Lectures 17-19

Greedy Algorithms

• Interval Scheduling
• Interval Partitioning
• Scheduling for Maximum Lateness

Adam Smith
Design paradigms so far

- Simple, iterative algorithms
  - Insertion sort
- Recursion-based algorithms
  - Divide and conquer
  - Backtracking
  - Dynamic programming
- Today: greedy algorithms
  - Simple, iterative structure: at each phase, “grab” make the best choice available, and never look back.
  - Typically very tricky to analyze
  - Easy to get wrong!!!
Weighted Interval Scheduling

• Weighted interval scheduling problem.
  – Job j starts at $s_j$, finishes at $f_j$, and has weight or value $v_j$.
  – Two jobs **compatible** if they don't overlap.
  – Goal: find maximum **weight** subset of mutually compatible jobs.

• Dynamic programming algorithm:
  – Either item j is in or out…

• Can we find a simpler algorithm for the “unweighted” case? (All weights equal)
Unweighted Interval Scheduling Problem

- Job j starts at $s_j$ and finishes at $f_j$.
- Two jobs are **compatible** if they do not overlap.
- **Find**: maximum subset of mutually compatible jobs.
Greedy: Counterexamples

- for earliest start time
- for shortest interval
- for fewest conflicts
Formulating Algorithm

• Arrays of start and finishing times
  – \( s_1, s_2, \ldots, s_n \)
  – \( f_1, f_2, \ldots, f_n \)

• Input length?
  – \( 2n = \Theta(n) \)
Greedy Algorithm

• **Earliest finish time:** ascending order of $f_j$.

```plaintext
Sort jobs by finish times so that $f_1 \leq f_2 \leq \ldots \leq f_n$.

A ← φ  // M Set of selected jobs
for j = 1 to n {
    if (job j compatible with A)
        A ← A ∪ {j}
}
return A
```

• **Implementation.**  \( O(n \log n) \) time; \( O(n) \) space.
  – Remember job \( j^* \) that was added last to \( A \).
  – Job \( j \) is compatible with \( A \) if \( s_j \geq f_{j^*} \).
Running time: $O(n \log n)$

Sort jobs by finish times so that $f_1 \leq f_2 \leq \ldots \leq f_n$.

$A \leftarrow \text{(empty)}$ \hspace{1cm} M \text{ Queue of selected jobs}

$j^* \leftarrow 0$

for $j = 1$ to $n$ {
  if $(f_{j^*} \leq s_j)$
    enqueue($j$ onto $A$)
}

return $A$
Analysis: Greedy Stays Ahead

• Theorem. Greedy algorithm is optimal.

• Proof strategy (by contradiction):
  – Suppose greedy solution G is not optimal.
  – Consider an optimal strategy... which one?
    • Consider the optimal strategy OPT that agrees with the greedy strategy for as many initial jobs as possible
    • Look at first place in list where OPT differs from greedy strategy
  – Show a new optimal strategy that agrees more with the greedy strategy

A. Smith; based on slides by E. Demaine, C. Leiserson, S. Raskhodnikova and K. Wayne
Analysis: Greedy Stays Ahead

• Theorem. Greedy algorithm is optimal.
• Pf (by contradiction): Suppose greedy is not optimal.
  – Let $i_1, i_2, \ldots, i_k$ denote set of jobs selected by greedy.
  – Let $j_1, j_2, \ldots, j_m$ be set of jobs in the optimal solution with $i_1 = j_1, i_2 = j_2, \ldots, i_r = j_r$ for the largest possible value of $r$.  

![Diagram of Greedy vs Optimal Jobs]

Greedy: $i_1, i_2, \ldots, i_r, i_{r+1}$
OPT: $j_1, j_2, \ldots, j_r, j_{r+1}$

why not replace job $j_{r+1}$ with job $i_{r+1}$?
Analysis: Greedy Stays Ahead

• Theorem. Greedy algorithm is optimal.
• Pf (by contradiction): Suppose greedy is not optimal.
  – Let $i_1, i_2, ... i_k$ denote set of jobs selected by greedy.
  – Let $j_1, j_2, ... j_m$ be set of jobs in the optimal solution with $i_1 = j_1, i_2 = j_2, ..., i_r = j_r$ for the largest possible value of $r$.

Greedy:

OPT:

job $i_{r+1}$ finishes before $j_{r+1}$

↓

solution still feasible and optimal, but contradicts maximality of $r$.
• **Recall**: Greedy algorithm works if all weights are 1.
  – Consider jobs in ascending order of finish time.
  – Add job to subset if it is compatible with previously chosen jobs.

• **Observation**: Greedy algorithm can fail spectacularly if arbitrary weights are allowed.
Interval Partitioning
Interval Partitioning Problem

• Lecture $j$ starts at $s_j$ and finishes at $f_j$.
• **Find**: minimum number of classrooms to schedule all lectures so that no two occur at the same time in the same room.
• **E.g.**: 10 lectures are scheduled in 4 classrooms.
Interval Partitioning

• Lecture \( j \) starts at \( s_j \) and finishes at \( f_j \).

• **Find**: minimum number of classrooms to schedule all lectures so that no two occur at the same time in the same room.

• **E.g.**: Same lectures are scheduled in 3 classrooms.
Lower Bound

• **Definition.** The **depth** of a set of open intervals is the maximum number that contain any given time.

• **Key observation.** Number of classrooms needed $\geq$ depth.

• **E.g.:** Depth of this schedule $= 3 \Rightarrow$ this schedule is optimal.

• **Q:** Is it always sufficient to have number of classrooms $=$ depth?

A, B, C all contain 9:30

<table>
<thead>
<tr>
<th>9</th>
<th>9:30</th>
<th>10</th>
<th>10:30</th>
<th>11</th>
<th>11:30</th>
<th>12</th>
<th>12:30</th>
<th>1</th>
<th>1:30</th>
<th>2</th>
<th>2:30</th>
<th>3</th>
<th>3:30</th>
<th>4</th>
<th>4:30</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>b</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>c</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>a, b, c all contain 9:30</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Greedy Algorithm

• Consider lectures in increasing order of start time: assign lecture to any compatible classroom.

```plaintext
Sort intervals by starting time so that \( s_1 \leq s_2 \leq \ldots \leq s_n \).
\( d \leftarrow 0 \) M Number of allocated classrooms
for \( j = 1 \) to \( n \) {
  if (lecture \( j \) is compatible with some classroom \( k \))
    schedule lecture \( j \) in classroom \( k \)
  else
    allocate a new classroom \( d + 1 \)
    schedule lecture \( j \) in classroom \( d + 1 \)
  \( d \leftarrow d + 1 \)
}
```

• Implementation. \( O(n \log n) \) time; \( O(n) \) space.
  – For each classroom, maintain the finish time of the last job added.
  – Keep the classrooms in a priority queue (main loop \( n \log(d) \) time)
Analysis: Structural Argument

• **Observation.** Greedy algorithm never schedules two incompatible lectures in the same classroom.

• **Theorem.** Greedy algorithm is optimal.

• **Proof:** Let $d =$ number of classrooms allocated by greedy.
  
  – Classroom $d$ is opened because we needed to schedule a lecture, say $j$, that is incompatible with all $d-1$ last lectures in other classrooms.
  
  – These $d$ lectures each end after $s_j$.
  
  – Since we sorted by start time, they start no later than $s_j$.
  
  – Thus, we have $d$ lectures overlapping at time $s_j + \varepsilon$.
  
  – Key observation $\Rightarrow$ all schedules use $\geq d$ classrooms. •
Scheduling to minimize lateness
Minimizing lateness problem.

- Single resource processes one job at a time.
- Job $j$ requires $t_j$ units of processing time and is due at time $d_j$.
- If $j$ starts at time $s_j$, it finishes at time $f_j = s_j + t_j$.
- Lateness: $\ell_j = \max\{0, f_j - d_j\}$.
- Goal: schedule all jobs to minimize maximum lateness $L = \max \ell_j$.

Ex:

<table>
<thead>
<tr>
<th>$t_j$</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>4</th>
<th>3</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$d_j$</td>
<td>6</td>
<td>8</td>
<td>9</td>
<td>9</td>
<td>14</td>
<td>15</td>
</tr>
</tbody>
</table>

Lateness:
- $\ell_3 = 2$
- $\ell_5 = 0$
- $\max \ell_j = 6$
Minimizing Lateness: Greedy Algorithms

Greedy template. Consider jobs in some order.

- [Shortest processing time first] Consider jobs in ascending order of processing time $t_j$.

- [Earliest deadline first] Consider jobs in ascending order of deadline $d_j$.

- [Smallest slack] Consider jobs in ascending order of slack $d_j - t_j$. 
Minimizing Lateness: Greedy Algorithms

Greedy template. Consider jobs in some order.

- [Shortest processing time first] Consider jobs in ascending order of processing time $t_j$.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>$t_j$</td>
<td>1</td>
</tr>
<tr>
<td>$d_j$</td>
<td>100</td>
</tr>
</tbody>
</table>

  counterexample

- [Smallest slack] Consider jobs in ascending order of slack $d_j - t_j$.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>$t_j$</td>
<td>1</td>
</tr>
<tr>
<td>$d_j$</td>
<td>2</td>
</tr>
</tbody>
</table>

  counterexample
Minimizing lateness problem.

- Single resource processes one job at a time.
- Job \( j \) requires \( t_j \) units of processing time and is due at time \( d_j \).
- If \( j \) starts at time \( s_j \), it finishes at time \( f_j = s_j + t_j \).
- Lateness: \( \ell_j = \max \{ 0, f_j - d_j \} \).
- Goal: schedule all jobs to minimize maximum lateness \( L = \max \ell_j \).

**Ex:**

| \( t_j \) | 3 | 2 | 1 | 4 | 3 | 2 |
| \( d_j \) | 6 | 8 | 9 | 9 | 14 | 15 |

\( d_3 = 9 \) \( d_2 = 8 \) \( d_6 = 15 \) \( d_1 = 6 \) \( d_5 = 14 \) \( d_4 = 9 \)

\( \text{lateness} = 2 \) \( \text{lateness} = 0 \) \( \text{max lateness} = 6 \)
Minimizing Lateness: Greedy Algorithms

**Greedy template.** Consider jobs in some order.

- **[Shortest processing time first]** Consider jobs in ascending order of processing time \( t_j \).

- **[Earliest deadline first]** Consider jobs in ascending order of deadline \( d_j \).

- **[Smallest slack]** Consider jobs in ascending order of slack \( d_j - t_j \).
Minimizing Lateness: Greedy Algorithms

Greedy template. Consider jobs in some order.

- [Shortest processing time first] Consider jobs in ascending order of processing time $t_j$.

  \[
  \begin{array}{|c|c|c|}
  \hline
  & 1 & 2 \\
  \hline
  t_j & 1 & 10 \\
  d_j & 100 & 10 \\
  \hline
  \end{array}
  \]

  counterexample

- [Smallest slack] Consider jobs in ascending order of slack $d_j - t_j$.

  \[
  \begin{array}{|c|c|c|}
  \hline
  & 1 & 2 \\
  \hline
  t_j & 1 & 10 \\
  d_j & 2 & 10 \\
  \hline
  \end{array}
  \]

  counterexample
Minimizing Lateness: Greedy Algorithm

Greedy algorithm. Earliest deadline first.

Sort n jobs by deadline so that $d_1 \leq d_2 \leq \ldots \leq d_n$

$t \leftarrow 0$

for $j = 1$ to $n$
    Assign job $j$ to interval $[t, t + t_j]$
    $s_j \leftarrow t$, $f_j \leftarrow t + t_j$
    $t \leftarrow t + t_j$

output intervals $[s_j, f_j]$
Minimizing Lateness: No Idle Time

**Observation.** There exists an optimal schedule with no *idle time.*

\[
\begin{array}{cccccc|cccc}
\text{d = 4} & \text{d = 6} & \text{d = 12} \\
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\
\end{array}
\]

**Observation.** The greedy schedule has no idle time.

\[
\begin{array}{cccccc|cccc}
\text{d = 4} & \text{d = 6} & \text{d = 12} \\
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \\
\end{array}
\]
Def. An inversion in schedule $S$ is a pair of jobs $i$ and $j$ such that: $i < j$ but $j$ scheduled before $i$.

Observation. Greedy schedule has no inversions.

Observation. If a schedule (with no idle time) has an inversion, it has one with a pair of inverted jobs scheduled consecutively.
**Def.** An inversion in schedule $S$ is a pair of jobs $i$ and $j$ such that: $i < j$ but $j$ scheduled before $i$.

**Claim.** Swapping two adjacent, inverted jobs reduces the number of inversions by one and does not increase the max lateness.

**Pf.** Let $\ell$ be the lateness before the swap, and let $\ell'$ be it afterwards.

- $\ell'_k = \ell_k$ for all $k \neq i, j$
- $\ell'_i \leq \ell_i$
- If job $j$ is late:

\[
\ell'_j = f'_j - d_j = f_i - d_j = f_i - d_i \leq \ell_i.
\]
Minimizing Lateness: Analysis of Greedy Algorithm

**Theorem.** Greedy schedule S is optimal.

**Pf.** Define S* to be an optimal schedule that has the fewest number of inversions, and let's see what happens.

- Can assume S* has no idle time.
- If S* has no inversions, then S = S*.
- If S* has an inversion, let i-j be an adjacent inversion.
  - swapping i and j does not increase the maximum lateness and strictly decreases the number of inversions
  - this contradicts definition of S*
Greedy Analysis Strategies

**Greedy algorithm stays ahead.** Show that after each step of the greedy algorithm, its solution is at least as good as any other algorithm's.

**Exchange argument.** Gradually transform any solution to the one found by the greedy algorithm without hurting its quality.

**Structural.** Discover a simple "structural" bound asserting that every possible solution must have a certain value. Then show that your algorithm always achieves this bound.