

## Objectives

- Greedy Algorithms: Interval Scheduling

## Review

- What is a greedy algorithm?

## Review: Greedy Algorithms

At each step, take as much as you can get  
→ “local” optimizations

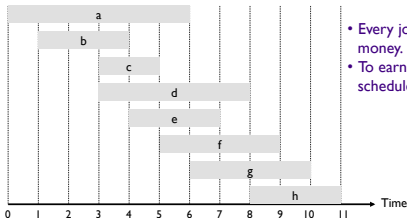
- Need a proof to show that the algorithm finds an optimal solution
- A counter example shows that a greedy algorithm does not provide an optimal solution

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? (rhetorical for now)

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

What is the worst case?

### Greedy Algorithm Pseudo-Code

```

Set Greedy (Set candidate){
  solution = new Set( );
  while candidate.isNotEmpty()
    next = candidate.select() //use selection criteria,
    //remove from candidate and return value
    if solution.isFeasible(next) //constraints satisfied
      solution.union(next)
    if solution.solves()
      return solution
  //No more candidates and no solution
  return null
}
    
```

In some specified order

### 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? (rhetorical for now)

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

What is the worst case?

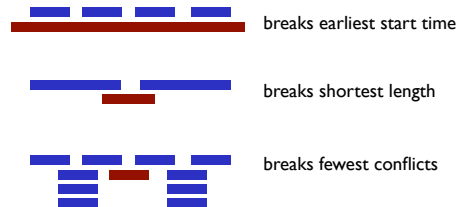
### Interval Scheduling

- 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 ...



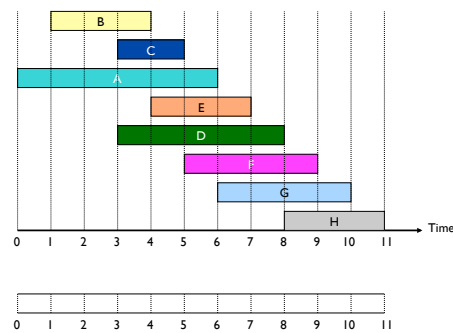
### Interval Scheduling: Greedy Algorithm

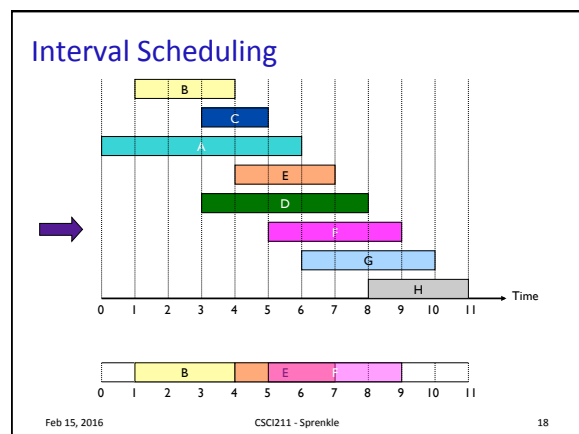
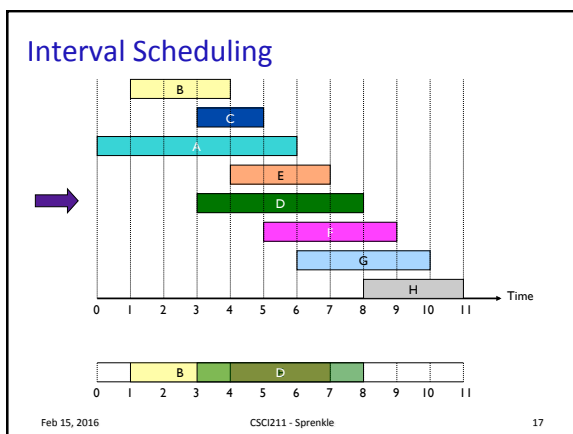
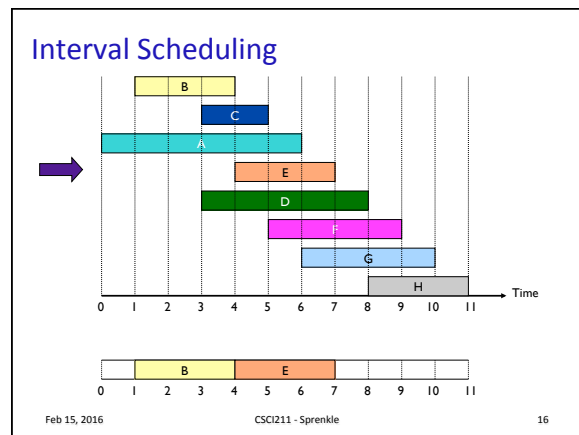
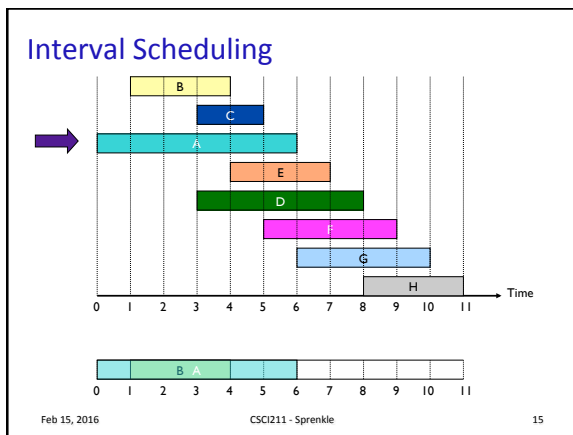
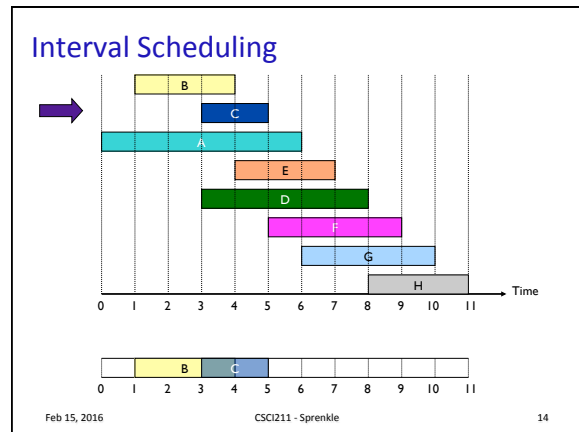
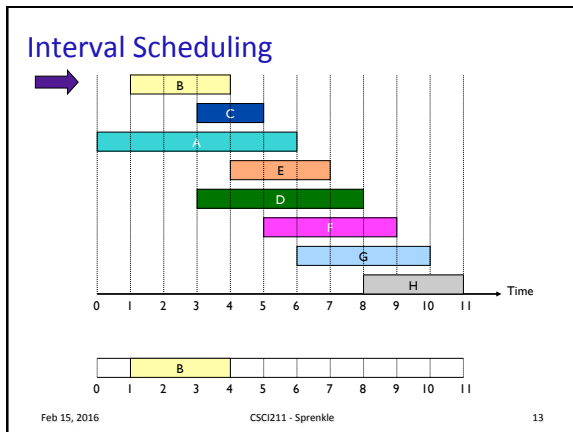
- Consider jobs in **increasing order of finish time**
- Take each job provided it's compatible with the ones already taken

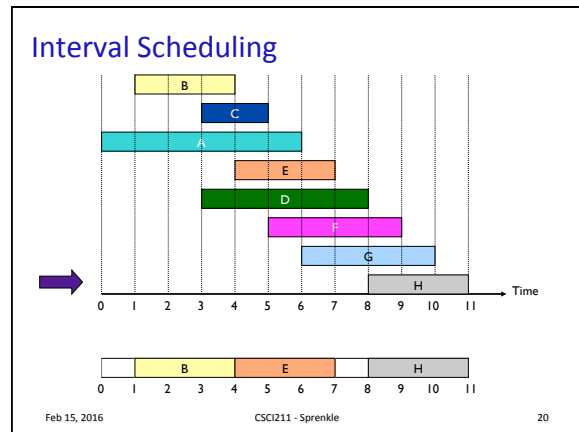
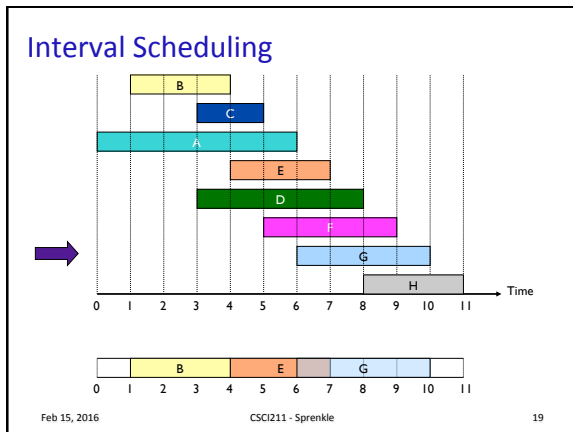
```

Sort jobs by finish times so that  $f_1 \leq f_2 \leq \dots \leq f_n$ 
selected
G = {}
for j = 1 to n
  if job j compatible with G
    G = G ∪ {j}
return G
    
```

### 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 Sort jobs by finish times so that  $f_1 \leq f_2 \leq \dots \leq f_n$ 
selected  $G = \{\}$ 
for  $j = 1$  to  $n$ 
  if job  $j$  compatible with  $G$ 
     $G = G \cup \{j\}$ 
return  $G$ 
    
```

Runtime of algorithm?

- Where/what are the costs?

Feb 15, 2016 CSCI211 - Spenkle 21

### Interval Scheduling: Greedy Algorithm

- Consider jobs in increasing order of finish time.
- Take each job provided it's compatible with the ones already taken.  $O(n \log n)$

```

jobs Sort jobs by finish times so that  $f_1 \leq f_2 \leq \dots \leq f_n$ 
selected  $G = \{\}$ 
for  $j = 1$  to  $n$ 
  if job  $j$  compatible with  $G$   $O(1)$ 
     $G = G \cup \{j\}$ 
return  $G$ 
    
```

$O(n)$

- Implementation.  $O(n \log n)$ 
  - Remember job  $j^*$  that was added last to  $A$
  - Job  $j$  is compatible with  $A$  if  $s_j \geq f_{j^*}$

Feb 15, 2016 CSCI211 - Spenkle 22

### Analyzing Interval Scheduling

- 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

Feb 15, 2016 CSCI211 - Spenkle 23

### Greedy Stays Ahead Proofs

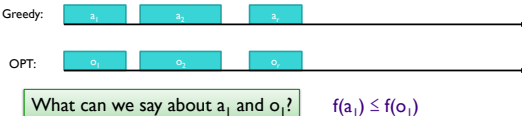
- Define your solutions
  - Describe the form of your greedy solution ( $A$ ) and of some other solution (possibly the optimal solution,  $O$ )
- Find a measure
  - Find a measure by which greedy *stays ahead* of the optimal solution
    - Ex: Let  $a_1, \dots, a_k$  be the first  $k$  measures of greedy algorithm and  $o_1, \dots, o_m$  be the first  $m$  measures of other solution (sometimes  $m = k$ )
- Prove greedy stays ahead
  - Show that greedy's partial solutions constructed are always just as good as the optimal solution's initial segments based on the measure
    - Ex: for all indices  $r \leq \min(k, m)$ , prove by induction that  $a_r \geq o_r$  or  $a_r \leq o_r$
  - Use the greedy algorithm to help you argue the inductive step
- Prove optimality
  - Prove that since greedy stays ahead of the other solution with respect to the measure, then the greedy solution is optimal

→ Make sure maps back to measure of optimality

Feb 15, 2016 CSCI211 - Spenkle 24

### Interval Scheduling: Analysis


- Theorem. Greedy algorithm is optimal, i.e., schedules the most jobs possible
- Pf. (by contradiction)
  - Assume greedy is not optimal
  - Let  $a_1, a_2, \dots, a_k$  denote set of jobs selected by greedy ( $k$  jobs)
  - Let  $o_1, o_2, \dots, o_m$  denote set of jobs in optimal solution ( $m$  jobs)
  - Both sets ordered by finish time for comparison ordering
  - Want to show that  $k = m$



Feb 15, 2016 CSC211 - Spenkile 25

### Interval Scheduling: Analysis

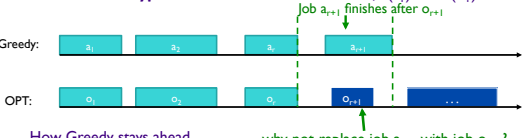
- Theorem. Greedy algorithm is optimal
  - i.e., schedules the most jobs possible
- Pf. (by contradiction)
  - Since we picked the first job to have the first finishing time, we know that  $f(a_1) \leq f(o_1)$
  - Want to show that Greedy “stays ahead”
    - Each interval finishes at least as soon as Optimal’s
  - Induction hypothesis: for all indices  $r \leq k$ ,  $f(a_r) \leq f(o_r)$
  - Prove for  $r+1$



Feb 15, 2016 CSC211 - Spenkile 26

### Interval Scheduling: Analysis

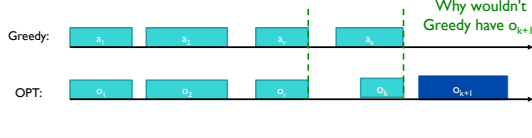
- Theorem. Greedy algorithm is optimal
  - i.e., schedules the most jobs possible
- Pf. (by contradiction)
  - Since we picked the first job to have the first finishing time, we know that  $f(a_1) \leq f(o_1)$
  - Want to show that Greedy “stays ahead”
    - Each interval finishes at least as soon as Optimal’s
  - Induction hypothesis: for all indices  $r \leq k$ ,  $f(a_r) \leq f(o_r)$



Feb 15, 2016 CSC211 - Spenkile 27

### Interval Scheduling: Analysis

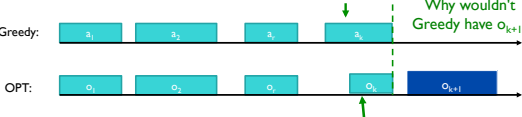
- Theorem. Greedy algorithm is optimal.
  - i.e., schedules the most jobs possible
- Pf. (by contradiction)
  - Assume Greedy is not optimal (i.e.,  $m > k$ )
    - Optimal solution has more jobs than Greedy
  - We already showed that for all indices  $r \leq k$ ,  $f(a_r) \leq f(o_r)$
  - Since  $m > k$ , there is a request  $o_{k+1}$  in Optimal



Feb 15, 2016 CSC211 - Spenkile 28

### Interval Scheduling: Analysis

- Theorem. Greedy algorithm is optimal.
  - i.e., schedules the most jobs possible
- Pf. (by contradiction)
  - Assume Greedy is not optimal (i.e.,  $m > k$ )
  - We already showed that for all indices  $r \leq k$ ,  $f(a_r) \leq f(o_r)$
  - Since  $m > k$ , there is a request  $o_{k+1}$  in Optimal
    - Starts after  $o_k$  ends  $\rightarrow$  after  $a_k$  ends
  - So, Greedy could also add  $o_k$ 
    - Contradiction because now Greedy has another job



Feb 15, 2016 CSC211 - Spenkile 29

### Greedy Algorithm Pseudo-Code

In some specified order

```

Set Greedy (Set candidate){
    solution = new Set( );
    while candidate.isNotEmpty()
        next = candidate.select() //use selection criteria,
        //remove from candidate and return value
        if solution.isFeasible(next) //constraints satisfied
            solution.union(next)
            if solution.solves()
                return solution

    //No more candidates and no solution
    return null
}
    
```

Feb 15, 2016 CSC211 - Spenkile 30

### 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

Feb 15, 2016

CSCI211 - Sprenkle

31

### INTERVAL PARTITIONING

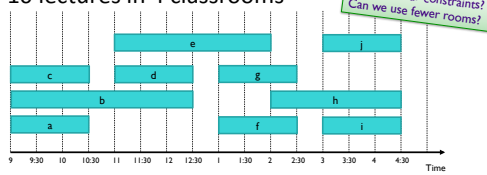
Feb 15, 2016

CSCI211 - Sprenkle

32

### 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



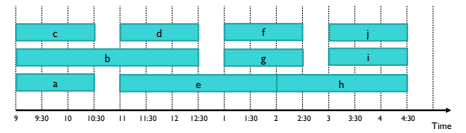
Feb 15, 2016

CSCI211 - Sprenkle

33

### 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



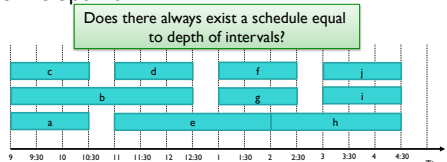
Feb 15, 2016

CSCI211 - Sprenkle

34

### 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  $\geq$  depth.
- Ex: Depth of schedule below = 3  $\Rightarrow$  schedule below is optimal.



Feb 15, 2016

CSCI211 - Sprenkle

35

### 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?

Feb 15, 2016

CSCI211 - Sprenkle

36

## Interval Partitioning: Greedy Algorithm

- Consider lectures in increasing order of start time: assign lecture to any compatible classroom

```
Sort intervals by starting time so that  $s_1 \leq s_2 \leq \dots \leq s_n$ 
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
```

Analyze algorithm

Feb 15, 2016

CSCI211 - Sprenkle

37

## Interval Partitioning: Greedy Algorithm

- Consider lectures in increasing order of start time: assign lecture to any compatible classroom

```
Sort intervals by starting time so that  $s_1 \leq s_2 \leq \dots \leq s_n$ 
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:  $O(n \log n)$ 
  - For each classroom  $k$ , maintain the finish time of the last job added.
  - Keep the classrooms in a priority queue by last job finish time.

Feb 15, 2016

CSCI211 - Sprenkle

38

## Interval Partitioning: Greedy Analysis

- Observation.** Greedy algorithm never schedules two incompatible lectures in the same classroom
- Theorem.** Greedy algorithm is optimal
- Pf Intuition**
  - When do we add more classrooms?
  - When would we add the  $d+1$  classroom?

Feb 15, 2016

CSCI211 - Sprenkle

39

## 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  $s_j + \epsilon$
  - $d$  is the depth of the set of lectures

Feb 15, 2016

CSCI211 - Sprenkle

40

## Assignments

- Journal for tonight
- Problem Set 4 for Friday

Feb 15, 2016

CSCI211 - Sprenkle

41