1 Introduction: Jobs with Deadlines

In this lecture, we consider the scheduling of jobs with deadlines. Table 1 shows an example instance consisting of four jobs $J_j$ with deadlines $d_j$. With such jobs, the scheduling objective can be as follows.

- Feasibility: to schedule all jobs by their deadlines (also known as hard real-time scheduling\footnote{‘Hard’ here means that the deadlines are strict. If a schedule misses a hard deadline of any job, it is considered as a (system) failure. ‘Soft’ deadlines can be missed with the quality of service degraded based on the tardiness \cite{1}.})

- To minimize the maximum or weighted sum of:
  - Lateness $L_j = C_j - d_j$  
    ($C_j$ is the completion time of job $J_j$.)
  - Tardiness $T_j = \max\{L_j, 0\}$
  - Unit penalty $U_j = \begin{cases} 1, & \text{if } C_j > d_j \\ 0, & \text{otherwise} \end{cases}$

2 Scheduling Jobs with Deadlines: Earliest Due Date

For scheduling jobs on one machine to meet all their deadlines, there is an optimal method called Earliest Due Date (EDD), Earliest Deadline First (EDF), or Jackson’s Rule. As the names suggest, it schedules the job with the earliest deadline first, and then repeatedly schedules one with the earliest deadline among remaining jobs. Fig. 1 shows an EDD schedule for jobs in Table 1. Note that this EDD schedule meets the deadlines of all jobs except $J_3$.

![Figure 1: EDD schedule for jobs with deadlines in Table 1.](image-url)
Claim 1. For any instance (set of jobs), if there exists a schedule that meets all the deadlines, then an EDD schedule also meets all the deadlines. A contrapositive statement of this is that if EDD cannot schedule an instance to meet all the deadlines of the jobs, then there exists no schedule for this instance that meets all the deadlines.

Claim 2. EDD is optimal for $1|\sum L_j, L_{\text{max}} = \max_j L_j$.

Claim 1 is equivalent to Claim 2. The intuition to verify this argument is as follows. Suppose that there exists a schedule that does not meet all deadlines of an instance. What is the minimum $x$ such that we can meet the deadlines $d_j + x$ for all jobs $J_j$ of this instance?

Hence, we will prove Claim 2 to validate both of the above claims, using Exchange Argument. This argument is widely used when proving the optimality or feasibility of a scheduling method.

2.1 Exchange argument

proof of Claim 2. Suppose that we have a schedule that is optimal for $1|\sum L_j, L_{\text{max}}$, but is not EDD. Then, in this schedule, there must exist two consecutive jobs $J_j$ and $J_k$ with deadlines $d_j > d_k$ (i.e., $J_j$ is scheduled right before $J_k$, but $J_j$ has later deadline than $J_k$). Exchange Argument: We will show that if we swap jobs $J_j$ and $J_k$ in the schedule, then $L_{\text{max}}$ of the new schedule is not greater than the original schedule’s $L_{\text{max}}$. We can apply this Exchange Argument repeatedly to eventually get an EDD schedule of no greater $L_{\text{max}}$ than the original schedule.

As shown in Fig. 2, swapping jobs $J_j$ and $J_k$ does not affect the time at which any other jobs are scheduled. Thus, it is enough to show that $\max(L'_k, L'_j) \leq \max(L_k, L_j) = L_k$, where $L'_k$ and $L'_j$ denote the lateness of jobs $J_k$ and $J_j$, respectively, in the new schedule $S'$. $\max(L_k, L_j) = L_k$ since $C_k > C_j$ and $d_k < d_j$.

- $L'_k \leq L_k$ is trivial as job $J_k$ completes earlier in the new schedule.

- $L'_j \leq L_k$, since $L'_j = C'_j - d_j = C_k - d_j < C_k - d_k = L_k$. ($C'_j$ is the completion time of job $J_j$ in the new schedule $S'$, $C'_j = t$ (starting time of $J_j$ in $S$) + $p_k + p_j = C_k$ as shown in Fig. 2. Also, $d_j > d_k$ from the assumption of this exchange argument.)

Thus, $\max(L'_k, L'_j) \leq \max(L_k, L_j)$, and $L_{\text{max}}$ does not increase by swapping these two jobs.  

![Figure 2: An illustration of the exchange argument. Consecutive jobs $J_j$ and $J_k$ in schedule $S$ are swapped in the new schedule $S'$. If $d_j > d_k$, then $L_{\text{max}}$ of $S'$ does not increase by this swapping.](image)
2.2 Jobs with precedence constraints (1|\text{prec}|L_{\text{max}})

Now, suppose that jobs have precedence constraints as well as the deadlines, and that we want to schedule them on one machine to minimize $L_{\text{max}}$. In this case (1|\text{prec}|L_{\text{max}}), if we do not have any pair of jobs $J_j$ and $J_k$ such that $J_j \rightarrow J_k$ (i.e., $J_j$ precedes $J_k$) and $d_j \geq d_k$, then EDD is optimal. For any instance, if we have such pair of jobs ($J_j \rightarrow J_k$ and $d_j \geq d_k$), then we can transform this instance so that no such pair exists, without changing $L_{\text{max}}$.

The instance is transformed as follows. In reverse topological order\footnote{In reverse topological order, job $J_k$ is visited before job $J_j$ if $J_j \rightarrow J_k$. (In topological order, $J_j$ is visited first.)} of jobs $J$, if $J_j \rightarrow J_k$ for some job $J_k$, let

$$d_j = \min\{d_j, d_k - p_k\}.$$ 

For example, Table 2 (a) shows an instance with the (direct) precedence constraints $J_4 \rightarrow J_3 \rightarrow J_2 \rightarrow J_1$, and Table 2 (b) shows the transformed instance with modified deadlines.

Table 2: An instance with precedence constraints and its transformation.

<table>
<thead>
<tr>
<th>(a) Original instance.</th>
<th>(b) Transformed instance.</th>
<th>(c) $C_j$ and $L_j$ of EDD for (b).</th>
</tr>
</thead>
<tbody>
<tr>
<td>$J_4 \rightarrow J_3 \rightarrow J_2 \rightarrow J_1$</td>
<td>$J_4 \rightarrow J_3 \rightarrow J_2 \rightarrow J_1$</td>
<td>$J \rightarrow C_j \rightarrow L_j$</td>
</tr>
<tr>
<td>$j$</td>
<td>$p_j$</td>
<td>$d_j$</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>12</td>
</tr>
</tbody>
</table>

After this transformation, for any pair of jobs $J_j$ and $J_k$ such that $J_j \rightarrow J_k$, $d_j < d_k$ holds. Thus, an EDD schedule for the transformed instance satisfies all the precedence constraints of both the original and the transformed instances. (The two instances differ only on deadlines.)

We want to show that the maximum lateness $L_{\text{max}}$ of this schedule is the same for both instances. During the transformation, if we did not change the value of $d_j$, then the value of $L_j$ remains the same. Otherwise, i.e., if the value of $d_j$ has been changed to $d_k - p_k$, then $L_j$ for the transformed instance will be larger than that for the original instance (since the deadline has decreased), but the transformed $L_j$ is less than or equal to $L_k$, as follows:

$$L_j = C_j - d_j = C_j - (d_k - p_k) = C_j - d_k + p_k = C_j + p_k - d_k \leq C_k - d_k \leq L_k.$$ 

Hence, if $d_j$ was modified, then $L_j \leq L_k$ for some $k$ and thus it will not affect $L_{\text{max}}$.

In conclusion, an EDD schedule on the transformed instance satisfies all the precedence constraints and its $L_{\text{max}}$, which is optimal for the transformed instance, is the same for the original instance.
2.3 Jobs with release times and preemptions ($1|r_j, \text{pmtn}|L_{\text{max}}$)

When jobs have both release times and deadlines, $1|r_j|L_{\text{max}}$ is NP-hard.

If preemptions are allowed, $1|r_j, \text{pmtn}|L_{\text{max}}$ can be scheduled with preemptive EDD, where the scheduling decision is made 1) at the beginning (at time 0) of the schedule, 2) when a job completes, and 3) when a job arrives. At every decision point, it schedules an active job (that has been released and not completed) with the earliest deadline if exists, with preemption if another job is running.

Table 3: An instance with four jobs. Each job $J_j$ has the release time $r_j$, processing time $p_j$, and deadline $d_j$.

<table>
<thead>
<tr>
<th>$j$</th>
<th>$r_j$</th>
<th>$p_j$</th>
<th>$d_j$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>2</td>
<td>9</td>
</tr>
</tbody>
</table>

Table 3 shows an example instance consisting of four jobs with release times and deadlines, and Fig. 3 shows a preemptive EDD schedule for this instance. As noted in Fig. 3, the completion times of the jobs are $C_1 = 18$, $C_2 = 10$, $C_3 = 4$, $C_4 = 7$, and thus the lateness are $L_1 = -2$, $L_2 = 0$, $L_3 = -4$, $L_4 = -2$.

Note that a preemptive EDD schedule cannot have jobs scheduled in the order of “$J_2, J_3, J_2, J_3$.” The first “$J_2, J_3$” suggests that $d_3 < d_2$, since $J_2$ resumes later, meaning that $J_3$ is preempting $J_2$ here. Then, this $J_3$ is preempted by $J_2$ (since $J_3$ resumes at the end) which cannot occur in an EDD schedule where $J_3$ has an earlier deadline than $J_2$.

To show the optimality of preemptive EDD, let’s introduce some notations. Let $S$ be a subset of jobs of an instance. Then, we define $r_{\text{min}}(S)$, $p(S)$, and $d_{\text{max}}(S)$ as follows.

- $r_{\text{min}}(S) = \min_{j \in S} r_j$
\[ p(S) = \sum_{j \in S} p_j \]
\[ d_{\text{max}}(S) = \max_{j \in S} d_j \]

The following claim sets a lower bound on the optimal \( L_{\text{max}} \).

**Claim 3.** Let \( J \) be the set of all jobs of an instance, and let \( L_{\text{max}}^* \) be the optimal \( L_{\text{max}} \) for this instance. Then, the following holds:

\[ L_{\text{max}}^* \geq \max_{S \subset J} \{ r_{\text{min}}(S) + p(S) - d_{\text{max}}(S) \}. \]

*Proof.* Let \( J_c \) be the last job to complete in \( S \). Then,

\[ L_c = C_c - d_c \geq r_{\text{min}}(S) + p(S) - d_c \geq r_{\text{min}}(S) + p(S) - d_{\text{max}}(S). \]

\( C_c \geq r_{\text{min}}(S) + p(S) \) since the earliest time that any job in \( S \) can start running is \( r_{\text{min}}(S) \) and it takes at least \( p(S) \) to run all the jobs in \( S \). \( d_c \leq d_{\text{max}}(S) \) since \( J_c \) is one job in \( S \). \( \square \)

Now, we will show that preemptive EDD achieves this lower bound.

**Claim 4.** Preemptive EDD has

\[ L_{\text{max}} = \max_{S \subset J} \{ r_{\text{min}}(S) + p(S) - d_{\text{max}}(S) \}. \]

*Proof.* Let \( J_c \) be a job with \( L_{\text{max}} = L_c \). Let \( t \) be the latest time such that every job \( J_j \) running in the interval \([t, C_c]\) has \( r_j \geq t \). Also, let \( S \) be jobs running in the interval \([t, C_c]\). Then, the following claims hold.

1) There is no idle time in \([t, C_c]\), and thus \( p(S) \geq C_c - t \).
   *(proof)* Suppose that there is idle time \([t_1, t_2]\) in \([t, C_c]\) \( (t < t_1 < t_2 < C_c) \). This interval \([t_1, t_2]\) can be idle only if there is no active (released and not completed) job in \([t_1, t_2]\). Thus, any job running after \( t_2 \) must be released after \( t_2 \). Since \( t < t_2 \), this contradicts the definition of \( t \).

2) \( r_{\text{min}}(S) = t \).
   *(proof)* Both \( r_{\text{min}}(S) < t \) and \( r_{\text{min}}(S) > t \) contradict the definition of \( t \).

IEOR E8100, Scheduling Algorithms — Lecture 5 5
3) \(d_{\text{max}}(S) = d_c.\)

\(\text{(proof)}\) Suppose not. Then, let \(t'\) be the latest time in \([t, C_c]\) in which a job \(J_j\) with \(d_j > d_c\) is processed \((t < t')\). Then, for any job \(J_k\) that runs in \([t', C_c]\), \(d_k \leq d_c\), and since \(d_c < d_j\), it follows that \(d_k \leq d_c < d_j\), i.e., \(J_j\) has a later deadline than \(J_k\). However, \(J_j\) was not preempted by \(J_k\) in preemptive EDD, so \(r_k \geq t'\). Thus, all jobs \(J_k\) running in the interval \([t', C_c]\) have \(r_k \geq t'\) and \(t < t'\), which contradicts the definition of \(t\).

From 1), 2), and 3),

\[
L_{\text{max}} = L_c = C_c - d_c \leq t + p(S) - d_{\text{max}} = r_{\text{min}}(S) + p(S) - d_{\text{max}}.
\]

From Claim 3 and Claim 4, preemptive EDD is optimal for \(1|r_j, \text{pmtn}|L_{\text{max}}\).

3 Approximation algorithms

Since \(1|r_j|L_{\text{max}}\) is NP-hard, let’s consider \(\rho\)--approximation algorithms for this problem. Let \(L^*_{\text{max}}\) denote the optimal \(L_{\text{max}}\) value. Then, a \(\rho\)--approximation algorithm should achieve

\[
L_{\text{max}} \leq \rho L^*_{\text{max}}.
\]

Note that \(L_{\text{max}}\) can be 0 or negative. If \(L^*_{\text{max}} = -10\), then for any algorithm it is impossible to achieve \(L_{\text{max}} \leq 2L^*_{\text{max}} = -20\), so no 2--approximation algorithm exists.

On the other hand, since \(L_j = C_j - d_j\), we can decrease all deadlines \(d_j\) by the same amount \(\delta\) to increase \(L_{\text{max}}\) to \(L_{\text{max}} + \delta\). Let’s assume that an instance has \(L^*_{\text{max}} = 3\). Then, a 2--approximation algorithm needs to achieve \(L_{\text{max}} \leq 6\). If we decrease all deadlines by 10000, then \(L^*_{\text{max}} = 10003\) and the requirement for a 2--approximation algorithm becomes \(L_{\text{max}} \leq 20006\). The two problem instances (before and after shifting the deadlines) are essentially equivalent from the scheduling perspective (an optimal schedule for one instance is also optimal for the other), but decreasing the deadlines makes the problem easier for approximation algorithms.

These observations may imply that \(L_{\text{max}}\) is not an appropriate metric to represent the quality of scheduling that we want to compare approximation algorithms with. Motivated by this argument, we consider another metric: delivery times [2]. Let

\[
q_j = -d_j.
\]

Then,

\[
L_j = C_j - d_j = C_j + q_j,
\]
and we want to work on $C_j + q_j$ instead of on $C_j - d_j$. We call this $q_j$ the delivery time.

With this, the problem is defined as follows. A job $J_j$ has release time $r_j$, processing time $p_j$, and delivery time $q_j$. Job $J_j$ can only be processed after $r_j$. After its completion at $C_j$, $J_j$ needs $q_j$ time to deliver the result. With a single processor, at most one job can be processed at any time, but the delivery time of different jobs can overlap. Delivery for $J_j$ is done at $L_j = C_j + q_j$, and we want to minimize this time. Thus, the objective for this problem is

$$\min \max_j L_j.$$

As before, let $L_{\text{max}} = \max_j L_j$. There is a simple 2-approximation algorithm minimizing this objective.

**Claim 5.** In List scheduling, whenever the processor becomes available, the next active (released and not completed) job on the list starts running. List scheduling is a 2-approximation algorithm for the above problem ($1|r_j|L_{\text{max}}$ defined with delivery times).

**Proof.** Let $L^*_\text{max}$ be the optimal value of the objective $L_{\text{max}}$. It is obvious that

$$L^*_\text{max} \geq \sum_j p_j$$

and

$$L^*_\text{max} \geq r_j + p_j + q_j, \ \forall j.$$  

Let $J_c$ be a job with $L_c = L_{\text{max}}$ in a List schedule. Then,

$$L_{\text{max}} = C_c + q_c$$

$$\leq r_c + \sum_j p_j + q_c$$

$$\leq (r_c + q_c) + L^*_\text{max} \quad \cdots \quad \text{from Eq. (1)}$$

$$\leq L^*_\text{max} + L^*_\text{max} \quad \cdots \quad \text{from Eq. (2)}$$

$$= 2L^*_\text{max}.$$  

The inequality (3) holds because there is no idle time in $[r_c, C_c]$ by List scheduling (after job $J_c$ is released, the processor cannot become idle before completing this active job on the list.)

On the next page, Table 4 contains a simple example instance with two jobs $J_1$ and $J_2$ to demonstrate the gap between a List schedule and an optimal schedule. Fig. 4 shows a list schedule where $J_1$ is processed from 0 to $M$, because only $J_1$ is released at 0, and then $J_2$ is processed from $M$ to $M + 1$. With this schedule, $L_{\text{max}}$ is $2M + 1$ since $J_2$ takes $M$ time units for the delivery. Fig. 5 shows an optimal schedule where it is idle at $[0, 1]$, and at $t = 1$ when $J_2$ is released, $J_2$ is processed for 1 time unit, so $L_2 = 2 + M$. $J_1$ is processed from 2 to $M + 2$ and its delivery time is 0, so $L_1 = M + 2$. Thus, $L_{\text{max}} = M + 2$ for the optimal schedule. As the constant $M$ increases, the ratio between the $L_{\text{max}}$ of the List schedule to that of the optimal schedule approaches 2.
Table 4: An instance with two jobs. Each job $J_j$ has the release time $r_j$, processing time $p_j$, and delivery time $q_j$. Let $M$ be a large constant.

<table>
<thead>
<tr>
<th>$j$</th>
<th>$r_j$</th>
<th>$p_j$</th>
<th>$q_j$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>$M$</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>$M$</td>
</tr>
</tbody>
</table>

Figure 4: A List schedule with $L_{\text{max}} = 2M + 1$ for the instance in Table 4.

Figure 5: An optimal schedule with $L_{\text{max}} = M + 2$ for the instance in Table 4.

References
