Greedy Algorithms

- Solve problems with the simplest possible algorithm
- The hard part: showing that something simple actually works
- Pseudo-definition
  - An algorithm is Greedy if it builds its solution by adding elements one at a time using a simple rule

Scheduling Theory

- Tasks
  - Processing requirements, release times, deadlines
- Processors
- Precedence constraints
- Objective function
  - Jobs scheduled, lateness, total execution time

Interval Scheduling

- Tasks occur at fixed times
- Single processor
- Maximize number of tasks completed

- Tasks \{1, 2, \ldots N\}
- Start and finish times, s(i), f(i)

What is the largest solution?

Greedy Algorithm for Scheduling

Let T be the set of tasks, construct a set of independent tasks I, A is the rule determining the greedy algorithm

I = \{

While T is not empty

- Select a task t from T by a rule A
- Add t to I
- Remove t and all tasks incompatible with t from T
Simulate the greedy algorithm for each of these heuristics

- Schedule earliest starting task
- Schedule shortest available task
- Schedule task with fewest conflicting tasks

Greedy solution based on earliest finishing time

- Example 1
- Example 2
- Example 3

Theorem: Earliest Finish Algorithm is Optimal

- Key idea: Earliest Finish Algorithm stays ahead
- Let $A = \{i_1, \ldots, i_k\}$ be the set of tasks found by EFA in increasing order of finish times
- Let $B = \{j_1, \ldots, j_m\}$ be the set of tasks found by a different algorithm in increasing order of finish times
- Show that for $r \leq \min(k, m)$, $f(i_r) \leq f(j_r)$

Stay ahead lemma

- $A$ always stays ahead of $B$, $f(i_r) \leq f(j_r)$
- Induction argument
  - $f(i_1) \leq f(j_1)$
  - If $f(i_{r-1}) \leq f(j_{r-1})$ then $f(i_r) \leq f(j_r)$

Completing the proof

- Let $A = \{i_1, \ldots, i_k\}$ be the set of tasks found by EFA in increasing order of finish times
- Let $O = \{j_1, \ldots, j_m\}$ be the set of tasks found by an optimal algorithm in increasing order of finish times
- If $k < m$, then the Earliest Finish Algorithm stopped before it ran out of tasks

Scheduling all intervals

- Minimize number of processors to schedule all intervals
How many processors are needed for this example?

Prove that you cannot schedule this set of intervals with two processors

Depth: maximum number of intervals active

Algorithm

Scheduling tasks

Example

Time | Deadline
--- | ---
2   | 2
3   | 4

Lateness 1

Time | Lateness
--- | ---
2   | 3
3   | 2

Lateness 3
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 $f_i \geq d_i$

Example

<table>
<thead>
<tr>
<th>Time</th>
<th>Deadline</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</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 algorithm is optimal
Analysis

• Suppose the jobs are ordered by deadlines, \( d_1 \leq d_2 \leq \ldots \leq d_n \)
• A schedule has an \textit{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 \( O \)

List the inversions

<table>
<thead>
<tr>
<th>Time</th>
<th>Deadline</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a_1 )</td>
<td>3</td>
</tr>
<tr>
<td>( a_2 )</td>
<td>4</td>
</tr>
<tr>
<td>( a_3 )</td>
<td>2</td>
</tr>
<tr>
<td>( a_4 )</td>
<td>5</td>
</tr>
</tbody>
</table>

Lemma: There is an optimal schedule with no idle time

Lemma

• If there is an inversion \( i, j \), there is a pair of adjacent jobs \( i', j' \) which form an inversion

Interchange argument

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

Proof by Bubble Sort

Determine maximum lateness
Real Proof

• There is an optimal schedule with no inversions and no idle time.
• Let O be an optimal schedule k inversions, we construct a new optimal schedule with k-1 inversions
• Repeat until we have an optimal schedule with 0 inversions
• This is the solution found by the earliest deadline first algorithm

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

[Diagram of caching example with sequence 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, C, 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 . . .