Math 416

Worksheet 17. Greedy scheduling

Roughly: the idea of a greedy algorithm is this ...

- Try to solve the problem *locally*: at each stage you choose the best possible move at that moment. There will typically be an **objective function** that you attempt to optimize at each stage.
- At the end hopefully you arrive at an optimal or near-optimal solution.
- When greedy algorithms work, they are typically very efficient. (The problem is that they don't always work.)

Interval-scheduling You are given n tasks, each with a start time $s(i)$ and a (later) finish time $f(i)$. Your goal is to schedule as many tasks as possible that do not overlap.

We want to design a greedy algorithm for interval scheduling. We need to find an objective function to optimize at each stage.

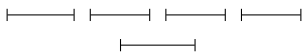
Problem 1. Here are three attempts at an objective function for a greedy algorithm. Do they work? (*Hint:* No. Explain why not.)

(i) Select the available request with earliest start time $s(i)$ so that the resource starts being used as quickly as possible.

(ii) Select the available request with the shortest duration, i.e., with $f(i) - s(i)$ minimal.

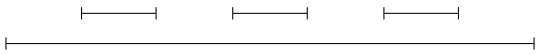
(iii) Select the available request with the fewest conflicts, i.e., the request i that is incompatible with the smallest number of requests j.

(*Hint:* You might find it helpful to add some intervals to the following picture.)

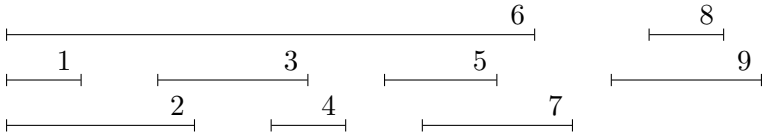


Problem 2. Come up with a greedy solution that does work.

(*Hint:* Maybe the example below will lead you in the right direction.)



Problem 3. Execute your greedy algorithm on the following example.



Problem 4. Invent a few more examples, and run your greedy algorithm on those. Try to get a feel for why it works before proceeding to the formal proof.

It is clear that this algorithm always returns a set of nonoverlapping intervals; but we need to show that it always returns an optimal solution, i.e., a set of nonoverlapping intervals of maximum size.

The greedy algorithm stays ahead Let O be an optimal solution (there may be more than one!) and let A be the solution given by our greedy algorithm. We must show that $|O| = |A|$. Say $A = \{i_1, i_2, \ldots, i_k\}$ ordered left to right, and $O = \{j_1, j_2, \ldots, j_m\}$ ordered left to right. (We mean $s(i_1) < f(i_1) < s(i_2) < f(i_2) < \cdots$ and similar for the j_ks.)

We prove that 'the greedy algorithm stays ahead':

Lemma. $f(i_r) \leq f(j_r)$ for all $r \leq k$.

Exercise (optional, if you want to get a better feel for the proof)**.** Return to the example from Problem 3. Label the greedy solution i_1, i_2, etc. Find a solution other than the optimal solution and label it j_1, j_2, etc.

Problem 5. Do the following to complete a proof of the lemma by induction on r.

(a) Why is the lemma true for $r = 1$?
(b) Assume that $f(i_{r-1}) \leq f(j_{r-1})$. The goal is to prove that $f(i_r) \leq f(j_r)$. We can assume that $i_r \neq j_r$ because _____
(c) Prove that the interval j_r does not overlap with the interval i_{r-1}.
(d) Explain why the previous part implies that $f(i_r) \leq f(j_r)$, completing the induction. □

Corollary. $|A| = |O|$ so A is optimal.

Problem 6. Prove the Corollary from the Lemma as follows.

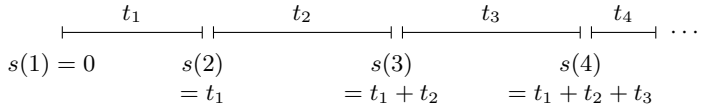
(a) Why do we know that $m \geq k$?
(b) Assume toward a contradiction that $m > k$. By the Lemma, $f(i_k) \leq f(j_k)$. Arrive at a contradiction by considering task j_{k+1}. This proves the Corollary.

Problem 7 (running time)**.** By initially sorting the n intervals by $f(i)$, this algorithm can be implemented to run in $\boxed{}$ time. Explain.

Interval scheduling with deadlines We are given n requests, each of the form (t, d), where t is a length of time and d is a deadline. We want to produce a schedule for the requests that minimizes the maximum lateness. In particular, if it's possible to schedule the jobs with no lateness, then our solution should do that.

- A *schedule* is a sequence of start times $s(1), \ldots, s(n)$ for the n requests (t_i, d_i), so that $s(i+1) \geq s(i) + t_i$. Then request i will use the resource during the interval $[s(i), f(i)]$, where $f(i) = s(i) + t_i$.

  Here's a schedule where we just schedule the jobs one after another in the order they're given to us:



- The *lateness* of a request is $l_i = \max(0, f(i) - d_i)$. We want to minimize the maximum lateness $\max_i l_i$.

Problem 8 (false starts)**.** The following attempts at an objective function don't work. Explain why by giving an example.

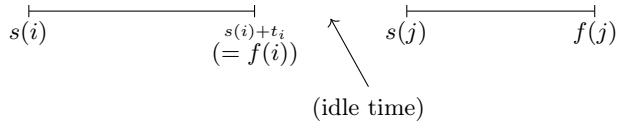
(a) Execute the jobs in increasing order of t_i to get short tasks out of the way first.
(b) Execute the jobs in increasing order of 'slack time' $d_i - t_i$.

Problem 9. Look again at your examples from Problem 8. Show that, in each of them, sorting the tasks by deadline (*earliest deadline first*) gives the optimal solution.

Even though it ignores half of the input data (the time required), this solution works. To prove that it works we use an *exchange argument*: we consider an optimal solution O, and we modify O to look like our solution A, preserving optimality at each stage.

Definition.

- An *inversion* in a schedule is a pair of jobs i, j where job j is scheduled before job i even though $d_i < d_j$.
- A schedule has *idle time* if there are consecutive tasks i and j in the schedule for which $f(i) < s(j)$:



Suppose the tasks are ordered by deadline: $d_1 \leq d_2 \leq \cdots \leq d_n$. Let A be the solution produced by our earliest-deadline-first algorithm. Our solution has no idle time and no inversions.

Claim 1. All schedules with no inversions and no idle time have the same maximum lateness.

Problem 10. Prove Claim 1 as follows.

(a) Two schedules with no inversions and no idle time can differ only ... (how?)
(b) Prove that if $(t_1, d), \ldots, (t_k, d)$ all have the same deadline, then reordering any of these k jobs does not change the max lateness.
(c) Explain why the Claim follows.

Claim 2. There is an optimal schedule with no inversions and no idle time.

Problem 11. Prove Claim 2 as follows.

(a) Explain why there is an optimal schedule with no idle time.
(b) Suppose that O has an inversion. Explain why we can find jobs i and j for which $d_j < d_i$ and job j is the next job in O scheduled after i.
(c) After swapping jobs i and j (from the previous part, we get a schedule O^* with one less inversion. Prove that the max lateness of O^* is $\leq$ (hence $=$) the max lateness of O.
(*Hint:* Job i is no later in O^* than job j was in O.)
(d) Explain how the Claim follows.

Problem 12. Explain how Claims 1 and 2 combine to prove the correctness of the earliest-deadline-first algorithm.

Problem 13 (bonus)**.** Consider the following variation of the Interval Scheduling Problem. You have a processor that can operate 24 hours a day, every day. People submit requests to run daily jobs on the processor. Each such job comes with a *start time* and an *end time*; if the job is accepted to run on the processor, then it must run continuously, every day, for the period between its start and end times. (Note that certain jobs can begin before midnight and end after midnight; this makes for a type of situation different from what we saw in the Interval Scheduling Problem.)

Given a list of n such jobs, your goal is to accept as many jobs as possible (regardless of their length), subject to the constraint that the processor can run at most one job at any given point in time. Find a greedy solution to this problem. How efficient is your algorithm?

(Example: Consider the following four jobs, specified by (start time, end time).

$$(6\ \text{PM}, 6\ \text{AM}), (9\ \text{PM}, 4\ \text{AM}), (3\ \text{AM}, 2\ \text{PM}), (1\ \text{PM}, 7\ \text{PM}).$$

The optimal solution would be to pick the two jobs (9 PM, 4 AM) and (1 PM, 7 PM).)