Chapter 4

Greedy Algorithms
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.
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
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
```

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 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 \( r \).

Greedy:  

|  \( i_1 \) |  \( i_1 \) |  \( i_r \) |  \( i_{r+1} \) |

OPT:  

|  \( j_1 \) |  \( j_2 \) |  \( j_r \) |  \( j_{r+1} \) |  \( \ldots \) |

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

why not replace job \( j_{r+1} \) with job \( i_{r+1} \)?
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 $r$.

Greedy:

```
| i_1 | i_1 | i_r | i_{r+1} |
```

OPT:

```
| j_1 | j_2 | j_r | i_{r+1} | ... |
```

job $i_{r+1}$ finishes before $j_{r+1}$

solution still feasible and optimal, but contradicts maximality of $r$. 
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.
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 \( s_1 \leq s_2 \leq \ldots \leq s_n \).

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

\[
\text{for } j = 1 \text{ to } n \{ \\
\quad \text{if } (\text{lecture } j \text{ is compatible with some classroom } k) \\
\quad \quad \text{schedule lecture } j \text{ in classroom } k \\
\quad \text{else} \\
\quad \quad \text{allocate a new classroom } d + 1 \\
\quad \quad \text{schedule lecture } j \text{ in classroom } d + 1 \\
\quad \quad d \leftarrow d + 1 \\
\}
\]

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. $\blacksquare$
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></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_j$</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>3</td>
<td>2</td>
</tr>
<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></th>
<th>d_3 = 9</th>
<th>d_2 = 8</th>
<th>d_6 = 15</th>
<th>d_1 = 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>
<td>6</td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>13</td>
<td>14</td>
<td>15</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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 $t_j$.

```
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_j$</td>
<td>1</td>
</tr>
<tr>
<td>$d_j$</td>
<td>100</td>
</tr>
</tbody>
</table>
```

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

```
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_j$</td>
<td>1</td>
</tr>
<tr>
<td>$d_j$</td>
<td>2</td>
</tr>
</tbody>
</table>
```
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
Observation. There exists an optimal schedule with no idle time.

Observation. The greedy schedule has no idle time.
Def. An inversion in schedule $S$ is a pair of jobs $i$ and $j$ such that: $i < j$ 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.
**Minimizing Lateness: Inversions**

**Def.** An inversion in schedule $S$ is a pair of jobs $i$ and $j$ such that: $i < j$ 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:
  
  $$
  \ell'_j = f'_j - d_j \quad (\text{definition})
  = f_i - d_j \quad (j \text{ finishes at time } f_i)
  \leq f_i - d_i \quad (i < j)
  \leq \ell_i \quad (\text{definition})
  $$

\[ \text{time at finish: } f'_j \]

\[ \text{time at finish: } f_j \]

\[ \text{inversion} \]

\[ \text{job } i \]

\[ \text{job } j \]
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.
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)
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.

**Pf.** (by induction on \( x \))

- Consider optimal way to change \( c_k \leq x < c_{k+1} \): greedy takes coin \( k \).
- 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
- 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.
Selecting Breakpoints
Selecting Breakpoints

Selecting breakpoints.

- Road trip from Princeton to Palo Alto 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 < ... < b_n = L

S ← \{0\} ← breakpoints selected
x ← 0 ← current location

while (x ≠ b_n)
  let p be largest integer such that b_p ≤ x + C
  if (b_p = x)
    return "no solution"
  x ← b_p
  S ← S ∪ \{p\}
return S
```

Implementation. $O(n \log n)$

- Use binary search to select each breakpoint $p$. 
Selecting Breakpoints: Correctness

Theorem. Greedy algorithm is optimal.

Pf. (by contradiction)

- Assume greedy is not optimal, and let's see what happens.
- Let \(0 = g_0 < g_1 < \ldots < g_p = L\) denote set of breakpoints chosen by greedy.
- Let \(0 = f_0 < f_1 < \ldots < f_q = L\) denote set of breakpoints in an optimal solution with \(f_0 = g_0, f_1 = g_1, \ldots, f_r = g_r\) for largest possible value of \(r\).
- Note: \(g_{r+1} > f_{r+1}\) by greedy choice of algorithm.

![Diagram showing greedy and optimal solutions]

- Why doesn't optimal solution drive a little further?
Theorem. Greedy algorithm is optimal.

Pf. (by contradiction)

- Assume greedy is not optimal, and let's see what happens.
- Let \( 0 = g_0 < g_1 < \ldots < g_p = L \) denote set of breakpoints chosen by greedy.
- Let \( 0 = f_0 < f_1 < \ldots < f_q = L \) denote set of breakpoints in an optimal solution with \( f_0 = g_0, f_1 = g_1, \ldots, f_r = g_r \) for largest possible value of \( r \).
- Note: \( g_{r+1} > f_{r+1} \) by greedy choice of algorithm.

Diagram:

- **Greedy:** \( g_0 \) to \( g_{r+1} \)
- **OPT:** \( f_0 \) to \( f_q \)

Another optimal solution has one more breakpoint in common ➞ contradiction