Objectives

- Greedy Algorithms
  - Interval Scheduling
  - Interval Partitioning

Review: Greedy Algorithms

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

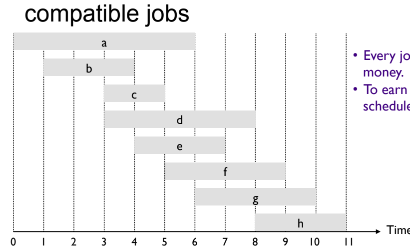
- Need a proof to show that the algorithm finds an optimal solution
- A counter example shows that a greedy algorithm does not provide an optimal solution

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?

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

What is the worst case?

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
}
```

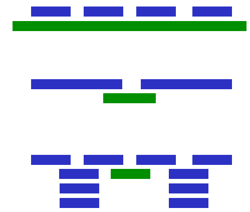
Interval Scheduling

- 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 …



breaks earliest start time

breaks shortest length

breaks fewest conflicts

Interval Scheduling: Greedy Algorithm

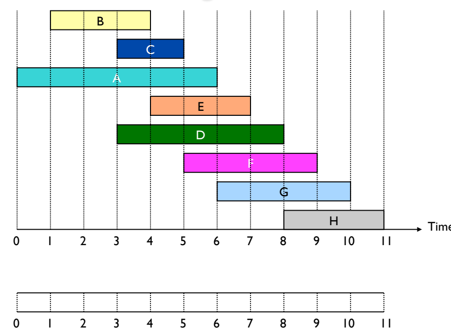
- Consider jobs in increasing order of finish time. Take each job provided it's compatible with the ones already taken.

jobs selected

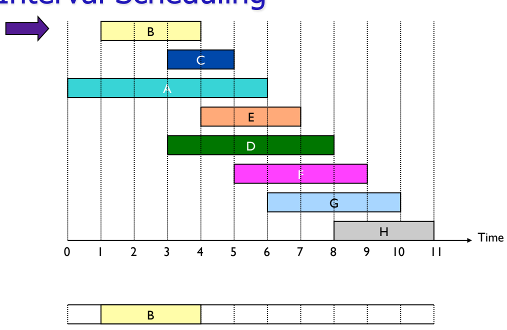
```
Sort jobs by finish times so that f1 ≤ f2 ≤ ... ≤ fn

G = {}
for j = 1 to n
   if job j compatible with G
      G = G ∪ {j}
return G
```

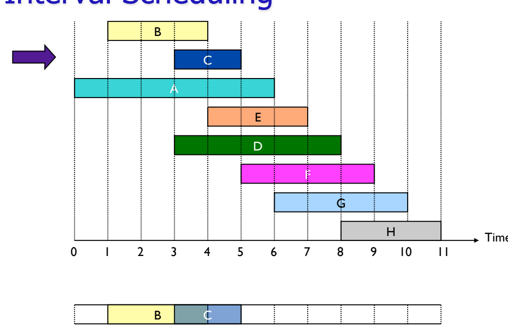

Interval Scheduling



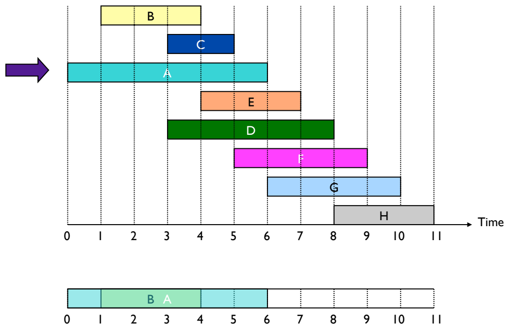
Interval Scheduling



Interval Scheduling

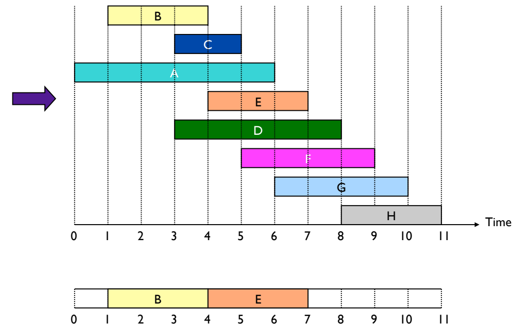


Interval Scheduling



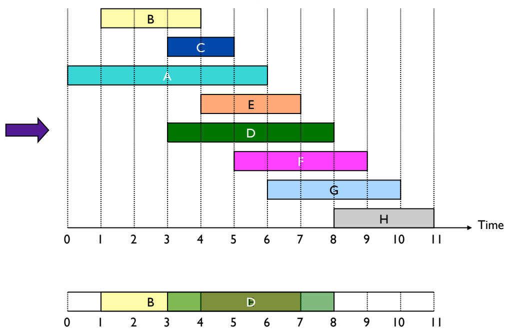
Feb 2, 2011 CSCI211 - Sprenkle 13

Interval Scheduling



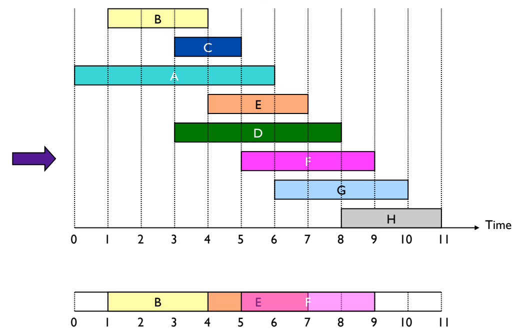
Feb 2, 2011 CSCI211 - Sprenkle 14

Interval Scheduling



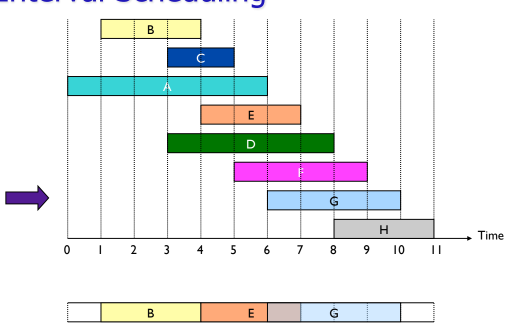
Feb 2, 2011 CSCI211 - Sprenkle 15

Interval Scheduling



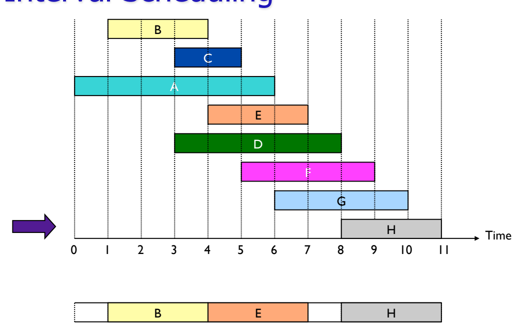
Feb 2, 2011 CSCI211 - Sprenkle 16

Interval Scheduling



Feb 2, 2011 CSCI211 - Sprenkle 17

Interval Scheduling



Feb 2, 2011 CSCI211 - Sprenkle 18

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 $s_j \geq f_j^*$

Interval Scheduling: Analysis

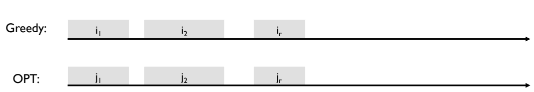
- Know that the intervals are compatible
  - Handled by the if statement
- But is it optimal?
  - What does it mean to be optimal?
  - Recall our goal for maximization

Greedy Stays Ahead Proofs

1. Define your solutions
   - Describe the form of your greedy solution and of some other solution (possibly the optimal solution)
     - Example: Let A be the solution constructed by the greedy algorithm and O be an solution.
2. Find a measure
   - Find a measure by which greedy stays ahead of the optimal solution
     - Ex: Let $a_1, \ldots, a_k$ be the first k measures of greedy algorithm and $o_1, \ldots, o_m$ be the first m measures of other solution (sometimes m = k)
3. Prove greedy stays ahead
   - Show that the partial solutions constructed by greedy are always just as good as the initial segments of the optimal solution, 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
4. Prove optimality
   - Prove that since greedy stays ahead of the other solution with respect to the measure, then the greedy solution is optimal.

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?

$f(i_1) \leq f(j_1)$

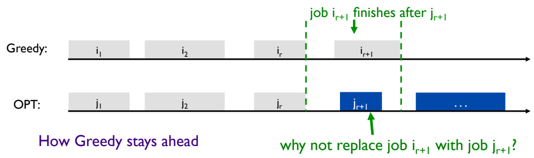
Interval Scheduling: Analysis

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

Prove for r+1

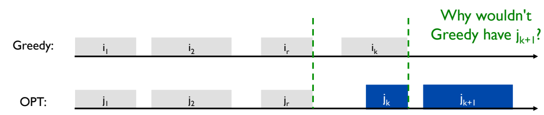
Interval Scheduling: Analysis

- 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 $r <= k$, $f(i_r) <= f(j_r)$



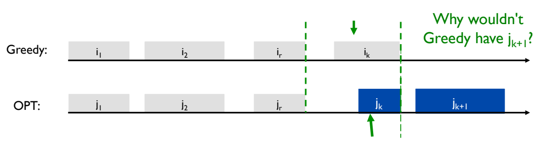
Interval Scheduling: Analysis

- Theorem. Greedy algorithm is optimal.
- Pf. (by contradiction)
  - Assume Greedy is not optimal (i.e., $m > k$)
  - We already showed that for all indices $r \leq k$, $f(i_r) \leq f(j_r)$
  - Since $m > k$, there is a request j_{k+1} in Optimal



Interval Scheduling: Analysis

- Theorem. Greedy algorithm is optimal.
- Pf. (by contradiction)
  - Assume Greedy is not optimal (i.e., $m > k$)
  - We already showed that for all indices $r \leq k$, $f(i_r) \leq f(j_r)$
  - Since $m > k$, there is a request j_{k+1} in Optimal
    - Starts after j_k ends → after i_k ends
  - So, Greedy could also add j_k
    - Contradiction because now Greedy has another job



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
}
```


Problem Assumptions

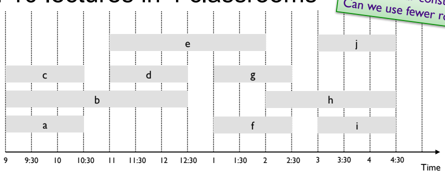
- All requests were known to scheduling algorithm
  - Online algorithms: make decisions without knowledge of future input
- Each job was worth the same amount
  - What if jobs had *different* values?
    - E.g., scaled with size
- Single resource requested
  - Rejected requests that didn't fit

INTERVAL PARTITIONING

Interval Partitioning

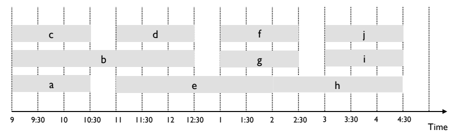
- Lecture j starts at s_j and finishes at f_j
- Goal: find minimum number of classrooms to schedule all lectures so that no two occur at the same time in the same room.
- Ex: 10 lectures in 4 classrooms

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

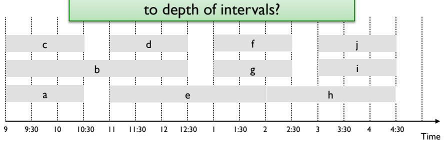


Interval Partitioning: Lower Bound on Optimal Solution

- Def. The depth of a set of open intervals is the maximum number that contain any given time.
- Key observation. # of classrooms needed ≥ depth.
- Ex: Depth of schedule below = 3 ⇒ schedule below is optimal.

a, b, c all contain 9:30

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?

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 s1 ≤ s2 ≤ ... ≤ sn
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
```


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 s1 ≤ s2 ≤ ... ≤ sn
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.

Assignments

- Read Chapter 4
- Friday: Problem Set 3