Chapter 4

Greedy Algorithms

Intro: Coin Changing

**Coin Changing**

**Goal.** Given currency denominations: 1, 5, 10, 25, 100, give change to customer using **fewest** number of coins.

**Ex:** 34¢.

**Cashier’s algorithm.** At each iteration, give the **largest** coin valued ≤ the amount to be paid.

**Ex:** $2.89.

**Algorithm is “Greedy”: One large coin better than two or more smaller ones**

**Coin-Changing: Does Greedy Always Work?**

**Observation.** Greedy algorithm is sub-optimal for US postal denominations: 1, 10, 21, 34, 70, 100, 350, 1225, 1500.

**Counterexample.** 140¢.

- Greedy: 100, 34, 1, 1, 1, 1, 1.
- Optimal: 70, 70.

**Algorithm is “Greedy”, but also short-sighted – attractive choice now may lead to dead ends later.**

**Correctness is key!**
Outline & Goals

“Greedy Algorithms”
what they are

Pros
intuitive
often simple
often fast

Cons
often incorrect!

Proof techniques
stay ahead
structural
exchange arguments

4.1 Interval Scheduling

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.

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

- What order? Does that give best answer? Why or why not?
- Does it help to be greedy about order?
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 $c_j$.

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

[Earliest start time] breaks earliest start time

[Earliest finish time] breaks shortest interval

[Fewest conflicts] breaks fewest conflicts

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 $f_1 \leq f_2 \leq \ldots \leq f_n$.

A \leftarrow \phi

for $j = 1$ to $n$ {
    if (job $j$ compatible with $A$)
        $A \leftarrow A \cup \{j\}$
}

return $A$

Implementation. $O(n \log n)$.

- Remember job $j^*$ that was added last to $A$.
- Job $j$ is compatible with $A$ if $s_j \leq f_{j^*}$.

Interval Scheduling: Greedy Algorithm

Interval Scheduling: Greedy Algorithm
Interval Scheduling

Time

0 1 2 3 4 5 6 7 8 9 10 11

B C A

E D

H

Interval Scheduling

Time

0 1 2 3 4 5 6 7 8 9 10 11

B C A

E D

H

Interval Scheduling

Time

0 1 2 3 4 5 6 7 8 9 10 11

B C A

E D

H

Interval Scheduling

Time

0 1 2 3 4 5 6 7 8 9 10 11

B C A

E D

H
**Interval Scheduling: Correctness**

**Theorem.** Greedy algorithm is optimal.

**Pf.** ("greedy stays ahead")

Let \( i_1, i_2, \ldots, i_k \) be jobs picked by greedy, \( j_1, j_2, \ldots, j_m \) those in some optimal solution.

Show \( f(i_r) \leq f(j_r) \) by induction on \( r \).

**Basis:** \( i_1 \) chosen to have min finish time, so \( f(i_1) \leq f(j_1) \).

**Ind:** \( f(i_r) \leq f(j_r) \), so \( j_{r+1} \) is among the candidates considered by greedy when it picked \( i_{r+1} \), & it picks min finish, so \( f(i_{r+1}) \leq f(j_{r+1}) \).

Similarly, \( k \geq m \), else \( i_{k+1} \) is among (nonempty) set of candidates for \( i_{k+1} \).

**4.1 Interval Partitioning**

**Proof Technique 2: "Structural"**

**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 as Interval Graph Coloring**

Vertices = classes;  
edges = conflicting class pairs;  
different colors = different assigned rooms

Note: graph coloring is very hard in general, but graphs corresponding to interval intersections are a much simpler special case.
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: A "Structural" 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.

**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 $s_1 \leq s_2 \leq \ldots \leq s_n$.
\[ d \leftarrow 0 \quad \text{--- number of allocated classrooms} \]
\[ \text{for } j = 1 \text{ to } n \{ \]
\[ \quad \text{if (lect j is compatible with some classroom k, 1 \leq k \leq d)} \]
\[ \quad \quad \text{schedule lecture j in classroom k} \]
\[ \quad \text{else} \]
\[ \quad \quad \text{allocate a new classroom } d + 1 \]
\[ \quad \quad \text{schedule lecture j in classroom } d + 1 \]
\[ \quad d \leftarrow d + 1 \]
\[ \}\]
```

**Implementation? Run-time? Next HW**

### Interval Partitioning: Greedy Analysis

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

**Theorem.** Greedy algorithm is optimal.

**Pf (exploit structural property).**
- 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 $d-1$ previously used 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 $s_j + \varepsilon$, i.e. depth $\geq d$.
- "Key observation" $\Rightarrow$ all schedules use $\geq$ depth classrooms, so $d =$ depth and greedy is optimal.
Interval Partitioning: Alt Proof (An “Exchange Argument”)

- When 4th room added, room 1 was free; why not swap it in there?
- (A: it conflicts with later stuff in schedule, which dominoes)
- But: room 4 schedule after 11:00 is conflict-free; so is room 1 schedule, so could swap both post-11:00 schedules
- Why does it help? Delays needing 4th room; repeat.

Cleaner: “Let S* be an opt sched with latest use of last room. When that room is added, all others in use (else we could swap, contradicting ‘latest’) so #rooms = depth, hence optimal”

4.2 Scheduling to Minimize 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: $l_j = \max(0, f_j - d_j)$.
- Goal: schedule all jobs to minimize maximum lateness $L = \max l_j$.

Ex:

<table>
<thead>
<tr>
<th>t_j</th>
<th>d_j</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>9</td>
</tr>
<tr>
<td>1</td>
<td>9</td>
</tr>
<tr>
<td>5</td>
<td>14</td>
</tr>
</tbody>
</table>

Lateness = 2
Lateness = 0
Max Lateness = 6

Scheduling to Minimize Lateness

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

- **[Smallest slack]** Consider jobs in ascending order of slack \( d_j - t_j \).
  
  \[
  \begin{array}{c|c|c}
  j & t_j & d_j \\
  1 & 1 & 10 \\
  2 & 2 & 10 \\
  \end{array}
  \]

Minimizing Lateness: Greedy Algorithm

**Greedy algorithm.** Earliest deadline first.

1. Sort \( n \) jobs by deadline so that \( d_1 \leq d_2 \leq \ldots \leq d_n \).
2. \( t \leftarrow 0 \)
3. For \( j = 1 \) to \( n \):
   - Assign job \( j \) to interval \([t, t+t_j]\):
     - \( s_j \leftarrow t \), \( f_j \leftarrow t + t_j \)
     - \( t \leftarrow t + t_j \)
4. Output intervals \([s_j, f_j]\)

Minimizing Lateness: No Idle Time

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

<table>
<thead>
<tr>
<th>( d )</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>( d = 4 )</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>( d = 6 )</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>( d = 12 )</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>

<table>
<thead>
<tr>
<th>( d )</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>( d = 4 )</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>( d = 6 )</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>( d = 12 )</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>

**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:

- deadline \( i < j \) but \( j \) scheduled before \( i \).

\[
\text{deadline} \quad \	ext{later deadline} \quad \text{earlier deadline}
\]

**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. (If \( j \) & \( i \) aren’t consecutive, then look at the job \( k \) scheduled right after \( j \). If \( d_k < d_j \), then \((j,k)\) is a consecutive inversion; if not, then \((k,i)\) is an inversion, & nearer to each other - repeat.)

**Observation.** Swapping adjacent inversion reduces # inversions by 1 (exactly)
Minimizing Lateness: Inversions

**Def.** An inversion in schedule $S$ is a pair of jobs $i$ and $j$ such that: deadline $i < j$ but $j$ scheduled before $i$. 

![Diagram showing an inversion with job $i$ scheduled after job $j$.

**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 $k \neq i, j$
- $\ell'_i = \ell_i$
- $\ell'_j = f_j - d_j$ (definition)
- If job $j$ is now late:
  - $f'_j = f_j - d_j$ (definition)
  - $f'_i = f_i - d_i$ (definition)

Minimizing Lateness: No Inversions

**Claim.** All inversion-free schedules $S$ have the same max lateness.

**Pf.** If $S$ has no inversions, then deadlines of scheduled jobs are monotonically nondecreasing, i.e., they increase (or stay the same) as we walk through the schedule from left to right.

Two such schedules can differ only in the order of jobs with the same deadlines. Within a group of jobs with the same deadline, the max lateness is the lateness of the last job in the group - order within the group doesn't matter.

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.
4.3 Optimal Caching

1. Cache

   Pronunciation: 'kash
   Function: noun
   Etymology: French, from cache to press, hide
   a hiding place especially for concealing and preserving provisions or implements

2. Cache

   Function: transitive verb
   to place, hide, or store in a cache
   - Webster's Dictionary

Optimal Offline Caching

Caching:
- Cache with capacity to store k items.
- Sequence of m item requests \(d_1, d_2, \ldots, d_m\).
- Cache hit: item already in cache when requested.
- Cache miss: item not already in cache when requested; must bring requested item into cache, and evict some existing item, if full.

Goal. Eviction schedule that minimizes number of cache misses.

Ex: \(k = 2\), initial cache = \(ab\), requests: \(a, b, c, b, c, a, a, b\).
Optimal eviction schedule: 2 cache misses.

4.4 Shortest Paths in a Graph

You've seen this in 326 or 373, so this section and next two on min spanning tree are review. I won't lecture on them, but you should review the material. Both, but especially shortest paths, are common problems, having many applications.
Shortest Path Problem

Shortest path network.
- Directed graph $G = (V, E)$.
- Source $s$, destination $t$.
- Length $\ell_e = \text{length of edge } e$.

Shortest path problem: find shortest directed path from $s$ to $t$.

cost of path = sum of edge costs in path

Cost of path $s\rightarrow 2 \rightarrow 3 \rightarrow 5 \rightarrow t$
$= 9 + 23 + 2 + 16$
$= 48$.

Dijkstra's Algorithm

Dijkstra's algorithm.
- Maintain a set of explored nodes $S$ for which we have determined the shortest path distance $d(u)$ from $s$ to $u$.
- Initialize $S = \{s\}$, $d(s) = 0$.
- Repeatedly choose unexplored node $v$ which minimizes
  \[ \pi(v) = \min_{e = (u, v) : u \in S} d(u) + \ell_e, \]
  add $v$ to $S$, and set $d(v) = \pi(v)$.

shortest path to some $u$ in explored part, followed by a single edge $(u, v)$