IEOR 8100: Scheduling Algorithms 6 October, 2016

Lecture 8 – Sum of Completion Times with Release Dates

Instructor: *Clifford Stein* Scribe: *Nishanth Mohan*

1 Recap

Last time, we studied the notion of sum of completion times as a metric and proved that in the non-preemptive case, sorting jobs by shortest processing time first (SPT) solves $(1 \mid\mid \sum_j C_j)$ exactly in polynomial time. In the case of unrelated machines $(R \mid\mid \sum_j C_j)$, we reduced it to a matching problem which was also solved in polynomial time. This time, we consider a slightly more general variant of $(1 \mid\mid \sum_j C_j)$ with release dates i.e. $(1 \mid r_j \mid \sum_j C_j)$.

2 $(1 \mid r_j \mid \sum_j C_j)$ - Hardness

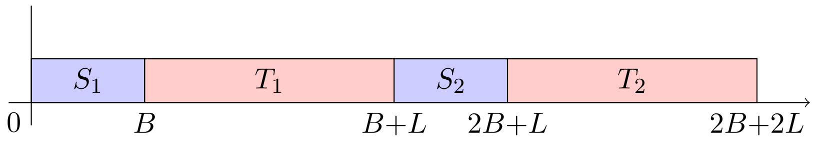
Claim. $(1 \mid r_j \mid \sum_j C_j)$ *is NP-hard.*

Proof. (Sketch) We reduce from 3-PARTITION. Recall that the statement of 3-PARTITION is as follows: given $3n$ numbers $x_1, x_2, \dots x_{3n}$ and a target B with $\sum_i x_i = nB$ and $B/4 \le x_i \le B/2$, does there exist a partition of these numbers into n groups $S_1, S_2, \dots S_n$ each with three elements such that the sum of each group S_i equals B?

Given an instance of 3-PARTITION, we construct an instance of $(1 \mid r_j \mid \sum_j C_j)$ where:

- For each x_i, we create a job J_i with $p_i = x_i$ and $r_i = 0$
- For each $t = 3n + k$ with $k \in \{1, 2, \dots n\}$, we create a set T_k of L jobs (where $L \gg B$), each with processing time 1 and release date $kB + (k-1)L$ i.e. $T_k = \{J_{k,1}, J_{k,2} \dots J_{k,L}\}$ with $r_{k,i} = kB + (k-1)L$ and $p_{k,i} = 1$.

Given a yes-instance of 3-PARTITION, there exists a schedule for $(1 \mid r_j \mid \sum_j C_j)$ like so:



Notice that each set T_k finishes at time $kB + kL$ and that in each group S_i, the 3 jobs finish in increasing order of processing times (as is consistent with SPT). Let S_{nd} be the set of non-dummy jobs (those with processing times p_k) and let S_d be the set of dummy jobs (those with processing times $p_{k,i}$). We have

$$\sum_{j \in S_{nd}} C_j \approx \frac{3(B+L)n^2}{2} \qquad \sum_{j \in S_d} C_j \approx \frac{L^2 n^2}{2}$$

Since $\left(\sum_{j\in S_d} C_j\right) \gg \left(\sum_{j\in S_{nd}} C_j\right)$, we are better off doing the dummy jobs as they are released. Further, the non-dummy jobs can be placed exactly in the "gaps" between the dummy jobs and moving them further back would only increase the sum of completion times. Thus, the schedule above is optimal.

Now, given a no-instance of 3-PARTITION, there exists a "gap" in a group S_i that can't be filled by a particular non-dummy job p_j. We have two options:

- Move p_j back without affecting the rest of the schedule. In this case, we increase p_j's completion time and thus increase the overall sum of completion times.
- Push some dummy jobs back which also increases the overall sum of completion times.

In either case, we don't have an optimal schedule (several pages of tedious algebra confirm this). $\square$

3 $(1 \mid r_j \mid \sum_j C_j)$ - Approximation Algorithms

In a previous lecture, we saw that sorting jobs by shortest remaining processing time first (SPRT) preemptively was optimal for $(1 \mid \text{pmtn}, r_j \mid \sum_j C_j)$. In fact, if C_j^P is the completion time of job j in the SRPT schedule and OPT is the sum of completion times in an optimal non-preemptive schedule, then

$$\sum_{i=1}^{n} C_j^P \leq OPT$$

It is thus reasonable to expect that we might glean something useful from the SRPT schedule for a $(1 \mid r_j \mid \sum_j C_j)$ instance.

3.1 `Extract-from-Preempt`

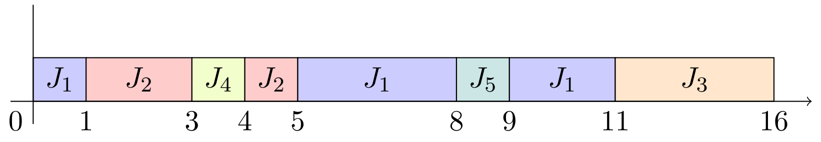
We now present a 2-approximation for $(1 \mid r_j \mid \sum_j C_j)$, `Extract-from-Preempt`, which is as follows:

1. Apply SRPT to obtain the optimal preemptive schedule P.
2. Reorder the jobs in increasing order of their completion times in P i.e. $j < i$ if $C_j^P < C_i^P$.
3. Schedule the jobs non-preemptively in this order to get the final schedule N.

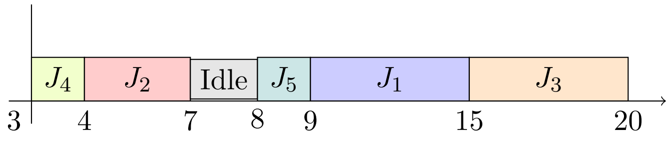
As an example, consider the following instance:

j	r_j	p_j
1	0	6
2	1	3
3	1	5
4	3	1
5	8	1

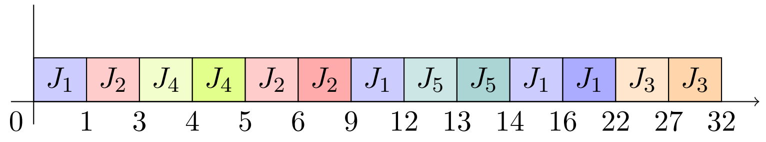
Running SRPT on this instance produces the following schedule:



while running `Extract-from-Preempt` produces the following schedule:



Claim. *`Extract-from-Preempt` is a 2-approximation for* $(1 \mid r_j \mid \sum_j C_j)$.

> **Intuition for proof:** Consider a schedule Z in which when J_i completes in P, we add another copy of p_i (i.e. schedule it non-preemptively) and move everything after up by p_i. Schedule Z for the instance above would look like this:
>
> 
>
>
> The extra p_i blocks are shaded in a darker color in the picture above compared to their preemptive counterparts. Note that before job J_i starts in Z, there are 2 copies of p_k if J_k finishes before J_i starts (the preemptive blocks and the extra block) and (possibly) some other preemptive blocks for jobs that haven't completed by then. Thus, removing the preemptive components in Z results in a valid non-preemptive schedule Z' that has at most twice the sum of completion times as P. All `Extract-from-Preempt` does is push the remaining blocks in Z' as far back as possible while respecting release dates.

Proof. Recall that we schedule jobs in order of their preemptive completion times. This gives us the following observations:

(i) $C_j^N \leq \max\{r_1, \ldots r_j\} + \sum_{k=1}^{j} p_k$ since all the jobs are available after $\max\{r_1, \ldots r_j\}$.

(ii) $C_j^P \geq \sum_{k=1}^{j} p_k$ since job j can only finish once all of the jobs before it have completed (by our ordering).

(iii) $C_j^P \geq \max\{r_1, \ldots r_j\}$ since job j can only start when all of the jobs until job j have been released (once again by our ordering).

Using the observations above, we have

$$C_j^N \leq \max\{r_1, \ldots r_j\} + \sum_{k=1}^{j} p_k \qquad (\ldots \text{ by observation (i)})$$

$$\Rightarrow C_j^N \leq 2C_j^P \qquad (\ldots \text{ by observations (ii) and (iii)})$$

$$\Rightarrow \sum_{j=1}^{n} C_j^N \leq 2\left(\sum_{j=1}^{n} C_j^P\right)$$

$$\Rightarrow \sum_{j=1}^{n} C_j^N \leq 2 \cdot OPT \qquad (\ldots \text{ since } \sum_j C_j^P \leq OPT)$$

□

3.2 `Schedule-by-α-points`

Definition (α-point)**.** *The α-point of a job j in a schedule is the first time at which αp_j fraction of the job has completed for $0 < \alpha \leq 1$.*

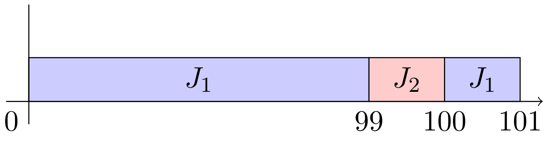
Consider the following algorithm, `Schedule-by-α-points`, which non-preemptively schedules jobs in increasing order of their preemptive α-points:

1. Apply SRPT to obtain the optimal preemptive schedule P.
2. Reorder the jobs in increasing order of their α-points.
3. Schedule the jobs non-preemptively in this order.

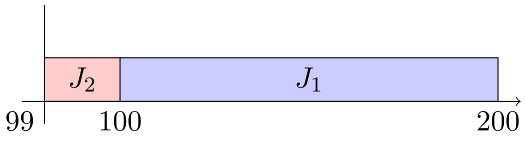
Note that `Extract-from-Preempt` is equivalent to `Schedule-by-α-points` when $\alpha = 1$. Scheduling by α-points can avoid certain bad cases. Consider the following instance:

j	r_j	p_j
1	0	100
2	99	1

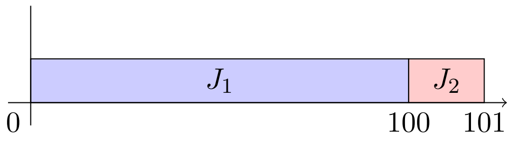
SRPT yields the following schedule with $\sum C_j = 200$:



However, `Extract-from-Preempt` yields the following schedule with $\sum C_j = 299$:



Finally, `Schedule-by-α-points` with $\alpha = 1/2$, yields the following schedule with $\sum C_j = 201$:



Claim. *`Schedule-by-a-points` is a $(1 + 1/\alpha)$-approximation for $(1 \mid r_j \mid \sum_j C_j)$.*

Proof. As before, we have the following observations:

- $C_j^N \leq \max\{r_1, \dots r_j\} + \sum_{k=1}^{j} p_k$
- $C_j^P \geq \alpha \left(\sum_{k=1}^{j} p_k \right)$
- $C_j^P \geq \max\{r_1, \dots r_j\}$

Thus,

$$C_j^N \leq \left(1 + \frac{1}{\alpha}\right) C_j^P$$

□

This bound is *tight* for certain cases. We thus do no better objectively than `Extract-from-Preempt` in the worst case since $(1 + 1/\alpha)$ is maximized when $\alpha = 1$. In order to avoid this, we can pick a random α so that we do well on "most" inputs. This approach is addressed in the next section.

3.3 `Schedule-by-random-α`

Consider the following algorithm, `Schedule-by-random-α`, which randomizes the choice of α in the previous algorithm:

1. Pick α' according to some PDF f on $(0, 1]$.
2. Run `Schedule-by-α-points` on the input with $\alpha = \alpha'$.

In order to analyze this algorithm, we introduce some notation:

- Let $S_i^P(\beta)$ be the set of jobs that complete *exactly* β-fraction of their processing before C_i^P (P is the preemptive schedule generated by SRPT).
- Let T_i be the idle time before C_i^P.
- Let $p\left(S_i^P(\beta)\right) = \sum_{j \in S_i^P(\beta)} p_j$.

We have

$$\begin{aligned} C_i^P &= T_i + \sum_{0 \leq \beta \leq 1} \beta p\left(S_i^P(\beta)\right) \\ &= T_i + \left(\sum_{0 \leq \beta \leq \alpha} \beta p\left(S_i^P(\beta)\right)\right) + \left(\sum_{\alpha < \beta \leq 1} \beta p\left(S_i^P(\beta)\right)\right) \end{aligned}$$

Let C_j^α be the completion time of job j in the schedule generated by `Schedule-by-random-α`.

Claim. *We have*

$$C_i^\alpha \leq T_i + \left[\sum_{0 \leq \beta \leq \alpha} \beta p\left(S_i^P(\beta)\right)\right] + \left[(1 + \alpha)\left(\sum_{\alpha < \beta \leq 1} \beta p\left(S_i^P(\beta)\right)\right)\right]$$

Proof. (Sketch) Similar to the analysis in the `Extract-from-Preempt` algorithm, for all $\beta \geq \alpha$, we can insert job j non-preemptively after its α-point and throw the remaining $(1 - \alpha)$-fraction of it away. Note that this is a non-preemptive schedule that respects release dates. Thus, we pay a factor of 1 for the non-preemptive component and a factor of α for the preemptive component leading to a total contribution of $(1 + \alpha)$. □

Claim. *We have*

$$\mathbf{E}\left[C_i^\alpha\right] \leq C_i^P \left(1 + \max_\beta \left(\int_0^1 f(\alpha) \left\lceil \frac{1+\alpha-\beta}{\beta} \right\rceil \, d\alpha\right)\right)$$

Proof. Next time. $\square$

Note that

$$\min_f \max_\beta \left(\int_0^1 f(\alpha) \left\lceil \frac{1+\alpha-\beta}{\beta} \right\rceil \, d\alpha\right) = \frac{e}{e-1} \approx 1.58$$

for $f(\alpha) = \frac{e^\alpha}{e-1}$, so `Schedule-by-random-α` is a 1.58-approximation algorithm for $(1 \mid r_j \mid \sum_j C_j)$ in expectation.