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 $1 \mid \sum C_j$ exactly in polynomial time. In the case of unrelated machines ($R \mid \sum C_j$), we reduced it to a matching problem which was also solved in polynomial time. This time, we consider a slightly more general variant of $(1 \mid \sum C_j)$ with release dates i.e. $(1 \mid r_j \mid \sum C_j)$.

2 $(1 \mid r_j \mid \sum C_j) - \text{Hardness}$

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

**Proof. (Sketch)** We reduce from 3-PARTITION. Recall that the statement of 3-PARTITION is as follows: given 3n numbers $x_1, x_2, \ldots, x_{3n}$ and a target $B$ with $\sum_i x_i = nB$ and $B/4 \leq x_i \leq B/2$, does there exist a partition of these numbers into n groups $S_1, S_2, \ldots, S_n$ 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 $(1 \mid r_j \mid \sum C_j)$ 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 $k \in \{1, 2, \ldots, n\}$, 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. $T_k = \{J_{k,1}, J_{k,2}, \ldots, J_{k,L}\}$ 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 $(1 \mid r_j \mid \sum C_j)$ 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

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

### 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 \((1 \mid \text{pttn}, r_j \mid \sum_j C_j)\). In fact, if \( C^P_j \) 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

\[
\sum_{i=1}^{n} C^P_j \leq OPT
\]

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 \( C^P_j < C^P_i \).

3. Schedule the jobs non-preemptively in this order to get the final schedule \( N \).

As an example, consider the following instance:

<table>
<thead>
<tr>
<th>( j )</th>
<th>( r_j )</th>
<th>( p_j )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>8</td>
<td>1</td>
</tr>
</tbody>
</table>

Running SRPT on this instance produces the following schedule:
while running \textbf{Extract-from-Preempt} produces the following schedule:

\begin{center}
\begin{tabular}{|c|c|c|c|c|c|c|}
\hline
3 & 4 & 7 & 8 & 9 & 15 & 20 \\
\hline
J_4 & J_2 & Idle & J_5 & J_1 & J_3 & \\
\hline
\end{tabular}
\end{center}

\textbf{Claim.} \textit{Extract-from-Preempt} is a 2-approximation for \((1 \mid r_j \mid \sum_j C_j)\).

\textbf{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:

\begin{center}
\begin{tabular}{|c|c|c|c|c|c|c|c|c|c|c|c|}
\hline
0 & 1 & 3 & 4 & 5 & 6 & 9 & 12 & 13 & 14 & 16 & 22 & 27 & 32 \\
\hline
J_1 & J_2 & J_1 & J_4 & J_2 & J_1 & J_5 & J_1 & J_1 & J_3 & J_3 & \\
\hline
\end{tabular}
\end{center}

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 \textbf{Extract-from-Preempt} does is push the remaining blocks in \(Z'\) as far back as possible while respecting release dates.

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

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

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

(iii) \(C^P_j \geq \max\{r_1, \ldots, r_j\}\) 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

\[ C^N_j \leq \max\{r_1, \ldots, r_j\} + \sum_{k=1}^j p_k \]

\[ \Rightarrow C^N_j \leq 2C^P_j \]

\[ \Rightarrow \sum_{j=1}^n C^N_j \leq 2 \left( \sum_{j=1}^n C^P_j \right) \]

\[ \Rightarrow \sum_{j=1}^n C^N_j \leq 2 \cdot OPT \]

\[ \quad (\ldots \text{since } \sum_j C^P_j \leq OPT) \]

\(\square\)
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 \( \alpha p_j \) 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:

\[
\begin{bmatrix}
  j & r_j & p_j \\
  1 & 0 & 100 \\
  2 & 99 & 1 \\
\end{bmatrix}
\]

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

```
J_1  J_2  J_1
0  99  100  101
```

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

```
J_2  J_1
99  100  200
```

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

```
J_1  J_2
0  100  101
```

Claim. Schedule-by-α-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:

- \( C_j^N \leq \max\{r_1, \ldots, r_j\} + \sum_{k=1}^{j} p_k \)
- \( C_j^P \geq \alpha \left( \sum_{k=1}^{j} p_k \right) \)
- \( C_j^P \geq \max\{r_1, \ldots, r_j\} \)
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 \(\alpha\) so that we do well on “most” inputs. This approach is addressed in the next section.

### 3.3 Schedule-by-random-\(\alpha\)

Consider the following algorithm, Schedule-by-random-\(\alpha\), which randomizes the choice of \(\alpha\) in the previous algorithm:

1. Pick \(\alpha'\) according to some PDF \(f\) on \((0, 1]\).
2. Run Schedule-by-\(\alpha\)-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 \(\beta\)-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 \(p(S_i^P(\beta)) = \sum_{j \in S_i^P(\beta)} p_j\).

We have

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

Let \(C_i^\alpha\) be the completion time of job \(j\) in the schedule generated by Schedule-by-random-\(\alpha\).

**Claim.** We have

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

**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 \(\alpha\)-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 \(\alpha\) for the preemptive component leading to a total contribution of \((1 + \alpha)\). \(\square\)
Claim. We have
\[ E[C_i^α] \leq C_i^P \left( 1 + \max_β \left( \int_0^1 f(α) \left( \frac{1 + α - β}{β} \right) dα \right) \right) \]

Proof. Next time. \qed

Note that
\[ \min_f \max_β \left( \int_0^1 f(α) \left( \frac{1 + α - β}{β} \right) dα \right) = \frac{e}{e - 1} \approx 1.58 \]

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