Lecture 6
Greedy Algorithms
• Interval Scheduling
• Interval Partitioning
• Scheduling to Minimize Lateness

Sofya Raskhodnikova
Optimization problems

• Coming up: 3 design paradigms
  – Greedy
  – Divide and Conquer
  – Dynamic Programming

• Illustrated on optimization problems
  – Set of feasible solutions
  – Goal: find the “best” solution according to some objective function
Design technique #1: Greedy Algorithms
Greedy Algorithms

- Build up a solution to an optimization problem at each step shortsightedly choosing the option that currently seems the best.
  - Sometimes good
  - Often does not work
- Key to designing greedy algorithms: find structure that ensures you don’t leave behind other options

S. Raskhodnikova; based on slides by E. Demaine, C. Leiserson, A. Smith, K. Wayne
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.

![Interval Scheduling Problem Diagram](image-url)
Possible Greedy Strategies

Consider jobs in some natural order. Take next job if it is compatible with the ones already taken.

- **Earliest start time:** ascending order of $s_j$.
- **Earliest finish time:** ascending order of $f_j$.
- **Shortest interval:** ascending order of $(f_j - s_j)$.
- **Fewest conflicts:** For each job $j$, count the number of conflicting jobs $c_j$. Schedule in ascending order of $c_j$. 
Greedy: Counterexamples

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

• Input: arrays of start and finishing times
  – $s_1$, $s_2$, …, $s_n$
  – $f_1$, $f_2$, …, $f_n$

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

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

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

A $\leftarrow \emptyset$  // Set of jobs selected so far
for $j = 1$ to $n$
    if (job $j$ compatible with A)
        A $\leftarrow$ A $\cup \{j\}$
return A

• Implementation:
  
  – How do we quickly test if j is compatible with A?
  – Store job $j^*$ that was added last to A.
  – Job j is compatible with A if $s_j \geq f_{j^*}$. 

S. Raskhodnikova; based on slides by E. Demaine, C. Leiserson, A. Smith, K. Wayne
Time and space analysis

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

\[ A \leftarrow (\text{empty}) \quad \// \text{Queue of selected jobs} \]
\[ j^{*} \leftarrow 0 \]
for \( j = 1 \) to \( n \)
  if \( (f_{j^{*}} \leq s_j) \)
    enqueue(\( j \) onto \( A \))
  \( j^{*} \leftarrow j \)

return \( A \)

\( O(n \log n) \) time; \( O(n) \) space.
Theorem. Greedy algorithm’s solution is optimal.

Proof strategy (by contradiction):

• Suppose greedy is not optimal.
• Consider an optimal solution…
  – which one?
  – optimal solution that agrees with the greedy solution for as many initial jobs as possible
• Look at the first place in the list where optimal solution differs from the greedy solution
  – Show a new optimal solution that agrees more w/ greedy
  – Contradiction!
Theorem: Greedy algorithm’s solution is optimal.

Proof (by contradiction): Suppose greedy 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 the optimal solution with $i_1 = j_1, i_2 = j_2, \ldots, i_r = j_r$ for the largest possible value of $r$.
- If $r < k$, then …?
Analysis: Greedy Stays Ahead

Theorem: Greedy algorithm’s solution is optimal.

Proof (by contradiction): Suppose greedy 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 the optimal solution with $i_1 = j_1, i_2 = j_2, \ldots, i_r = j_r$
  for the largest possible value of $r$.

– If $r < k$, then …?

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

**Theorem:** Greedy algorithm’s solution is optimal.

**Proof (by contradiction):** Suppose greedy 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 the optimal solution with
  \[ i_1 = j_1, i_2 = j_2, \ldots, i_r = j_r \]
  for the largest possible value of \( r \).
- If \( r < k \), then we get contradiction.

\[ \text{Greedy:} \quad \begin{array}{cccc}
  i_1 & i_2 & i_r & i_{r+1}
\end{array} \]

\[ \text{OPT:} \quad \begin{array}{cccc}
  j_1 & j_2 & j_r & i_{r+1}
\end{array} \]

Could it be that \( r = k \) but \( k < m \)?

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

**Theorem:** Greedy algorithm’s solution is optimal.

**Proof (by contradiction):** Suppose greedy 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 the optimal solution with $i_1 = j_1, i_2 = j_2, \ldots, i_r = j_r$ for the largest possible value of $r$.
- If $r < k$, we get a contradiction by replacing $j_{r+1}$ with $i_{r+1}$ because we get an optimal solution with larger $r$.
- If $r = k$ but $m > k$, we get a contradiction because greedy algorithm stopped before all jobs were considered.

9/12/2016
Alternate Way to See the Proof

- Induction statement
  \[ P(k) : \text{There is an optimal solution that agrees with the greedy solution in the first } k \text{ jobs.} \]
  
- \( P(n) \) is what we want to prove.

- Base case: \( P(0) \)

- We essentially proved the induction step…
Interval Partitioning
Interval Partitioning

- Lecture $j$ starts at $s_j$ and finishes at $f_j$.
- Input: $s_1, \ldots, s_n$ and $f_1, \ldots, f_n$.
- Goal: 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.

9/12/2016
Interval Partitioning

- Lecture $j$ starts at $s_j$ and finishes at $f_j$.
- **Input**: $s_1, \ldots, s_n$ and $f_1, \ldots, f_n$.
- **Goal**: 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 scheduled in 3 classrooms.

![Diagram of lecture scheduling with time slots and classrooms labeled a, b, c, d, e, f, g, h, i, j.]}
Lower Bound

- **Definition.** The depth of a set of open intervals is the maximum number that contain any given time.
- **Key lemma.** 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 $= \text{depth}$?
Greedy Algorithm

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

```
Sort intervals by starting time so that s_1 ≤ s_2 ≤ ... ≤ s_n.
d ← 0     // 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 ← 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
• Using a heap, main loop takes \( O(n \log d) \) 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 lemma $\Rightarrow$ all schedules use $\geq d$ classrooms. □
Duality

• Our first example of “duality”!

• High-level overview of proof of correctness:
  – Identify obstacles to scheduling in few classrooms
    • Sets of overlapping lectures
  – Show that our algorithm’s solution matches some obstacle
    • If our solution uses $d$ classrooms, then there is a set of $d$ overlapping lectures
  – Conclude that our solution cannot be improved

9/12/2016

S. Raskhodnikova; based on slides by E. Demaine, C. Leiserson, A. Smith, K. Wayne
Scheduling to minimize lateness
Scheduling to Minimizing 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 $L = \max \ell_j$.
Greedy strategies?
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 Strategies

Greedy template: consider jobs in some order.

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

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_j$</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>$d_j$</td>
<td>100</td>
<td>10</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_j$</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>$d_j$</td>
<td>2</td>
<td>10</td>
</tr>
</tbody>
</table>
Minimizing Lateness: Greedy Algorithm

• [Earliest deadline first]

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

\[
\begin{align*}
t & \leftarrow 0 \\
\text{for } j = 1 \text{ to } n & \\
& \quad \text{Assign job } j \text{ to interval } [t, t + t_j] \\
& \quad s_j \leftarrow t, f_j \leftarrow t + t_j \\
& \quad t \leftarrow t + t_j \\
\text{output intervals } [s_j, f_j]
\end{align*}
\]

max lateness = 1

<table>
<thead>
<tr>
<th>( d_1 = 6 )</th>
<th>( d_2 = 8 )</th>
<th>( d_3 = 9 )</th>
<th>( d_4 = 9 )</th>
<th>( d_5 = 14 )</th>
<th>( d_6 = 15 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>
Minimizing Lateness: No Idle Time

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

• **Observation.** The greedy schedule has no idle time.
Minimizing Lateness: Inversions

- An **inversion** in schedule S is a pair of jobs i and j such that $d_i < d_j$ 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.
Minimizing Lateness: Inversions

• An **inversion** in schedule $S$ is a pair of jobs $i$ and $j$ such that $d_i < d_j$ but $j$ scheduled before $i$.

<table>
<thead>
<tr>
<th>before swap</th>
<th>j</th>
<th>i</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>after swap</td>
<td>i</td>
<td>j</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

• **Proof:** Let $\ell$ be the lateness before the swap, and let $\ell'$ be the lateness afterwards.
  
  $\ell'_k = \ell_k$ for all $k \neq i, j$
  
  $\ell'_i \leq \ell_i$
  
  If job $j$ is late:

  $\ell'_j = f'_j - d_j$ (definition)
  
  $= f_i - d_j$ ($j$ finishes at time $f_i$)
  
  $\leq f_i - d_i$ ($d_i < d_j$)
  
  $\leq \ell_i$ (definition)
**Minimizing Lateness: Analysis**

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

**Proof:** Define $S^*$ to be an optimal schedule that has the fewest number of inversions.

- 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 the definition of $S^*$. □
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.

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.

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