Chapter 4
Greedy Algorithms

4.1 Interval Scheduling

Interval Scheduling

- Job j starts at $s_j$ and finishes at $f_j$.
- Two jobs are compatible if they don’t overlap.
- Goal: find maximum subset of mutually compatible jobs.

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

- What order? Does that give the best answer? Why or why not?
- Does it help to be greedy about order?
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 $c_j$.

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 ← Ø

for j = 1 to n {
    if (job j compatible with A)
        A ← A ∪ {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: Correctness

**Theorem.** Greedy algorithm is optimal.

**Pf.** ("greedy stays ahead")

Let $i_1, i_2, ..., i_k$ be jobs picked by greedy, $j_1, j_2, ..., j_m$ those in some optimal solution.

Show $f(i_r) \leq f(j_r)$ by induction on $r$.

**Basis:** $i_1$ chosen to have min finish time, so $f(i_1) \leq f(j_1)$

**Ind.** $f(i_r) \leq f(j_r)$, so $j_{r+1}$ is among the candidates considered by greedy when it picked $i_{r+1}$, & it picks min finish, so $f(i_{r+1}) \leq f(j_{r+1})$

Similarly, $k \geq m$, else $j_{k+1}$ is among (nonempty) set of candidates for $i_{k+1}$

---

4.1 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 as Interval Graph Coloring

Vertices = classes; edges = conflicting class pairs; different colors = different assigned rooms

Note: graph coloring is very hard in general, but graphs corresponding to interval intersections are a much simpler special case.
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.

Example: This schedule uses only 3.

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.

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

Interval Partitioning: Greedy Analysis

Observation. Greedy algorithm never schedules two incompatible lectures in the same classroom.

Theorem. Greedy algorithm is optimal.

Pf.

1. Let \( d \) = number of classrooms that the greedy algorithm allocates.
2. Classroom \( d \) is opened because we needed to schedule a job, say \( j \), that is incompatible with all \( d-1 \) previously used classrooms.
3. Since we sorted by start time, all these incompatibilities are caused by lectures that start no later than \( s_j \).
4. Thus, we have \( d \) lectures overlapping at time \( s_j + \epsilon \), i.e. depth = \( d \).
5. "Key observation" \( \Rightarrow \) all schedules use \( \geq \) depth classrooms, so \( d = \text{depth} \) and greedy is optimal.
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: $l_j = \max(0, f_j - d_j)$.
- Goal: schedule all jobs to minimize maximum lateness $L = \max l_j$.

Ex:

<table>
<thead>
<tr>
<th>Job</th>
<th>$d_j$</th>
<th>$t_j$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>15</td>
<td>9</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
<td>14</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>9</td>
</tr>
<tr>
<td>6</td>
<td>14</td>
<td>9</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 $t_j$:

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

(Smallest slack) Consider jobs in ascending order of slack $d_j - t_j$:

<table>
<thead>
<tr>
<th>$j$</th>
<th>$t_j$</th>
<th>$d_j$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>10</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 3 4 5 6 7 8 9 10 11</td>
<td></td>
<td></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: deadline $i < j$ but $j$ scheduled before $i$.

<table>
<thead>
<tr>
<th>$d_j = 6$</th>
<th>$d_j = 8$</th>
<th>$d_j = 9$</th>
<th>$d_j = 14$</th>
<th>$d_j = 15$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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. (If $j$ & $i$ aren’t consecutive, then look at the job $k$ scheduled right after $j$. If $d_k < d_j$, then $(j,k)$ is a consecutive inversion; if not, then $(k,i)$ is an inversion, & nearer to each other - repeat.)

Observation. Swapping adjacent inversion reduces # inversions by 1 (exactly)
Minimizing Lateness: Inversions

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

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

**Pf.** Let $l$ be the lateness before the swap, and let $l'$ be it afterwards.
- $l'_k = l_k$ for all $k \neq i, j$
- $l'_i \leq l'_j$
- If job $j$ is now late: $l'_j = f_j - d_j$ (definition)  
  $f_j = f_i + d_j$ (j finishes at time $f_i$)  
  $d_i \leq d_j$ (since $i$ had later deadline, so is less tardy than $i$ was)
  $l'_i = l_i$ (definition)

Minimizing Lateness: No Inversions

**Claim.** All inversion-free schedules $S$ have the same max lateness

**Pf.** If $S$ has no inversions, then deadlines of scheduled jobs are monotonically nondecreasing, i.e., they increase (or stay the same) as we walk through the schedule from left to right.

Two such schedules can differ only in the order of jobs with the same deadlines. Within a group of jobs with the same deadline, the max lateness is the lateness of the last job in the group - order within the group doesn't matter.

Minimizing Lateness: Correctness of Greedy Algorithm

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

**Pf.** Let $S^*$ be an optimal schedule with the fewest number of inversions

Can assume $S^*$ has no idle time.

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 $S^*$

So, $S^*$ has no inversions. But then $\text{Lateness}(S) = \text{Lateness}(S^*)$

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.

**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.

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

1. **cache**
   - Pronunciation: 'kash
   - Function: noun
   - Etymology: French, from cacher to press, hide
   - a hiding place especially for concealing and preserving provisions or implements

2. **cache**
   - Function: transitive verb
   - to place, hide, or store in a cache

- Webster’s Dictionary

---

**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: a b c d e f
- future queries:
  - g a b c d a b b a c d e a f a d e f g h . . .

- cache miss
- eject this one

**Theorem.** [Bellady, 1960s] FF is optimal eviction schedule.
**Pf.** Algorithm and theorem are intuitive; proof is subtle.

---

4.4 Shortest Paths in a Graph

You’ve seen this in 373, so this section and next two on min spanning tree are review. I won’t lecture on them, but you should review the material. Both, but especially shortest paths, are common problems with many applications.
Shortest Path Problem

Shortest path network:
- Directed graph $G = (V, E)$.
- Source $s$, destination $t$.
- Length $l_e$ length of edge $e$.

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

Cost of path $s$-2-3-5-$t$ = $9 + 23 + 2 + 16 = 48$.

Dijkstra’s Algorithm

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 choose unexplored node $v$ which minimizes $d(v) = \min_{u \in S} d(u) + l_{uv}$.
- Add $v$ to $S$, and set $d(v) = \pi(v)$.

Coin Changing

Greed is good. Greed is right. Greed works.
Greed clarifies, cuts through, and captures the essence of the evolutionary spirit.
- Gordon Gekko (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.

Q. Is cashier’s algorithm optimal?

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

\[ S \leftarrow \emptyset \]

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 \( c_k = x < c_{k+1} \): 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, …, ( k ) 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>( \geq 1 )</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>( N \geq 2 )</td>
<td>( 4 \cdot 5 = 9 )</td>
</tr>
<tr>
<td>4</td>
<td>25</td>
<td>( Q \geq 3 )</td>
<td>( 20 \cdot 4 = 24 )</td>
</tr>
<tr>
<td>5</td>
<td>100</td>
<td>no limit</td>
<td>( 75 \cdot 24 = 99 )</td>
</tr>
</tbody>
</table>

Coin-Changing: Analysis of Greedy Algorithm

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.
- Optimal: 70, 70.