IEOR 8100 SCHEDULING ALGORITHMS

Lecturer: Clifford Stein | **Lecture 1**
Scribe: Vladlena Powers | Date: 8 Sep 2016

1 Models. Notation. Applications.

In a scheduling problem the number of jobs is denoted by n and the number of machines by m.

All definitions in this chapter, except the definition of idle time and response time, are borrowed from [1, p. 14-19].

One of the schedule manipulation interfaces is **Gantt chart** (see Figure 1). The Gantt chart is the usual horizontal bar chart with the x-axis representing the time and the y-axis, the various machines.

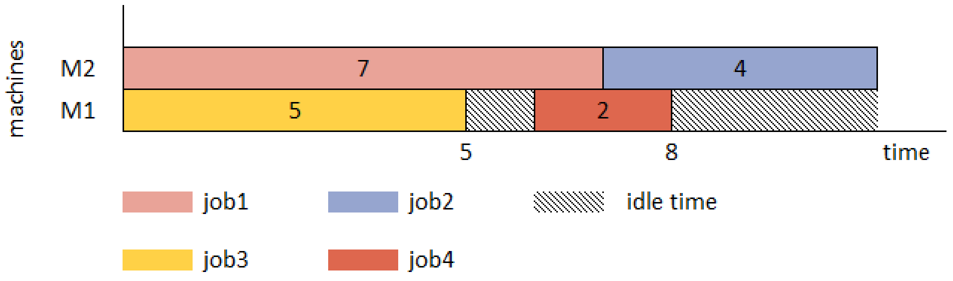

Figure 1: Example of a Gantt chart. Numbers specify lengths of jobs.

The following pieces of data are associated with job j:

Processing time (length) p_{ij}. The p_{ij} represents the processing time of job j on machine i. The subscript i is omitted if the processing time of job j does not depend on the machine or if job j is only to be processed on one given machine.

Release date r_j. It is the earliest time at which job j can start its processing.

Due date d_j. The due date d_j of job j represents the committed shipping or completion date. When a due date must be met, it is referred to as a ***deadline***.

Weight w_j. The weight w_j of job j is basically a priority factor, denoting the importance of job j relative to the other jobs in the system.

A scheduling problem is described by a triplet $\boldsymbol{\alpha} \mid \boldsymbol{\beta} \mid \boldsymbol{\gamma}$, representing ***machine*** | ***constraints*** | ***objective***. The α field describes the machine environment and contains a single entry. The β field provides details of processing characteristics and constraints and may contain no entry at all or multiple entries. The γ field describes the objective to be minimized and usually contains a single entry.

Single machine 1. The case of a single machine is the simplest of all possible machine environments and is a special case of all other more complicated machine environments.

Identical machines in parallel P_m. There are m identical machines in parallel. Job j requires a single operation and may be processed on any one of the m machines or on any one that belongs to a given subset.

Machines in parallel with different speeds Q_m. There are m machines in parallel with different speeds. The speed of machine i is denoted by v_i. The time p_{ij} that job j spends on machine i is equal to p_i/v_i (assuming job j receives all its processing from machine i).

Unrelated machines in parallel R_m. This environment is a generalization of the previous one. There are m different machines in parallel. Machine i can process job j at speed v_{ij} (assuming job j receives all its processing from machine i).

Flow shop F_m. There are m machines in series. Each job has to be processed on each of the m machines. All jobs have to follow the same route. After completion on one machine, a job joins the queue at the next machine. Usually, all queues are assumed to operate under the *First In First Out (FIFO)* discipline. If the *FIFO* discipline is in effect, the flow shop is referred to as a *permutation* flow shop and the β field includes the entry *prmu*.

Job shop J_m. In a job shop with m machines, each job has its own predetermined route to follow. A distinction is made between job shops in which a job may visits each machine at most once and job shops in which a job may visit each machine more than once. In the latter case, the β-field contains the entry *recrc* for *recirculation.*

Open shop O_m. There are m machines. Each job has to be processed on each of the m machines. However, some of these processing times may be zero.

Sequence dependent setup times s_{jk}. The s_{jk} represents the sequence dependent setup time between jobs j and k; s_{0k} denotes the setup time for job k if job k is first in the sequence and s_{j0} the cleanup time after job j if job j is last in the sequence.

Preemptions *prmp.* Preemptions imply that it is not necessary to keep a job on a machine, once started, until completion. The schedule is allowed to interrupt the processing of a job at any point in time and put a different job on the machine instead.

Precedence constraints *prec.* Precedence constraints may appear in a single machine or in a parallel machine environment, requiring that one or more jobs may have to be completed before another job is allowed to start its processing.

Completion time C_{ij}. The completion time of the operation of job j on machine i. The time job j exits the system is denoted by C_j.

Response time F_j. $F_j = C_j - r_j$.

Idle time I_j. $I_j = C_j - r_j - p_j$.

Lateness L_j. $L_j = C_j - d_j$.

Tardiness T_j. $T_j = \max(C_j - d_j, 0) = \max(L_j, 0)$.

Unit penalty U_j. $U_j = \begin{cases} 1, & \text{if } C_j > d_j \\ 0, & \text{otherwise} \end{cases}$

Examples of possible **objective functions** to be minimized are:

Makespan C_{max}. The makespan, define as $\max(C_1, \ldots, C_n)$, is equivalent to the completion time of the last job to leave the system. A minimum makespan usually implies a high utilization of the machine(s).

Maximum Lateness L_{max}. The maximum lateness is defined as $\max(L_1, \ldots, L_n)$. It measures the worst violation of the due dates.

Total weighted completion time $\sum w_j C_j$. The sum of the weighted completion times of the n jobs gives an indication of the total holding or inventory costs incurred by the schedule. The sum of the completion times if often referred to in the literature as the *flow time.*

Total weighted tardiness $\sum w_j T_j$.

2 Minimizing Schedule Length

Lemma 1. *A scheduling problem with the makespan objective without constraints is a special case of a scheduling problem with the total weight completion time objective and with precedence constraints.* $\alpha \,|\,|\, C_{max} \;\propto\; \alpha \,|\, prec \,|\, \sum w_j c_j$.

Proof. We want to prove that $\alpha \,|\,|\, C_{max}$ reduces to $\alpha \,|\, prec \,|\, \sum w_j c_j$. Assume we know how to solve $\alpha \,|\, prec \,|\, \sum w_j c_j$. Consider an instance of $\alpha \,|\,|\, C_{max}$ with n jobs j_i, $i = 1, \ldots, n$. Construct the precedence constraints in the following way (see Figure 2). Add a new job j_0 with length 0. j_0 can be started after all j_i, $i = 1, \ldots, n$ jobs are finished. Add weights $w_0 = 1$ and $w_j = 0$, $i = 1, \ldots, n$.

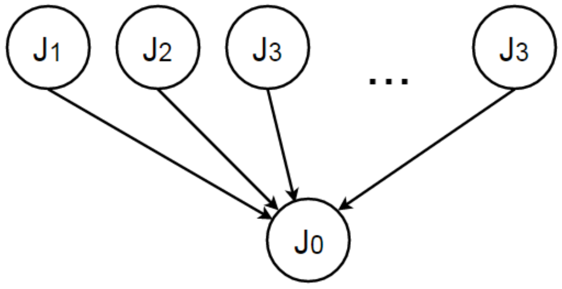

Figure 2: Precedence constraints for Lemma 1.

Notice that min $\sum_0^n w_j c_j =$ min $C_0 =$ min C_{max}, since $p_0 = 0$ and j_0 has to be processed after all j_i, $i = 1, \ldots, n$. The constructed problem is in the form $\alpha \,|\, prec \,|\, \sum w_j c_j$ and has the same value of objective function as the original problem. We conclude that $\alpha \,|\,|\, C_{max}$ is a special case of $\alpha \,|\, prec \,|\, \sum w_j c_j$.

□

The following algorithm and argument about its correctness is borrowed from [1, p. 106].

Algorithm 1. ***Minimizing Makespan with Preemptions***

Step 1 *Take the n jobs and process them one after another on a single machine in any sequence. The makespan is them equal to the sum of the n processing times and is less than or equal to mC^*_{max}. $C^*_{max} = \max\left(p_{max}, \sum_{j=1}^n p_j/m\right)$ and, p_{max} is the largest job.*

Step 2 *Take this single machine schedule and cut it into m parts. The first part constitutes the interval $[0, C^*_{max}]$, the second part the interval $[C^*_{max}, 2C^*_{max}]$, the third part the interval $[2C^*_{max}, 3C^*_{max}]$, and so on.*

Step 3 *Take as the schedule for machine 1 the processing sequence of the first interval; take as the schedule for machine 2 the processing sequence of the second interval; and so on.*

Proof. (*Correctness of the algorithm*)

The resulting schedule is feasible. Part of a job may appear at the end of the schedule for machine i while the remaining part may appear at the beginning of the schedule for machine $i + 1$. As preemptions are allowed and the processing time of each job is less or equal to C^*_{max}, such a schedule is feasible. C^*_{max} is a lower bound for $P_m\,|prmp|\,C_{max}$. As this schedule has $C_{max} = C^*_{max}$, it is also optimal. For an example see Figure 3.

$\square$

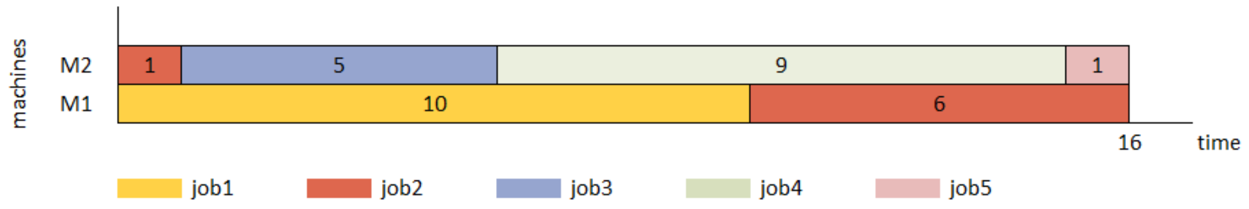

Figure 3: Optimal schedule for an instance of $P_2\,|prmp|\,C_{max}$.

Theorem 1. *The decision version of a scheduling problem with two machines and with the makespan objective function ($P_2\,||\,C_{max}$) is NP-complete.*

Proof. Consider the corresponding decision problem. If there a schedule of makespan $\le B = (1/2)\sum_{j=1}^{n} p_j$? If we are given a schedule we can verify in polynomial time whether its makespan $\le B$, given a polynomial encoding of the problem.

We will show reduction from PARTITION. Recall the definition of the PARTITION problem. Given positive integers $a_1, \ldots, a_t$ and $B = (1/2)\sum_{j=1}^{t} a_j$, do there exist two disjoint subsets S_1 and S_2 such that $\sum_{j\in S_i} a_j = B$ for $i = 1, 2$? [1, p. 538]

Assume we are given an instance of the PARTITION problem. We form a scheduling problem as follows, $n = t$, $p_j = a_j$, $j = 1, \ldots, n$. We will show that there exists a schedule with an objective value less than or equal to $(1/2)\sum_{j=1}^{n} p_j$ if and only if there exists a solution for the PARTITION problem.

If there is a solution to the PARTITION problem, then schedule jobs with indexes from S_1 on machine 1, and jobs with indexes from S_2 on machine 2 in any order. This schedule is feasible since we don't have preemption and it satisfies $C_{max} \le B$.

Now suppose that there is a schedule with an objective value less than or equal to $(1/2)\sum_{j=1}^{n} p_j$. Recall that B is also a lower bound for this scheduling problem. We conclude that indexes of jobs on servers represent S_i, $i = 1, 2$ and there exists a solution to the PARTITION problem.

$\square$

Theorem 2. ***List scheduling for $P_m\,||\,C_{max}$.***
List scheduling for $P_m\,||\,C_{max}$ gives 2-approximation.

Proof. We adopt the notations from Algorithm 1 and [1, p. 94-95].

We are given n jobs j_i, $i = 1, \dots, n$. At $t = 0$ Assign first two job to the machines. After that, whenever a machine is freed, the next job on the list is put on the machine. See Figure 4.

We want to show $\frac{C_{max}(LIST)}{C_{max}(OPT)} \leq 2$. $C_{max}(LIST)$ denotes the makespan of the List schedule and $C_{max}(OPT)$ denotes the makespan of the optimal schedule. Let T be the latest time when all machines are busy. There is no job that starts after T. To prove the last statement we assume that there is a job j_i that starts after T, then this violates the list scheduling greedy approach described above, since by definition of T there must be a machine that finishes processing a job at T. Let p_{avg} be $\frac{\sum_{j=1}^{n} p_j}{m}$. Notice that $T = p_{avg}$, when all machines stopped processing all jobs exactly at time T. If there are some machines that are still working on some jobs after time T, then p_{avg} is larger in this case than the previous, since by the definition of T all machines are busy at T, so $T < p_{avg}$. We conclude that in general $T \leq p_{avg}$.

$C_{max}(LIST) \leq T + p_{max} \leq p_{avg} + p_{max} \leq 2C^*_{max} \leq C_{max}(OPT)$.

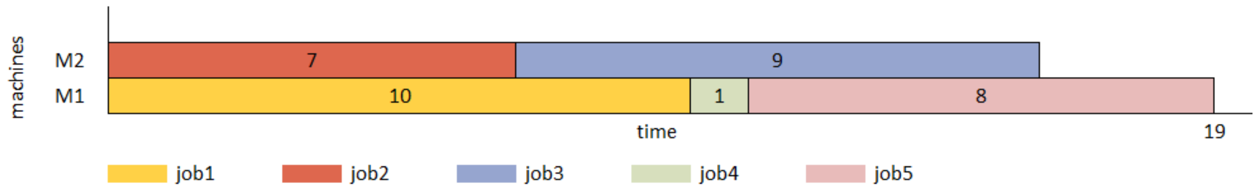

Figure 4: Example of a list scheduling algorithm for an instance of $P_2 \,||\, C_{max}$.

We can prove $(2 - 1/m)$-approximation using the idea from [1, p.95]:

$$C_{max}(LIST) \leq p_{max} + \frac{\sum_{j=1}^{n-1} p_j}{m} = p_{max}\left(1 - \frac{1}{m}\right) + \frac{\sum_{j=1}^{n} p_j}{m}.$$

It follows that

$$\frac{C_{max}(LIST)}{C_{max}(OPT)} \leq \frac{p_{max}(1 - 1/m)}{C_{max}(OPT)} + \frac{\sum_{j=1}^{n} p_j/m}{C_{max}(OPT)} \leq 1 - \frac{1}{m} + 1 = 2 - \frac{1}{m}.$$

□

Theorem 3. ***LPT rule for $P_m \,||\, C_{max}$.***
LPT rule for $P_m \,||\, C_{max}$ gives 4/3-approximation.

Proof. This proof contains ideas from [1, p.94-96].

The Longest Processing Time first (LPT) rule assigns at $t = 0$ the m largest jobs to the m machines.

After that, whenever a machine is freed, the longest job among those not yet processed is put on the machine. This heuristic ties to place the shorter jobs toward the end of the schedule, where they can be used for balancing the loads. [1, p. 94]

By contradiction. Assume that there exists one or more counterexamples where the ratio is strictly larger than 4/3. If more than one such counterexample exist, there must exist an example with the smallest number of jobs. Consider this smallest counterexample and assume it has n jobs. This smallest counterexample has a useful property: under LPT, the shortest job is the last job to start its processing and also the last job to finish its processing. Also, if this job is not the last to complete its processing, the deletion of this smallest job will result in a counterexample with fewer jobs. Assume p_n is the shortest job. [1, p. 95]

Let T be defined the same as in the Theorem 2. $C_j \leq T + p_{min}$, for $j = 1, \dots, n \implies C_{max}(LPT) \leq T + p_{min}$. There are two possible cases:

1. $p_{min} \leq \frac{1}{3}C_{max}(OPT) \rightarrow C_{max} \leq \frac{4}{3}C_{max}(OPT)$.
2. $p_{min} > \frac{1}{3}C_{max}(OPT)$. In this case there has to be fewer than three jobs per machine and LPT rule gives the optimal schedule. It can be proven by induction.

We conclude that there are no examples of the problem for which the ratio is strictly larger than 4/3. Contradiction to the assumption of the existence of such examples. The statements is proved.

See Figure 5 for an example of the application of the LPT rule.

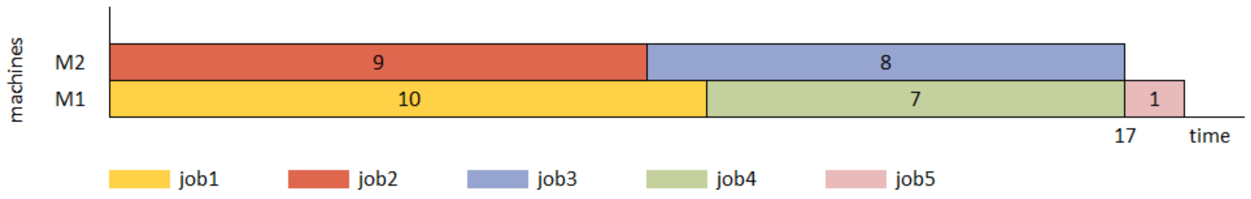

Figure 5: Example of the application of the LPT rule for an instance of $P_2 \,||\, C_{max}$.

□

Theorem 4 and Example 1 are borrowed from [1, p. 95-96].

Theorem 4. ***LPT rule for $P_2 \,||\, C_{max}$.***

LPT rule for $P_2 \,||\, C_{max}$ gives $\frac{4}{3} - \frac{1}{3m}$-approximation.

Proof. Similarly to Theorem 3 we prove the statement by contradiction. Assume there exists the smallest counterexample. It has a property: under LPT, the shortest job is the last job to start its processing and also the last job to finish its processing. Let p_n be the shortest job. It follows that

$$C_{max}(LPT) - p_n \le \frac{\sum_{j=1}^{n-1} p_j}{m}.$$

The right-hand side is an upper bound on the starting time of the shortest job. This upper bound is achieved when scheduling the first $n-1$ jobs according to LPT results in each machine having exactly the same amount of processing to do.

$$C_{max}(LPT) \le p_n + \frac{\sum_{j=1}^{n-1} p_j}{m} = p_n\left(1 - \frac{1}{m}\right) + \frac{\sum_{j=1}^{n} p_j}{m}.$$

Since

$$C_{max}(OPT) \ge \frac{\sum_{j=1}^{n} p_j}{m},$$

the following series of inequalities holds for the counterexample:

$$\frac{4}{3} - \frac{1}{3m} < \frac{C_{max}(LPT)}{C_{max}(OPT)} \le \frac{p_n\,(1-1/m)}{C_{max}(OPT)} + \frac{\sum_{j=1}^{n} p_j/m}{C_{max}(OPT)} \le \frac{p_n\,(1-1/m)}{C_{max}(OPT)} + 1.$$

Thus

$$\frac{4}{3} - \frac{1}{3m} < \frac{p_n\,(1-1/m)}{C_{max}(OPT)} + 1$$

and

$$C_{max}(OPT) < 3p_n.$$

This implies that for the smallest counterexample the optimal schedule can result in at most two jobs on each machine. In this case the LPT schedule is optimal and the ration of the two makespans is equal to one. This contradiction completes the proof of the theorem.

□

Example 1. ***A worst case example of LPT rule for $P_m \,||\, C_{max}$.***

Consider four parallel machines and nine jobs whose processing times are given in the following table:

jobs	1	2	3	4	5	6	7	8	9
p_j	7	7	6	6	5	5	4	4	4

Scheduling the jobs according to LPT results in a makespan of 15, while the optimal schedule has a makespan of 12. This particular instance is thus a worst case when there are four machines in parallel (see Figure 6).

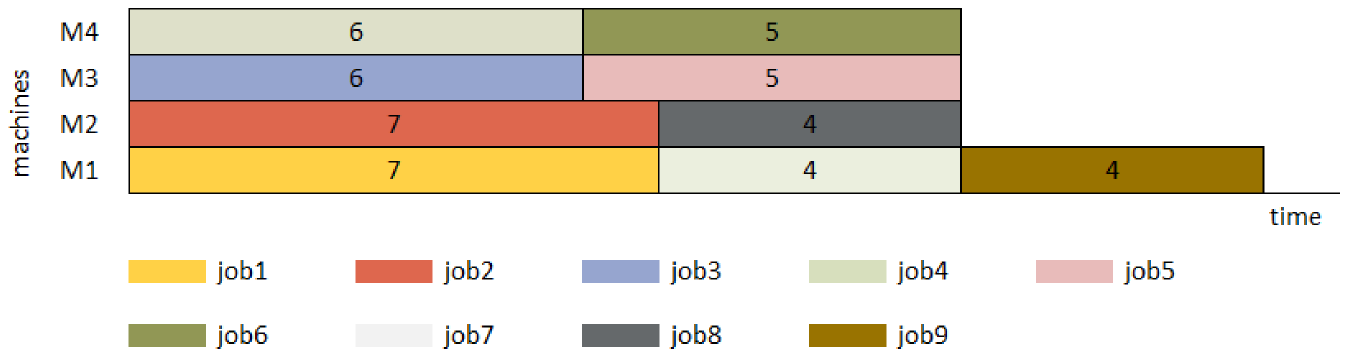

Figure 6: A worst case example of LPT

References

[1] Pinedo, M. *Scheduling. Theory, algorithms, and systems.*. Upper Saddle River, New Jersey: Prentice Hall, 2002.