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.
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 \( f_1 \leq f_2 \leq \ldots \leq f_n \).

\[
\begin{align*}
\text{A} & \leftarrow \emptyset \\
\text{for } j = 1 \text{ to } n \{ \\
\quad \text{if (job } j \text{ compatible with A)} \\
\quad \quad \text{A} \leftarrow \text{A} \cup \{j\} \\
\} \\
\text{return A}
\end{align*}
\]

**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

A: 0-6
B: 2-4
C: 3-5
D: 4-8
E: 6-7
F: 7-9
G: 8-10
H: 9-11
Interval Scheduling

A
B
C
D
E
F
G
H

Time
Interval Scheduling
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

```
<table>
<thead>
<tr>
<th>Time</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>B</td>
<td></td>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>E</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>H</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
Interval Scheduling

The diagram illustrates the scheduling of intervals on a timeline. Each bar represents an interval, with its start and end times indicated by its position on the timeline.

- **A**: Starts at 0 and ends at 6.
- **B**: Starts at 1 and ends at 3.
- **C**: Starts at 4 and ends at 5.
- **D**: Starts at 2 and ends at 7.
- **E**: Starts at 3 and ends at 6.
- **F**: Starts at 6 and ends at 9.
- **G**: Starts at 7 and ends at 10.
- **H**: Starts at 9 and ends at 11.

The intervals are arranged to show which intervals overlap or are independent of each other.
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 a set of jobs selected by greedy.
- Let $j_1, j_2, \ldots, j_m$ denote a set of jobs in an 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 comparison between greedy and optimal solutions.](image-url)
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 a set of jobs selected by greedy.
- Let $j_1, j_2, \ldots, j_m$ denote a set of jobs in an optimal solution with $i_1 = j_1, i_2 = j_2, \ldots, i_r = j_r$ for the largest possible value of $r$.

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

OPT: $j_1, j_2, \ldots, j_r, i_{r+1}$

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

solution still feasible and optimal, but contradicts the 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.
**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.

```
| 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 \( s_1 \leq s_2 \leq \ldots \leq s_n \).

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

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

**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.
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 \( d \) was allocated 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
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>d₂ = 8</th>
<th>d₆ = 15</th>
<th>d₁ = 6</th>
<th>d₅ = 14</th>
<th>d₄ = 9</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15</td>
<td>lateness = 2</td>
<td>lateness = 0</td>
<td>max lateness = 6</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 $t_j$.
  
  \[
  \begin{array}{ccc}
  1 & 2 & \text{counterexample} \\
  t_j & 1 & 10 \\
  d_j & 100 & 10 \\
  \end{array}
  \]

- [Smallest slack] Consider jobs in ascending order of slack $d_j - t_j$.
  
  \[
  \begin{array}{ccc}
  1 & 2 & \text{counterexample} \\
  t_j & 10 & 1 \\
  d_j & 10 & 2 \\
  \end{array}
  \]
Minimizing Lateness: Greedy Algorithm

**Greedy algorithm.** Earliest deadline first.

Sort n jobs by deadline so that \( d_1 \leq d_2 \leq \ldots \leq d_n \)

\[
\begin{align*}
t &\leftarrow 0 \\
\text{for } j = 1 \text{ to } n \\
&\quad (\text{Assign job } j \text{ to interval } [t, t + t_j]) \\
&s_j \leftarrow t, f_j \leftarrow t + t_j \\
t &\leftarrow t + t_j \\
\text{output intervals } [s_j, f_j]
\end{align*}
\]

Max lateness = 1

<p>| | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>( d_1 = 6 )</td>
<td>( d_2 = 8 )</td>
<td>( d_3 = 9 )</td>
<td>( d_4 = 9 )</td>
<td>( d_5 = 14 )</td>
<td>( d_6 = 15 )</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<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>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
</tr>
</tbody>
</table>
Minimizing Lateness: No Idle Time

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

<table>
<thead>
<tr>
<th>d = 4</th>
<th>d = 6</th>
<th>d = 12</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2</td>
<td>3 4 5 6</td>
<td>7 8 9 10 11</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>d = 4</th>
<th>d = 6</th>
<th>d = 12</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2</td>
<td>3 4 5 6</td>
<td>7 8 9 10 11</td>
</tr>
</tbody>
</table>

**Observation.** The greedy schedule has no idle time.
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 $i$.

**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 $i$.

![Diagram showing an inversion between jobs $i$ and $j$ before and after swap]

**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:

\[
\ell'_j = f'_j - d_j \quad \text{(definition)}
\]
\[
= f_i - d_j \quad \text{(j finishes at time $f_i$)}
\]
\[
\leq f_i - d_i \quad \text{(i < j)}
\]
\[
= \ell_i \quad \text{(definition)}
\]
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.
Chapter 4

Greedy Algorithms
shortest path from Princeton CS department to Einstein's house
Shortest Path Problem

Shortest path network.
- Directed graph $G = (V, E)$.
- Source $s$, target $t$.
- Length $\ell_e = \text{length of edge } e$.

Shortest path problem: find shortest directed path from $s$ to $t$.

Cost of path $s$-2-3-5-$t$:

\[ \text{Cost} = 9 + 23 + 2 + 16 = 50. \]
Dijkstra's algorithm.

- Maintain a set of explored nodes $S$ for which we have determined the shortest path distance $d(u)$ from $s$ to $u$.

- Initialize $S = \{s\}$, $d(s) = 0$.

- Repeatedly (greedily) choose unexplored node $v \notin S$ which minimizes

$$\partial(v) = \min_{e = (u,v) : u \in S} d(u) + \ell_e,$$

add $v$ to $S$, and set $d(v) = \partial(v)$. 

shortest path to some $u$ in explored part, followed by a single edge $(u, v)$
Dijkstra's Shortest Path Algorithm

Find shortest path from $s$ to $t$. 
Dijkstra's Shortest Path Algorithm

\[ S = \{ \} \]
\[ PQ = \{ s, 2, 3, 4, 5, 6, 7, t \} \]
Dijkstra's Shortest Path Algorithm

\[ S = \{ \} \]
\[ PQ = \{ s, 2, 3, 4, 5, 6, 7, t \} \]
Dijkstra's Shortest Path Algorithm

\[ S = \{ s \} \]
\[ PQ = \{ 2, 3, 4, 5, 6, 7, t \} \]

distance label

distance label

decrease key

\[ 0 \]
\[ 15 \]
\[ 9 \]
\[ 2 \]
\[ 24 \]
\[ 3 \]
\[ 30 \]
\[ 11 \]
\[ 16 \]
\[ 19 \]
\[ 5 \]
\[ 44 \]
\[ 4 \]
\[ 2 \]
\[ 6 \]
\[ 6 \]
\[ 6 \]
\[ 14 \]
\[ 14 \]
\[ 14 \]
\[ 11 \]
\[ 6 \]
Dijkstra's Shortest Path Algorithm

\[ S = \{ s \} \]
\[ PQ = \{ 2, 3, 4, 5, 6, 7, t \} \]
Dijkstra's Shortest Path Algorithm

$S = \{ s, 2 \}$

$PQ = \{ 3, 4, 5, 6, 7, t \}$
Dijkstra's Shortest Path Algorithm

$S = \{ s, 2 \}$
$PQ = \{ 3, 4, 5, 6, 7, t \}$
Dijkstra's Shortest Path Algorithm

\[ S = \{ s, 2 \} \]
\[ PQ = \{ 3, 4, 5, 6, 7, t \} \]
Dijkstra's Shortest Path Algorithm

\[ S = \{ s, 2, 6 \} \]
\[ PQ = \{ 3, 4, 5, 7, t \} \]
Dijkstra's Shortest Path Algorithm

\[ S = \{ s, 2, 6 \} \]
\[ PQ = \{ 3, 4, 5, 7, t \} \]
Dijkstra's Shortest Path Algorithm

\[ S = \{ s, 2, 6, 7 \} \]
\[ PQ = \{ 3, 4, 5, t \} \]
Dijkstra's Shortest Path Algorithm

\[ S = \{ s, 2, 6, 7 \} \]
\[ PQ = \{ 3, 4, 5, t \} \]
Dijkstra's Shortest Path Algorithm

\[ S = \{ s, 2, 3, 6, 7 \} \]
\[ PQ = \{ 4, 5, t \} \]
Dijkstra's Shortest Path Algorithm

\[ S = \{ s, 2, 3, 6, 7 \} \]
\[ PQ = \{ 4, 5, t \} \]
Dijkstra's Shortest Path Algorithm

\[ S = \{ s, 2, 3, 5, 6, 7 \} \]
\[ PQ = \{ 4, t \} \]
Dijkstra's Shortest Path Algorithm

$S = \{ s, 2, 3, 5, 6, 7 \}$

$PQ = \{ 4, t \}$
Dijkstra's Shortest Path Algorithm

\[ S = \{ s, 2, 3, 4, 5, 6, 7 \} \]
\[ PQ = \{ t \} \]
Dijkstra's Shortest Path Algorithm

\[ S = \{ s, 2, 3, 4, 5, 6, 7 \} \]
\[ PQ = \{ t \} \]
Dijkstra's Shortest Path Algorithm

\[ S = \{ s, 2, 3, 4, 5, 6, 7, t \} \]

\[ PQ = \{ \} \]
Dijkstra's Shortest Path Algorithm

\[ S = \{ s, 2, 3, 4, 5, 6, 7, t \} \]
\[ PQ = \{ \} \]
Dijkstra's Algorithm: Proof of Correctness

**Invariant.** For each node \( u \in S \), \( d(u) \) is the length of the shortest \( s-u \) path.

**Pf.** (by induction on \( |S| \))

**Base case:** \( |S| = 1 \) is trivial.

**Inductive hypothesis:** Assume true for \( |S| = k \geq 1 \).

- Let \( v \) be next node added to \( S \), and let \((u,v)\) be the chosen edge.
- The shortest \( s-u \) path plus \((u,v)\) is an \( s-v \) path of length \( \partial(v) \).
- Consider any \( s-v \) path \( P \). We'll see that it's no shorter than \( \partial(v) \).
- Let \( x-y \) be the first edge in \( P \) that leaves \( S \), and let \( P' \) be the subpath to \( x \).
- \( P \) is already too long as soon as it leaves \( S \).

\[
\begin{align*}
\| (P) & \geq \| (P') + \| (x,y) \\
& \geq d(x) + \| (x, y) \\
& \geq \partial(y) \\
& \geq \partial(v)
\end{align*}
\]

- nonnegative weights
- inductive hypothesis
- defn of \( \partial(y) \)
- Dijkstra chose \( v \) instead of \( y \)
Dijkstra's Algorithm: Implementation

For each unexplored node \( v \notin S \), explicitly maintain
\[
\partial(v) = \min_{e = (u, v): u \in S} d(u) + \ell_e,
\]

- Next node to explore = node \( v \notin S \) with minimum \( \partial(v) \).
- When exploring \( v \), for each incident edge \( e = (v, w) \), \( w \notin S \), update
\[
\partial(w) = \min \{ \partial(w), \partial(v) + \ell_e \}.
\]

Efficient implementation. Maintain a priority queue of unexplored nodes, prioritized by \( \partial(v) \).

<table>
<thead>
<tr>
<th>PQ Operation</th>
<th>Dijkstra</th>
<th>Array</th>
<th>Binary heap</th>
<th>d-way Heap</th>
<th>Fib heap†</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insert</td>
<td>( n )</td>
<td>( n )</td>
<td>( \log n )</td>
<td>( d \log d n )</td>
<td>1</td>
</tr>
<tr>
<td>ExtractMin</td>
<td>( n )</td>
<td>( n )</td>
<td>( \log n )</td>
<td>( d \log d n )</td>
<td>( \log n )</td>
</tr>
<tr>
<td>ChangeKey</td>
<td>( m )</td>
<td>( 1 )</td>
<td>( \log n )</td>
<td>( \log d n )</td>
<td>1</td>
</tr>
<tr>
<td>IsEmpty</td>
<td>( n )</td>
<td>( 1 )</td>
<td>( 1 )</td>
<td>( 1 )</td>
<td>1</td>
</tr>
<tr>
<td>Total</td>
<td>( n^2 )</td>
<td>( m \log n )</td>
<td>( m \log_{m/n} n )</td>
<td>( m + n \log n )</td>
<td></td>
</tr>
</tbody>
</table>

† Individual ops are amortized bounds