Objectives

- Greedy Algorithms
  - Interval partitioning
  - Minimizing Lateness
- Exchange argument

Review

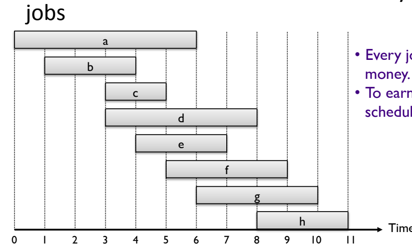
- What is the template for a greedy solution?
- What problem did we solve optimally with a greedy algorithm?
- How did we prove optimality?

Review: Greedy Algorithms

- Template
  1. Consider jobs (or whatever) in some order
     - Decision: What order is best?
  2. Take each job provided it's compatible with the ones already taken
- At each step, take as much as you can get
  - Feasible – satisfy problem's constraints
  - Locally optimal – best local choice among available feasible choices
  - Irrevocable – after decided, no going back

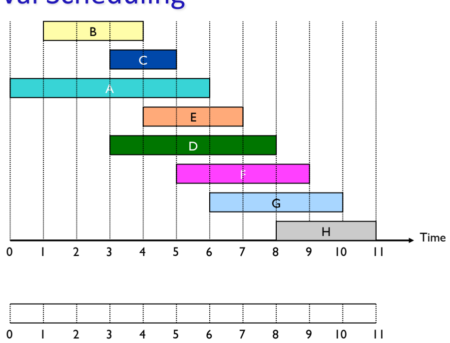
Review: Interval Scheduling

- 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

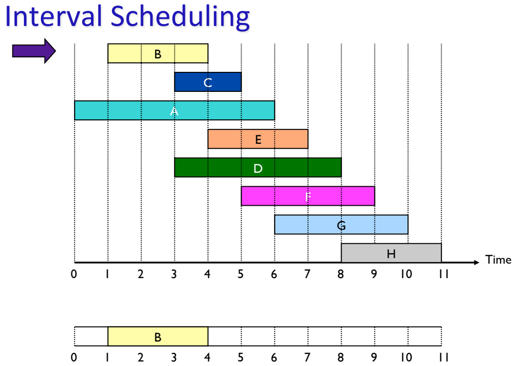


- Every job is worth equal money.
- To earn the most money → schedule the most jobs

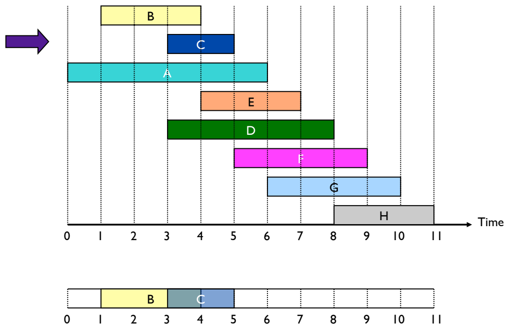
Interval Scheduling



Interval Scheduling

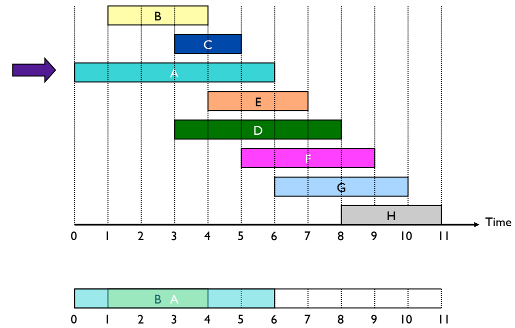


Interval Scheduling



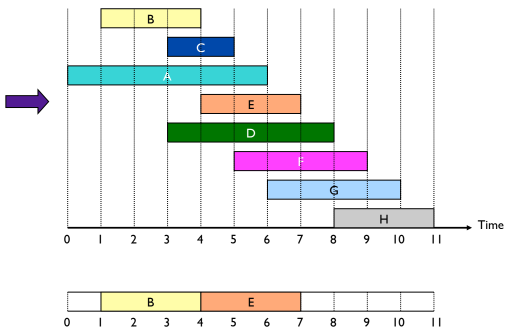
Feb 17, 2016 CSCI211 - Sprenkle 7

Interval Scheduling



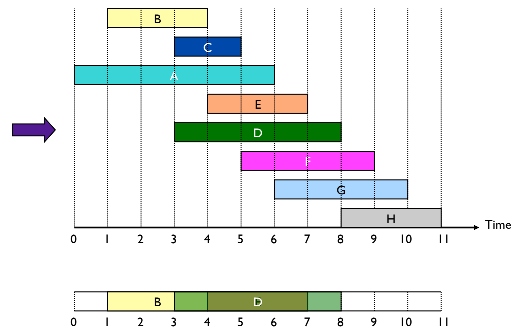
Feb 17, 2016 CSCI211 - Sprenkle 8

Interval Scheduling



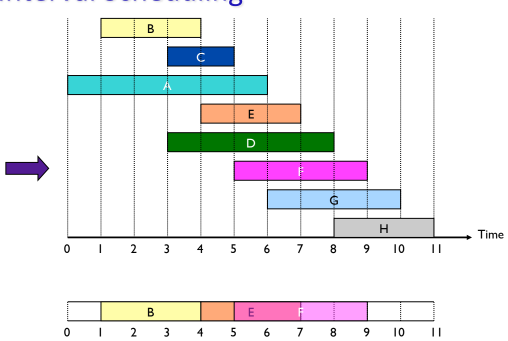
Feb 17, 2016 CSCI211 - Sprenkle 9

Interval Scheduling



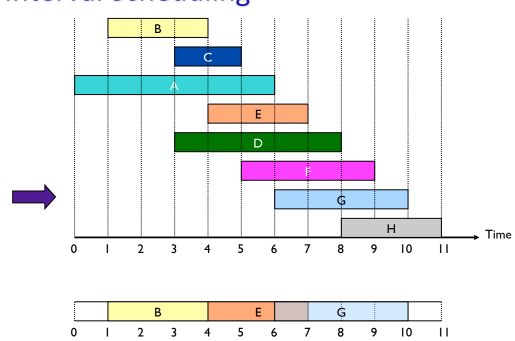
Feb 17, 2016 CSCI211 - Sprenkle 10

Interval Scheduling



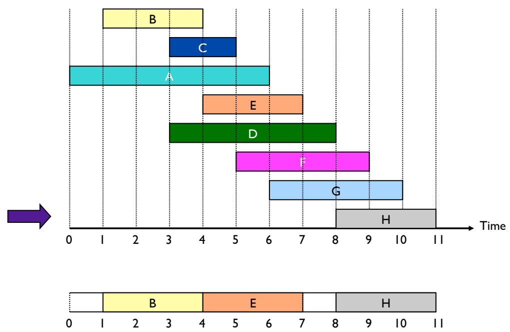
Feb 17, 2016 CSCI211 - Sprenkle 11

Interval Scheduling



Feb 17, 2016 CSCI211 - Sprenkle 12

Interval Scheduling

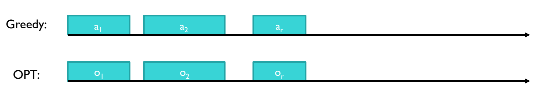


Review: Greedy Stays Ahead Proofs

1. Define your solutions
   - Describe the form of your greedy solution (**A**) and of some other solution (possibly the optimal solution, **O**)
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 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
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, i.e., schedules the most jobs possible
- Pf. (by contradiction)
  - Assume greedy is not optimal
  - Let $a_1, a_2, \ldots, a_k$ denote set of jobs selected by *greedy* (*k* jobs)
  - Let $o_1, o_2, \ldots, o_m$ denote set of jobs in *optimal* solution (*m* jobs)
  - Both sets ordered by finish time for comparison ordering
  - Want to show that *k = m*

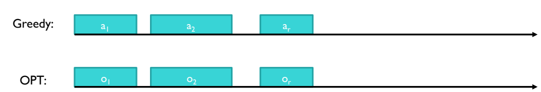


What can we say about a_1 and o_1? $f(a_1) \leq f(o_1)$

Interval Scheduling: Analysis

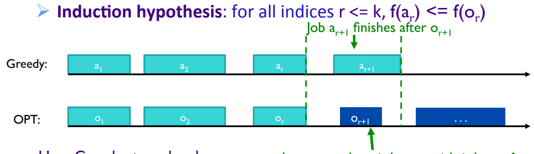
- Theorem. Greedy algorithm is optimal
  - i.e., schedules the most jobs possible
- Pf. (by contradiction)
  - Since we picked the first job to have the first finishing time, we know that $f(a_1) <= f(o_1)$
  - Want to show that Greedy "stays ahead"
    - Each interval finishes at least as soon as Optimal's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

Prove for r+1



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 $f(a_1) <= f(o_1)$
  - Want to show that Greedy "stays ahead"
    - Each interval finishes at least as soon as Optimal's
  - **Induction hypothesis**: for all indices $r <= k$, $f(a_r) <= f(o_r)$

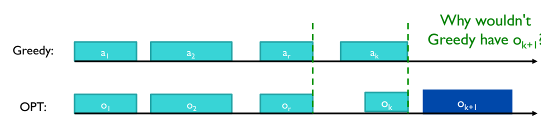


How Greedy stays ahead

why not replace job a_{r+1} with job o_{r+1}?

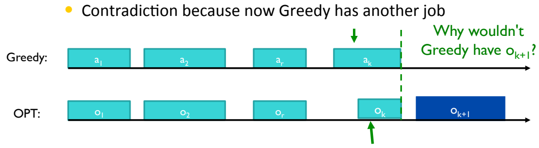
Interval Scheduling: Analysis

- 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



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 ≤ k, $f(i_r) \le f(j_r)$
  - Since m > k, there is a request o_{k+1} in Optimal
    - Starts after o_k ends → after a_k ends
  - So, Greedy could *also* add o_k
    - Contradiction because now Greedy has another job



Problem Assumptions

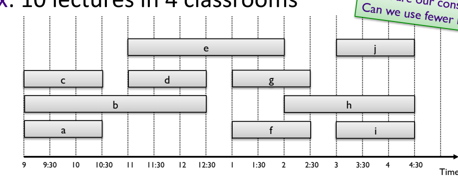
- All requests were known to scheduling algorithm
  - Online algorithms: make decisions without knowledge of future input
- Each job was worth the same amount
  - What if jobs had *different* values?
    - E.g., scaled with size
- Single resource requested
  - Rejected requests that didn't fit

INTERVAL PARTITIONING

Interval Partitioning

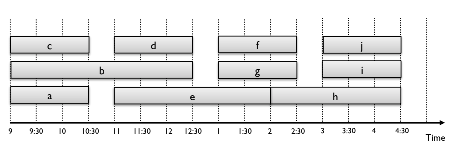
- Lecture j starts at s_j and finishes at f_j
- Goal: find minimum number of classrooms to schedule all lectures so that no two occur at the same time in the same room.
- Ex: 10 lectures in 4 classrooms

What are our constraints? Can we use fewer rooms?



Interval Partitioning

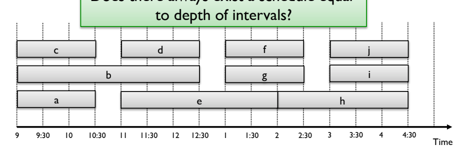
- Lecture j starts at s_j and finishes at f_j
- Goal: find minimum number of classrooms to schedule all lectures so that no two occur at the same time in the same room.
- Alternative schedule uses only 3 classrooms



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?

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
```

Analyze algorithm

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 by last job finish time.

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?

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

Exchange argument

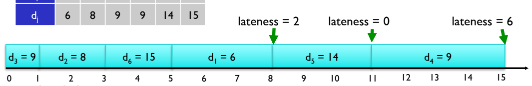
SCHEDULING TO MINIMIZE MAX LATENESS

Scheduling to Minimizing Max Lateness

- Single resource processes one job at a time
- Job j requires t_j units of processing time and is due at time d_j (its deadline)
- If j starts at time s_j, it finishes at time $f_j = s_j + t_j$
- Lateness: $\ell_j = \max \{ 0, f_j - d_j \}$
- **Goal**: schedule all jobs to ***minimize*** **maximum lateness** $L = \max \ell_j$

Input:

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



One schedule

Note: **not** a sum total

Greedy Algorithms

- Greedy template.
  Consider jobs in some order
- What do we want to optimize?
- What order?
  - Intuition of order?
  - Counter examples for order being optimal?

Minimizing Lateness: Greedy Algorithms

- Greedy template. Consider jobs in some order.
  - Shortest processing time first. Consider jobs in ascending order of processing time t_j.

Counter example

	1	2
t_j	1	10
d_j	100	10

  - Smallest slack. Consider jobs in ascending order of slack $d_j - t_j$.

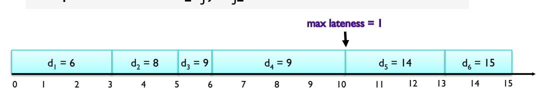
Counter example

	1	2
t_j	1	10
d_j	2	10

Minimizing Lateness: Greedy Algorithm

- Earliest deadline first.

```
Sort n jobs by deadline so that d1 ≤ d2 ≤ … ≤ dn
t = 0
for j = 1 to n
   Assign job j to interval [t, t + tj]
   sj = t
   fj = t + tj
   t = t + tj
output intervals [sj, fj]
```



What can we say about this algorithm/its results?

Looking Ahead

- Problem Set 4 due Friday