \text{if } (i, j') \to (i, j) \end{aligned}$$

For every disjunctive edge, we have two variables $X_{(i,j),(i,j')}$ and $X_{(i,j'),(i,j)}$ with the semantic that the first is 1 if the edge is oriented from $(i, j)$ to $(i, j')$, and 0 otherwise. The latter has exactly the opposite semantic. Thus we add the following set of constraints for every disjunctive edge to capture the above.

$$M \cdot X_{(i,j'),(i,j)} + C_{ij'} \geq C_{ij} + p_{ij'} \tag{2}$$
$$M \cdot X_{(i,j),(i,j')} + C_{ij} \geq C_{ij'} + p_{ij} \tag{3}$$
$$X_{(i,j'),(i,j)} + X_{(i,j),(i,j')} = 1 \tag{4}$$
$$X_{(i,j'),(i,j)}, X_{(i,j),(i,j')} \in \{0,1\} \tag{5}$$

where $M$ is a large number with the following property. If $X_{(i,j'),(i,j)} = 1$, that is the edge is oriented from $(i,j')$ to $(i,j)$, then I need only the second constraint to be true, and not the first. Thus, $M$ should be so large that the first is true any way. Choosing $M$ to be the sum of *all* processing time suffices.

Thus, now the integer program is as follows:

$$\min \quad C \tag{6}$$
$$\forall (i,j); \qquad C \geq C_{ij} \qquad \forall (i,j) \tag{7}$$
$$\forall \text{ fixed } (i,j) \rightarrow (i',j) \qquad C_{i'j} \geq C_{ij} + p_{i'j} \tag{8}$$
$$\forall \text{ disjunctive edge } (i,j) - (i,j')$$
$$M \cdot X_{(i,j'),(i,j)} + C_{ij'} \geq C_{ij} + p_{ij'} \tag{9}$$
$$M \cdot X_{(i,j),(i,j')} + C_{ij} \geq C_{ij'} + p_{ij} \tag{10}$$
$$X_{(i,j'),(i,j)} + X_{(i,j),(i,j')} = 1 \tag{11}$$
$$X_{(i,j'),(i,j)}, X_{(i,j),(i,j')} \in \{0,1\} \tag{12}$$