

## Homework 1 Solutions

1. (a) **Problem:** You are given a set of  $n$  jobs. Job  $i$  starts at a fixed time  $s_i$ , ends at a fixed time  $e_i$ , and results in a profit  $q_i$ . Only one job may run at a time. The goal is to choose a subset of the available jobs to maximize total profit.

**Algorithm:** First, sort the jobs so that  $q_1 \geq q_2 \geq \dots \geq q_n$ . Then, try to add each job to the schedule in turn. If job  $i$  does not conflict with any jobs schedule so far, add it to the schedule; otherwise, discard it

**This algorithm is not optimal.**

Consider three jobs with starts  $\{1, 1, 3\}$ , ends  $\{4, 2, 4\}$ , and profits  $\{3, 2, 2\}$ . The given algorithm schedules the first job from time 1 to time 4, at a profit of 3, and must discard the other two jobs. The optimal solution schedules the other two jobs, for a profit of 4.

- (b) **Problem:** You are given a set of  $n$  jobs, each of which runs in unit time. Job  $i$  has an integer-valued deadline time  $d_i \geq 0$  and a real-valued penalty  $p_i \geq 0$ . Jobs may be scheduled to start at any integer time  $(0, 1, 2, \text{etc})$ , and only one job may run at a time. If job  $i$  completes at or before time  $d_i$ , then it incurs no penalty; otherwise, it incurs penalty  $p_i$ . The goal is to schedule all jobs so as to minimize the total penalty incurred.

**Algorithm:** Define slot  $k$  in the schedule to run from time  $k - 1$  to time  $k$ . First, sort the jobs so that  $p_1 \geq p_2 \geq \dots \geq p_n$ . Then, add each job to the schedule in turn. When adding job  $i$ , if any time slot between 1 and  $d_i$  is available, then schedule  $i$  in the latest such slot. Otherwise, schedule job  $i$  in the latest available slot  $\leq n$ .

**The given algorithm is indeed optimal.**

First, observe that, without loss of generality, we need consider only schedules that use all slots  $[0, 1, \dots, n - 1]$ . Any schedule without this property can be changed by moving jobs back in time to fill any empty slots, which can only increase the number of jobs that meet their deadlines.

Second, we will consider the slightly more general problem in which the input is a list  $J$  of jobs and a list  $T$  of blocked slots. Formulating the general problem on  $J \times T$  will make our substructure argument easier.

**Greedy Choice Property:** Let  $J \times T$  be any problem instance. Let  $j_1$  be the first job scheduled, and let  $t_1$  be the slot in which it is scheduled. Then there exists an optimal solution to  $J \times T$  that schedules  $j_1$  at time  $t_1$ .

**Pf:** Let  $S^*$  be any optimal solution. If  $S^*$  schedules  $j_1$  at time  $t_1$ , we are done. Otherwise,  $S^*$  puts  $j_1$  at some time  $t \neq t_1$  and schedules some other job  $j_k \neq j_1$  at time  $t_1$ .

Suppose we create a new solution  $S$  from  $S^*$  by moving the jobs  $j$  and  $j_1$  to each others' slots.  $S$  makes the greedy choice, but we must show that it is still optimal. We proceed by cases:

- i. Suppose  $t_1 \leq d_1$ , and that  $j_1$  incurs no penalty in  $S^*$ . Then  $t < t_1$ , since  $j_1$  cannot be scheduled without penalty in any slot  $> t_1$ . Conclude that exchanging  $j_k$  with  $j_1$

leaves  $j_1$  without penalty and moves  $j_k$  to an earlier slot, which leaves its penalty the same or less than before. Hence,  $S$ 's penalty is no greater than that of  $S^*$ .

- ii. Suppose  $t_1 \leq d_1$ , and that  $j_1$  incurs a penalty in  $S^*$ . Then  $t > t_1$ , and the exchange moves  $j$  forward. The total penalty of  $S$  is therefore at most that of  $S^*$  plus  $p_k - p_1$ , since  $j_k$  may miss its deadline in  $S$ , but  $j_1$  does not. By the greedy choice,  $p_1 \geq p_k$ , so  $S$ 's penalty remains at most that of  $S^*$ .
- iii. Suppose  $t_1 > d_1$ . Then there is *no* free slot in which to schedule  $j_1$  without penalty, and so  $j_1$  incurs a penalty in every solution to  $J \times T$ . Moreover, by the greedy choice,  $t_1$  is the latest free slot, so  $t < t_1$ . Hence, exchanging  $j_k$  with  $j_1$  does not change the penalty for  $j_1$  and may eliminate the penalty for  $j_k$ . Hence,  $S$ 's penalty remains at most that of  $S^*$ .

Conclude that  $S$  is always at least as good as  $S^*$  and so is optimal. QED

**Substructure 1:** After making the greedy choice, we are left with a smaller instance of the same problem.

**Pf:** Let  $J' = J - \{j_1\}$ , and let  $T' = T \cup \{t_1\}$ . Then  $J' \times T'$  is a smaller instance of the original problem. QED

**Substructure 2:** Let  $S'$  be an optimal solution to the subproblem  $J' \times T'$ . Then the solution  $S$  that adds  $j_1$  at time  $t_1$  to  $S'$  optimally solves the original problem  $J \times T$ .

**Pf:** First, since  $T'$  blocks slot  $t_1$ , we can add  $j_1$  at time  $t_1$  to  $S'$ . Second, observe that

$$\text{penalty}(S) = \text{penalty}(S') + p(j_1, t_1),$$

where  $p(j_1, t_1)$  is the penalty incurred by placing  $j_1$  at  $t_1$ . The usual contradiction argument completes the proof. QED

2. *You are an alien spy, worried that the world's radio telescopes will discover your home planet. After decades of undercover work as a PhD student in Physics, you've become the scheduler for the biggest world radio telescope. Your home planet will be particularly susceptible to being noticed during a time interval  $[S, F]$ . During this time, there are a set of research projects that have been proposed:  $A = \{a_1 \dots a_n\}$ , which can justify using the radio telescope from an interval  $[s_i, f_i]$ . (These intervals cannot be lengthened or shortened and may overlap).*

*You're goal is to avoid suspicion, but minimize the chance that your home planet will be discovered. That is, you must create an efficient algorithm to select the **fewest** possible jobs from  $A$  such that at least one job is running at all times (otherwise, the schedule will look suspicious).*

Our greedy algorithm is as follows: beginning at time  $S$ , repeatedly select the research project that overlaps the current time, and whose interval extends furthest in time toward  $F$  (i.e. the  $a_i$  with maximum  $f_i$ ).

Here is some pseudocode that takes a set  $A$  of projects  $a_i$  and an interval  $[S, F]$  and selects projects according to the above rule. It will break if no project is available to cover some time between  $S$  and  $F$ , but we'll ignore that case here.

```
SCHEDULE( $A, [S, F]$ )
  sort  $A$  in nondecreasing order of  $s_i$ 
```

```

 $T \leftarrow \emptyset$  // solution set
 $i \leftarrow 1$ 
 $e \leftarrow S$ 
while  $e < F$ 
     $f_i^* = S$ 
    while  $s_i < e$ 
        if  $f_i > f_i^*$ 
             $f_i^* \leftarrow f_i$ 
             $i^* \leftarrow i$ 
         $i ++$ 
     $T \leftarrow T \cup \{a_{i^*}\}$ 
     $e \leftarrow f_i^*$ 
return  $T$ 

```

The code looks at each project exactly once. For each successive end  $e$  of the covered part of  $[S, F)$ , it considers all projects who start before  $e$  and chooses the one that ends furthest to the right. That project is added to the solution, and the endpoint  $e$  is updated. The sitters not chosen all end at time  $\leq e$ , so they may subsequently be ignored.

The code spends  $\Theta(1)$  time per project, so the main loop is  $\Theta(n)$ . The dominant cost of the algorithm is therefore the sort, making it  $\Theta(n \log n)$  overall.

**Greedy Choice Property:** Let  $A, [S, F)$  be an instance of the problem, and let  $a_i$  be a sitter with  $s_i \leq S$ , such that  $f_i$  is maximal for  $a_i$  among all such projects. Then there exists an optimal solution containing  $a_i$ .

**Pf:** Let  $T^*$  be any optimal solution to  $A, [S, F)$ . If  $a_i \in T^*$ , we are done. Otherwise, let  $a_j$  be the sitter in  $T^*$  with minimal  $f_j$ ;  $a_j$  must have  $s_j \leq S$ . Construct a new solution  $T$  from  $T^*$  by discarding  $a_j$  and adding  $a_i$ . Because  $f_i \geq f_j$  and  $s_i \leq S$ , the new solution  $T$  still covers the interval  $[S, f_j)$ . Moreover,  $T$  contains just as many projects as  $T^*$ , so  $T$  is still optimal. QED

**Substructure 1:** after making the greedy choice  $a_i$ , we are left with a subproblem  $A', [f_i, F)$ , where  $A'$  is  $A$  with  $a_i$  and possibly some other  $a_j$ 's removed. This problem has no extra constraints compared to the original. (Note that we don't bother removing *all* useless projects from  $A$ , since the problem does not specify that such projects cannot exist, and the algorithm discards them provided that at least one non-useless project is found.)

**Substructure 2:** Let  $T'$  be an optimal solution to  $A', [f_i, F)$ , and let  $T = T' \cup \{a_i\}$ , where  $a_i$  is the greedy choice. Then  $T$  is optimal for  $A, [S, F)$ .

**Pf:** First, since  $a_i$  runs from  $\leq S$  to  $f_i$ ,  $T$  is a feasible solution, since it spans from  $S$  to  $F$ . Second, observe that  $|T| = |T'| + 1$ . The usual contradiction argument follows. QED

3. “Survivor Anchorage” proposes to take the popular TV show up north. One of the challenges is based on the entire team skiing quickly down a mountainside. Knowing that you aren't the best skier, you need to help your team by defining an efficient algorithm to match skis to skiers. In particular, skiers go fastest with skis whose length is about their height.

Your team consists of  $n$  members, with heights  $h_1, h_2, \dots, h_n$ , and your team gets a delivery of  $n$  pairs of skis, with lengths  $l_1, l_2, \dots, l_n$ . Your goal is to write an algorithm to assign one

pair of skis to each skier to minimize the sum of the absolute differences between the height of the skier  $h_i$  and its length of the corresponding ski  $l_j$ .

Here is an algorithm for this problem. First, sort the ski lengths in increasing order  $\{l_1, l_2, \dots\}$  by size, and sort the known masses in increasing order  $\{h_1, h_2, \dots\}$  by size. Then, assign the  $i$ th measured mass to the  $i$ th known mass. This algorithm is clearly  $\Theta(n \log n)$ , since it takes this much time to sort each list and  $\Theta(n)$  to do the assignment.

**Greedy Choice Property:** Let  $(L, H)$  be an instance of the problem, with  $L$  the lengths and  $H$  the heights. Then there exists an optimal assignment that pairs the length  $l_1$  to the smallest height  $h_1$ .

**Pf:** let  $A^*$  be an optimal assignment of lengths to heights. If  $A^*$  pairs  $l_1$  with  $h_1$ , we are done. Otherwise,  $A^*$  pairs  $l_1$  with some other  $h'$  and  $h_1$  with some other  $l'$ . Note that, by construction,  $h' \geq l_1$  and  $l' \geq h_1$ .

Let  $A$  be the assignment obtained from  $A^*$  by swapping these pairs, so that the greedy choice is made and  $l'$  is paired with  $w'$ . We claim that  $A$ 's cost is at most that of  $A^*$ . Observe that

$$\text{cost}(A) = \text{cost}(A^*) - |l_1 - h'| - |l' - h_1| + |l_1 - h_1| - |l' - h'|.$$

We need to show that the difference  $\text{cost}(A) - \text{cost}(A^*)$  is zero or negative.

There are six possible orderings of the four values  $l_1$ ,  $h_1$ ,  $l'$ , and  $h'$  that are consistent with the inequalities  $l_1 \leq l'$  and  $h_1 \leq h'$ . We will consider three of them; the other three are symmetric (just swap the roles of  $l$  and  $h$ ):

(a) If  $l_1 \leq h_1 \leq l' \leq h'$ , then we have that

$$\begin{aligned} |l_1 - h_1| + |l' - h'| - |l_1 - h'| - |l' - h_1| &= h_1 - l_1 + h' - l' - (h' - l_1) - (h_1 - l') \\ &= 0. \end{aligned}$$

(b) If  $l_1 \leq h_1 \leq h' \leq l'$ , then we have that

$$\begin{aligned} |l_1 - h_1| + |h' - l'| - |l_1 - h'| - |h' - l_1| &= h_1 - l_1 + h' - l' - (h' - l_1) - (h_1 - l') \\ &= 2(h_1 - l') \\ &\leq 0. \end{aligned}$$

(c) If  $l_1 \leq h_1 \leq l' \leq h'$ , then we have that

$$\begin{aligned} |l_1 - h_1| + |l' - h'| - |l_1 - h'| - |l' - h_1| &= h_1 - l_1 + l' - h' - (h' - l_1) - (l' - h_1) \\ &= 2(h_1 - h') \\ &\leq 0. \end{aligned}$$

Conclude that, in every case,  $A$ 's cost is at most that of  $A^*$ , and so  $A$  is also optimal. QED

**Substructure 1:** After making the greedy choice, we are left with a smaller instance of the same problem.

**Pf:** Let  $(L, H)$  be the original problem instance. After making the greedy choice, set  $L' = L - \{l_1\}$  and  $H' = H - \{h_1\}$ . Then we are left with a subproblem  $(L', H')$  of matching  $n - 1$  lengths to  $n - 1$  heights. QED

**Substructure 2:** Suppose  $A'$  is an optimal assignment for subproblem  $(L', H')$ ; then  $A = A' \cup (l_1, h_1)$  is an optimal psolution to the original.

**Pf:** The cost of  $A$  can be decomposed as

$$\text{cost}(A) = \text{cost}(A') + |l_1 - h_1|.$$

Apply the usual contradiction argument. QED

4. You are given  $n$  events where each takes one unit of time. Event  $i$  will provide a profit of  $g_i$  dollars ( $g_i > 0$ ) if started at or before time  $t_i$  where  $t_i$  is an arbitrary real number. (Note: If an event is not started by  $t_i$  then there is no benefit in scheduling it at all. All events can start as early as time 0.) Given the most efficient algorithm you can to find a schedule that maximizes the profit.

We first argue that there always exists an optimal solution in which all of the events start at integral times. Take an optimal solution  $S$  — you can always have the first job in  $S$  start at time 0, the second start at time 1, and so on. Hence, in any optimal solution, event  $i$  will start at or before time  $\lfloor t_i \rfloor$ .

This observation leads to the following greedy algorithm. First, we sort the jobs according to  $\lfloor t_i \rfloor$  (sorted from largest to smallest). Let time  $t$  be the current time being considered (where initially  $t = \lfloor t_1 \rfloor$ ). All jobs  $i$  where  $\lfloor t_i \rfloor = t$  are inserted into a priority queue with the profit  $g_i$  used as the key. An extractMax is performed to select the job to run at time  $t$ . Then  $t$  is decremented and the process is continued. Clearly the time complexity is  $O(n \log n)$ . (The sort takes  $O(n \log n)$  and there are at most  $n$  insert and extractMax operations performed on the priority queue, each which takes  $O(\log n)$  time.

We now prove that this algorithm is correct by showing that the greedy choice and optimal substructure properties hold.

**Greedy Choice Property:** Consider an optimal solution  $S$  in which  $x + 1$  events are scheduled at times  $0, 1, \dots, x$ . Let event  $k$  be the last job run in  $S$ . The greedy schedule will run event 1 last (at time  $\lfloor t_1 \rfloor$ ). From the greedy choice property we know that  $\lfloor t_1 \rfloor \geq \lfloor t_k \rfloor$ . We consider the following cases:

**Case 1:**  $\lfloor t_1 \rfloor = \lfloor t_k \rfloor$ . By our greedy choice, we know that  $g_1 \geq g_k$ . If event 1 is not in  $S$  then we can just replace event  $k$  by event 1. The resulting solution  $S'$  is at least as good as  $S$  since  $g_1 \geq g_k$ . The other possibility is that event 1 is in  $S$  at an earlier time. Since  $\lfloor t_1 \rfloor = \lfloor t_k \rfloor$ , we can switch the times in which they run to create a schedule  $S'$  which has the same profit as  $S$  and is hence optimal.

**Case 2:**  $\lfloor t_k \rfloor < \lfloor t_1 \rfloor$ . In this case,  $S$  does not run any event at time  $\lfloor t_1 \rfloor$  since job  $k$  was its last job. If event 1 is not in  $S$ , then we could add it to  $S$  contradicting the optimality of  $S$ . If event 1 is in  $S$  we can run it instead at time  $\lfloor t_1 \rfloor$  creating a schedule  $S'$  that makes the greedy choice and has the same profit as  $S$  and is hence also optimal.

**Optimal Substructure Property:** Let  $P$  be the original problem of scheduling events  $1, \dots, n$  with an optimal solution  $S$ . Given that event 1 is scheduled first we are left with the subproblem  $P'$  of scheduling events  $2, \dots, n$ . Let  $S'$  be an optimal solution to  $P'$ . Clearly  $\text{profit}(S) = \text{profit}(S') + g_1$  and hence an optimal solution for  $P$  includes within it an optimal solution to  $P'$ .