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

$A \leftarrow \emptyset$

for $j = 1$ to $n$

if (job $j$ compatible with $A$)

$A \leftarrow A \cup \{j\}$

end if

end for

return $A$

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

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

**Pf.** (by contradiction)
1. Assume greedy is not optimal, and let's see what happens.
2. Let \( i_1, i_2, \ldots, i_k \) denote set of jobs selected by greedy.
3. 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:

- \( i_1 \)
- \( i_2 \)
- \( \ldots \)
- \( i_k \)

OPT:

- \( j_1 \)
- \( j_2 \)
- \( \ldots \)
- \( j_m \)

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

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

4.1 Interval Partitioning

Interval partitioning.
1. Lecture \( j \) starts at \( s_j \) and finishes at \( f_j \).
2. 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

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.

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?

Sort intervals by starting time so that \( s_1 \leq s_2 \leq \ldots \leq s_n \).

\( d = 0 \) — number of allocated classrooms

\( d \leftarrow d + 1 \)

Implementation? Run-time? Next HW
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 $s_j + \varepsilon$, i.e. depth $\geq d$.
- "Key observation" $\Rightarrow$ all schedules use $\geq$ depth classrooms, so $d =$ depth and greedy is optimal.

Interval Partitioning: Alt Proof (exchange argument)

When 4th room added, $rm_1$ was free; why not swap it in there?
(A: it conflicts with later stuff in schedule, which dominoes)

But: $rm_4$ schedule after 11:00 is conflict-free; so is $rm_1$ schedule, so could swap both post-11:00 schedules

Why does it help? Delays needing 4th room; repeat.
Cleaner: "Let $S^*$ be an opt sched with latest use of last room; ... swap; ... contradiction"

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></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>$d_j$</th>
<th>9</th>
<th>8</th>
<th>15</th>
<th>6</th>
<th>14</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>lateness = 2</td>
<td>lateness = 0</td>
<td>max lateness = 6</td>
<td></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 Algorithm

Greedy algorithm. Earliest deadline first.

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

\[ t \leftarrow 0 \]

\[ \text{for } j = 1 \text{ to } n \]

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 \([s_j, f_j]\)

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

Observation. The greedy schedule has no idle time.

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: No Idle Time

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

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 $i$.

**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 adjacent, 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 $k \neq i, j$
- $l'_i \leq l_i$
- If job $j$ is now late:

  $l'_j = l_j - (j$ finishes at time $f_i$)
  $= f_i - d_j$ (definition)
  $\leq f_i - d_i$ ($i < j$, so $d_i < d_j$)
  $\leq l_i$ (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

- 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.

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

$$
\pi(v) = \min_{u \in S, e \in E} d(u) + l_e
$$

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

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

$$
\pi(v) = \min_{u \in S, e \in E} d(u) + l_e
$$

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: \(c_1 < c_2 < \ldots < c_n\).

```java
coins selected
S ← ∅
while (x ≠ 0) {
    let k be largest integer such that \(c_k ≤ x\)
    if (k = 0)
        return "no solution found"
    x ← x - c_k
    S ← S ∪ {k}
}
return S
```

**Coin-Changing: Analysis of Greedy Algorithm**

**Theorem.** Greedy 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 ≤ 4)</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>(N ≤ 1)</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>(N + D ≤ 2)</td>
<td>(4 + 5 = 9)</td>
</tr>
<tr>
<td>4</td>
<td>25</td>
<td>(Q ≤ 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>

**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.