Chapter 4

Greedy Algorithms
Goals

Understand that sometimes greed is good optimal!

Be able to analyze whether a greedy algorithm is optimal
• show it “stays ahead” of any other algorithm
• inductively
• lower bound the optimal solution, show that greedy achieves this bound
• exchangability and other problem structure

Problems:
• Interval scheduling
• Coin changing
• Optimal caching
• Shortest path
• Minimum spanning tree
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.

![Diagram showing interval scheduling with jobs a through h]
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 $c_j$. 
Interval Scheduling: Greedy Algorithms

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

- Breaks earliest start time
- Breaks shortest interval
- Breaks 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 \( f_1 \leq f_2 \leq \ldots \leq f_n \).
  - jobs selected
  - \( A \leftarrow \emptyset \)
  - 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 \( s_j \geq f_{j^*} \).
Interval Scheduling Example

A

B

C

D

E

F

G

H
Interval Scheduling Example

A

B

C

D

E

F

G

H
Interval Scheduling Example

Time Schedule Example
Interval Scheduling Example

Time

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

A B C D E F G H
Interval Scheduling Example
Interval Scheduling Example
Interval Scheduling Example

Time

A

B

C

D

E

F

G

H
Interval Scheduling Example

Diagram showing the scheduling of tasks A, B, C, D, E, F, G, and H over a time interval from 0 to 11.
Interval Scheduling Example
Interval Scheduling Example
Interval Scheduling: Analysis

**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 showing Greedy and OPT schedules with jobs $i_1, i_2, \ldots, i_r$, and $j_1, j_2, \ldots, j_r$. There is a question mark indicating why not replace job $j_{r+1}$ with job $i_{r+1}$?](image)

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.

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 showing greedy and optimal solutions with job $i_{r+1}$ finishes before $j_{r+1}$, solution still feasible and optimal, but contradicts maximality of $r$.]
Greed is good. Greed is right. Greed works. Greed clarifies, cuts through, and captures the essence of the evolutionary spirit.

- Gordon Gecko (Michael Douglas)
Coin Changing

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

**Ex:** 34¢.

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

**Ex:** $2.89.
Coin-Changing: Greedy Algorithm

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

Sort coins denominations by value: \( c_1 < c_2 < \ldots < c_n \).

coins selected
\[
S \leftarrow \emptyset
\]

while \( (x \neq 0) \) { 
  let \( k \) be largest integer such that \( c_k \leq x \)
  if \( (k = 0) \)
    return "no solution found"
  \( x \leftarrow x - c_k \)
  \( S \leftarrow S \cup \{k\} \)
}

return \( S \)

Q. Is cashier's algorithm optimal?
Coin-Changing: Analysis of Greedy Algorithm

Theorem. Greed is optimal for U.S. coinage: 1, 5, 10, 25, 100.

Proof (by induction on \( x \))

1. Consider optimal way to change \( c_k \leq x < c_{k+1} \): greedy takes coin \( k \).
2. We claim that any optimal solution must also take coin \( k \).
   - if not, it needs enough coins of type \( c_1, \ldots, c_{k-1} \) to add up to \( x \)
   - table below indicates no optimal solution can do this
3. Problem reduces to coin-changing \( x - c_k \) cents, which, by induction, is optimally solved by greedy algorithm.

<table>
<thead>
<tr>
<th>( k )</th>
<th>( c_k )</th>
<th>All optimal solutions must satisfy</th>
<th>Max value of coins 1, 2, \ldots, ( k-1 ) in any OPT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>( P \leq 4 )</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>( N \leq 1 )</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>( N + D \leq 2 )</td>
<td>( 4 + 5 = 9 )</td>
</tr>
<tr>
<td>4</td>
<td>25</td>
<td>( Q \leq 3 )</td>
<td>( 20 + 4 = 24 )</td>
</tr>
<tr>
<td>5</td>
<td>100</td>
<td>no limit</td>
<td>( 75 + 24 = 99 )</td>
</tr>
</tbody>
</table>
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, 1.
- Optimal: 70, 70.
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.
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.

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 \( s_1 \leq s_2 \leq \ldots \leq s_n \).
\[ d \leftarrow 0 \quad \text{number of allocated classrooms} \]

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.
- 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.