Greedy Algorithms

- Solve problems with the simplest possible algorithm
- The hard part: showing that something simple actually works
- Today’s problem
  - Homework Scheduling
  - Optimal Caching

Homework Scheduling

- Tasks to perform
- Deadlines on the tasks
- Freedom to schedule tasks in any order

Scheduling tasks

- Each task has a length $t_i$ and a deadline $d_i$
- All tasks are available at the start
- One task may be worked on at a time
- All tasks must be completed

- Goal minimize maximum lateness
  - Lateness = $f_i - d_i$ if $f_i >= d_i$

Example

<table>
<thead>
<tr>
<th>Task</th>
<th>Lateness</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>12</td>
</tr>
</tbody>
</table>
Greedy Algorithm

• Earliest deadline first
• Order jobs by deadline

This result may be surprising, since it ignores the job lengths

Analysis

• Suppose the jobs are ordered by deadlines, $d_1 \leq d_2 \leq \ldots \leq d_n$
• A schedule has an inversion if job $j$ is scheduled before $i$ where $j > i$
• The schedule $A$ computed by the greedy algorithm has no inversions.
• Let $O$ be the optimal schedule, we want to show that $A$ has the same maximum lateness as $O$

Proof

• Lemma: There is an optimal schedule with no idle time.
• Lemma: There is an optimal schedule with no inversions and no idle time.

Interchange argument

• Suppose there is a pair of jobs $i$ and $j$, with $i < j$, and $j$ scheduled immediately before $i$. Interchanging $i$ and $j$ does not increase the maximum lateness. Recall, $d_i \leq d_j$

Result

• Earliest Deadline First algorithm constructs a schedule that minimizes the maximum lateness

Extensions

• What if the objective is to minimize the sum of the lateness?
  – EDF does not seem to work
• If the tasks have release times and deadlines, and are non-preemptable, the problem is NP-complete
• What about the case with release times and deadlines where tasks are preemptable?
Optimal Caching

• Caching problem:
  – Maintain collection of items in local memory
  – Minimize number of items fetched

Caching example

A, B, C, D, A, E, B, A, D, A, C, B, D, A

Optimal Caching

• If you know the sequence of requests, what is the optimal replacement pattern?
• Note – it is rare to know what the requests are in advance – but we still might want to do this:
  – Some specific applications, the sequence is known
  – Competitive analysis, compare performance on an online algorithm with an optimal offline algorithm

Farthest in the future algorithm

• Discard element used farthest in the future

A, B, C, A, D, C, B, C, A, D

Correctness Proof

• Sketch
• Start with Optimal Solution O
• Convert to Farthest in the Future Solution F-F
• Look at the first place where they differ
• Convert O to evict F-F element
  – There are some technicalities here to ensure the caches have the same configuration . . .