Objectives

- Greedy Algorithms
  - Interval Scheduling
  - Interval Partitioning

Greedy Algorithms

- At each step, take as much as you can get
  - "local" optimizations

Example of Greedy Algorithm

- How do you make change to give out the fewest coins?
- Determine for 34¢

Example of Greedy Algorithm

- How do you make change to give out the fewest coins?
  - Local optimum: coin of the highest value, less than the remaining change owed

```
while change > 0:
        if change >= 25:
                print "Quarter"
                change -= 25
        elif change >= 10:
                print "Dime"
                change -= 10
        …
```

Let's generalize …

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.

Sort coins' denominations by value: $c_1 < c_2 < \ldots < c_n$.

coins selected

```
S = ϕ
while x ≠ 0
   let k be largest integer such that c_k ≤ x
   if k = 0
      return "no solution found"   ← How could this happen?
   x = x - c_k
   S = S ∪ {k}
return S
```

Is cashier's algorithm ***optimal***?

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 $c_k \le x < c_{k+1}$
    - Greedy takes coin k
  - 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. ▪

k	c_k	All optimal solutions must satisfy	Max value of coins 1, 2, …, k-1 in any OPT
1	1	$P \le 4$	-
2	5	$N \le 1$	4
3	10	$N + D \le 2$	4 + 5 = 9
4	25	$Q \le 3$	20 + 4 = 24
5	100	no limit	75 + 24 = 99

If don't take c_k

Coin-Changing: Analysis of Greedy Algorithm

- Observation. Greedy algorithm is sub-optimal for US postal denominations:
  - 500 100 98 79 78 64 44 28 17 2 1
- Counterexample. 158¢.
  - Greedy: 100, 44, 2, 2, 2, 2, 2, 2, 2.
  - Optimal: 79, 79.

Proving Greedy Algorithms Work

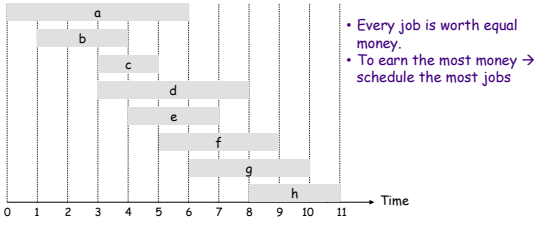
- Specifically, produce an **optimal** solution
- Two approaches:
  - Greedy algorithm stays ahead
    - Does better than any other algorithm at each step
  - Exchange argument
    - Transform any solution into a greedy solution

Greedy algorithm stays ahead

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



- Every job is worth equal money.
- To earn the most money → schedule the most jobs

Greedy Algorithm Template

- Consider jobs (or whatever) in some order
  - Decision: What order is best?
- Take each job provided it's compatible with the ones already taken

What are options for orders?

What is our goal? What are we trying to minimize/maximize?

What is the worst case?

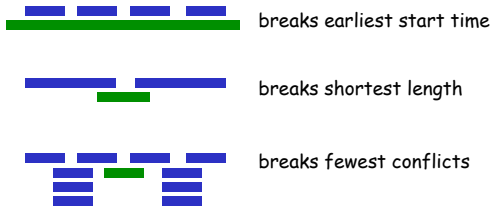
Interval Scheduling

- Earliest start time. Consider jobs in ascending order of start time s_j
  - Utilize CPU as soon as possible
- Earliest finish time. Consider jobs in ascending order of finish time f_j
  - Resource becomes free ASAP
  - Maximize time left for other requests
- 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

Can we "break" any of these? i.e., prove they're not optimal?

Counterexamples to Optimality of Various Job Orders

Not optimal when …

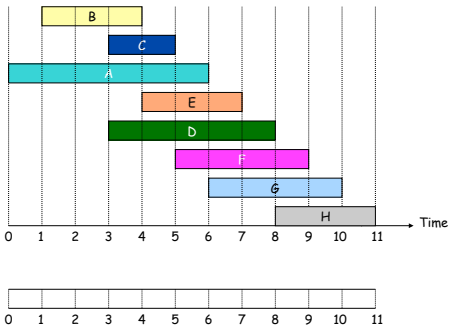


Interval Scheduling: Greedy Algorithm

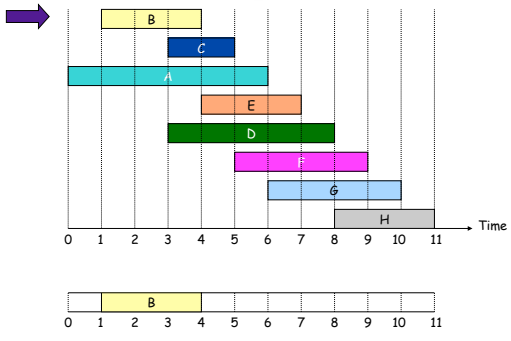
- Consider jobs in increasing order of finish time. Take each job provided it's compatible with the ones already taken.

```
jobs selected
   Sort jobs by finish times so that f1 ≤ f2 ≤ ... ≤ fn
   G = {}
   for j = 1 to n
      if job j compatible with G
         G = G ∪ {j}
   return G
```

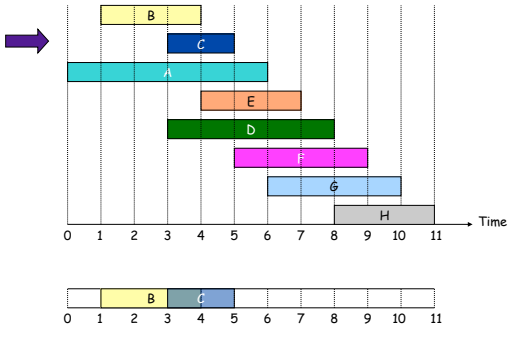

Interval Scheduling



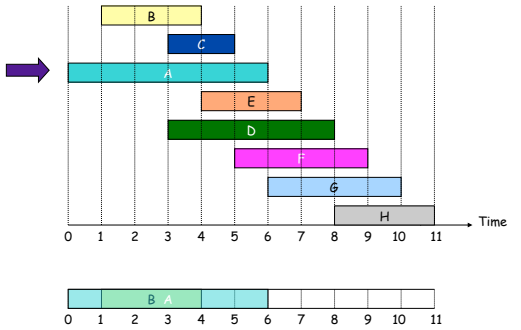
Interval Scheduling



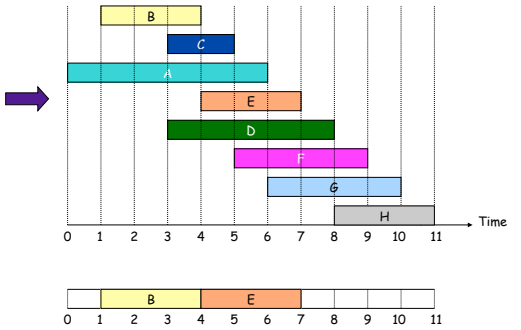
Interval Scheduling



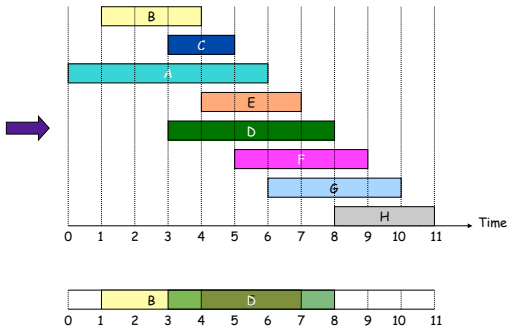
Interval Scheduling



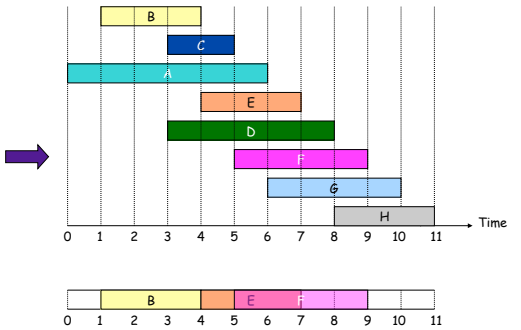
Interval Scheduling



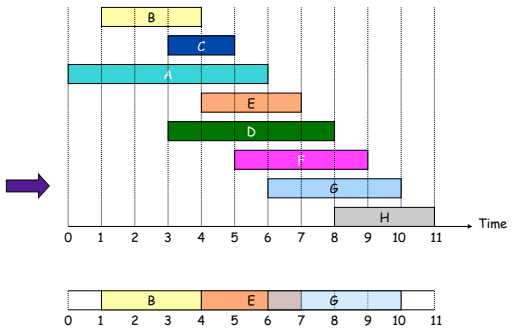
Interval Scheduling



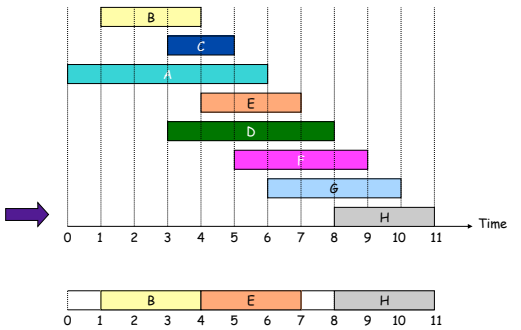
Interval Scheduling



Interval Scheduling



Interval Scheduling



Interval Scheduling: Greedy Algorithm

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

jobs selected

```
Sort jobs by finish times so that f1 ≤ f2 ≤ ... ≤ fn
G = {}
for j = 1 to n
   if job j compatible with G
      G = G ∪ {j}
return G
```

- Runtime of algorithm?
  - Where/what are the costs?

Interval Scheduling: Greedy Algorithm

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

O(n logn)

jobs selected

```
Sort jobs by finish times so that f1 ≤ f2 ≤ ... ≤ fn
G = {}
for j = 1 to n
   if job j compatible with G   O(1)   } O(n)
      G = G ∪ {j}
return G
```

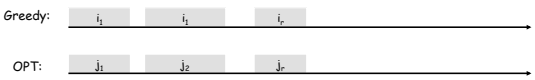
- Implementation. O(n log n)
  - Remember job j* that was added last to A
  - Job j is compatible with A if sj ≥ fj*

Interval Scheduling: Analysis

- Know that the intervals are compatible
  - Handled by the if statement
- But is it optimal?
  - What does it mean to be optimal?
  - Recall our goal for maximization

Interval Scheduling: Analysis

- 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 set of jobs selected by *greedy* (*k* jobs)
  - Let $j_1, j_2, \ldots, j_m$ denote set of jobs in the *optimal* solution (*m* jobs)
  - Same ordering, by finish times because compatible jobs
  - Want to show that *k* = *m*



What can we say about i_1 and j_1? $f(i_1) \le f(j_1)$

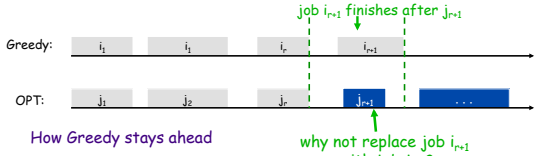
Interval Scheduling: Analysis

- Theorem. Greedy algorithm is optimal.
- Pf. (by contradiction)
  - Since we picked the first job to have the first finishing time, we know that $f(i_1) <= f(j_1)$
  - Want to show that Greedy "stays ahead"
  - Each interval finishes at least as soon as Optimal's
  - Induction hypothesis: for all indices r <= k, $f(i_r) <= f(j_r)$

Prove for r+1

Interval Scheduling: Analysis

- Theorem. Greedy algorithm is optimal.
- Pf. (by contradiction)
  - Since we picked the first job to have the first finishing time, we know that $f(i_1) <= f(j_1)$
  - Want to show that Greedy "stays ahead"
  - Each interval finishes at least as soon as Optimal's
  - Induction hypothesis: for all indices r <= k, $f(i_r) <= f(j_r)$



How Greedy stays ahead

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

Problem Assumptions

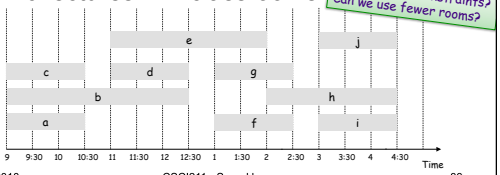
- All requests were known to scheduling algorithm
  - Online algorithms: make decisions without knowledge of future input
- Each job was worth the same amount
  - What if jobs had *different* values?
    - E.g., scaled with size
- Single resource requested
  - Rejected requests that didn't fit

INTERVAL PARTITIONING

Interval Partitioning

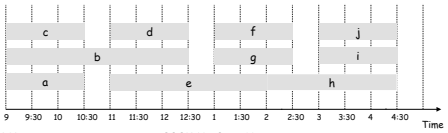
- Lecture j starts at s_j and finishes at f_j.
- Goal: find minimum number of classrooms to schedule all lectures so that no two occur at the same time in the same room.
- Ex: 10 lectures in 4 classrooms

What are our constraints? Can we use fewer rooms?



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.
- Alternative schedule uses only 3 classrooms



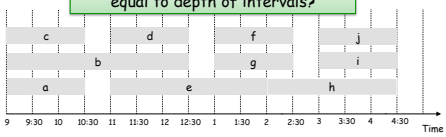
Interval Partitioning: Lower Bound on Optimal Solution

- Def. The depth of a set of open intervals is the maximum number that contain any given time.
- Key observation. # of classrooms needed ≥ depth.

a, b, c all contain 9:30

- Ex: Depth of schedule below = 3 ⇒ schedule below is optimal.

Does there always exist a schedule equal to depth of intervals?



Interval Partitioning Discussion

- Does there always exist a schedule equal to depth of intervals?
- Can we make decisions locally to get a global optimum?
  - Or are there long-range obstacles that require more resources?

Assignments

- Read Chapter 4
- Friday: Problem Set 3
- Today at 3:30 – Professor Crowley's talk