COP 4531
Complexity & Analysis of Data Structures & Algorithms

Greedy Algorithms

Thanks to the text authors and K. Wayne and Piyush Kumar who contributed to these slides
What are greedy algorithms?

• Similar to dynamic programming and also used for optimization problems

• Idea
  - When a choice needs to be made, make the one that looks the best right now. Thus make a locally optimal choice in hopes of getting a globally optimal solution
  - We will define some general characteristics of when greedy algorithms can give optimal solutions
  - We have already seen that greedy algorithms naturally arise in the minimum spanning tree problem and that we were able to determine a greedy choice, the safe edge, in that problem which gave us two good solution

• Will explore several problems that can be solved using greedy algorithms
Greedy Algorithms

- For some problems, “Greed is good” works.
- For some, it finds a good solution which is not global optimal
  - Heuristics
  - Approximation Algorithms
- For others, it can do very bad.
Activity Selection = Interval Scheduling

- n activities (jobs) require exclusive use of a common resource. For example use of a classroom.
- Set of activities $S = \{a_1, \ldots, a_n\}$
- Job $a_j$ needs resource during period $[s_j, f_j)$
  - $s_j$ is the start time of the job
  - $f_j$ is the finish time of the job
- Goal: select the largest possible set of non-overlapping (mutually compatible) jobs
Example

- Activity Selection.
  - Job $a_j$ starts at $s_j$ and finishes at $f_j$.
  - Two jobs compatible if they don’t overlap.
  - Goal: find maximum subset of mutually compatible jobs.

optimal is set \{b, e, h\}
note that the smallest interval c is not in the optimal set
Making a greedy choice

- Choose an activity to add to optimal solution before solving subproblems. For the activity-selection problem, we can get away with considering only the greedy choice: the activity that leaves the resource available for as many other activities as possible.
- Question: Which activity leaves the resource available for the most other activities? Answer: The first activity to finish. (If more than one activity has earliest finish time, can choose any such activity.) Since activities are sorted by finish time, just choose activity \( a_1 \).
- That leaves only one subproblem to solve: finding a maximum size set of mutually compatible activities that start after \( a_1 \) finishes.
- This is clearly recursive. By optimal substructure, if \( a_1 \) is in an optimal solution, then an optimal solution to the original problem consists of \( a_1 \) plus all activities in an optimal solution to \( S_1 \).
- Only need to prove that \( a_1 \) is always part of some optimal solution.
Activity Selection: Greedy Algorithms

- **Greedy template.** Consider jobs in some order. Take each job provided it's compatible with the ones already taken.

  - **[Earliest start time]** Consider jobs in ascending order of start time $s_j$.

  - **[Earliest finish time]** Consider jobs in ascending order of finish time $f_j$.

  - **[Shortest interval]** Consider jobs in ascending order of interval length $f_j - s_j$.

  - **[Fewest conflicts]** For each job, count the number of conflicting jobs $c_j$. Schedule in ascending order of conflicts $c_j$. 

Activity Selection: Greedy Algorithms

- **Greedy template.** Consider jobs in some order. Take each job provided it's compatible with the ones already taken. Note that these three strategies do not work and are easily shown to be not optimal:

  - Breaks earliest start time
  - Breaks shortest interval
  - Breaks fewest conflicts
Activity Selection: Greedy Algorithm

- **Greedy algorithm.** Consider jobs in increasing order of finish time. Take each job provided it's compatible with the ones already taken.

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

  jobs selected

  A ← \( \phi \)

  for j = 1 to n {
    if (job j compatible with A)
      A ← A ∪ \{ j \}
  }

  return A

- **Implementation.** \( O(n \log n) \).
  - Remember job \( j^* \) that was added last to A.
  - Job j is compatible with A if \( s_j \geq f_{j^*} \).
More precise algorithm

\textbf{GREEDY-ACTIVITY-SELECTOR}(s, f)
\begin{align*}
n &= s.\text{length} \\
A &= \{a_1\} \\
k &= 1 \\
\text{for } m &= 2 \text{ to } n \\
    \text{if } s[m] &\ge f[k] \\
        A &= A \cup \{a_m\} \\
        k &= m \\
\text{return } A
\end{align*}

Time Complexity
\(\Theta(n)\) for this algorithm. Need to sort jobs which is \(O(n \log n)\)
Interval Scheduling: Analysis

- Theorem. Greedy algorithm is optimal.

- Pf. (by contradiction)
  - Assume greedy is not optimal, and let's see what happens.
  - Let \( i_1, i_2, \ldots, i_k \) denote set of jobs selected by greedy.
  - Let \( j_1, j_2, \ldots, j_m \) denote 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 \( r \).

\[ \begin{array}{c}
\text{Greedy:} \\
\hline
i_1 & i_1 & i_r & i_{r+1} \\
\hline
\text{OPT:} \\
\hline
j_1 & j_2 & j_r & j_{r+1} & \ldots \\
\hline
\end{array} \]

Why not replace job \( j_{r+1} \) with job \( i_{r+1} \)?

\( \text{job } i_{r+1} \text{ finishes before } j_{r+1} \)
Interval Scheduling: Analysis

- Theorem. Greedy algorithm is optimal.

- Pf. (by contradiction)
  - Assume greedy is not optimal, and let's see what happens.
  - Let $i_1, i_2, \ldots i_k$ denote set of jobs selected by greedy.
  - Let $j_1, j_2, \ldots j_m$ denote 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]

Greedy: $i_1, i_1, i_r, i_{r+1}$

OPT: $j_1, j_2, j_r, i_{r+1}$

$\downarrow$

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

$\downarrow$

solution still feasible and optimal, but contradicts maximality of $r$. 
4.1 Interval Partitioning
Interval Partitioning

- Interval partitioning.
  - Lecture $j$ starts at $s_j$ and finishes at $f_j$.
  - Goal: find minimum number of classrooms to schedule all lectures so that no two occur at the same time in the same room.

- Ex: This schedule uses 4 classrooms to schedule 10 lectures.
Interval Partitioning

- Interval partitioning.
  - Lecture $j$ starts at $s_j$ and finishes at $f_j$.
  - Goal: find minimum number of classrooms to schedule all lectures so that no two occur at the same time in the same room.

- Ex: This schedule uses only 3.
Interval Partitioning: Lower Bound on Optimal Solution

- **Def.** 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.

- **Ex:** Depth of schedule below = 3 $\Rightarrow$ schedule below is optimal. $a$, $b$, $c$ all contain 9:30.

- **Q.** Does there always exist a schedule equal to depth of intervals?
Interval Partitioning:
Greedy Algorithm

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

Sort intervals by starting time so that $s_1 \leq s_2 \leq \ldots \leq s_n$.

d $\leftarrow$ 0  \quad \text{number of allocated classrooms}

\textbf{for} j = 1 \textbf{ to } n \textbf{ \{} \\
  \quad \text{if (lecture j is compatible with some classroom k)} \\
  \qquad \text{schedule lecture j in classroom k} \\
  \quad \text{else} \\
  \qquad \text{allocate a new classroom d + 1} \\
  \qquad \text{schedule lecture j in classroom d + 1} \\
  \qquad d \leftarrow d + 1 \\
\textbf{\}}
Interval Partitioning: Greedy Analysis

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

- Theorem. Greedy algorithm is optimal.

- Pf.
  - Let \( d \) = number of classrooms that the greedy algorithm allocates.
  - Classroom \( d \) is opened because we needed to schedule a job, say \( j \), that is incompatible with all \( d-1 \) other classrooms.
  - Since we sorted by start time, all these incompatibilities are caused by lectures that 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.
4.2 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: $\text{lateness}_j = \max \{ 0, f_j - d_j \}$.
- Goal: schedule all jobs to minimize maximum lateness $L = \max \text{lateness}_j$.

Note: a related problem of finding a maximal size subset of jobs that meets deadline is NP-hard.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_j$</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>3</td>
<td>2</td>
</tr>
<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>

Ex:

- Lateness = 0
- Lateness = 2
- 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 $t_j$.

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

    counterexample

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

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

    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 $[s_j, f_j]$

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 1 2 3</td>
<td>4 5 6</td>
<td>7 8 9</td>
<td>10 11 12</td>
<td>13 14 15</td>
<td></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

• 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.

• (In the above, remember that jobs sorted in ascending order of due dates. Thus $i < j$ means
  job $i$ is less than $j$ in this sorted order)
Minimizing Lateness: Inversions

- 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 \( \bar{x} \) be the lateness before the swap, and let \( \bar{x}' \) be it afterwards.
  - \( \bar{x}'_k = \bar{x}_k \) for all \( k \neq i, j \)
  - \( \bar{x}'_i \leq \bar{x}_i \) (this is trivially true)
  - If job \( j \) is late:
    - Note that this implies that the max lateness can only decrease (or be the same with the swap)

\[
\begin{align*}
\bar{x}_j &= f'_j - d_j \quad \text{(definition)} \\
&= f_i - d_j \quad \text{\((j \text{ finishes at time } f_i)\)} \\
&\leq f_i - d_i \quad \text{\((i < j)\)} \\
&\leq \bar{x} \quad \text{(definition)}
\end{align*}
\]
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.

- Myopic greed: "at every iteration, the algorithm chooses the best morsel it can swallow, without worrying about the future"
  - Objective function. Does not explicitly appear in greedy algorithm!
  - Hard, if not impossible, to precisely define "greedy algorithm."