

## Greedy II: Interval Scheduling

Suppose you are given  $n$  jobs to schedule on a machine. Each job  $i$  (where  $i \in \{1, \dots, n\}$ ) has a start time  $s(i)$  and a finish time  $f(i)$ . You would like to schedule *as many* jobs as possible given that the machine can only process one job at a time, and the jobs must run from their start time to finish time uninterrupted to be processed. That is, the machine cannot process two jobs that overlap.

What *greedy* algorithm should you use to schedule the jobs? By what metric is it greedy? (See **Step 2.**) Here are some ideas.

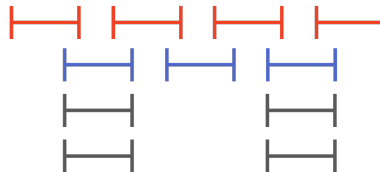
- Shortest jobs. Counterexample, with red optimal solution and blue greedy solution:



- Earliest start time. Counterexample, with red optimal solution and blue greedy solution:



- Fewest conflicts. Counterexample, with red optimal solution and blue greedy solution:



- Earliest finish time. This is the correct metric that we will prove is optimal.

Prove that your algorithm is optimal by a **Greedy-Stays-Ahead** proof.

**Step 1: Define your solutions.** Describe the form your greedy solution takes, and what form some other solution takes (possibly the optimal solution). For example, let  $A$  be the solution constructed by the greedy algorithm, and let  $O$  be a (possibly optimal) solution.

Let  $A = \{i_1, \dots, i_k\}$  be the set of requests selected by our greedy algorithm, in the order in which they were added. Let  $O = \{j_1, \dots, j_m\}$  be the requests selected by an optimal solution, ordered by their finish times.

**Step 2: Find a measure.** Find a *measure* by which greedy stays ahead of the other solution you chose to compare with. Let  $a_1, \dots, a_k$  be the first  $k$  measures of the greedy algorithm, and let  $o_1, \dots, o_m$  be the first  $m$  measures of the other solution ( $m = k$  sometimes).

We will compare  $A$  and  $O$  by their jobs' finish times, that is, we define the measures  $a_r = f(i_r)$  and  $o_r = f(j_r)$  for all  $r \leq k$ , and we show that for all  $r \leq k$ ,  $a_r \leq o_r$  (i.e. that  $f(i_r) \leq f(j_r)$ ). This can be shown by induction on  $r$ .

**Step 3: Prove greedy stays ahead.** Show that the partial solutions constructed by greedy are always just as good as the initial segments of your other solution, based on the measure you selected.

- For all indices  $r \leq \min(k, m)$ , prove (often by induction) that  $a_r \geq o_r$  or that  $a_r \leq o_r$ , whichever the case may be. Don't forget to use your algorithm to help you argue the inductive step.

Formally, for all  $r \leq k$ , we will prove the claim that  $a_r \leq o_r$  by induction. (If you want, you can call this  $P(r)$ , the property for  $r$  that we aim to prove.) We want to show that this ( $P(r)$ ) is true for all  $1 \leq r \leq k$ .

*Base Case* ( $r = 1$ ): Since the algorithm selects the job with the earliest finish time, it must be the case that  $a_1 = f(i_1) \leq f(j_1) = o_1$ .

*Inductive Hypothesis*: Suppose that the claim holds for some fixed  $r - 1$  with  $r > 1$ , that is, that  $a_{r-1} = f(i_{r-1}) \leq f(j_{r-1}) = o_{r-1}$ .

*Inductive Step* ( $r =$ ): Now we prove that  $P(r)$  is true using the IH that  $P(r - 1)$  is true. That is, we prove that  $a_r \leq o_r$ . Recall that by the inductive hypothesis,  $f(i_{r-1}) \leq f(j_{r-1})$ , and so any jobs that are compatible with the first  $r - 1$  jobs in the optimal solutions are certainly compatible with the first  $r - 1$  jobs of our greedy solution. Therefore, we could add  $j_r$  to our greedy solution, and since we take the compatible job with the smallest finish time, it must be the case that  $f(i_r) \leq f(j_r)$ , that is, that  $a_r \leq o_r$ , as desired.

Thus we have shown that for all  $r \leq k$ ,  $f(i_r) \leq f(j_r)$ .

**Step 4: Prove optimality.** Prove that since greedy stays ahead of the other solution with respect to the measure you selected, then it is optimal.

Our inductive claim implies that, in particular,  $f(i_k) \leq f(j_k)$ . If  $A$  is not optimal, then it must be the case that  $m > k$ , and so there is a job  $j_{k+1}$  in  $O$  that is not in  $A$ . This job must start after  $O$ 's  $k^{th}$  job finishes at time  $f(j_k)$ , and hence after  $f(i_k)$ . But then this job is compatible with all the jobs in  $A$ , and so  $A$  would have added it during the greedy algorithm. This is a contradiction, and thus  $A$  has as many elements as  $O$ .

**Step 5: Analyze runtime.** This is always our last step.

The algorithm begins by sorting the  $n$  requests in order of finishing time, which takes time  $O(n \log n)$ . Each time we select an interval, we proceed past any incompatible intervals in our list; that is, we proceed through our list exactly once. This part of the algorithm takes time  $O(n)$ ; therefore, the total running time is  $O(n \log n)$ .