4. Greed

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, 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: $.89
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: \( c_1 < c_2 < \ldots < c_n \).

\[
\begin{align*}
S & \leftarrow \emptyset \\
\text{while } (x \neq 0) \{ \\
& \text{let } k \text{ be largest integer such that } c_k \leq x \\
& \quad \text{if } (k = 0) \\
& \quad \quad \text{return } \text{"no solution found"} \\
& \quad x \leftarrow x - c_k \\
& \quad S \leftarrow S \cup \{k\} \\
\} \\
\text{return } S
\end{align*}
\]

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

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

**Pf.**
- Optimum has ≤4 pennies (will choose 5 instead of 5*1)
- Optimum has ≤1 nickel (will choose 1*10 instead of 2*5)
- Optimum has ≤ 2 dimes (Optimum will choose 25+5 instead of 3*10)
- Optimum doesn’t have 2 dimes and a nickel (otherwise, will choose 25)
- Opt & greedy have at most 24 in P, N, D. Above 24 uses as many Q as possible: remainder P, N, D
- By inspection, Opt & Greedy agree on 1..24

<table>
<thead>
<tr>
<th>k</th>
<th>(c_k)</th>
<th>All optimal solutions must satisfy</th>
<th>Max value of coins 1, 2, ..., 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>
</tbody>
</table>
Observation. Greedy algorithm is sub-optimal without nickels.

Counterexample. 30¢.

- Greedy: 25, 1, 1, 1, 1, 1.
- Optimal: 10, 10, 10.
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.
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

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

\text{return } A

Implementation. $O(n \log n)$.

- Remember job $j^*$ that was added last to $A$.
- Job $j$ is compatible with $A$ if $s_j \geq f_{j^*}$.
**Theorem.** Greedy algorithm is optimal.

**Pf.** (by induction)

- There is an optimal solution $O$ whose choices $O_1..O_n$ match the greedy algorithm's choices $G_1..G_n$
- Base case (first job)
  - Take any optimal solution $O$. If $O_1 \neq G_1$, then replace $G_1$ with $O_1$ (it remains compatible since $\text{start}(O_1) \geq \text{finish}(G_1)$)
- Induction (assume there is an optimal solution $O$ whose choices $O_1..O_i$ match $G_1..G_i$). Show there exists an optimal solution $O'$ starting with $G_1..G_{i+1}$
  - If $O_{i+1} \neq G_{i+1}$, then $\text{finish}(O_{i+1}) \geq \text{finish}(G_{i+1})$ (by algorithm)
  - $O_1..O_i,G_{i+1},O_{i+2},..O_n$ is compatible and optimal (splice-in)
- Greedy algorithm can't have fewer choices than optimal (by algorithm)

<table>
<thead>
<tr>
<th>Greedy:</th>
<th>$G_1$</th>
<th>$G_2$</th>
<th>$G_i$</th>
<th>$G_{i+1}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$O$:</td>
<td>$O_1$</td>
<td>$O_2$</td>
<td>$O_i$</td>
<td>$O_{i+1}$</td>
</tr>
<tr>
<td>$O'$:</td>
<td>$O_1$</td>
<td>$O_2$</td>
<td>$O_i$</td>
<td>$G_{i+1}$</td>
</tr>
</tbody>
</table>

---

Interval Scheduling: Analysis
Selecting Breakpoints
Selecting Breakpoints

Selecting breakpoints.

- Road trip from San Diego to Seattle along fixed route.
- Refueling stations at certain points along the way.
- Fuel capacity = $C$.
- Goal: makes as few refueling stops as possible.

Greedy algorithm. Go as far as you can before refueling.
Selecting Breakpoints: Greedy Algorithm

Truck driver’s algorithm.

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

\[
\begin{align*}
S &\leftarrow \{0\} \quad \text{breakpoints selected} \\
x &\leftarrow 0 \quad \text{current location} \\
\textbf{while} \ (x < b_n) \\
&\quad \text{let } p \text{ be largest integer such that } b_p \leq x + C \\
&\quad \text{if } (b_p = x) \\
&\quad \quad \text{return "no solution"} \\
&\quad x \leftarrow b_p \\
S &\leftarrow S \cup \{p\} \\
\textbf{return } S
\end{align*}
\]

Implementation. \( O(n \log n) \)

- Use binary search to select each breakpoint \( p \).
Theorem. Greedy algorithm is optimal.

Pf. (by induction)

- Lemma: For any optimal solution, \(O_i \leq G_i\), for \(1 \leq i \leq k\) (\(k = \#\) breakpoints in \(O\)) (proof by induction)
- Base case (first job)
  - \(G_1\) is maximum possible, so \(O_1 \leq G_1\)
- Induction (assume that for any optimal solution \(O\), \(O_{i-1} \leq G_{i-1}\)). Show \(O_i \leq G_i\)
  - If \(O_{i+1} > G_{i+1}\), then greedy algorithm would have chosen it. (algorithm chooses largest possible)
- \(|G| = |O|\). If not, then there’s a \(G_{k+1}\). But \(O_k = b_n\) and \(G_k \geq O_k\) (by lemma). But, algorithm stops when \(G_i \geq b_n\), so there’s no such \(G_{k+1}\).

Selecting Breakpoints: Analysis

<table>
<thead>
<tr>
<th>Greedy:</th>
<th>G_1</th>
<th>G_2</th>
<th>G_{i-1}</th>
<th>G_k</th>
</tr>
</thead>
<tbody>
<tr>
<td>O:</td>
<td>O_1</td>
<td>O_2</td>
<td>O_{k-1}</td>
<td>O_k</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>b_n</td>
</tr>
</tbody>
</table>

"finish line"
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.

<table>
<thead>
<tr>
<th>9</th>
<th>9:30</th>
<th>10</th>
<th>10:30</th>
<th>11</th>
<th>11:30</th>
<th>12</th>
<th>12:30</th>
<th>1</th>
<th>1:30</th>
<th>2</th>
<th>2:30</th>
<th>3</th>
<th>3:30</th>
<th>4</th>
<th>4:30</th>
</tr>
</thead>
</table>
| a, b, c all contain 9:30

**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 $s_1 \leq s_2 \leq \ldots \leq s_n$.

```plaintext
\[
d \leftarrow 0 \quad \text{number of allocated classrooms}
\]

```plaintext
\begin{align*}
\text{for } j = 1 \text{ to } n \{ & \\
\quad \text{if (lecture } j \text{ is compatible with some classroom } k) } & \text{schedule lecture } j \text{ in classroom } k \\
\quad \text{else} & \text{allocate a new classroom } d + 1 \\
& \text{schedule lecture } j \text{ in classroom } d + 1 \\
& d \leftarrow d + 1
\}
\end{align*}

\]

**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.
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.
- 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.
4.2 Scheduling to Minimize Lateness
Scheduling to Minimizing 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:

<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>

lateness = 2  lateness = 0  max lateness = 6

$d_3 = 9$  $d_2 = 8$  $d_6 = 15$  $d_1 = 6$  $d_5 = 14$  $d_4 = 9$

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
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 Algorithms

Greedy template. Consider jobs in some order.

- **[Shortest processing time first]** Consider jobs in ascending order of processing time $t_j$.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_j$</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>$d_j$</td>
<td>100</td>
<td>10</td>
</tr>
</tbody>
</table>

counterexample

- **[Smallest slack]** Consider jobs in ascending order of slack $d_j - t_j$.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_j$</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>$d_j$</td>
<td>2</td>
<td>10</td>
</tr>
</tbody>
</table>

counterexample
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 $[t, t + t_j]$
    $s_j \leftarrow t$, $f_j \leftarrow t + t_j$
    $t \leftarrow t + t_j$
output intervals $[s_j, f_j]$
```

max lateness = 1

<table>
<thead>
<tr>
<th></th>
<th>$d_1$</th>
<th>$d_2$</th>
<th>$d_3$</th>
<th>$d_4$</th>
<th>$d_5$</th>
<th>$d_6$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
<td>13</td>
</tr>
<tr>
<td>14</td>
<td>15</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Minimizing Lateness: No Idle Time

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

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Observation.** The greedy schedule has no idle time.
Def. An inversion in schedule $S$ 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.

Observation. If a schedule (with no idle time) has an inversion, it has one with a pair of inverted jobs scheduled consecutively.
Minimizing Lateness: Inversions

**Def.** An inversion in schedule $S$ is a pair of jobs $i$ and $j$ such that $d_i < d_j$ but $j$ scheduled before $i$.

**Claim.** Swapping two adjacent, 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 late:
  \[
  \ell'_j = f'_j - d_j \quad \text{(definition)} \\
  = f_i - d_j \quad \text{($j$ finishes at time $f_i$)} \\
  \leq f_i - d_i \quad \text{($i < j$)} \\
  \leq \ell_i \quad \text{(definition)}
  \]
Minimizing Lateness: Analysis of Greedy Algorithm

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

**Pf.** Define $S^*$ to be an optimal schedule that has the fewest number of inversions, and let's see what happens.

- Can assume $S^*$ has no idle time.
- If $S^*$ has no inversions, then $S = S^*$.
- 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
  - this contradicts definition of $S^*$
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 algorithm's.

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

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.
4.3 Optimal Caching
Optimal Offline Caching

Caching.
- Cache with capacity to store $k$ items.
- Sequence of $m$ item requests $d_1, d_2, ..., d_m$.
- 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: \[ a \ b \ c \ d \ e \ f \]

Future queries: \[ g \ a \ b \ c \ e \ d \ a \ b \ b \ a \ c \ d \ e \ a \ f \ a \ d \ e \ f \ g \ h \ldots \]

Cache miss \[ \uparrow \] eject this one

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

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

**Claim.** Given any unreduced schedule S, can transform it into a reduced schedule S' with no more cache misses.

**Pf.** (by induction on number of unreduced items)

- Suppose S brings d into the cache at time t, without a request.
- Let c be the item S evicts when it brings d into the cache.
- **Case 1:** d evicted at time t', before next request for d.
- **Case 2:** d requested at time t' before d is evicted.

![Diagram](Case 1)

![Diagram](Case 2)
Farthest-In-Future: Analysis

Theorem. FF is optimal eviction algorithm.

Pf. (by induction on number or requests j)

Let S be reduced schedule that satisfies invariant through j requests.

We produce S' that satisfies invariant after j+1 requests.

- Consider (j+1)\textsuperscript{st} 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 $f \neq e$).
  - begin construction of $S'$ from S by evicting e instead of f

<table>
<thead>
<tr>
<th></th>
<th>$j$</th>
<th></th>
<th>$j+1$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$S$</td>
<td>same</td>
<td>e</td>
<td>f</td>
</tr>
<tr>
<td>$S'$</td>
<td>same</td>
<td>e</td>
<td>f</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$S$</td>
<td>same</td>
<td>e</td>
<td>d</td>
</tr>
<tr>
<td>$S'$</td>
<td>same</td>
<td>d</td>
<td>f</td>
</tr>
</tbody>
</table>

- now $S'$ agrees with $S_{FF}$ on first $j+1$ requests; we show that 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 $j'$.

- 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 $e' \neq e$, $S'$ 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 $j+1$
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 $j'$.

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 $j'$.

- Case 3c: $g \neq e, f$. $S$ must evict $e$. Make $S'$ evict $f$; now $S$ and $S'$ have the same cache. ▪
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.

FF with direction of time reversed!

Theorem. FF is optimal offline eviction algorithm.
- Provides basis for understanding and analyzing online algorithms.
- LRU is k-competitive.
- LIFO is arbitrarily bad.