Chapter 4

Greedy Algorithms

Coin Changing

“Greed is good. Greed is right. Greed works. Greed clarifies, cuts through, and captures the essence of the evolutionary spirit.”

- Gordon Gecko (Michael Douglas)
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.
Interval Scheduling: Greedy Algorithms

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

- **[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 $c_j$.

---

Counterexamples for:

- Earliest start time
- Shortest interval
- 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 $f_1 \leq f_2 \leq \ldots \leq f_n$.

\[ A \leftarrow \emptyset \]

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

return $A$

---

**Interval Scheduling**

![Interval Scheduling Diagram](image)
Interval Scheduling

Time

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

A
B
C
D
E
F
G
H

Interval Scheduling

Time

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

A
B
C
D
E
F
G
H

Interval Scheduling: Greedy Algorithm

**Greedy algorithm.** Consider jobs in increasing order of finish time.

```plaintext
Sort jobs by finish times so that \( f_1 \leq f_2 \leq \ldots \leq f_n \).
A ← ∅
for \( j = 1 \) to \( n \) {
    if (job \( j \) compatible with \( A \))
        A ← A ∪ {j}
}
return A
```

**Runtime Analysis.** \( O(\text{??}) \).
- Remember job \( j^* \) that was added last to \( A \).
- Job \( j \) is compatible with \( A \) if \( s_j \geq f_r \).
- Don't forget time to sort!
- \( O(n \log n) \)

Interval Scheduling: Analysis

**Theorem.** Greedy algorithm is optimal.

**Pf.** (direct proof)
- 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 any other solution.
- Consider the largest \( r \) such that \( i_1 = j_1, i_2 = j_2, \ldots, i_r = j_r \).

Greedy: \( i_1, i_2, \ldots, i_k \)
Other: \( j_1, j_2, \ldots, j_m \)

Job \( i_{k+1} \) finishes before \( j_{r+1} \)

Why not replace job \( j_{r+1} \) with job \( i_{k+1} \)?
Interval Scheduling: Analysis

**Theorem.** Greedy algorithm is optimal.

**Pf.** (direct proof)
- 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 any other solution.
- Consider the largest $r$ such that $i_1 = j_1, i_2 = j_2, \ldots, i_r = j_r$.

Greedy: $i_1$ $i_2$ $i_r$ $i_{r+1}$

Other: $j_1$ $j_2$ $j_r$ $i_{r+1}$

- The greedy algorithm always "stays ahead" of any other solution!

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.

Ex:

This schedule uses 3. 

Ex:
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.

```plaintext
Sort intervals by starting time so $s_1 \leq s_2 \leq \ldots \leq s_n$.
d $\leftarrow$ 0

for $j = 1$ to $n$
    if (lecture $j$ is compatible with some classroom $k$)
        schedule lecture $j$ in room $k$
    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 (key with finish time above).
Interval Partitioning: Greedy Analysis

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.

*These \( d \) jobs each end after \( s_j \).

*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.

Administrative Stuff

Add/Drop Class. Please do so as soon as possible as there are students on the waiting list.

Late Days. Due to feedback, you now have 2 late days. RECALL: Due time is 2:30!! After class is considered late!!

Homework Solutions. Will not post or hand out "official" solutions, but will distribute an "ideal" solution each week.
### 4.2 Scheduling to Minimize Lateness

**Minimizing lateness problem.**
- Single machine 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>3</th>
<th>2</th>
<th>1</th>
<th>4</th>
<th>3</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$d_j$</td>
<td>6</td>
<td>8</td>
<td>9</td>
<td>9</td>
<td>14</td>
<td>15</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$d_4 = 9$</th>
<th>$d_2 = 8$</th>
<th>$d_6 = 15$</th>
<th>$d_4 = 6$</th>
<th>$d_5 = 14$</th>
<th>$d_4 = 9$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
</tbody>
</table>

max lateness = 6

lateness = 2

lateness = 0

$\text{ex}$
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 $d_j - t_j$.

\begin{center}
\begin{tabular}{c|c|c}
\hline
$i$ & $1$ & $2$ \\
\hline
$t_j$ & 1 & 10 \\
$d_j$ & 100 & 10 \\
\hline
\end{tabular}
\end{center}

counterexample

\begin{center}
\begin{tabular}{c|c|c}
\hline
$i$ & $1$ & $2$ \\
\hline
$t_j$ & 1 & 10 \\
$d_j$ & 2 & 10 \\
\hline
\end{tabular}
\end{center}

counterexample
Minimizing Lateness: Greedy Algorithm

Greedy algorithm. Earliest deadline first.

Sort \( n \) jobs by deadline so \( d_1 \leq d_2 \leq \ldots \leq d_n \)
\( t \leftarrow 0 \)
for \( j = 1 \) to \( n \)
  Assign job \( j \) to interval \([t, t + t_j]\)
  \( s_j \leftarrow t, \ f_j \leftarrow t + t_j \)
  \( t \leftarrow t + t_j \)
output intervals \([s_j, f_j]\)

max lateness = 1

\[
\begin{array}{cccccccc}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\hline
d_1 = 6 & d_2 = 8 & d_3 = 9 & d_4 = 9 & d_5 = 14 & d_6 = 15 \\
\end{array}
\]

Minimizing Lateness: No Idle Time

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

\[
\begin{array}{cccc}
0 & 1 & 2 & 3 \\
\hline
d = 4 & d = 6 & d = 12 \\
7 & 8 & 9 & 10 & 11 \\
\end{array}
\]

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

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

\[
\text{before swap} \quad \begin{array}{ccccccc}
& & & & & & \\
& & & & & & \\
& & & & & & \\
& & & & & & \\
& & & & & & \\
& i & j & i & & & \\
\end{array}
\]

\[
\text{after swap} \quad \begin{array}{ccccccc}
& & & & & & \\
& & & & & & \\
& & & & & & \\
& & & & & & \\
& & & & & & \\
& i & j & i & & & \\
\end{array}
\]

**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 $\ell$ be the lateness before the swap, and let $\ell'$ be it afterwards.

- $\ell'_{i,j} = \ell_k$ for all $k \neq i, j$
- $\ell'_{i,j} \leq \ell_i$
- If job $j$ is late:

\[
\ell'_j = f'_j - d_j \quad \text{(definition)}
\]

\[
= f_i - d_j \quad (j \text{ finishes at time } f_j)
\]

\[
\leq f_i - d_i \quad (i < j)
\]

\[
\leq \ell_i \quad \text{(definition)}
\]
Minimizing Lateness: Analysis of Greedy Algorithm

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

Theorem. Greedy schedule $S$ is optimal.

Pf. Define $S^*$ to be an optimal schedule.

- Can assume $S^*$ has no idle time by earlier observation.
- If $S^*$ has no inversions, then $S = S^*$.
- If $S^*$ has an inversion, then it has an adjacent inversion, say $i$-$j$.
  - By the claim above, swapping $i$ and $j$ does not increase the maximum lateness and strictly decreases the number of inversions.
  - Let this new optimal solution be $S^*_2$.
- We can repeat above steps until no inversions remain.
- Hence, $S = S^*_k$ is optimal. □

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 (e.g. Interval Scheduling).

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 (e.g. Interval Partitioning).

Exchange argument. Gradually transform an optimal solution to the one found by the greedy algorithm without hurting its quality (e.g. Minimizing Lateness).
In-Class Exercise

The Problem: You are a stamp collector, and treat yourself to one expensive stamp each year. However, these items go up in price exponentially the longer you wait. There are $n$ stamps you want. Assume stamp $i$ has price $100a_i^t$ if you buy it $t$ years from now (where $a_i > 1$).

What sequence of purchases will minimize the amount you spend?

1. Find at least two “natural” orders for a greedy algorithm.
2. Find a counterexample for as many of these orders as possible.
3. Choose a possible order, and try to prove the greedy solution works.

Variation. You shop at a fashion boutique where accessories are expensive, but devalue quickly. You can purchase item $i$ for $100a_i^t$ if you buy it $t$ weeks from now (where $a_i < 1$). There are $n$ accessories you want.

What sequence of purchases will minimize the amount of money you spend? Answer questions 1-3 as above.

4.3 Optimal Caching
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.

Optimal Offline Caching: Farthest-In-Future

Farthest-in-future. Evict item in the cache that is not requested until farthest in the future.

current cache: \[ \begin{array}{c}
  a \\
b \\
c \\
d \\
e \\
f \\
\end{array} \]

future queries: \[ \begin{array}{cccccccccccc}
g & a & b & c & d & a & b & b & a & c & d & e & f & a & d & e & f & h \ldots
\end{array} \]

\[ \begin{array}{c}
  \text{cache miss} \\
  \text{eject this one} \\
\end{array} \]

Theorem. [Bellady, 1960s] FF is optimal eviction schedule.

Pf. Algorithm and theorem are intuitive; proof is subtle.
Reduced Eviction Schedules

**Def.** A reduced schedule is a schedule that only inserts an item into the cache in a step in which that item is requested.

**Intuition.** Can transform an unreduced schedule into a reduced one with no more cache misses.

---

Farthest-In-Future: Analysis

**Theorem.** FF is optimal eviction algorithm.

**Pf.** (by induction on number of requests $j$)

**Induction Hypothesis:** There exists an optimal reduced schedule $S$ that makes the same eviction schedule as $S_{FF}$ through the first $j$ requests.

Let $S$ be reduced schedule that satisfies invariant through $j$ requests. We produce $S'$ that satisfies invariant after $j+1$ requests.

- **Consider** $(j+1)^{th}$ request $d = d_{j+1}$.
- **Since** $S$ and $S_{FF}$ have agreed up until now, they have the same cache contents before request $j+1$.

**Case 1:** $(d$ is already in the cache).

- $S' = S$ satisfies invariant.

**Case 2:** $(d$ is not in the cache and $S$ and $S_{FF}$ evict the same element).

- $S' = S$ satisfies invariant.
Farthest-In-Future: Analysis

Pf. (continued)

- Case 3: (d is not in the cache; S_{FF} evicts e; S evicts f ≠ e).
  - begin construction of S' from S by evicting e instead of f

\[
\begin{array}{ccc}
  \text{j} & \text{same} & \text{e} & \text{f} \\
  \text{j+1} & \text{same} & \text{e} & \text{d} \\
  \end{array}
\]

\[
\begin{array}{ccc}
  \text{S} & \text{S'} = S_{FF} \\
  \end{array}
\]

- now S' agrees with S_{FF} on first j+1 requests

- must show that for S', having element f in cache is no worse than having element e

Farthest-In-Future: Analysis

Let j' be the first time after j+1 that S and S' take a different action, and let g be item requested at time j'.

\[
\begin{array}{ccc}
  \text{j'} & \text{same} & \text{e} & \text{f} \\
  \text{S} & \text{S'} \\
  \end{array}
\]

must involve e or f (or both)

- Case 3a: g = e. Can't happen with Farthest-In-Future since there must be a request for f before e.

- Case 3b: g = f. Element f can't be in cache of S, so let e' be the element that S evicts.
  - if e' = e, S' accesses f from cache; now S and S' have same cache
  - if e' ≠ e, S' evicts e' and brings e into the cache; now S and S' have the same cache

Note: S' is no longer reduced, but can be transformed into a reduced schedule that agrees with S_{FF} through step j+1
Farthest-In-Future: Analysis

Let \( j' \) be the first time after \( j+1 \) that \( S \) and \( S' \) take a different action, and let \( g \) be item requested at time \( j' \).

If \( j' \) is \( \text{same} \) then \( e \) or \( f \) (or both) must involve \( e \) or \( f \) (or both).

\[
\begin{array}{c|c|c}
\text{j'} & e & f \\
\hline
S & \text{same} & e \\
S' & \text{same} & f \\
\end{array}
\]

otherwise \( S' \) would take the same action

- **Case 3c**: \( g \neq e, f \). \( S \) must evict \( e \).

  Make \( S' \) evict \( f \); now \( S \) and \( S' \) have the same cache.

\[
\begin{array}{c|c|c}
\text{j'} & g & g \\
\hline
S & \text{same} & g \\
S' & \text{same} & g \\
\end{array}
\]

Caching Perspective

**Online vs. offline algorithms.**
- *Offline*: full sequence of requests is known a priori.
- *Online (reality)*: requests are not known in advance.
- Caching is among most fundamental online problems in CS.

- **LIFO.** Evict page brought in most recently.
- **LRU.** Evict page whose most recent access was earliest.

**Theorem.** FF is optimal offline eviction algorithm.

- Provides basis for understanding and analyzing online algorithms.
- LRU is \( k \)-competitive. [Section 13.8]
- LIFO is arbitrarily bad.
In-Class Exercise

Answer T/F for the following questions:

1. $2^{n+1} \in O(2^n)$
2. $2^{2n} \in O(2^n)$
3. $\log_2(8n^{10}) \in O(\log_{10} n)$
4. $4^{\log_2 n} \in O(n^3)$
5. $n^{1/3} + \log_2 n \in O(\log_2 n)$