Objectives

Greedy Algorithms

Greedy Algorithms

At each step

- Decision: 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

Proving Greedy Algorithms Work

Specifically, produce an **optimal** solution

Two approaches:

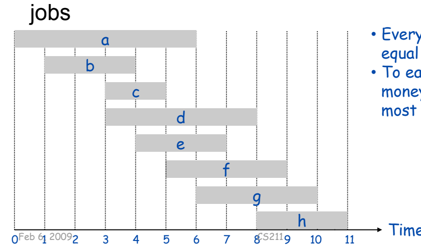
- Greedy algorithm stays ahead
  - Does better than any other algorithm at each step
- Exchange argument
  - Transform any solution into a greedy solution

Interval Scheduling

Job j starts at s_j and finishes at f_j

Two jobs *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

Interval Scheduling: Greedy Algorithms

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 number of conflicting jobs c_j. Schedule in ascending order of conflicts c_j

Interval Scheduling: Greedy Algorithms

Not optimal when …

breaks earliest start time

breaks shortest interval

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.

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

A = {}
for j = 1 to n
   if (job j compatible with A)
      A = A ∪ {j}
return A
```

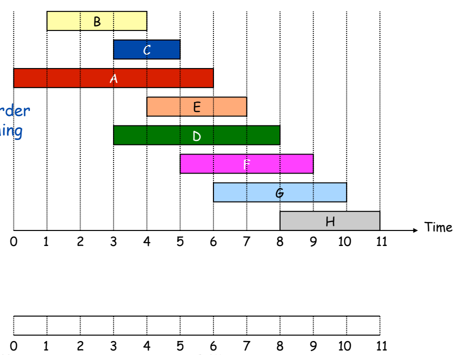
jobs selected

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

Jobs in order of finishing time



Interval Scheduling: Analysis

Know that the intervals are compatible

- Handle by the if statement

But is it optimal?

- What are we looking for?

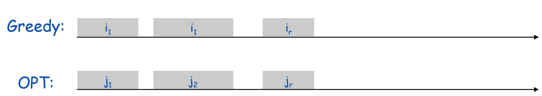
Interval Scheduling: Analysis

Theorem. Greedy algorithm is optimal.

Proof Setup: (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

➡Want to show that *k = m*



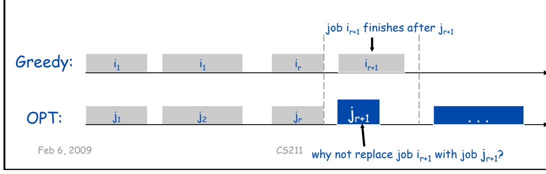
What can we say about i_1 and j_1? $f(i_1) <= f(j_1)$

Interval Scheduling: Analysis

Lemma. For all indices $r \leq k$, $f(i_r) \leq f(j_r)$

Pf. (by induction)

- Base case: Since Greedy's first job has the first finishing time, we know that $f(i_1) \leq f(j_1)$
- Want to show that Greedy "stays ahead" of Optimal
  - Each interval finishes at least as soon as Optimal's
- Induction hypothesis: assume that $f(i_r) <= f(j_r)$
- For that not to be true for r+1, Greedy would need to fall behind



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 m == k

Why wouldn't Greedy have j_{k+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

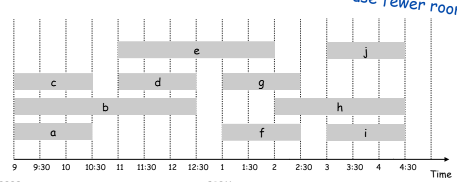
Interval Partitioning

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: 4 classrooms, 10 lectures

What are our constraints?
Can we use fewer rooms?

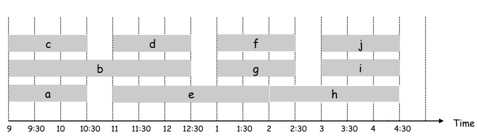


Interval Partitioning

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 Ex: This schedule uses only 3.



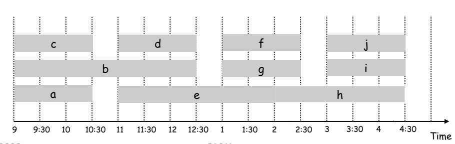
Interval Partitioning: Lower Bound on Optimal Solution

Def. The *depth* of a set of open intervals is the maximum number that contain any given time

Key observation. Number of classrooms needed $\geq$ depth

a, b, c all contain 9:30

Ex: Depth of schedule = 3 $\Rightarrow$ schedule is optimal



Interval Partitioning

Q. 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
```

Runtime/Implementation?

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 finish time of last job added
- Keep the classrooms in a priority queue

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 LATENESS

Scheduling to Minimizing 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

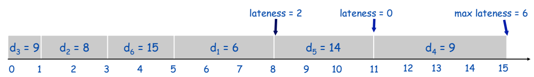
If j starts at time s_j, it finishes at time $f_j = s_j + t_j$

Lateness: $\ell_j = \max \{ 0, \; f_j - d_j \}$

Goal: schedule all jobs to minimize maximum lateness $L = \max \ell_j$

Ex:

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



Minimizing Lateness: 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.

	1	2
t_j	1	10
d_j	100	10

Counter example

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

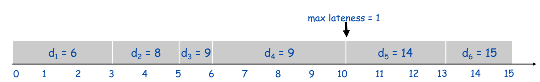
	1	2
t_j	1	10
d_j	2	10

Counter example

Minimizing Lateness: Greedy Algorithm

Greedy algorithm. Earliest deadline first.

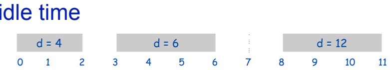
```
Sort n jobs by deadline so that d1 ≤ d2 ≤ … ≤ dn
t = 0
for j = 1 to n
   Assign job j to interval [t, t + tj]
   sj = t
   fj = t + tj
   t = t + tj
output intervals [sj, fj]
```

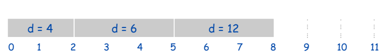


 What can we say about this algorithm/its results?

Minimizing Lateness: No Idle Time

Observation. There exists an optimal schedule with no idle time





Observation. The greedy schedule has no idle time

Proving Optimality

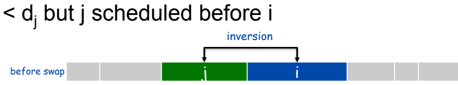
Goal: Prove greedy algorithm produces optimal solution

Approach: **Exchange argument**

- Start with an optimal schedule Opt
- Gradually modify Opt
  - –Preserving its optimality
- Transform into a schedule identical to greedy's schedule

Minimizing Lateness: Inversions

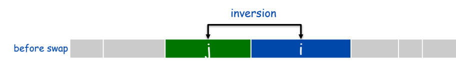
Def. An inversion in schedule S is a pair of jobs i and j such that:
$d_i < d_j$ but j scheduled before i



Can Greedy's solution have any inversions?

Minimizing Lateness: Inversions

Def. An inversion in schedule S is a pair of jobs i and j such that:
$d_i < d_j$ but j scheduled before i



Observation. Greedy schedule has no inversions

Minimizing Lateness: Inversions

Claim. Swapping two adjacent jobs with the same deadline does not increase the max lateness

Pf Sketch. Let ℓ be the lateness before the swap, and let ℓ' be it afterwards

- Lateness of other jobs?
- Lateness of i? j?