Greedy Algorithms

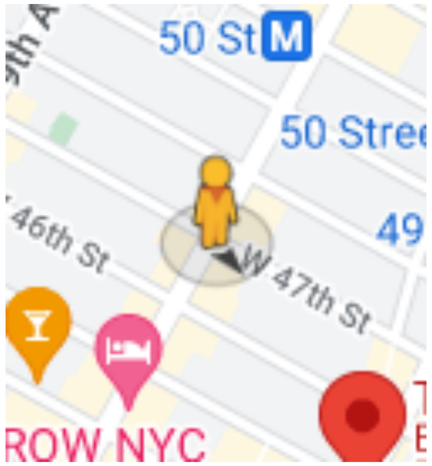
Set of Algorithm Design Paradigms

- Greedy Algorithms ⟵
- Divide and Conquer
- Dynamic Programming
- Network flow

Greedy: Make Locally Optimal Choices

Greedy algorithms build solutions by making **locally optimal** choices at each step of the algorithm. Our hope is that we eventually reach a our goal.

- Intuitive example: How do you navigate the Manhattan street grid on foot?
- Suppose you are trying to get to a location that is South and East of your starting location



- Bill's Navigation Algorithm: Choose a direction (South or East) and walk until you hit a red light or reach your target street. Then walk in the other direction until you hit a red light or reach your target street.
  - Each decision uses only local information, but your choices always bring you closer to your goal (always makes progress)
- Surprisingly, greedy algorithms *sometimes* produce globally optimal solutions!

An Optimal Greedy Example

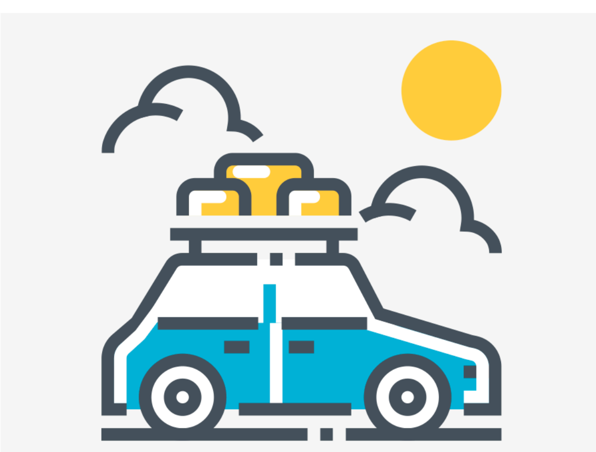
- What is the algorithm to return change in US currency?
  - It is greedy!
  - To make change for \$$r$, start with biggest denomination less than \$$r$, subtract and repeat

- The greedy change algorithm is optimal for US coins!
- But it is not optimal in general:
  - Imagine 25c, 20c, 10c, 5c, 1c coins
  - How to make change for 40c?
    - Greedy: 25c, 10c, 5c
    - Optimal: 20c, 20c

An Optimal Greedy Example: Filling Up on Gas

Suppose you are on a road trip on a long straight highway

- **Goal**: minimize the number of times you stop to get gas
- Many possible ways to choose which gas station to stop at
- Greedy: wait until you are just about to run out of gas (i.e., you won't make it to the *next* gas station), then stop for gas
    - This turns out to be an optimal solution!

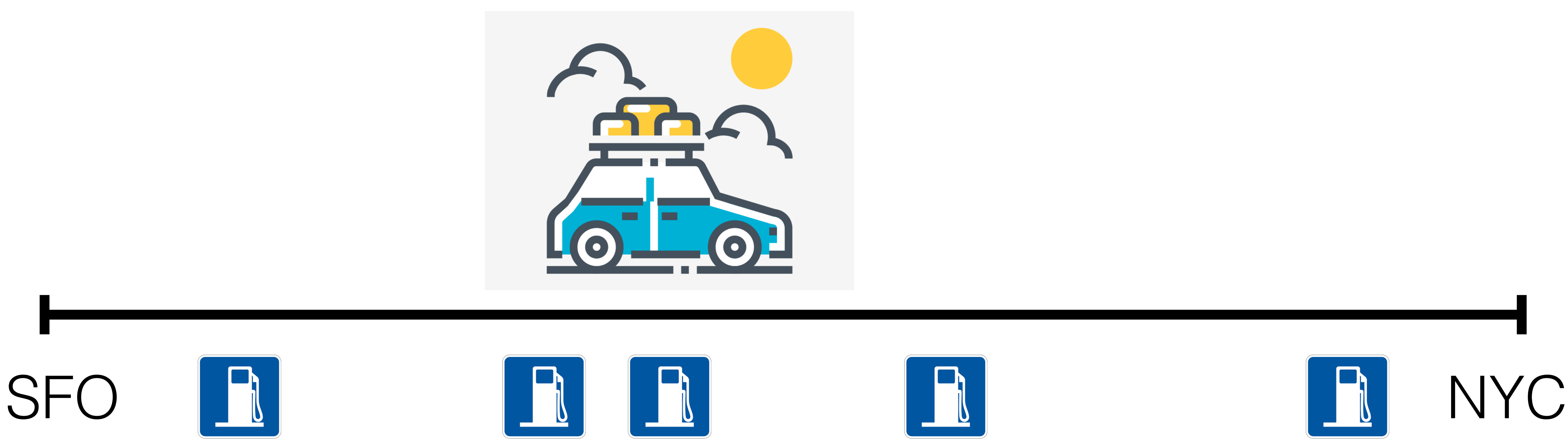


A Typical Problem Structure

Have a **global objective.** Want to minimize or maximize a quantity

Make **local optimizations.** At every step, an algorithm can make several choices; a greedy algorithm makes this choice *myopically*

- For some problems, a greedy algorithm ends up being optimal
    - Greedy happens to be *one way* to reach an optimal solution



High-Level Problem Solving Steps

- Formalize the problem
- Design the algorithm to solve the problem
    - Usually this is natural/intuitive/easy for greedy
- Prove that the algorithm is correct
  - This means proving that greedy is optimal (i.e., the resulting solution minimizes or maximizes the global problem objective)
    - This is the hard part! (which is why we will focus on it)
- Analyze running time
    - Often straightforward

Problem Example: Class Scheduling

Class scheduling. Suppose you have a single classroom.

You are given the list of start times $s_1, \ldots, s_n$ and finish times $f_1, \ldots, f_n$ of n classes (labeled $1, \ldots, n$).

What is the maximum number of non-conflicting classes you can schedule?

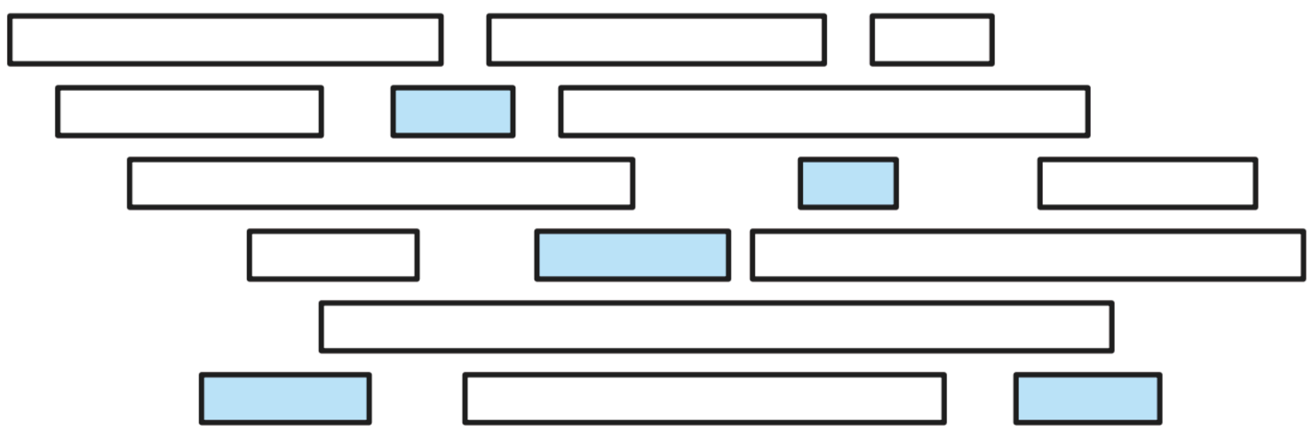


Figure 4.1. A maximum conflict-free schedule for a set of classes.

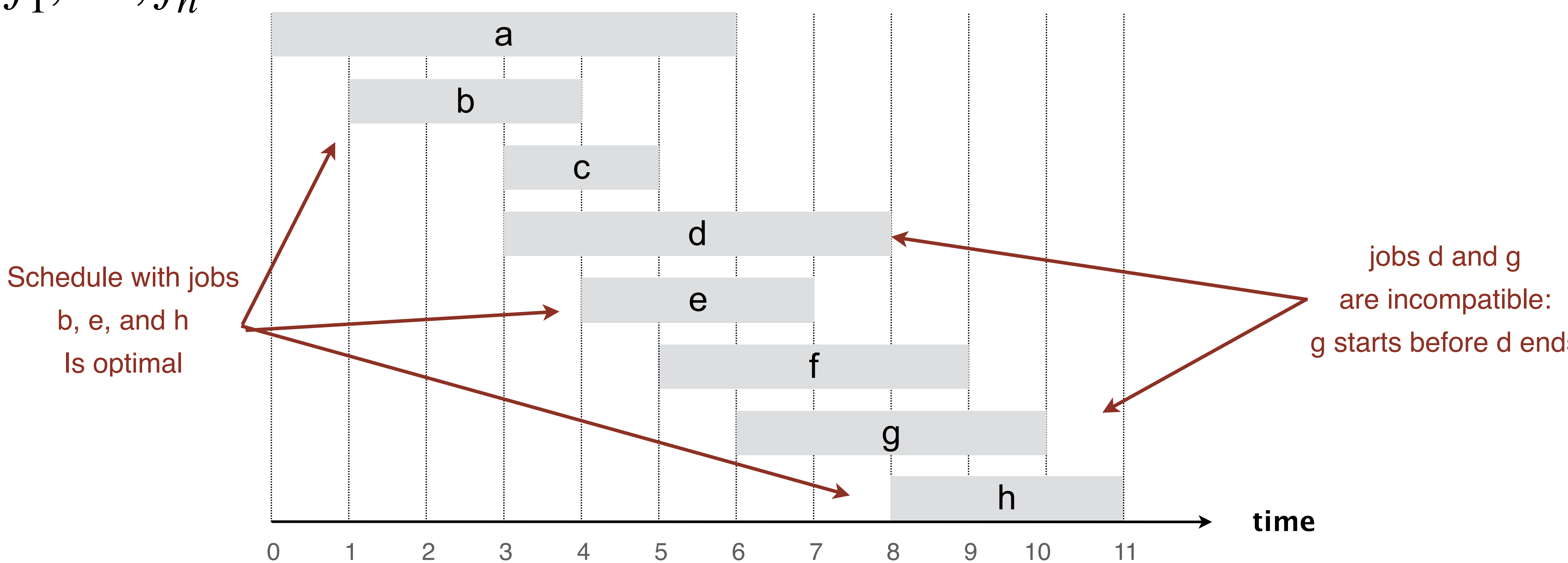
From Erickson's Algorithms Book

Problem Example: Interval Scheduling

Job scheduling. Here is a general job scheduling problem:

Suppose you have a machine that can run one job at a time.

You are given n job requests with start and finish times: $s_1, \ldots, s_n$ and $f_1, \ldots, f_n$.



How do you determine the maximum number of compatible requests?

What to be Greedy About?

Algorithmic idea: Pick a criterion to be greedy about. Keep choosing compatible jobs based on chosen criterion.

- Lets start with some of the obvious ones: **job start time**
  - **Greedy algorithm 1:** schedule jobs with **earliest start time** first
- Is this the best way?
  - If not, can we come up with a counter example?

counterexample for earliest start time

Many Ways to be Greedy

Algorithmic idea: Pick a criterion to be greedy about. Keep choosing compatible jobs based on chosen criterion.

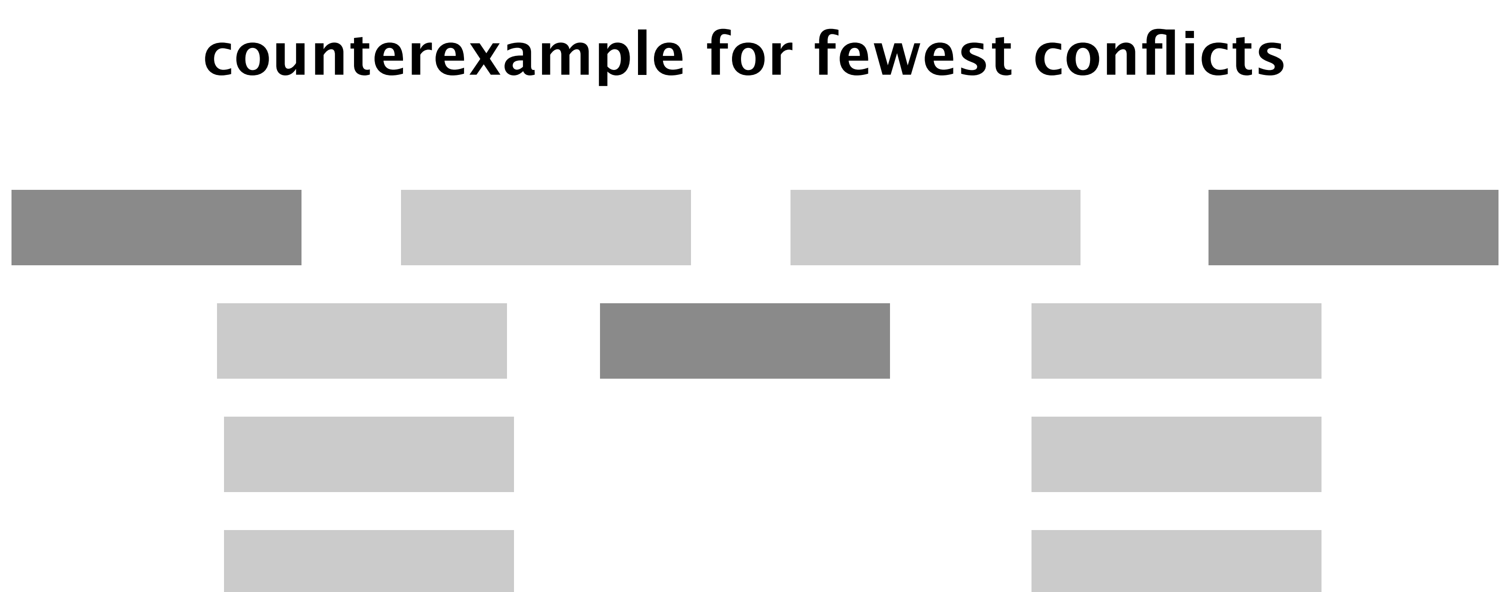
- **Greedy algorithm 2:** schedule jobs with **shortest interval** first
  - That is, smallest value of $f_i - s_i$
- Is this the best way?
  - If not, can we come up with a counter example?

counterexample for shortest interval

Many Ways to be Greedy

Algorithmic idea: Pick a criterion to be greedy about. Keep choosing compatible jobs based on chosen criterion.

- **Greedy algorithm 3:** schedule jobs that **conflict with the fewest other jobs** first
- Is this the best way?
  - If not, can we come up with a counter example?



Many Ways to be Greedy. Not all are equal…

Algorithmic idea: Pick a criterion to be greedy about. Keep choosing compatible jobs based on chosen criterion.

- We've identified criteria that do not work:
  - Earliest start time first
  - Shortest interval first
  - Fewest conflicts first
- How about: **earliest finish time first?**
  - Surprisingly, this results in an optimal algorithm!
  - But we need to ***prove*** why it is optimal
    - General idea: earliest finish time first frees the shared resource as soon as possible

Earliest-Finish-Time-First Algorithm

EARLIEST-FINISH-TIME-FIRST $(n, s_1, s_2, \ldots, s_n, f_1, f_2, \ldots, f_n)$

SORT jobs by finish times and renumber so that $f_1 \leq f_2 \leq \ldots \leq f_n$.

$S \leftarrow \varnothing$. ← set of jobs selected

FOR $j = 1$ TO n

  IF job j is compatible with S

    $S \leftarrow S \cup \{ j \}$.

RETURN S.

Proving Algorithm Correctness

- Output Set S consists of compatible requests
  - This is try by construction!
- We want to prove our solution S is optimal,
  - That is, it schedules the maximum number of jobs
  - Note: there can be more than one optimal solution
- If we let $\mathcal{O}$ be be some optimal set of jobs, then
  - **Goal:** show $|S| = |\mathcal{O}|$, i.e., our greedy solution also selects the same number of jobs and is therefore optimal

Exchange Argument

Idea behind **proof by exchange argument**:

- Transform O into G one step at a time, without hurting solution (that is, each of our transformations must *preserve optimality*)

- Let $O = o_1, o_2, \ldots, o_m$ be the sequence of jobs scheduled by the optimal algorithm, and let $G = g_1, g_2, \ldots, g_k$ be the sequence of jobs scheduled by greedy, such that $O \neq G$

- Our goal is to modify O to produce a new solution O' that is:
    - No worse than O, and
    - Closer to G in some measurable way

O (optimal) $\rightarrow$ O' (optimal) $\rightarrow$ O'' (optimal) $\rightarrow \cdots \rightarrow$ G (optimal)

Exchange Argument Proof Example

- Let $O = o_1, o_2, \ldots, o_m$ be the sequence of jobs scheduled by the optimal algorithm, and
  Let $G = g_1, g_2, \ldots, g_k$ be the sequence of jobs scheduled by greedy, both ordered by increasing finish time
- By induction, we will show that we can exchange each job scheduled by optimal with a non-conflicting job scheduled by greedy to create a new optimal schedule

Base case: $j = 1$. In the beginning, greedy picks the job with the earliest finish time, so $f_{g_1} \leq f_{o_1}$, thus g_1 does not conflict with any of the jobs $o_2, \ldots, o_m$

- We can therefore exchange o_1 with g_1 to get a new conflict-free optimal schedule $g_1, o_2, o_3, \ldots, o_m$

Exchange Argument Proof Example

Inductive hypothesis: Assume we have an optimal conflict-free schedule that is the same as greedy from job 1 up to job $j-1$

  - In other words, we have: $O' = g_1, g_2, \ldots, g_{j-1}, o_j, \ldots, o_m$

- Because both G and O' consist on non-conflicting jobs, neither g_j nor o_j conflict with $g_1, g_2, \ldots, g_{j-1}$
- Recall, greedy picks earliest finish time among non-conflicting jobs
  - Since $f_{g_j} \leq f_{o_j} \leq s_{o_{j+1}}$ which means g_j does not conflict with any remaining jobs $o_{j+1}, \ldots o_m$
- We can exchange o_j with the greedy choice g_j to construct a new optimal schedule $g_1, g_2, \ldots, g_j, o_{j+1}, \ldots, o_m$

Are We Done? Almost

- We can keep replacing every job scheduled by the optimal algorithm with a non-conflicting job scheduled by greedy until we have an optimal schedule that contains all the greedy jobs

Lemma 2. Greedy is optimal, that is, $k = m$.

Proof. (By **contradiction**) Suppose $m > k$.

- That is, we assume that there is a job o_{k+1} that starts after g_k ends
- What is the contradiction?
  - Greedy keeps selecting jobs until no more compatible jobs left. Since $f_{g_k} \leq f_{o_k}$, greedy would also select compatible job o_{k+1}

$(\Rightarrow\!\Leftarrow)$ ∎

Review: Exchange Argument Idea

- Assume there is an optimal solution O that is different from the greedy solution G
- Show that we can modify O to produce a new solution O' that is:
  - No worse than O
  - Closer to G in some measurable way

Idea behind proof by exchange argument:

- Transform O into G one step at a time, without hurting solution (that is, each transformation preserves optimality)

O (optimal) $\rightarrow$ O' (optimal) $\rightarrow$ O'' (optimal) $\rightarrow \cdots \rightarrow$ G (optimal)

Caution: Not Uniquely Optimal

We did not prove that greedy was the only optimal solution: there can be more than one optimal solution

Greedy: Proof Techniques

The textbook (reading) talks about two approaches to proving correctness of greedy algorithms

- **Greedy stays ahead**: Partial greedy solution is, at all times, as good as an "equivalent" portion of any other solution
    - Simple induction, *often has an implicit exchange argument at its heart*
- **Exchange Property**: An optimal solution can be transformed into a greedy solution without sacrificing optimality

Can use any approach that proves correctness

Example: Running Time Analysis

Let's analyze all the steps of our job-scheduling algorithm:

- Sorting and relabelling jobs by finish times
    - $O(n \log n)$
- For each selected job i, find next job j such that $s_j \geq f_i$
    - We work our way through the list from $i = 1 \ldots n$, considering each job once
    - Identifying compatibility is $O(1)$ per interval (job), so
        - $O(n)$
- Overall $O(n \log n)$ time

Review: Problem Solving Steps

- Formalize the problem
- Design the algorithm to solve the problem
  - Usually this is natural/intuitive/easy for greedy
- Prove that the algorithm is correct
  - This means proving that greedy is optimal (i.e., the resulting solution minimizes or maximizes the global problem objective)
    - This is the hard part! (which is why we spent most of our time on it)
- Analyze running time
  - Often straightforward, since greedy rules are often simple

Acknowledgments

- The pictures in these slides are taken from
  - Kleinberg Tardos Slides by Kevin Wayne (https://www.cs.princeton.edu/~wayne/kleinberg-tardos/pdf/04GreedyAlgorithmsI.pdf)
  - Jeff Erickson's Algorithms Book (http://jeffe.cs.illinois.edu/teaching/algorithms/book/Algorithms-JeffE.pdf)
- Much of the content was based on slides developed by Shikha Singh