Scheduling Independent Tasks

Setup: We are given n tasks, numbered 1, 2, ..., n, with running times t_1, t_2, ..., t_n respectively, and m processors. We would like to schedule all n task on the m processors so as to *minimize the finish time of the last job*.

We will see two heuristics methods for solving this problem.

1. List Schedule

A simple strategy is to assign the jobs sequentially, each job going to the processor that has the smallest load at that time.

e.g. Suppose we have 6 tasks with running times 10, 5, 4, 11, 3, 5, and 2 processors. Using to the list schedule heuristic, we get:

1	$t_1 = 10$		$t_5 = 3$	$t_6 = 5$
2	$t_2 = 5$	$t_3 = 4$	$t_4 = 11$	

Using list scheduling, we get 2 units of idle time on processor 1.
Note that the optimal schduling is

1	$t_1 = 10$	$t_2 = 5$	$t_3 = 4$
2	$t_4 = 11$	$t_5 = 3$	$t_6 = 5$

The optimal scheduling leaves no idle time on either processor, and finishes one time unit earlier. We look at performance by considering the ratio between the schedule finish time of the list schedule and the optimal schedule.
Let t_{LIST} denote the length of the list schedule (where the length is the finish time of the last job).
Let t_{OPT} denote the length of the optimum schedule.
Then,

$$\frac{t_{\text{LIST}}}{t_{\text{OPT}}} = \frac{20}{19}$$

Now, suppose we were given the tasks in a different order, say 10, 5, 5, 4, 3, 11. Using the list schedule heuristic, we now get

1	$t_1 = 10$		$t_4 = 4$	
2	$t_2 = 5$	$t_3 = 5$	$t_5 = 3$	$t_6 = 11$

Processor 1 now has 10 units of idle time, and the finish time of the last job is 24 units of time. For this particular sequence, the performance compared to the optimal is

$$\frac{t_{\text{LIST}}}{t_{\text{OPT}}} = \frac{24}{19}$$

We would like to know something about how well list schedule performs in general.

Theorem 1
For an independent task scheduling problem with m processors, the length t_{LIST} of the schedule obtained by the list schedule heuristic satisfies

$$\frac{t_{\text{LIST}}}{t_{\text{OPT}}} \leqslant \left(2 - \frac{1}{m}\right)$$

Proof
Let t be the running time of the last job to finish in the list schedule. Before time $t_{\text{LIST}} - t$, all processors must be busy, otherwise the job would have been scheduled earlier. So, up until time $t_{\text{LIST}} - t$, none of the processors were idle at any time. So,

$$\left(\sum_{i=1}^{n} t_i\right) - t \;\geq\; m(t_{\text{LIST}} - t) \quad (1)$$

Also, the optimum scheduling can never be better than the total time taken by all the tasks, divided by the number of processors available. That is,

$$t_{\text{OPT}} \;\geq\; \frac{1}{m}\sum_{i=1}^{n} t_i$$

$$\Leftrightarrow m \cdot t_{\text{OPT}} \;\geq\; \sum_{i=1}^{n} t_i$$

Plugging into (1), we get

$$\begin{aligned} & m(t_{\text{LIST}} - t) && \leq && m \cdot t_{\text{OPT}} - t \\ \Leftrightarrow \quad & m \cdot t_{\text{LIST}} && \leq && m \cdot t_{\text{OPT}} - t + mt \\ \Leftrightarrow \quad & t_{\text{LIST}} && \leq && t_{\text{OPT}} + \frac{m-1}{m} t \qquad (2) \end{aligned}$$

We also know that the length of any schedule, including the optimum, must be at least as long as the length of any task, so

$$t_{\text{OPT}} \;\geq\; t$$

Plugging into (2), we get that

$$\begin{aligned} & t_{\text{LIST}} && \leq && t_{\text{OPT}} + \frac{m-1}{m} t \\ \Leftrightarrow \quad & t_{\text{LIST}} && \leq && t_{\text{OPT}} + \frac{m-1}{m} t_{\text{OPT}} \\ \Leftrightarrow \quad & \frac{t_{\text{LIST}}}{t_{\text{OPT}}} && \leq && 1 + \frac{m-1}{m} \\ \Leftrightarrow \quad & \frac{t_{\text{LIST}}}{t_{\text{OPT}}} && \leq && 2 - \frac{1}{m} \end{aligned}$$

□

This bound is actually a tight upper bound. Consider the case with $m = 6$ processors and 11 tasks with running times 5, 5, 5, 5, 5, 1, 1, 1, 1, 1, 6, given in that order.

Then list schedule will give us

1	5					6
2	5					
3	5					
4	5					
5	5					
6	1	1	1	1	1	

Clearly the optimum schedule has length 6, and so we get that

$$\frac{t_{\text{LIST}}}{t_{\text{OPT}}} = \frac{11}{6} = 2 - \frac{1}{6}$$

2. List Decreasing

We can note looking at examples above that we seem to get poorer results when we finish on very long jobs. In the last example, the fact that we kept 6 for last caused our solution to be rather suboptimal.
This observation leads to the list decreasing heuristic: given n tasks with running times t_1, t_2, ..., t_n, we sort the tasks by decreasing runtime to get $t_1 \geqslant t_2 \geqslant \ldots \geqslant t_n$, and then apply the list schedule heuristic as described above.

Lemma

If $t_n > \frac{t_{\text{OPT}}}{3}$ then the list decreasing heuristic finds the optimum schedule, i.e. $t_{\text{LIST}} = t_{\text{OPT}}$

Proof

Exercise.

Theorem 2

For an independent task scheduling problem with m processors, the length t_{LIST} of the schedule obtained by the list schedule heuristic satisfies

$$\frac{t_{\text{LIST}}}{t_{\text{OPT}}} \leqslant \left(\frac{4}{3} - \frac{1}{3m} \right)$$

Proof

First, we will assume that job n finishes last. We have two cases

- $t_n > \frac{t_{\text{OPT}}}{3}$

  By the lemma above, we know that $t_{\text{LIST}} = t_{\text{OPT}} > \frac{t_{\text{OPT}}}{3}$

- $t_n \leqslant \frac{t_{\text{OPT}}}{3}$ (3)

As shown in the proof of theorem 1, in equation (2), we have

$$L \leqslant t_{\mathrm{OPT}} + \frac{m-1}{m} t$$

where t is the last job to finish, in this case t_n. Plugging in (3), we get

$$\begin{aligned} L &\leqslant t_{\mathrm{OPT}} + \frac{m-1}{m} \cdot \frac{t_{\mathrm{OPT}}}{3} \\ \Leftrightarrow \quad \frac{L}{t_{\mathrm{OPT}}} &\leqslant 1 + \frac{m-1}{3m} \\ \Leftrightarrow \quad \frac{L}{t_{\mathrm{OPT}}} &\leqslant \frac{4}{3} + \frac{1}{3m} \end{aligned}$$

Note that if it is not the case that t_n finsihed last, then we may remove it from the list without changing the finish time of the list schedule. We continue to remove jobs until the last job in the list does finish last, and then apply the same proof detailed above.

□