2 Longest Processing Time Rule (LPT)

The example of the previous section shows a drawback of list scheduling – the longest jobs should be processed earlier. In fact that precisely is the LPT rule: Sort the jobs in decreasing order of their processing times and run the list scheduling algorithm with this list.

Theorem 2.1. *LPT returns a schedule S such that $C^S_{max} \leq (4/3 - 1/3m)OPT$.*

Proof. Suppose, as in the previous proof, l is the last job to finish. We can assume that l is actually the last job. The reason is if not, then consider the jobs from $\{1, \ldots, l\}$ (assuming $p_1 \geq p_2 \geq \ldots$). LPT will return the same schedule and thus will have the same makespan as with $\{1, \ldots, n\}$ jobs. The optimum only decreases. Thus, if we show that the makespan on LPT on the first l jobs is within $4/3$ of the optimum of the first l jobs, we will be done. Thus, we will assume that $p_l = p_{min}$.

Lemma 2.2. *If $p_{min} > OPT/3$, then $C^S_{max} = OPT$.*

Proof. We actually show that if no optimum schedule processes more than two jobs per machine, then LPT will return the optimal schedule. This will imply the lemma. (Why?)

To see this, for every machine i, let i_1 and i_2 be the two jobs that the optimum schedules on it (i_2 being empty if only one job is scheduled) and suppose $p_{i_1} \geq p_{i_2}$. Also, order the machines from 1 to m such that $p_{i_1} \geq p_{i'_1}$ if $i < i'$. That is the largest job goes on the first machine, the second largest on the second machine, and so on. We claim that we may assume $p_{i_2} \leq p_{i'_2}$ for all $i < i'$. If not, a simple interchange argument will not decrease C_{max}.

We now claim that LPT returns the same schedule. Note that if LPT also puts at most two jobs on a machine then it precisely returns the above schedule. Suppose LPT processes three jobs on a machine. Let j be the first job which is put on a machine with two jobs already. Note that there at most $2m$ jobs. Call a job a *loner* if it is the only job processed by a machine. Note that if LPT processes three jobs on a machine there must be a loner job in LPT which is not a loner in the OPT schedule. Furthermore, when the job j is being processed by LPT, the loner must have already been processed. Therefore, the loner has processing time at least $2p_{min} > 2OPT/3$ (for otherwise j would be processed with the loner). Since it is not a loner in OPT, the job which goes with it in the optimum schedule must have processing time $< OPT/3$ contradicting the premise. □

Now we are almost done. Since the job which finishes last, l has $p_l = p_{min}$, if $p_l > OPT/3$, we get $C^S_{max} = OPT$. Otherwise, from the inequality in the previous proof

$$C^S_{max} \leq OPT + p_l(1 - 1/m) \leq (4/3 - 1/3m)OPT$$

□

Let us see that the algorithm is tight for $m = 2$ machines. Consider the list of five jobs with processing times $\{3, 3, 2, 2, 2\}$. LPT will return a schedule with $C^S_{max} = 7$, while $OPT = 6$ with jobs 1 and 2 processed on machine 1, and rest of the jobs on machine 2.

Theorem 1 *LPT is a 4/3-approximation.*

Proof Order the n jobs such that $p_1 \leq p_2 \leq \cdots \leq p_n$. Without loss of generality, we assume job n finishes last. If it did not finish last, removing it would not change the schedule of LPT, but only decrease OPT. Thus, if the bound holds for the case where n finishes last, it must also hold for cases where it does not, i.e., OPT is worse: $LPT \leq 4/3 \cdot OPT \wedge OPT' \geq OPT \Rightarrow LPT \leq 4/3 \cdot OPT'$. So, the makespan is

$$START_n + p_n \leq \frac{1}{m} \sum_{i \neq n} p_i + p_n \leq OPT + p_n.$$

Using the same technique as in the case of LS gets us an approximation ratio of 2. Instead of refining our bounds on OPT, we will try to bound p_n. Clearly, if $p_n \leq OPT/3$, then LPT is 4/3-OPT. We only have to show that larger p_n values don't violate our bounds. We claim if $p_n > OPT/3$, then LPT gives an optimal schedule.

Claim 2 *If $p_n > OPT/3$, LPT is optimal.*

Proof We prove this by contradiction. Assume the l-th job completes after time OPT. We consider the schedule before the l-th job. It is easy to see that in this schedule each machine must be assigned at least one job and at most two (since otherwise the length of the schedule would already be bigger than OPT). Now order the machines and select i such that the machines M_1 through M_i have a single job assigned to them and machines M_{i+1} through M_m have two. Let's call the jobs on the first i machines long jobs, the ones on the remaining ones short jobs (l is also considered a short job). LPT would schedule l so that it finishes after OPT. This means l can't be scheduled on a machine with a long job, and since l is smaller than any job that is already scheduled (by the rules of LPT) no short or long job can be scheduled with another long job. Hence, to not violate OPT any schedule needs i machines to schedule the long jobs and the short jobs must be scheduled on the remaining machines ($m - i$ many). However, as there are $2(m - i) + 1$ short jobs and each machine can execute at most two jobs, this is clearly impossible and we conclude that there is no schedule better than LPT and our claim holds. ■

We have shown that LPT returns either an optimal schedule or one with length at most 4/3OPT, thus LPT is indeed a 4/3-approximation algorithm. ■

The 4/3 bound is tight, an infinite family of instances showing this is given below.

Instance: we are given m machines, and $2m+1$ jobs. There are three jobs with processing time m, and 2 jobs with processing times $m+1, m+2, \ldots, 2m-1$ each. In case of LPT, all but one of the machines get two jobs with a total processing time of $3m - 1$, and a single machine gets three jobs with a total of $4m - 1$ processing time. Thus, the makespan is $4m - 1$. OPT schedules the three m jobs on a single machine, and the remaining jobs on the remaining $m-1$ machines, such that each of those machines get jobs with a total processing time of $3m$, thus the makespan of OPT is $3m$. As m grows towards infinity the approximation ratio approaches 4/3.

There is an even better approximation algorithm, a so-called PTAS[1], for this problem, but it is not covered by this lecture.

1.2 Set-Cover Problem

The next problem we consider is the Set-Cover Problem. We are given a universe of n elements $U = \{e_1, e_2, \cdots, e_n\}$, a collection of subsets of the universe $T = \{s_1, s_2, \cdots, s_k\}$, and a cost function

[1]Polynomial Time Approximation Scheme: For any $\epsilon > 0$ we can find a $(1 + \epsilon)$-approximate solution in time that is polynomial for any fixed ϵ (for example, the running time may be $O(n^{1/\epsilon})$).