Lecture 4 : Greedy Algorithms

Precise def. is difficult. A greedy algorithm makes "locally" optimal choices according to some cost or score to try to construct a global solution. When this leads to an optimal / near-optimal solution, that is "interesting".

Problem: Interval Scheduling (Sec 4.1)

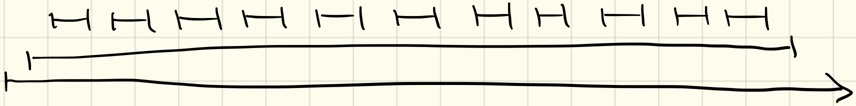
Given: A set R of n requests $\{1, 2, \ldots, n\}$, where the i^{th} request is an interval of time from s_i to f_i.

Find: The maximum cardinality subset of compatible intervals

Note: intervals i and j are compatible if $\underbrace{f_j \le s_i \text{ or } f_i \le s_j}_{\text{the intervals don't overlap in time}}$.

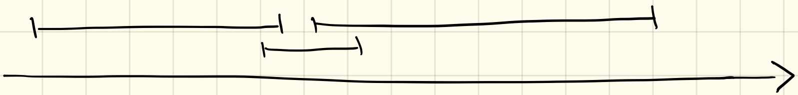
What are some "natural" greedy approaches to this problem?

(a) Earliest Interval : Greedily choose the interval that "starts next"
will it work?



NO

(b) Shortest Interval : Choose the shortest available interval next



NO

(c) Fewest Conflicts : Choose the available interval with smallest # of conflicts

No

(d) Earliest Finish : Choose the next available interval that finishes first

Yes

Alg GIS:

R = all requests, $A = \emptyset$

While $|R| > 0$:

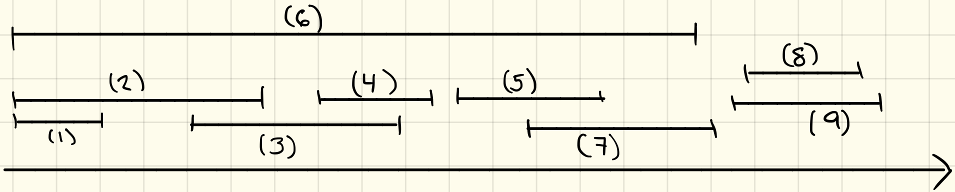
  Select $i \in R$ such that f_i is smallest

  $A = A \cup \{i\}$

  $R = R / (\{i\} \cup \{j \in R \mid j \text{ conflicts with } i\})$

end while

return A



Claim: The set A returned by GIS is compatible. (why?)

How do we show that A is optimal?

- Will show that $|A| = |O|$ for an optimal solution. In particular, will show that the greedy algorithm "keeps up" (does as well as OPT) on every ordered prefix of intervals.

Let $i_1, \ldots, i_k$ be the set of intervals in the order they are added to A. (note $|A| = k$).

Let $j_1, \ldots, j_m$ be the set of intervals in O in left-to-right order ($|O| = m$). Note, the requests in O must be compatible, so order of starting times equals the order of the finish times.

Need to show $k = m$

(4.2) Lemma: For all $r \leq k$, $f_{i,r} \leq f_{j,r}$ ($f_{i,r}$ is finish time in A, $f_{j,r}$ is finish time in O)
(Induction)

Proof:

Base case: $r=1$ is true by def. of the greedy algorithm

Inductive Hypothesis: $f_{i,r-1} \leq f_{j,r-1}$

Inductive Step: We know that $s_{j,r} \geq f_{j,r-1}$ (because O's schedule is compat)

but, by IH, $f_{j,r-1} \geq f_{i,r-1}$ so that $s_{j,r} \geq f_{i,r-1}$. So, when GIS

selects i_r, the interval j_r is in the set R of available intervals.

But, GIS selects the interval with the smallest finish time,

so $f_{i,r} \leq f_{j,r}$. ■

(4.3) Proposition: GIS returns an optimal set A.

(contradiction)

Proof: Assume A is not optimal. Then $(m = |O|) > (k = |A|)$.

By (4.2) with $r = k$, we know that $f_{i,k} \le f_{j,k}$. Since $m > k$ by assumption there must exist some request j_{k+1} in O (otherwise both would be of size k). This request, j_{k+1} starts after j_k ends. But, $f_{i,k} \le f_{j,k}$, so, after removing from R all requests not compatible with $i_1, i_2, \ldots, i_k$ the request j_{k+1} must still exist in R. However, the greedy algorithm terminated with i_k, but should only terminate when R is empty.

$\rightarrow\leftarrow$ (contradiction). $\blacksquare$

This shows that the solution, A, produced by GIS has the same cardinality as some optimal solution O. Since all optimal solutions have the same cardinality, A is optimal.

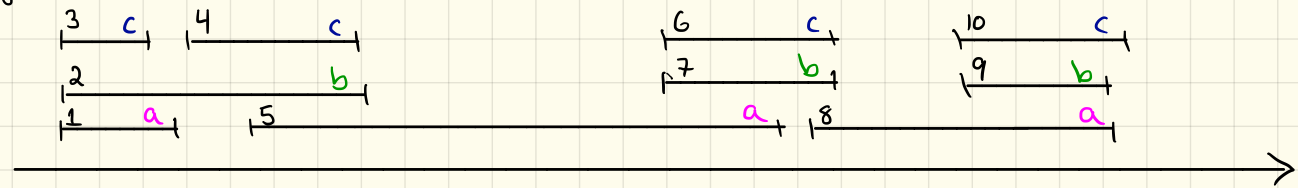
Could there be other optimal solutions?

Problem: Interval Partitioning

Given: A collection $I_1, I_2, \ldots, I_n$ of intervals.

Find: A partitioning of the intervals into the smallest number of compatible sets, such that each set can be satisfied by a single resource.

E.g. Here, the number denotes the interval, and the letter is the resource satisfying it.



Q: What is the absolute minimum # of resources required to schedule all requests?

(can the example above be scheduled with 2 resources? why or why not?)

A: The <u>depth</u> of the interval set is a lower bound on the resource requirement.

(4.4) Lemma: In an instance of Interval Partitioning, the # of resources needed is at least the depth of the set of intervals.

<u>Proof</u>: Suppose R has depth d, and intervals $I_1, I_2, \ldots, I_d$ pass over a common point in time. Each of these intervals must be scheduled on a different resource, so R requires at least d resources.

Alg GIP:

Let $I_1, I_2, \ldots, I_n$ be intervals sorted in order by their <u>start</u> times

For $j = 1, \ldots, n$:

For each I_i that precedes I_j and conflicts with it

Exclude label of I_i from consideration for I_j

If $\exists\, l \in \{1, 2, \ldots, d\}$ that hasn't been excluded

Assign label l to I_j

Else

leave I_j unlabeled

return the labeling.

(4.5) <u>Proposition</u>: GIP will assign a label to <u>every</u> interval <u>and</u> no two conflicting intervals will receive the same label.

<u>Proof</u>: (1) All intervals are labeled. Consider I_j, and assume (wlog) that t other preceding intervals overlap it. These intervals, along with I_j, form a set of $t+1$ intervals that pass over a common point in time.

Thus, $t+1 \leq d \Rightarrow t \leq d-1$.

So, one of the d labels is not excluded by the t intervals and is available to label I_j.

(2) No overlapping intervals share a label. Let I, I' be 2 overlapping intervals with $I \leq I'$ (smaller start time). When GIP considers I', I's label is <u>excluded</u> from consideration, so GIP will <u>not</u> assign I's label to I'.

Since this algorithm labels all intervals in a conflict-free way using d (the minimum possible # of labels), it produces an optimal solution.

Problem: Minimum Lateness Scheduling (aka Job Scheduling) [Ch 4.2]

Given: A set of request/length/deadline tuples (i, t_i, d_i).

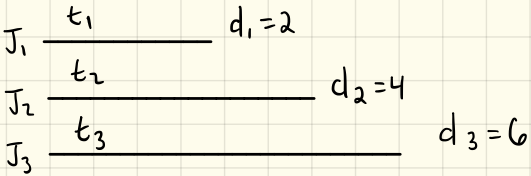
Find: A schedule for these requests (using a single resource), that <u>minimizes</u> the <u>maximum</u> lateness of any request.

The <u>lateness</u>, l_i of request i is $l_i = f_i - d_i$ if $f_i > d_i$ and 0 otherwise

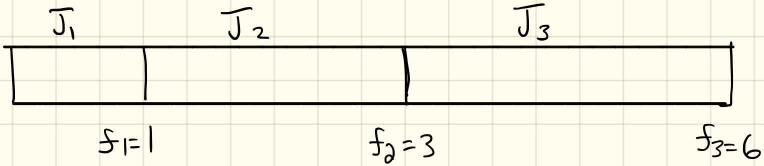
$$l_i = \begin{cases} f_i - d_i & \text{if } f_i > d_i \\ 0 & \text{otherwise} \end{cases}$$

Call $L(S)$ the maximum lateness of schedule S, where $L(S) = \max_i l_i$

E.g.



Solution



How to maximize our objective?

- Schedule jobs in order of t_i? No
  - ignores deadlines, consider $(t_1=1, d_1=100)$, $(t_2=10, d_2=10)$

- Earliest Deadline First (EDF) rule (note: this ignores length).
  - ensure that jobs with earliest deadlines are scheduled first.

Alg EDF:

Order jobs by their deadlines and assume we have $d_1 \le d_2 \le \dots \le d_n$

$S = S_1 = 0$

$f = S$

For $i = 1, 2, \dots, n$

$s_i = f$ ← start next job asap

$f_i = f + t_i$ ← it runs for t_i time

$f = f + t_i$ ← all i jobs finish at f

return the schedule $[(s_1, f_1), (s_2, f_2), \dots, (s_n, f_n)] = A$

We want to show that EDF is optimal - no other schedule could have a lesser maximum lateness

We will show this using an exchange argument. Starting with some optimal solution O, and modifying it into A, showing that these modifications do not increase the max lateness.

Key terms:

- gap or idle time: time between the finishing of job i and the start of job $i+1$. The time during which our resource is idle.

- inversion: A schedule A' has an inversion if job i with deadline d_i is scheduled before job j with deadline $d_j < d_i$ (j has an earlier deadline).

(4.7) Lemma: There is an optimal schedule with no idle time.

Proof: This is obvious, as we could eliminate idle time without increasing L.

(4.8) Lemma: There is an optimal O with no inversions or idle time.

Proof: No idle time (4.7). What about inversions?

(a) If O has an inversion, then there is a pair of jobs i, j such that j is immediately after i and $d_j < d_i$ (why?)

- Suppose O has at least 1 inversion, and let i, j be the adjacent pair of jobs that are inverted. Swapping i and j eliminates this inversion without creating a new inversion, so

(b) Swapping i and j produces a schedule with 1 fewer inversion

(c) This new swapped schedule has a maximum lateness no larger than O

- assume that in O each request r is scheduled from s_r to f_r and has lateness l'_r. Let $L' = \max_r l'_r$.

- Let $\overline{O}$ be the swapped schedule with $\overline{s}_r, \overline{f}_r, \overline{l}_r$ and $\overline{L}$ defined similarly.

- Consider the adjacent inverted jobs i, j

- Then f_j <u>before</u> the swap is the same as f_i after the swap because $t_i + t_j = t_j + t_i$ and the overall start time of the pair of jobs is the same in both schedules. As a result, all jobs other than i, j finish at identical times in the two schedules.

- Job j finishes earlier in $\bar{O}$, so swap can't increase lateness of j

- Job i may finish later in $\bar{O}$, but this cannot increase the <u>overall</u> lateness. Why?

    - If job i is late in $\bar{O}$, the lateness is $\bar{l}_i = \bar{f}_i - d_i = f_j - d_i$

    But, because $d_i > d_j$ (by def of inversion), job i cannot be later in $\bar{O}$ than j was in O. Specifically,

$$\left(\bar{l}_i = f_j - d_i\right) < \left(f_j - d_j = l'_j\right)$$

Since the lateness of O was $L' \geq l'_j > \bar{l}_i$, the swap <u>can not</u> increase the maximum lateness. ■

Note: Since the initial optimal schedule can have at most $\binom{n}{2}$ inversions, we can <u>transform</u> this into a schedule with no inversions with at most $\binom{n}{2}$ swaps. By (c), this can be done without increasing the maximum lateness.

(4.10) Proposition: The schedule A produced by EDF has optimal maximum lateness L.

<u>Proof</u>: An optimal schedule with no inversions exists (4.9), but all schedules with no inversions have the same maximum lateness (4.8).

So, the maximum lateness of schedule A with no inversions must not be greater than this <u>optimal</u> maximum lateness.

$\blacksquare$