CSE 421: Algorithms

Winter 2014

Lecture 8: Greedy algorithms II

Reading: Sections 4.1-4.5

Don't Forget:

Industry Affiliates Recruiting Fairs

are next Tuesday (1/28) and Wednesday (1/29)!

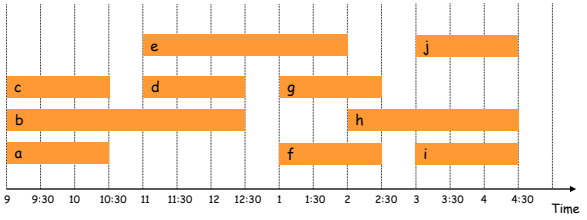
Please pick up your name badge & lanyard in advance.

Where: Main Office
When: Friday or Monday, 10:30-11:30 am or 1:30-3 pm
or Tuesday morning, 10:30-11:30 am.

Over 60 companies will be represented, including:

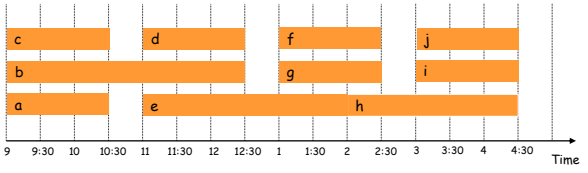
interval partitioning

- 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.
- Example: This schedule uses 4 classrooms to schedule 10 lectures.



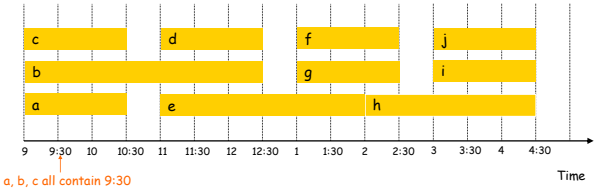
interval partitioning

- 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.
- Example: This schedule uses only 3 classrooms.



lower bound on optimal solutions

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



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

a simple greedy algorithm

Sort requests in increasing order of start times
$(s_1,f_1),\ldots,(s_n,f_n)$

For $i=1$ to n
  $j \leftarrow 1$
  While (request i not scheduled)
    $last_j \leftarrow$ finish time of the last request currently scheduled on resource j
    if $s_i \geq last_j$ then schedule request i on resource j
    $j \leftarrow j+1$
  End While
End For

interval partitioning: greedy analysis

- Observation. Greedy algorithm never schedules two incompatible lectures in the same classroom.

interval partitioning: greedy analysis

- Observation. Greedy algorithm never schedules two incompatible lectures in the same classroom.
- Theorem. Greedy algorithm is optimal.
- Proof.
  - 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 $s_j + \varepsilon$.
  - Key observation ⇒ all schedules use ≥ d classrooms. ▪

a simple greedy algorithm

Sort requests in increasing order of start times
$(s_1,f_1),\ldots,(s_n,f_n)$ **O(n log n)** time

For $i=1$ to n
  $j \leftarrow 1$
  While (request i not scheduled)
    $last_j \leftarrow$ finish time of the last request
        currently scheduled on resource j
    if $s_i \geq last_j$ then schedule request i on resource j
    $j \leftarrow j+1$
  End While
End For

May be slow: **O(nd)**
which could be $\Omega(n^2)$

a simple greedy algorithm

Sort requests in increasing order of start times
$(s_1,f_1),\ldots,(s_n,f_n)$ **O(n log n)** time

For $i=1$ to n
  $j \leftarrow 1$
  While (request i not scheduled)
    $last_j \leftarrow$ finish time of the last request
        currently scheduled on resource j
    if $s_i \geq last_j$ then schedule request i on resource j
    $j \leftarrow j+1$
  End While
End For

a more efficient implementation

Sort requests in increasing order of start times $(s_1,f_1),\ldots,(s_n,f_n)$ **O(n log n)**

$d \leftarrow 1$
Schedule request 1 on resource 1
$last_1 \leftarrow f_1$
Insert 1 into priority queue Q with key = $last_1$
For $i=2$ to n
  $j \leftarrow$findmin(Q)
  if $s_i \geq last_j$ then
    schedule request i on resource j
    $last_j \leftarrow f_i$
    Increasekey(j,Q) to $last_j$
  else
    $d \leftarrow d+1$
    schedule request i on resource d
    $last_d \leftarrow f_i$
    Insert d into priority queue Q with key = $last_d$
End For

O(n log d)

O(n log n) time

greedy analysis strategies

- **Greedy algorithm stays ahead.** Show that after each step of the greedy algorithm, its solution is at least as good as any other algorithm's.
- **Exchange argument.** Gradually transform any solution to the one found by the greedy algorithm without hurting its quality.
- **Structural.** Discover a simple "structural" bound asserting that every possible solution must have a certain value. Then show that your algorithm always achieves this bound.

scheduling to minimize lateness

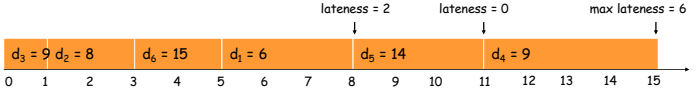
Scheduling to minimize lateness

- Single resource as in interval scheduling but instead of start and finish times request l has
  - Time requirement t_l which must be scheduled in a contiguous block
  - Target deadline d_l by which time the request would like to be finished
  - Overall start time s
- Requests are scheduled by the algorithm into time intervals $[s_l, f_l]$ such that $t_l = f_l - s_l$
- Lateness of schedule for request l is
  - If $d_i < f_i$ then request i is late by $L_i = f_i - d_i$ otherwise its lateness $L_i = 0$
- Maximum lateness $L = \max_l L_l$
- **Goal:** Find a schedule for **all** requests (values of s_l and f_l for each request l) to minimize the maximum lateness, L

scheduling to minimize lateness

Example:

	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.
  - [Shortest processing time first] Consider jobs in ascending order of processing time t_j.
  - [Earliest deadline first] Consider jobs in ascending order of deadline d_j.
  - [Smallest slack] Consider jobs in ascending order of slack $d_j - t_j$.

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

counterexample

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

counterexample

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

	1	2
t_j	1	10
d_j	2	10

counterexample

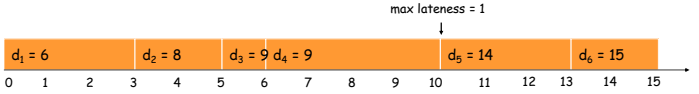
earliest deadline

- Order requests in increasing order of deadlines
- Schedule the request with the earliest deadline as soon as the resource becomes available

minimizing lateness: greedy algorithm

- Greedy algorithm. Earliest deadline first.

Sort deadlines in increasing order $(d_1 \le d_2 \le \ldots \le d_n)$
$f \leftarrow s$
for $i \leftarrow 1$ to n to
  $s_i \leftarrow f$
  $f_i \leftarrow s_i + t_i$
  $f \leftarrow f_i$
end for



proof for greedy algorithm: exchange argument

We will show that if there is another schedule **O** (think optimal schedule) then we can gradually change **O** so that
- at each step the maximum lateness in **O** never gets worse
- it eventually becomes the same cost as **A**

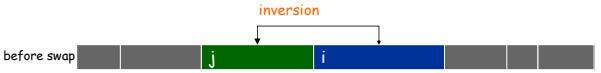
minimizing lateness: no idle time

- Observation. There exists an optimal schedule with no idle time.

- Observation. The greedy schedule has no idle time.

minimizing lateness: inversions

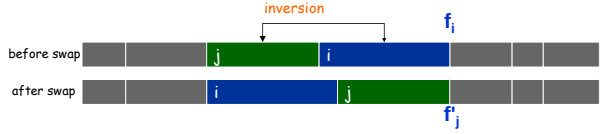
- Definition. An inversion in schedule S is a pair of jobs i and j such that $d_i < d_j$ but j scheduled before i.



- Observation. Greedy schedule has no inversions.
- Observation. If a schedule (with no idle time) has an inversion, it has one with a pair of inverted jobs scheduled consecutively (by transitivity of <).

minimizing lateness: inversions

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



- Claim. Swapping two adjacent, inverted jobs reduces the number of inversions by one and does not increase the max lateness.

minimizing lateness: inversions

If $d_j > d_i$ but j is scheduled in O immediately before i then swapping requests i and j to get schedule O' does not increase the maximum lateness

- Lateness $L_i' \leq L_i$ since i is scheduled earlier in O' than in O
- Requests i and j together occupy the same total time slot in both schedules

  All other requests $k \neq i,j$ have $L_k' = L_k$

  $f_j' = f_i$ so $L_j' = f'_j - d_j = f_i - d_j < f_i - d_i = L_i$
- Maximum lateness has not increased!

optimal schedules and inversions

- Claim: There is an optimal schedule with no idle time and no inversions
- Proof:
  - By previous argument there is an optimal schedule **O** with no idle time
  - If **O** has an inversion then it has a **consecutive** pair of requests in its schedule that are inverted and can be swapped without increasing lateness

optimal schedules and inversions

- Eventually these swaps will produce an optimal schedule with no inversions
  - Each swap decreases the number of inversions by **1**
  - There are a bounded number of (at most **n(n-1)/2**) inversions (we only care that this is finite.)

QED

idleness and inversions are the only issue

- Claim: All schedules with no inversions and no idle time have the same maximum lateness.
- Proof
  - Schedules can differ only in how they order requests with equal deadlines
  - Consider all requests having some common deadline **d**
  - Maximum lateness of these jobs is based only on the finish time of the last of these jobs but the set of these requests occupies the same time segment in both schedules

    Last of these requests finishes at the same time in any such schedule.

earliest deadline first is optimal

- We know that
  - There is an optimal schedule with no idle time or inversions
  - All schedules with no idle time or inversions have the same maximum lateness
  - EDF produces a schedule with no idle time or inversions
- Therefore
  - EDF produces an optimal schedule