CSC373— Algorithm Design, Analysis, and Complexity — Spring 2018

Suggested Solutions for Tutorial Exercise 1: Greedy Algorithms

---

1. Interval Scheduling. Make sure you understand the proof of the correctness of the interval scheduling algorithm given on slides 14 through 18 of the lecture notes. There are two questions on the top of p.18 that require some details to be filled in, and we look at those questions here.

First prove that, in case (2b), $J(k+1)$ is compatible with $\{J(j_m)|m = 1, \ldots, r\}$. Recall the notation here is that loop invariant $L(k)$ refers to an optimal solution, $\mathcal{O}$, which has been unpacked as $\mathcal{O} = \{J(j_m) \mid m = 1, \ldots, p\}$, where $p > r$. Hint: A suitable argument is very short but crystal clear.

In addition, prove that $J(k + 1)$ is compatible with $\{J(j_m) \mid m = r + 2, \ldots, p\}$. Hint: Recall jobs $J(k)$ have been sorted by finish time, so $f_1 \leq f_2 \ldots \leq f_n$.

Solution For the first question, note that in case (2b) the algorithm has chosen to add $J(k + 1)$ to $S(k)$, so it must be compatible with all the intervals in $S(k)$. Moreover, recall that $S(k) = \{J(j_m)|m = 1, \ldots, r\}$, so the result follows. ■

For the second question, in order to show $J(k + 1)$ is compatible with the subsequent intervals in $\mathcal{O}$ which come **after** job $J(j_{r+1})$, namely $J(j_m)$ for $m > r + 1$, first recall that the indices j_m of jobs in $\mathcal{O}$ are sorted in increasing order for $m = 1, 2, \ldots, p$. Also recall that the original jobs are sorted by finish time, and therefore the finish times of the jobs in $\mathcal{O}$, namely, f_{j_m}, must also be in non-decreasing order (in terms of m). Since $\mathcal{O}$ is compatible, it must also be the case that the start times satisfy $f_{j_m} \leq s_{j_{m+1}}$ for $m = 1, 2, \ldots, p-1$. That is, the next job in $\mathcal{O}$ must start sometime on or after the previous job finishes. (Otherwise, it would be the case that $f_{j_m} > s_{j_{m+1}}$ and the two intervals would intersect over the open interval $(s_{j_{m+1}}, f_{j_m})$, which contradicts the compatibility of $\mathcal{O}$.)

Currently we are in case (2b) on the lecture slides, so we have $k + 1 < j_{r+1}$ and therefore the corresponding finish times satisfy $f_{k+1} \leq f_{j_{r+1}}$. Using the above relationship between the start and finish times of jobs in $\mathcal{O}$, we find $f_{k+1} \leq f_{j_{r+1}} \leq s_{j_m}$ for all $m > r + 1$. Specifically, this shows that $J(k + 1)$ is compatible with the remaining jobs in $\mathcal{O}$, i.e., $\{J(j_m) \mid m = r + 2, \ldots, p\}$. ■

2. Certificate for Minimum Max-Lateness. Consider the problem of minimizing the maximum lateness for a set of jobs, as discussed on pp.20-24 of the lecture notes. The input data has the form $\{(t_i, d_i)\}_{i=1}^n$. Here we assume t_i and d_i are all strictly positive integers. Consider the following function,

$$T(d_j) \equiv \sum_{\{i \ \mid \ d_i \leq d_j\}} t_i, \tag{1}$$

where "≡" denotes a definition. That is, $T(d_i)$ is simply the total of the execution times of all the jobs that have deadlines no later than d_i. Moreover, define $L^* \equiv \max\{0, \max\{T(d_j) - d_j \mid j = 1, \ldots, n\}\}$. Note that L^* can be easily computed given the data; it does not depend on first finding a suitable schedule.

2a) Let P denote any feasible schedule of the given jobs (that is, P specifies the order all jobs are scheduled and no two jobs overlap in time). Show the maximum lateness of this schedule, say $L(P)$, satisfies

$$L(P) \geq L^*. \tag{2}$$

2b) Use part (2a) to prove that the greedy algorithm on p. 24 of the lecture notes is correct. Hint: Consider the lower bound you obtained in part (2a) in relation to the schedule P generated by the greedy algorithm.

Solution 2a) Let P be a feasible schedule. We can assume, without loss of generality, that P does not include gaps in the processing, that is, all the jobs are scheduled with one starting immediately after another finishes, as is illustrated by the schedules sketched in the lecture notes (and copied below). (Since otherwise we could construct a new schedule by simply removing these gaps, and this modification must either decrease the max lateness or leave it the same.)

Any such schedule P can be represented by the appropriate permutation $(\sigma(1), \sigma(2), \dots, \sigma(n))$, where job $\sigma(k)$ is the k^{th} job to be run in execution order specified by the schedule. We define $S(k)$, for $k = 1, \dots, n$ to be the set of the first k jobs scheduled in P, that is

$$S(k) = \{i \mid i = \sigma(j) \text{for } j = 1, \dots k\}. \tag{3}$$

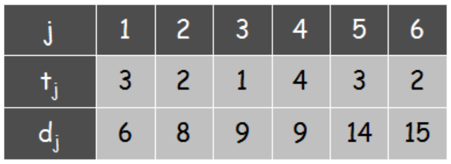
It is also useful to use $F(k)$ to denote the finish time t of these first k jobs, that is,

$$F(k) = \sum_{i \in S(k)} t_i. \tag{4}$$

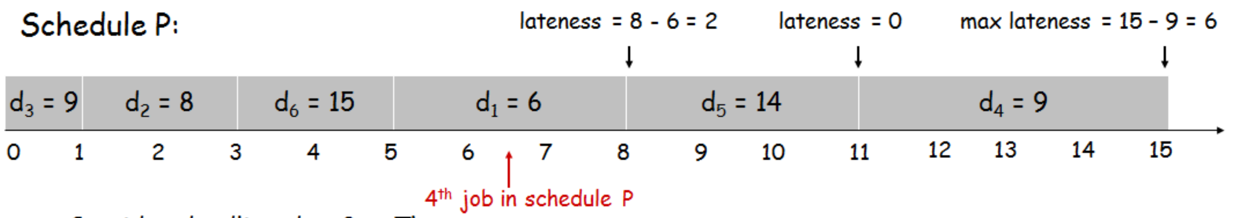
Note the deadline for this k^{th} job in the schedule is $d_{\sigma(k)}$ and the lateness will be

$$\max(0, F(k) - d_{\sigma(k)}). \tag{5}$$

The schedule shown in class provides a concrete example (see below). Here $(\sigma(1), \dots \sigma(6)) = (3, 2, 6, 1, 5, 4)$.



j	1	2	3	4	5	6
t_j	3	2	1	4	3	2
d_j	6	8	9	9	14	15



Consider deadline $d_2 = 8$. Then:
$T(d_2) = t_1 + t_2 = 5$ (the sum of durations of all jobs with deadline <= 8)
$K(8) = 4$ (all jobs in P with deadlines <= 8 are scheduled within the first 4 in P).
$S(K(8)) = S(4) = \{3, 2, 6, 1\}$ (the j's for the first 4 jobs in P).
$F(4) = t_3 + t_2 + t_6 + t_1 = 8$ (the finish time of the 4 jobs in P, coincidentally equal to d_2)

Let d_j be any deadline given in the problem data. Define $K(d_j)$ to be the position of the last job in the schedule P which has a deadline of d_j or less. That is,

$$K(d_j) = \max\{k \mid d_{\sigma(k)} \le d_j\}. \tag{6}$$

By construction of $K(d_j)$ it follows that each of the jobs with deadlines less than or equal to d_j must be scheduled within the first $K(d_j)$ jobs, that is

$$\{i \mid d_i \le d_j\} \subseteq S(K(d_j)). \tag{7}$$

Since $t_i > 0$ for all i, we then find

$$T(d_j) \equiv \left[\sum_{\{i \mid d_i \le d_j\}} t_i \right] \le \left[\sum_{\{i \in S(K(d_j))\}} t_i \right] \equiv F(K(d_j)). \tag{8}$$

The reason for this inequality is that, by (7), every term on the left hand side appears on the right, and every term on the right is non-negative. That is, extra terms can only appear on the right and these will all be positive.

Next consider the lateness of the job scheduled in position $K(d_j)$, which corresponds to job $i = \sigma(K(d_j))$, that is, it has the runtime t_i and the deadline d_i. Note, by construction of $K(d_j)$ (as the last job scheduled with deadline less than or equal to d_j), this last job i, with data (t_i, d_i), must itself have the property that $d_i \leq d_j$. From this we can conclude the difference in finish time and deadline at the end of this job satisfies

$$\begin{aligned} F(K(d_j)) - d_{\sigma(K(d_j))} &= F(K(d_j)) - d_i && \text{since } i = \sigma(K(d_j)) \ , && (9)\\ &\geq F(K(d_j)) - d_j, && \text{since } d_i \leq d_j, && (10)\\ &\geq T(d_j) - d_j && \text{by (8).} && (11) \end{aligned}$$

Since $L(P)$ is the maximum lateness over all n intervals, and the above gives us a lower bound just for the $K(d_j)^{th}$, we can maximize this lower bound over j. That is, we find

$$L(P) \geq \max\{0, \max\{T(d_j) - d_j \mid 1 \leq j \leq n\}\} \equiv L^*, \tag{12}$$

where the threshold at zero comes from the definition of lateness.

Since P was an arbitrary schedule, this is what we were asked to show. ■

Solution 2b) Suppose we run the greedy algorithm. The algorithm is straight forward, it clearly stops and outputs a feasible schedule, say G. In this computed solution find the finish time t at which the maximum lateness, say $M = L(G)$, is achieved (if there is a tie, then use the first such job). Suppose the deadline of the job finishing at t is d_j, so the max lateness is

$$L(G) = M = \max\{0, t - d_j\}. \tag{13}$$

That is, we have used the greedy algorithm to pick out a suitable M, t and j for this instance of the problem, and we will use these quantities in the remainder of the proof.

Due to the sorting done in the greedy algorithm, any jobs with an earlier deadline $d_i < d_j$ must have been scheduled before any job with deadline d_j, and therefore they must be scheduled before time t. Moreover, if there are more than one jobs with deadline equal to d_j, then all these jobs are scheduled one after the other in G, and the latenesses of successive jobs increases by each of these jobs' durations (since $t_i > 0$ for all i). Since we picked the maximum lateness, we would have picked the last job with deadline equal to d_j. Therefore we find that **all** jobs with deadlines less than or equal to d_j must be completed on or before time t.

Moreover, by the design of the algorithm, it is **only** the jobs with deadlines less than or equal to d_j that can be scheduled before t in G. (Use contradiction, and the specification of the sorting used in the greedy algorithm.) Therefore, t must be the sum of the times taken by exactly these jobs, that is, $t = T(d_j)$. Moreover, we have

$$\begin{aligned} M &= \max\{0, t - d_j\} && \text{from (13),} && (14)\\ &= \max(0, T(d_j) - d_j) && \text{since } t = T(d_j), && (15)\\ &\leq \max(0, \max\{T(d_i) - d_i \mid 1 \leq i \leq n\}), && \text{since } j \in \{1, 2, \ldots, n\}, && (16)\\ &\equiv L^*. && && (17) \end{aligned}$$

That is, we have shown that the maximum lateness produced by the greedy algorithm, $L(G)$, which equals M, satisfies $L(G) = M \leq L^*$.

But in part (2a) we showed that, for any schedule P, $L(P) \geq L^*$ for any schedule. Therefore, it must be the case that $L(G) = L^*$. That is, no schedule can achieve a smaller lateness. Therefore the greedy algorithm produces a schedule that minimizes the maximum lateness, which is what we set out to show. ■

3. Cell Phone Towers Consider the problem of choosing cell phone tower locations on a long straight road. We are giving a list of distances along the road at which there are customers' houses, say $\{x_k\}_{k=1}^K$. No towers have yet been built (so no customers currently have service). However, a survey of the road has provided a set of J possible tower locations, $\{t_j\}_{j=1}^J$, where these potential tower locations are also measured in terms of the distance along the road. (These indexed lists of house and tower locations might be in any order. You can assume that all these distances are integers.)

Each customer will get service (at home) if and only if they are within a range $R > 0$ of at least one cell phone tower that gets built, that is, the house at x_k will have service iff there exists a tower that has been built at some t_j with $|x_k - t_j| \leq R$.

You can assume that if all the towers were built then every home would get service. However, such a solution could be overly expensive for the phone company. How can we minimize the number of towers that need to be built but still provide service at every house? Note that a suitable solution may still leave some parts of the road without cell service, even though each custormers' house will have service.

3a) Describe a plausible greedy algorithm for choosing the minimum number M of cell phone towers required, along with a suitable set of locations, say $T = \{t_{j(m)}\}_{m=1}^{M}$, such that every house has service. That is, for each $k = 1, \ldots, K$, there must exist an $m \in \{1, 2, \ldots M\}$ such that $x_k \in [t_{j(m)} - R, t_{j(m)} + R]$ (i.e, x_k can get service from tower $t_{j(m)}$).

3b) Prove that your algorithm in (a) is correct. That is, it will select the minimum possible number of towers to be built and identify an appropriate set of locations for these towers.

Solution 3a

```
sort (and replace) {x_k | k=1...K} and {t_j | j=1...J} so that they
    are in increasing order.
T = {}
Loop through the (sorted) house locations, with increasing k.
   If x_k is not covered by any tower in T:
     Find tower t_j such that x_k in [t_j-R, t_j+R] and t_j is as
     large as possible.
     If such a t_j exists, set T = T U {t_j}
     else report error "No Solution''
   end if
end loop
return T
```

Solution 3b We use a proof that is similar to the one for the truck driver algorithm that was given in the lecture notes. Note that we have assumed that all the towers would cover all the houses, so we know an optimum solution exists, although it may not be unique.

After each execution of the loop the house x_k is either covered by towers in the resulting set T, or an error has been reported. The algorithm visits each house x_k in order and clearly terminates.

First, we show a simple property for any feasible solution that will be useful to have later in the proof.

Claim 1. Let $F = \{t_{f(n)}\}_{n=1}^{N}$ denote any feasible solution, where we have sorted the indices such that $t_{f(1)} < t_{f(2)} < \cdots < t_{f(N)}$. Then for any $j \in \{1, 2, \ldots, N\}$, the set of houses left uncovered by the first j towers, i.e., at $\{t_{f(i)}\}_{i=1}^{j}$ has the form $U(j) = x_s | s \geq a$. That is, the uncovered set $U(j)$ is the set of houses at or beyond a particular house x_a. That is, the uncovered set $U(j)$ is a suffix of the sorted tuple $(x_1, x_2, \ldots, x_K)$.

Pf: By contradiction. Suppose x_a is the minimum in the uncovered set $U(j)$, and assume there exists an $x_b > x_a$ such that $x_b \notin U(j)$. For this to be true there must exist a tower $t_{f(j)}$ with $1 \leq j \leq i$ that covers x_b. So $x_b \in [t_{f(i)} - R, t_{f(i)} + R]$. But since x_a is uncovered by the first j towers it cannot be covered by $t_{f(i)}$. Therefore either $x_a > t_{f(i)} + R$ (which would contradict $x_a < x_b$) or $x_a < t_{f(i)} - R$. But this latter inequality, and the sorting of the $t_{f(m)}$ in F then implies that x_a would not be covered by **any** tower $t_{f(m)}$ for $m \geq i$. However, we chose x_a as a point that was not covered by any tower $t_{f(m)}$ for $m \leq j$. Since $j \geq i$ we conclude that x_a is not covered by any tower in F. Which contradicts F being a feasible solution. Therefore such an

x_a and x_b cannot exist and the claim follows. ■

Let $T = \{t_{g(m)}\}_{m=1}^{M}$ be the tower locations provided by the greedy algorithm.

Claim 2. The algorithm returns $t_{g(1)} < t_{g(2)} < \cdots < t_{g(M)}$. and does not report a "No Solution" error

Pf: If the alg did produce a "No Solution" error, just before returning this error, the house position x_k must be such that there is no possible tower location from the given set $\{t_j\}_{j=1}^{J}$ that provides service. This contradicts our assumption about the set of possible tower locations.

Next, the ordering of the towers $t_{g(m)}$ provided by the greedy algorithm follows from the property that we have sorted the x_k's. Specifically, it can be shown that if x_k is uncovered and $k > 1$ then the previous tower location, say $t_{g(m)}$, must satisfy $x_k > t_{g(m)} + R$. (See Claim 1 above.) That is, x_k is out of range and further down the road than the previous tower $t_{g(m)}$. In order for x_k to get service we must choose the next tower location such that $|x_k - t_{g(m+1)}| \leq R$. So, in particular $x_k - t_{g(m+1)} \leq R$. Combining these two inequalities gives $t_{g(m)} + R < x_k \leq t_{g(m+1)} + R$, which implies $t_{g(m)} < t_{g(m+1)}$, as desired. ■

Claim 3. Let $F = \{t_{f(n)}\}_{n=1}^{N}$ denote any solution, where we assume the tower locations have been sorted such that $t_{f(1)} < t_{f(2)} < \cdots < t_{f(N)}$. Then $t_{f(j)} \leq t_{g(j)}$ for each $j \in \{1, 2, \ldots, \min(M, N)$. That is, the j^{th} tower chosen by the greedy algorithm is at least as far down the road than the j^{th} tower in F.

Pf: The proof is similar to that of the Claim given in the lecture notes for the Truck Driver problem.

Suppose Claim 3 is not true. Then there must be an F for which $t_{f(j)} > t_{g(j)}$ for some j. Let j be the first time F "gets ahead".

Note that on the first iteration ($j = k = 1$) the algorithm chooses $t_{g(1)} = t_j$ where t_j is the largest tower location such that $x_1 \in [t_j - R, t_j + R]$, which is equivalent to $t_j \in [x_1 - R, x_1 + R]$. If F is already ahead at $k = 1$, then it must have chosen a tower location $t_{f(1)} = t_k$, with $t_k > t_j$. Also, in order for the greedy algorithm not to have chosen t_k, it must be the case that $t_k > x_1 + R$. That is the first house will not get service from tower $t_{f(1)}$. But since all the other towers in F are even further down the road, x_1 will not get service from any of them either. But this contradicts F being a feasible solution. Therefore, the solution F cannot be ahead on the first tower, that is, $t_{f(1)} \leq t_{g(1)}$, and the first index for this to happen must be some $j > 1$.

Suppose the first time F "gets ahead" is for $j > 1$. Due to the increasing order to the towers $t_{f(i)}$, we can consider the first house x_a that is left uncovered by the previous towers in F, but gets covered by $t_{f(j)}$. That is,

$$t_{f(j-1)} + R < x_a \text{ and } t_{f(j)} - R \leq x_a. \tag{18}$$

Since $t_{f(j-1)} \leq t_{g(j-1)}$ we have $t_{f(j-1)} + R \leq t_{g(j-1)} + R$ and,from above, we have $t_{f(j-1)} + R < x_a$. So there are two possible orderings for these three numbers, either (1) $t_{f(j-1)} + R < x_a \leq t_{g(j-1)} + R$, or (2) $t_{f(j-1)} + R \leq t_{g(j-1)} + R < x_a$. In the first case x_a is already covered by the first $j - 1$ greedy towers. While in the second case x_a is not already covered by the first $j - 1$ greedy towers.

In case (1), the first house that is left uncovered by the first $j - 1$ greedy towers, say x_b, must be further down the road than x_a (since x_a is already covered). That is, $x_b > x_a$. However, tower $t_{f(j)}$ covers x_a, so $t_{f(j)} - x_a \leq R$. But this implies $t_{f(j)} - x_b \leq R$, which implies that the greedy algorithm could have chosen the tower at $t_{f(j)}$ instead, or towers that are still further down the road than $t_{f(j)}$, which are all further down the road than $t_{g(j)}$. This contradicts the specification of the algorithm.

In case (2), the house x_a is left uncovered by the first $j - 1$ towers in both F and G, and is the first house left uncovered by these towers in F. It follows from $t_{f(j-1)} \leq t_{g(j-1)}$ that it must also be the first tower left uncovered by these towers in G. (Any house, x_b, left uncovered by the first $j - 1$ towers in G must have $x_b > t_{g(j-1)} + R \geq t_{f(j-1)} + R$, so it must also left uncovered by the first $j - 1$ towers in F.) That is, in this case, $x_a = x_b$. This situation is similar to that for the very first iteration, but now with x_a playing the role of the "first" house on the road. A similar argument shows that it must be the case that $t_{g(j)} \geq t_{f(j)}$. But this contradicts the assumption that j is the first index for F gets ahead.

Therefore, both cases (1) and (2) lead to contradictions. Therefore there cannot be any j at which F first gets ahead of G. This proves Claim 3. ■

Finally, we need to show the greedy algorithm produces a minimum number of towers, $G = \{t_{g(j)}\}_{j=1}^{q}$. We do this using contradiction and Claim 3. Assume there exists a suitable set of towers $F = \{t_{f(j)}\}_{j=1}^{p}$ with fewer towers, i.e., $p < q$. Then, after all the towers in F have been built, the last house must be covered, so $t_{f(p)} + R \geq x_K$. But Claim 3 ensures that $t_{g(p)} \geq t_{f(p)}$, so $t_{g(p)} + R \geq t_{f(p)} + R \geq x_K$. That is, the first p towers in G already cover all the houses. The assumption that the algorithm continues to select towers beyond this contradicts the algorithm specification. Therefore there cannot be any such F. And therefore the greedy algorithm must produce a set of towers with the minimun number of towers. ■