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 choices 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

Proof Technique 1: “greedy stays ahead”

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?
Interval Scheduling: Greedy Algorithms

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

[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 \( c_j \).

[Earliest start time] Consider jobs in ascending order of start time \( s_j \).

[Earliest finish time] Consider jobs in ascending order of finish time \( f_j \).

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 \( f_1 \leq f_2 \leq \ldots \leq f_n \).

\[ A \leftarrow \emptyset \]

\[
\text{for } j = 1 \text{ to } n \{ \\
\quad \text{if job } j \text{ compatible with } A \\
\quad \quad A \leftarrow A \cup \{j\}
\}
\]

return \( A \)

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

Interval Scheduling

![Diagram of interval scheduling](image1.png)

Interval Scheduling

![Diagram of interval scheduling](image2.png)
Interval Scheduling

Time

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

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

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 $j_r$ 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 $j_{k+1}$ is among (nonempty) set of candidates for $i_{k+1}$

Greedy:

OPT:
4.2 Scheduling to Minimize Lateness

Proof Technique 2: “Exchange” Arguments

Minimizing Lateness: Greedy Algorithms

Greedy template. Consider jobs in some order.

[Shortest processing time first]
Consider jobs in ascending order of processing time \( t_j \).

[Smallest slack]
Consider jobs in ascending order of slack \( d_j - t_j \).

[Earliest deadline first]
Consider jobs in ascending order of deadline \( d_j \).

Minimizing Lateness: Greedy Algorithm

Greedy algorithm. Earliest deadline first.

1. Sort \( n \) jobs by deadline so that \( d_1 ≤ d_2 ≤ ... ≤ d_n \).
2. \( t ← 0 \)
3. for \( j = 1 \) to \( n \) do
   1. Assign job \( j \) to interval \( [t, t + t_j] \):
      1. \( s_j ← t \), \( f_j ← t + t_j \)
      2. \( t ← t + t_j \)
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 = 4</th>
<th>d = 6</th>
<th>d = 12</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>10</td>
<td>11</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.

**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.

**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.

**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.

**Observation.** Swapping adjacent inversion reduces # inversions by 1 (exactly)

**Claim.** Swapping two consecutive, inverted jobs reduces the number of inversions by one and does not increase the max lateness.

**Pf.**
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 $i$.

**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 now late:
  \[
  \ell'_j = f'_j - d_j = (f_j - d_j) \leq (f_i - d_j) = \ell_i
  \]
  \[
  \ell'_j = f'_j - d_j = (f_j - d_j) \leq (f_i - d_j) = \ell_i
  \]

Minimizing Lateness: Correctness of Greedy Algorithm

**Theorem.** Greedy schedule $S$ is optimal

**Pf.** Let $S^*$ be an optimal schedule with the fewest number of inversions.

Can assume $S^*$ has no idle time.

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 $S^*$.

So, $S^*$ has no inversions. But then $\text{Lateness}(S) = \text{Lateness}(S^*)$.

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

- Solve some special cases.
- Guess at some algorithms that might work.
- Try to distinguish between them by coming up with inputs on which they do different things.

Once you have a plausible candidate, try one of the following strategies for proving optimality:

**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. (Part of the cleverness is deciding what “good” is)

**Exchange argument.** Gradually transform any solution to the one found by the greedy algorithm without hurting its quality.
Problem

Given sequence $S$ of $n$ purchases at a stock exchange, possibly containing some events multiple times.
e.g.
Buy Amazon, Buy Google, Buy eBay, Buy Google, Buy Google, Buy Oracle
And another sequence $S'$ of $m$ purchases: Determine if $S'$ is a subsequence of $S$ in linear time.

Problem

You have $n$ jobs $J_1, J_2, \ldots, J_n$ each consisting of two stages:
- Preprocessing stage on a supercomputer
- Finishing stage on a PC

Second stage can be done in parallel (first stage has to be done sequentially).
- Job $J_i$ needs $p_i$ seconds of time on the supercomputer followed by $f_i$ seconds of time on a PC.

Design an algorithm that finds a schedule (order in which to process on supercomputer) that minimizes the completion time of the last job.