EE360C: Algorithms

Greedy Algorithms

Summer 2019

Department of Electrical and Computer Engineering
University of Texas at Austin

Greedy Algorithms

Greedy Algorithms

- An algorithm is greedy if it builds up a solution in small steps, choosing a decision at each step myopically to optimize some underlying criterion.
- Locally, incrementally optimizing some measure on its way to a solution.
- We will prove that a greedy algorithm produces an optimal solution to a problem. Two approaches
  - That the greedy algorithms stays ahead.
  - The exchange argument: Any possible solution and gradually transform it into a solution found by the greedy algorithm without hurting its quality.
- We will illustrate these approaches using examples of Interval Scheduling.
- Applications of greedy algorithms: Shortest path in a graph, Minimum Spanning Tree, Huffman codes.

Interval Scheduling

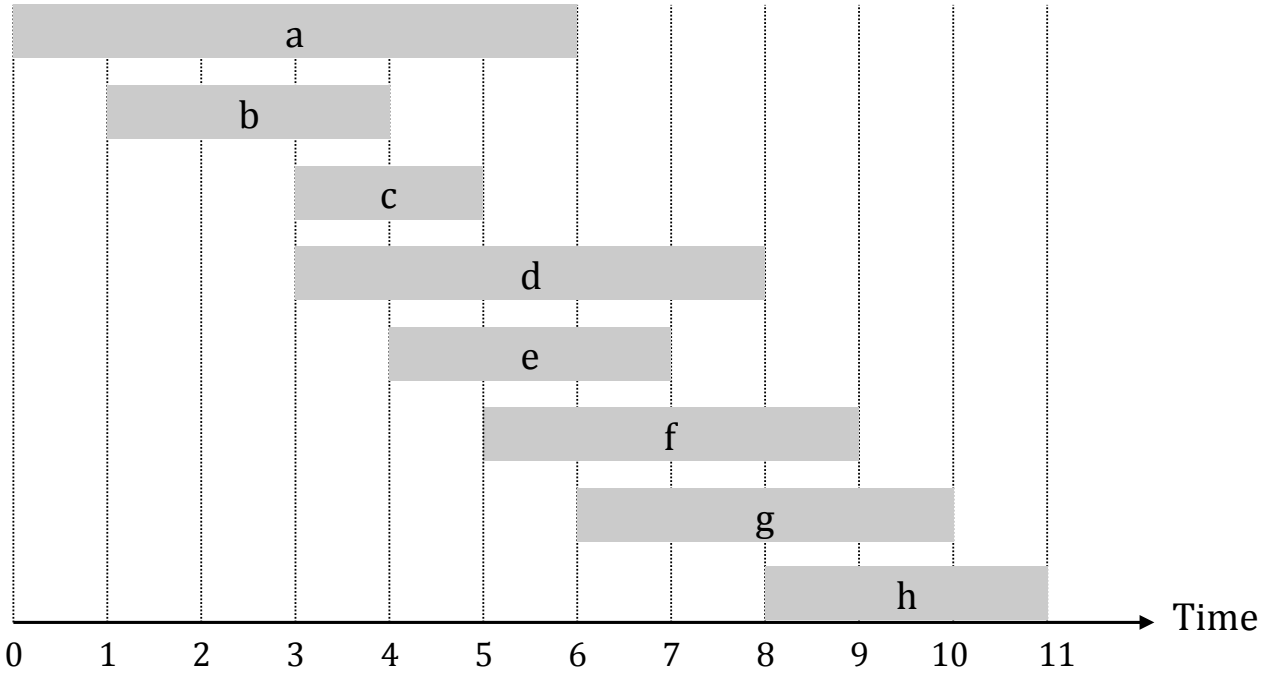
Interval Scheduling: The Problem

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: a Greedy Choice

Consider the jobs in some order. Take each job if it is compatible with the ones already taken.

- **Earliest start time** Consider jobs in order of start time s_j

- **Shortest interval** Consider jobs in order of interval length $f_j - s_j$

- **Fewest conflicts** Consider jobs in order of number of conflicts

- **Earliest finish time** Consider jobs in order of finish time f_j

Interval Scheduling: Greedy Algorithm

Greedy Algorithm

Consider jobs in increasing order of finish time. Take each job provided that it is compatible with the ones already taken.

```
Sort jobs by finish times so that f_1 ≤ f_2 ≤ ... ≤ f_n.

jobs selected
A ← ϕ
for j = 1 to n {
   if (job j compatible with A)
      A ← A ∪ {j}
}
return A
```

Implementation

We can implement this is $O(n \log n)$ running time.

- Remember job $j*$ that was most recently added to A
- Job j is compatible with A if $s_j \geq f_{j*}$

Interval Scheduling: Analysis

To show optimality of the greedy algorithm, we need to show that for the set of intervals A returned by the greedy algorithm

- All the intervals are compatible.
- Comparable to the optimal set of intervals $\mathcal{O}$.

We can immediately declare

Fact

A is a compatible set of requests.

Interval Scheduling: Analysis

Ideally, we would like to show that $A = \mathcal{O}$. But there may be many optimal solutions, so at best we can show $|A| = |\mathcal{O}|$.

What is our strategy?

We will try to show that our greedy algorithm "stays ahead" of the optimal solution $\mathcal{O}$.

Compare partial solution of our greedy algorithm to initial segments of $\mathcal{O}$, and we will show that the greedy algorithm is doing better in a step-by-step fashion.

Interval Scheduling: Notation

Let $i_1, i_2, \ldots i_k$ be the set of requests in A in the order that they were added to A.

Therefore, $|A| = k$.

Similarly, let $j_1, j_2, \ldots j_k$ be the set of requests in $\mathcal{O}$ and $|\mathcal{O}| = m$.

Our goal is to prove $k = m$.

Interval Scheduling: Analysis

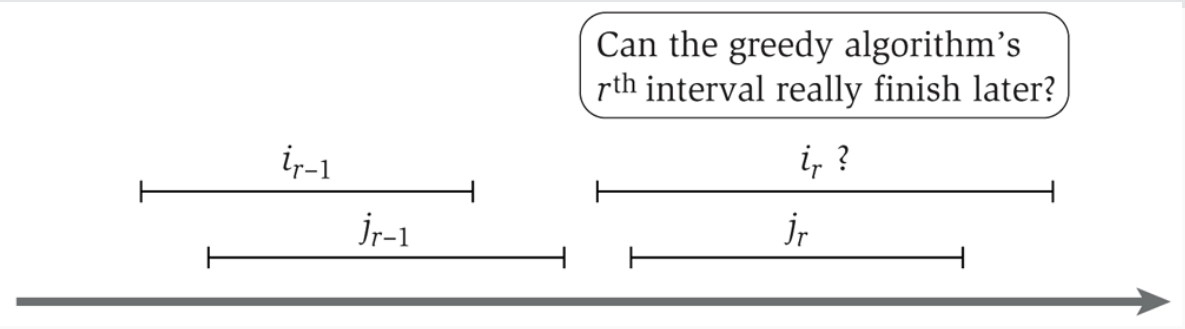
Greedy Algorithm Stays Ahead of Optimal Solution

When i counts indices in the greedy solution and j counts indices in the previous solution, for all indicies $r \leq k$, $f_{i_r} \leq f_{j_r}$.

Proof (by induction)

Base case: $r = 1$; the claim is clearly true, since the greedy algorithm selects the interval with the earliest finish time.

Inductive step: Let $r > 1$, and assume the claim is true for $r - 1$. By the inductive hypothesis, we have $f_{i_{r-1}} \leq f_{j_{r-1}}$. For our greedy algorithm to fall behind, the next interval would have to finish later. But by the definition of the greedy algorithm, at each step, the algorithm will choose the next non-conflicting job with the earliest finish time. The greedy algorithm would have the option of picking j_r as it's next request, thus fulfilling the inductive step.



Interval Scheduling: Analysis

Theorem

The greedy algorithm returns an optimal set A.

Proof (by contradiction)

- A is not optimal, then $\mathcal{O}$ must have more requests than A, that is $m > k$.
- By previous theorem, with $r = k$, we have $f_{i_k} \leq f_{j_k}$.
- Since, $m > k$, there is a request j_{k+1} in $\mathcal{O}$. This request starts after request j_k ends and hence after i_k ends (Greedy algorithm always ahead).
- That means there is one more request that is compatible to A, but the greedy algorithm terminated, which is a contradiction.

Interval Scheduling: Complications

This was the simplest framing of the problem. Additional potential considerations:

- the algorithm doesn't know all of the requests *a priori* — online algorithms
- different requests may have different weights — weighted interval scheduling

Interval Partitioning

Interval Partitioning

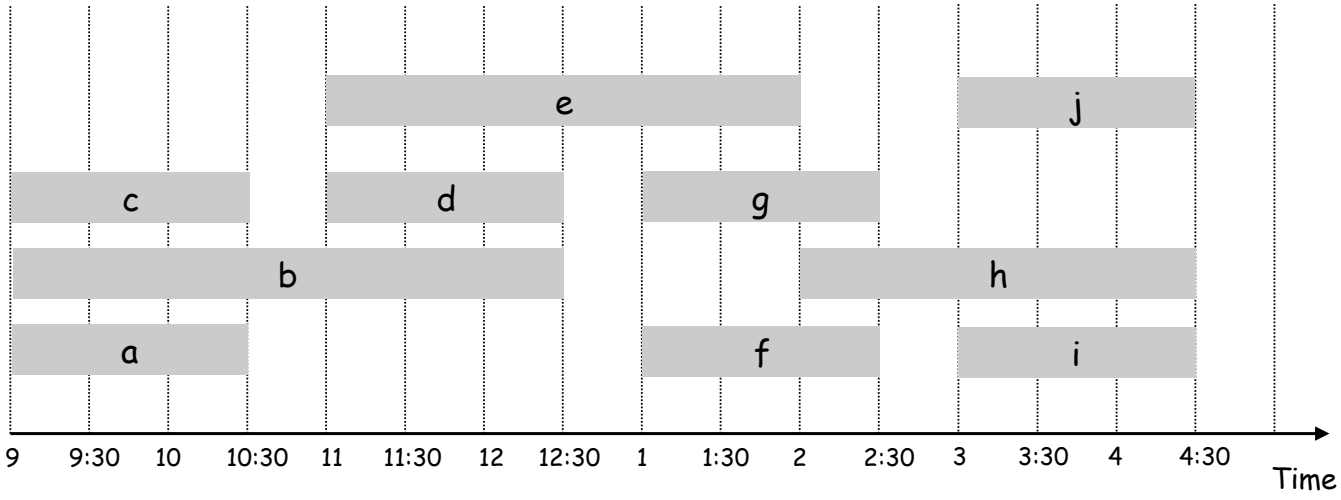
Consider the problem in which one must schedule *all* requests

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.



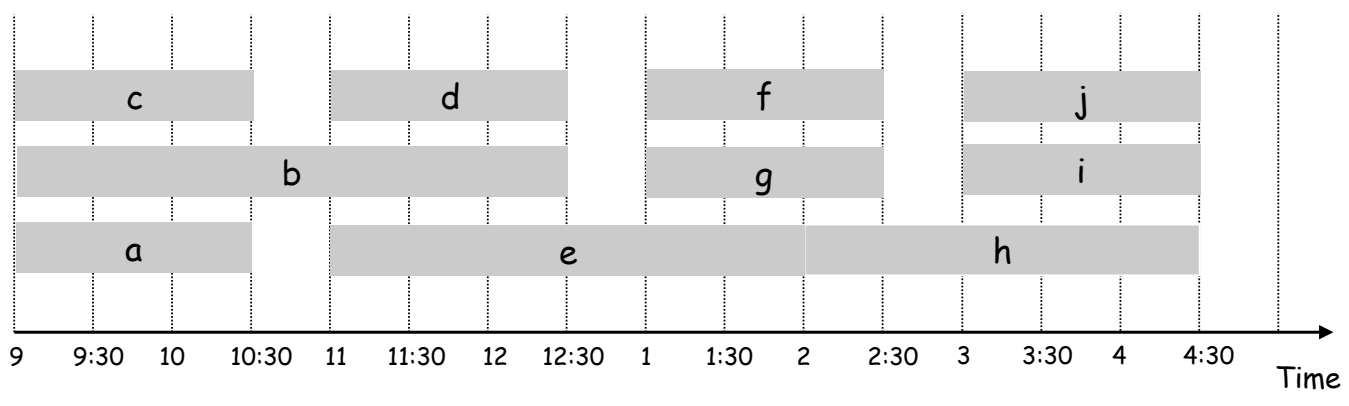
Interval Partitioning

Consider the problem in which one must schedule *all* requests using the fewest "resources" (e.g., processors, classrooms)

Interval Partitioning

- Lecture j starts at s_j and finishes at f_j.
- Goal: find the minimum number of classrooms to schedule all lectures so that no two occur at the same time in the same room.

This schedule uses 3 classrooms to schedule 10 lectures:



Interval Partitioning: Lower Bound

Definition

The depth of a set of open intervals is the maximum number that contain any given time.

Key Observation

The number of classrooms needed $\geq$ depth

Example

The depth of the schedule on the previous slide was three; the schedule is optimal.

Question

Does there always exist a schedule equal to the 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 ≤ s_2 ≤ ... ≤ s_n.
d ← 0   ← number of allocated classrooms

for j = 1 to n {
   if (lecture j is compatible with some classroom k)
      schedule lecture j in classroom k
   else
      allocate a new classroom d + 1
      schedule lecture j in classroom d + 1
      d ← d + 1
}
```

Implementation

- For each classroom k, maintain the finish time of the last job added.
- Keep the classrooms in a priority queue.
- Running time: $O(n \log n)$

Interval Partitioning: Greedy Analysis

Observation

The greedy algorithm never schedules two incompatible lectures in the same classroom.

Theorem

The greedy algorithm is optimal.

Proof

- Let d be the number of classrooms the greedy algorithm uses.
- Classroom d is opened because we need 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, at time $s_j + \epsilon$, we have d overlapping lectures.
- Key observation: all schedules use $\geq d$ classrooms.

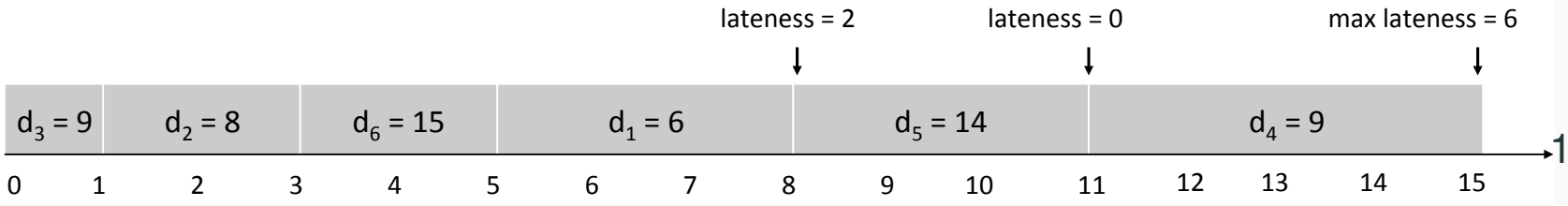
Scheduling to Minimize Lateness

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 $L = \max l_j$

	1	2	3	4	5	6
t_j	3	2	1	4	3	2
d_j	6	8	9	9	14	15



Greedy Template: Greedy Choice

Greedy Choice

We have to consider the jobs in some order:

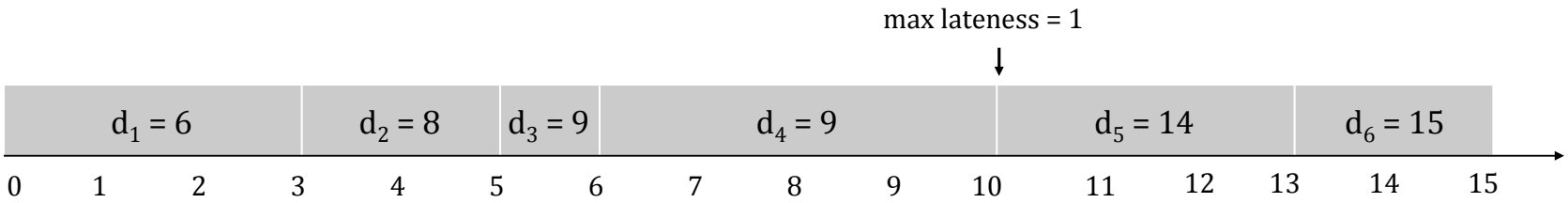
- **Shortest processing time first.** Consider jobs in ascending order of processing time t_j.
- **Earliest deadline first.** Consider jobs in ascending order of deadline d_j.
- **Smallest slack.** Consider jobs in ascending order of slack $d_j - t_j$.

Can you construct counter examples for any of these options?

Greedy Algorithm

Sort n jobs by deadline so that $d_1 \le d_2 \le \ldots \le d_n$

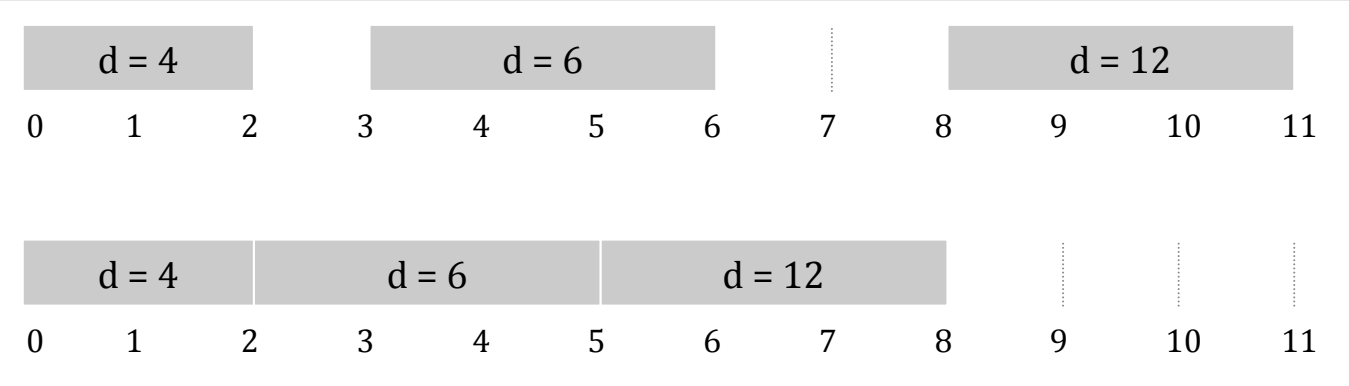
$t \leftarrow 0$
for j = 1 to n
  Assign job j to interval $[t, t + t_j]$
  $s_j \leftarrow t$, $f_j \leftarrow t + t_j$
  $t \leftarrow t + t_j$
output intervals $[s_j, f_j]$



No Idle Time

Observation 1

There exists an optimal schedule with no idle time.



Observation 2

The greedy schedule has no idle time.

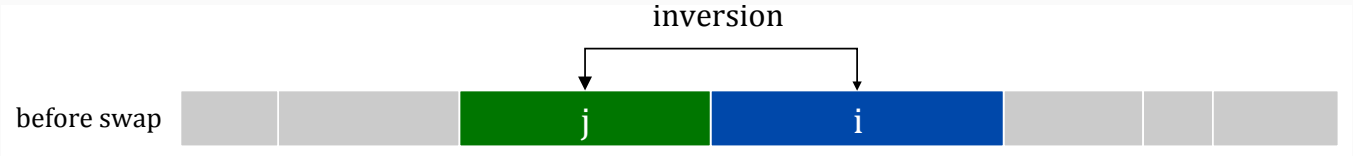
Process

Consider an optimal schedule *O*. Gradually modify *O*, preserving its optimality at each step, but eventually transforming it into a schedule *A* that our greedy algorithm would give us. This is an exchange argument.

Inversions

Definition

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

A 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.

Inversions (cont.)

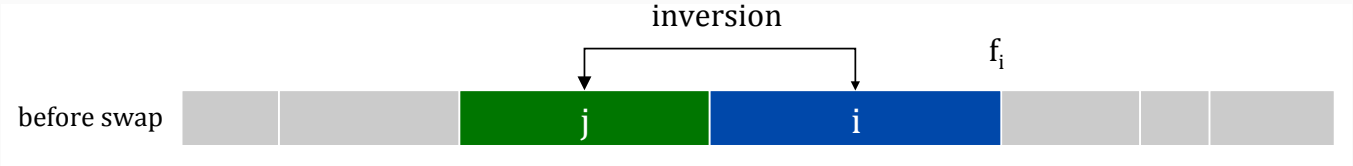
Claim

Swapping two adjacent, inverted jobs reduces the number of inversions by one and does not increase the maximum lateness.

Proof

Let l be the lateness before the swap, and let l' be it after.

- $l'_k = l_k$ for all $k \neq i, j$ and $l'_i \leq l_i$
- If job j is late:
  - $l'_j = f'_j - d_j$ (definition)
  - $l'_j = f_i - d_j$ (j finishes at time f_i)
  - $l'_j \leq f_i - d_i$ ($i < j$)
  - $l'_j \leq l_i$



Analysis of Greedy Algorithm

Theorem

Greedy schedule S is optimal.

Proof

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

- We can assume that S^* has no idle time.
- If S^* has no inversions, then $S = S^*$. (There is only one schedule with no idle time and no inversions.)
- If S^* has an inversion, then let *i*-*j* be an adjacent inversion.
  - Swapping *i* and *j* does not increase the maximum lateness and strictly decreases the number of inversions.
  - Therefore the swapped schedule is at least as good as S^* (it's also optimal) and has fewer inversions.

Recap: 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 optimal solution must have a certain value. Then show that your algorithm always achieves this bound.

Questions