Greedy Algorithms

Kleinberg and Tardos, Chapter 4

The road trip algorithm
- Road trip from Fort Collins to Durango on a given route with length \( L \), and fuel stations at positions \( b_i \).
- Fuel capacity \( = C \) miles.
- Goal: make as few refueling stops as possible.

Fort Collins       Durango

Greedy algorithm. Go as far as you can before refueling.
In general: determine a global optimum via locally optimal choices.

Selecting Breakpoints: Greedy Algorithm

The road trip algorithm.

Sort breakpoints so that: \( 0 = b_0 < b_1 < b_2 < \ldots < b_n = L \).

\[ S = \emptyset \quad \text{breakpoints selected} \]
\[ x = 0 \quad \text{current distance} \]

while (\( x \neq b_n \))
    let \( p \) be largest integer such that \( b_p \leq x + C \)
    if \( (b_p = x) \) return “no solution”
    \[ x = b_p \]
    \[ S = S \cup \{p\} \]
return \( S \)

Coin Changing

Goal. Given currency denominations: 1, 5, 10, 25, 100, devise a method to pay amount to customer using fewest number of coins.

Example: 34¢.

Cashier’s algorithm. At each iteration, add coin of the largest value that does not take us past the amount to be paid.

Example: $2.89.

Coin-Changing: Greedy doesn’t always work

Greedy algorithm works for US coins.
Fails when changing 40 when the denominations are 1, 5, 10, 20, 25.
Interval Scheduling

1. Job $j$ starts at $s_j$ and finishes at $f_j$.
2. Two jobs compatible if they don't overlap.

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

- (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 $c_j$.

Which of these don’t work? Let’s find a counter example.

Interval Scheduling: Greedy Algorithms

**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 ← φ
j ← 1
for i = 2 to n {
    if (job $i$ compatible with $A$)
        $A ← A ∪ \{i\}$
}
return $A$
```

Implementation.

- When is job $i$ compatible with $A$?

Interval Scheduling: Greedy Algorithms

**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 ← φ
j ← 1
for i = 2 to n {
    if $f_i \geq F_j$
        $A ← A ∪ \{i\}$
    j ← i
}
return $A$
```

Running time? $O(n \log n)$.
Theorem. The greedy algorithm is optimal.

Proof. (by contradiction)
- Assume greedy is not optimal, and let’s see what happens.
- Let \( i_1, i_2, \ldots, i_k \) be the jobs selected by greedy.
- Let \( j_1, j_2, \ldots, j_m \) be the 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 \( r \).

What did we do?
We assumed there was a non-greedy optimal solution, then we stepwise morphed this solution into a greedy optimal solution, thereby showing that the greedy solution works in the first place.

The book: "the Greedy Algorithm stays ahead"
Independent Set

Let's try to find a greedy algorithm for the independent set problem.

Is this algorithm optimal?

Conclusion: Greedy Algorithms

At every step, greedy algorithm makes the locally optimal choice, "without worrying about the future".

Not all problems have a greedy solution.
None of the NP-complete problems (e.g., TSP) have a greedy solution.

Other greedy algorithms: Kruskal, Prim, Dijkstra, Huffman, ...

Interval Scheduling

- 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.

This schedule uses 4 classrooms to schedule 10 lectures:

Interval Scheduling: Lower Bound

Key observation. Number of classrooms needed $\geq$ depth (maximum number of intervals at a time point)

Example: Depth of schedule below = 3 $\Rightarrow$ schedule is optimal.

Q. Does there always exist a schedule equal to depth of intervals?

Interval Scheduling: Greedy Algorithm

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

Allocate $d$ labels ($d =$ depth)
Sort the intervals by starting time: $I_1, I_2, \ldots, I_n$
For $j = 1$ to $n$
    - For each interval $I_j$ that precedes and overlaps with $I_i$, exclude its label for $I_j$
    - Pick a remaining label for $I_j$
Greedy works

allocate d labels (d = depth)
sort the intervals by starting time: \( I_1, I_2, \ldots, I_n \)

for \( j = 1 \) to \( n \)

for each interval \( I_j \) that precedes and overlaps with \( I_i \), exclude its label for \( I_j \)
pick a remaining label for \( I_j \)

Observations:
- There is always a label for \( I_j \)
- No overlapping intervals get the same label

Minimizing Lateness: Greedy Strategies

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 \( d_j - t_j \)

Minimizing Lateness: Greedy Algorithm

Greedy algorithm. Earliest deadline first.

sort 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 \( [t, t + t_j] \)

\( t \leftarrow t + t_j \)

max lateness = 1

Minimizing Lateness: No Idle Time

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

Example:

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14

*counterexample*

d = 4
d = 6
d = 12

Observation. The greedy schedule has no idle time.
Minimizing Lateness: Inversions

**Def.** Given a schedule A, an inversion is a pair of jobs i and j such that: \( d_i < d_j \) but j scheduled before i.

**Observation.** Greedy schedule has no inversions.

---

Minimizing Lateness: Inversions

**Def.** Given a schedule A, an inversion is a pair of jobs i and j such that: \( d_i < d_j \) but j scheduled before i.

**Observation.** All schedules with no inversions and no idle time have the same maximum lateness. How are they different? In the order of jobs with the same deadline are scheduled.

---

Minimizing Lateness: Inversions

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

**Proof.** Let \( l \) be the lateness before the swap, and let \( l' \) be the lateness afterwards.
- \( l'_k = l_k \) for all \( k \neq i, j \)
- \( l'_i = l_i \)

\[
\begin{align*}
l & = f_i - d_i \\
l & = f_j - d_j \\
l & = f_i - d_j \\
l' & = f'_i - d_i \\
l' & = f'_i - d_j \\
l' & = f_i - d_j
\end{align*}
\]

If we show that there is an optimal schedule with no inversions \( \Rightarrow \) the greedy schedule is optimal.

---

Minimizing Lateness: Analysis of Greedy Algorithm

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

**Proof.** Let S* be an optimal schedule.
- Can assume S* has no idle time.
- If S* has no inversions, then \( l(S) = l(S*) \).
- 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.
  - Continue until no inversions left.
- At each step lateness not increased, i.e \( l(S) = l(S*) \), and since S* is optimal, \( l(S) = l(S*) \).
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.

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. Incrementally transform any solution to the greedy one without hurting its quality.

Other greedy algorithms. Kruskal, Prim, Dijkstra, Huffman, ...