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.

Proof Technique 1: “greedy stays ahead”

Interval Scheduling: Greedy Algorithms

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.

[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 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 ← ∅
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 $s_j \geq f_{j^*}$.

Interval Scheduling

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

Time

B

A

E

D

C

F

H

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

Ind: if \( f(i_r) = f(j_r) = s(j_{r+1}) \), 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 \( j_{k+1} \) is among (nonempty) set of candidates for \( i_{k+1} \).

Greedy: \( i_1, i_2, i_3, \ldots, i_k \)
OPT: \( j_1, j_2, j_3, \ldots, j_m \)
4.1 Interval Partitioning

Proof Technique 2: “Structural”

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.

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 ≥ depth.

Ex: Depth of schedule below = 3 ⇒ 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.

Implementation. Run-time?

Exercises

4.2 Scheduling to Minimize Lateness

Proof Technique 3: “Exchange” Arguments
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: \( \ell_j = \max \{0, f_j - d_j\} \).
- Goal: schedule all jobs to minimize maximum lateness \( L = \max \ell_j \).

Ex:

\[
\begin{array}{cccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 \\
\hline
\tau_j & 1 & 2 & 3 & 4 & 5 & 6 \\
\eta_j & 6 & 6 & 9 & 9 & 14 & 15 \\
\end{array}
\]

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 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 \([t, t + t_j]\):
   \[ s_j \leftarrow t, \quad f_j \leftarrow t + t_j \]
   \[ t \leftarrow t + t_j \]
output intervals \([s_j, f_j]\)

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:

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.

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

Pf. Let \( \ell_i \) be the lateness before the swap, and let \( \ell_i' \) be it afterwards.

1. \( \ell_k' = \ell_k \) for all \( k \neq i, j \)
2. \( \ell_j' \leq \ell_j \)
3. If job \( j \) is now late:

\[
\ell_j' = \ell_j - d_j \quad \text{(definition)}
\]

\[
\leq \ell_i - d_j \quad \text{(definition)}
\]

only job moves later, but it's not later than \( i \) was, so max not increased
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. (Part of the cleverness is deciding what's "good.")

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. (Cleverness here is usually in finding a useful structural characteristic.)

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