CSE 417, Winter 2012

Greedy Algorithms

Ben Birnbaum
Widad Machmouchi

Slides adapted from Larry Ruzzo, Steve Tanimoto, and Kevin Wayne
Chapter 4

Greedy Algorithms
4.1 Interval Scheduling
Interval scheduling.

- Job $j$ 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.
Interval Scheduling: Greedy Algorithms

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

- [Earliest start time] Consider jobs in ascending order of \( s_j \).
- [Earliest finish time] Consider jobs in ascending order of \( f_j \).
- [Shortest interval] Consider jobs in 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 template. Consider jobs in some natural order. Take each job provided it's compatible with the ones already taken.

counterexample for earliest start time

counterexample for shortest interval

counterexample for fewest conflicts
**Interval Scheduling: 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 ≤ f_2 ≤ ... ≤ f_n.

set of jobs selected
A ← φ
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 $s_j ≥ f_{j^*}$.
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](https://via.placeholder.com/150)

**Greedy:** $i_1$, $i_2$, $i_r$, $i_{r+1}$

**OPT:** $j_1$, $j_2$, $j_r$, $j_{r+1}$, $\ldots$

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

why not replace job $j_{r+1}$ with job $i_{r+1}$?
Theorem. Greedy algorithm is optimal.

Proof. (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 $r$. 

\[
\begin{align*}
\text{Greedy:} & \quad i_1 \quad | \quad i_2 \quad | \quad i_r \quad | \quad i_{r+1} \\
\text{OPT:} & \quad j_1 \quad | \quad j_2 \quad | \quad j_r \quad | \quad i_{r+1} \quad | \quad \ldots
\end{align*}
\]

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

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.

![Diagram showing interval partitioning with lectures and time intervals.]

<table>
<thead>
<tr>
<th>Time</th>
<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>b</td>
<td></td>
<td>c</td>
<td></td>
<td>d</td>
<td></td>
<td>e</td>
<td></td>
<td>f</td>
<td></td>
<td>g</td>
<td></td>
<td>i</td>
<td></td>
</tr>
<tr>
<td>2</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>
<td></td>
<td></td>
</tr>
<tr>
<td>3</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>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- 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.

```plaintext
Sort intervals by starting time so that \( s_1 \leq s_2 \leq \ldots \leq s_n \).
\( d \leftarrow 0 \quad \text{number of allocated classrooms} \)

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.** \( O(n \log n) \).
- For each classroom \( k \), maintain the finish time of the last job added.
- Keep the classrooms in a priority queue.
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.
- These $d$ jobs each end after $s_j$.
- 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: \( \ell_j = \max \{ 0, f_j - d_j \} \).
- Goal: schedule all jobs to minimize maximum lateness \( L = \max \ell_j \).

Ex:

\[
\begin{array}{ccccccc}
  & 1 & 2 & 3 & 4 & 5 & 6 \\
  t_j & 3 & 2 & 1 & 4 & 3 & 2 \\
  d_j & 6 & 8 & 9 & 9 & 14 & 15 \\
\end{array}
\]

\[
\begin{array}{cccccccc}
  d_3 = 9 & d_2 = 8 & d_6 = 15 & d_1 = 6 & d_5 = 14 & d_4 = 9 \\
  0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 & 14 & 15 \\
\end{array}
\]

lateness = 2 \hspace{1cm} lateness = 0 \hspace{1cm} 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 $t_j$.

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

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

  \[
  \begin{array}{c|cc}
  & 1 & 2 \\
  \hline
  t_j & 1 & 10 \quad \text{counterexample} \\
  d_j & 2 & 10 \\
  \end{array}
  \]
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></th>
<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>
<td>6</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>4</td>
<td>5</td>
<td>7</td>
<td>9</td>
<td>11</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>9</td>
<td>12</td>
<td>12</td>
<td>13</td>
<td>15</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. Given a schedule S, an inversion 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.
Minimizing Lateness: Inversions

Def. Given a schedule $S$, an inversion is a pair of jobs $i$ and $j$ such that: $i < j$ but $j$ scheduled before $i$.

Claim. Swapping two consecutive, 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:

$$
\ell'_j = f' - d_j \quad \text{(definition)}
= f_i - d_j \quad \text{($j$ finishes at time $f_i$)}
\leq f_i - d_i \quad \text{($i < j$)}
\leq \ell_i \quad \text{(definition)}
$$
**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.

**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.

**Other greedy algorithms.** Kruskal, Prim, Dijkstra, Huffman, ...
4.5 Minimum Spanning Tree
Minimum Spanning Tree

**Minimum spanning tree.** Given a connected graph $G = (V, E)$ with real-valued edge weights $c_e$, an MST is a subset of the edges $T \subseteq E$ such that $T$ is a spanning tree whose sum of edge weights is minimized.

$G = (V, E)$

$T$, $\sum_{e \in T} c_e = 50$

**Cayley's Theorem.** There are $n^{n-2}$ spanning trees of $K_n$.  

↑

can't solve by brute force
Applications

MST is fundamental problem with diverse applications.

- Network design.
  - telephone, electrical, hydraulic, TV cable, computer, road

- Approximation algorithms for NP-hard problems.
  - traveling salesperson problem, Steiner tree

- Indirect applications.
  - max bottleneck paths
  - LDPC codes for error correction
  - image registration with Renyi entropy
  - learning salient features for real-time face verification
  - reducing data storage in sequencing amino acids in a protein
  - model locality of particle interactions in turbulent fluid flows
  - autoconfig protocol for Ethernet bridging to avoid cycles in a network
**Greedy Algorithms**

**Kruskal's algorithm.** Start with $T = \emptyset$. Consider edges in ascending order of cost. Insert edge $e$ in $T$ unless doing so would create a cycle.

**Prim's algorithm.** Start with some root node $s$ and greedily grow a tree $T$ from $s$ outward. At each step, add the cheapest edge $e$ to $T$ that has exactly one endpoint in $T$.
- Uses the same approach as Dijkstra's algorithm that you've seen before.

**Remark.** All these algorithms produce an MST.
Greedy Algorithms

Simplifying assumption. All edge costs $c_e$ are distinct.

Cut property. Let $S$ be any subset of nodes, and let $e$ be the min cost edge with exactly one endpoint in $S$. Then the MST contains $e$.

Cycle property. Let $C$ be any cycle, and let $f$ be the max cost edge belonging to $C$. Then the MST does not contain $f$. 

- $e$ is in the MST
- $f$ is not in the MST
Cycles and Cuts

**Cycle.** Set of edges the form a-b, b-c, c-d, ..., y-z, z-a.

![Diagram of a cycle]

**Cutset.** A cut is a subset of nodes S. The corresponding cutset D is the subset of edges with exactly one endpoint in S.

![Diagram of a cut]

\[
\text{Cycle } C = 1-2, 2-3, 3-4, 4-5, 5-6, 6-1
\]

\[
\text{Cut } S = \{4, 5, 8\}
\]

\[
\text{Cutset } D = 5-6, 5-7, 3-4, 3-5, 7-8
\]
**Claim.** A cycle and a cutset intersect in an even number of edges.

**Pf.** (by picture)
Greedy Algorithms

Simplifying assumption. All edge costs $c_e$ are distinct.

Cut property. Let $S$ be any subset of nodes, and let $e$ be the min cost edge with exactly one endpoint in $S$. Then the MST $T^*$ contains $e$.

Pf. (exchange argument)

- Suppose $e$ does not belong to $T^*$, and let's see what happens.
- Adding $e$ to $T^*$ creates a cycle $C$ in $T^*$.
- Edge $e$ is both in the cycle $C$ and in the cutset $D$ corresponding to $S$.
  - $\Rightarrow$ there exists another edge, say $f$, that is in both $C$ and $D$.
- $T' = T^* \cup \{e\} - \{f\}$ is also a spanning tree.
- Since $c_e < c_f$, $\text{cost}(T') < \text{cost}(T^*)$.
- This is a contradiction. □
Greedy Algorithms

Simplifying assumption. All edge costs $c_e$ are distinct.

Cycle property. Let $C$ be any cycle in $G$, and let $f$ be the max cost edge belonging to $C$. Then the MST $T^*$ does not contain $f$.

Pf. (exchange argument)
- Suppose $f$ belongs to $T^*$, and let's see what happens.
- Deleting $f$ from $T^*$ creates a cut $S$ in $T^*$.
- Edge $f$ is both in the cycle $C$ and in the cutset $D$ corresponding to $S$.
  $\Rightarrow$ there exists another edge, say $e$, that is in both $C$ and $D$.
- $T' = T^* \cup \{ e \} - \{ f \}$ is also a spanning tree.
- Since $c_e < c_f$, $\text{cost}(T') < \text{cost}(T^*)$.
- This is a contradiction. □
Kruskal's Algorithm: Proof of Correctness

Kruskal's algorithm. [Kruskal, 1956]
- Consider edges in ascending order of weight.
- Case 1: If adding \( e \) to \( T \) creates a cycle, discard \( e \) according to cycle property.
- Case 2: Otherwise, insert \( e = (u, v) \) into \( T \) according to cut property where \( S = \) set of nodes in \( u \)'s connected component.

Case 1

Case 2
Lexicographic Tiebreaking

To remove the assumption that all edge costs are distinct: perturb all edge costs by tiny amounts to break any ties.

Impact. Kruskal and Prim only interact with costs via pairwise comparisons. If perturbations are sufficiently small, MST with perturbed costs is MST with original costs. e.g., if all edge costs are integers, perturbing cost of edge e, by $i / n^2$

Implementation. Can handle arbitrarily small perturbations implicitly by breaking ties lexicographically, according to index.

Running Time: $O(m \log n)$
MST Algorithms: Theory

Deterministic comparison based algorithms.

- \( O(m \log n) \)  
  [Jarník, Prim, Dijkstra, Kruskal, Boruvka]
- \( O(m \log \log n) \).  
  [Cheriton-Tarjan 1976, Yao 1975]
- \( O(m \beta(m, n)) \).  
  [Fredman-Tarjan 1987]
- \( O(m \log \beta(m, n)) \).  
  [Gabow-Galil-Spencer-Tarjan 1986]
- \( O(m \alpha(m, n)) \).  
  [Chazelle 2000]

Holy grail. \( O(m) \).

Notable.

- \( O(m) \) randomized.  
  [Karger-Klein-Tarjan 1995]
- \( O(m) \) verification.  
  [Dixon-Rauch-Tarjan 1992]

Euclidean.

- 2-d: \( O(n \log n) \).  
  compute MST of edges in Delaunay
- k-d: \( O(k n^2) \).  
  dense Prim