CS1510 Greedy Problems 6 & 7

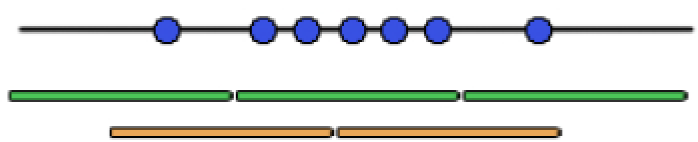
Rebecca Negley, Sean Myers

September 9, 2011

6. Consider the following problem. The input is a collection $A = \{a_1, \ldots, a_n\}$ of n points on the real line. The problem is to find a minimum cardinality collection S of unit intervals that cover every piont in A. Another way to think about this same problem is the following. You know a collection of time (A) that trains will arrive at a station. When a train arrives there must be someone manning the station. Due to union rules, each employee can work at most one hour at the station. The problem is to find a scheduling of employees that covers all the times in A and uses the fewest number of employees.

   (a) Prove or disprove that the following algorithm correctly solves this problem. Let I be the interval that covers the most number of points in A. Add I to the solution set S. Then recursively continue on the points A not covered by I.

   Solution:



   Let unit length be the size of one of the green or orange bars in the image above. (It does not matter which one as they are all the same size.) According to the greedy algorithm, the middle green bar in the diagram would be the first interval chosen, since it has five points within the interval and no other interval could match that accumulation of points. When this happens, the remaining points to the left and right of this interval each need their own interval, giving a cardinality of 3. The optimal interval cardinality, as shown below the greedy in orange, is 2. Therefore, this greedy algorithm is not optimal.

   (b) Prove or disprove that the following algorithm correctly solves this problem. Let a_j be the smallest (leftmost) point in A. Add the interval $I = (a_j, a_j + 1)$ to the solution set S. then recursively continue on the points in A not covered by I.

   Solution: Let's assume that this algorithm, which we will call Left-Most First (or LMF), is incorrect. In other words, there exists some input L such that $LMF(L)$ is not an optimal schedule of employees that cover all times in A. Choose an optimal solution, $OPT(L)$. Order both $LMF(L)$ and $OPT(L)$ from leftmost to rightmost. Let the k^{th} interval be the first interval where $LMF(L)$ and $OPT(L)$ differ. At this kth step, OPT chooses interval I_a and LMF chooses interval I_b. Let's now attempt to construct an $OPT'(L)$ that agrees with $LMF(L)$ for one more step and is still optimal.

In order to do this, we must let $OPT'(L) = OPT(L) - I_a + I_b$.

Now that we replaced I_a with I_b, we must show that $OPT'(L)$ is still an optimal solution. There are only three cases in which we need to concern ourselves with: if the interval I_a was before, after or the same as interval I_b. Intuitively, we know that the two intervals cannot be the same, since we already stated that they are different.

Suppose I_a begins after I_b. There is some point $a_j \in A$ such that a_j is the leftmost point in I_b (according to LMF). If I_a is to the right of I_b, I_a cannot cover a_j. The first $(k-1)$ intervals do not cover a_j because a_j was uncovered at the k^{th} step of LMF, and LMF and OPT (and OPT' by the way it was constructed) agree for the first $(k-1)$ steps. Therefore, we must have another interval $I_c \in OPT(L)$ that covers a_j since the optimal solution must cover every point. However, since a_j is to the left of I_a, I_c must be to the left of I_a. Since all intervals to the left of I_a in $OPT(L)$ were in the first $(k-1)$ steps (because of the left-right ordering), this gives that I_c must be in the first $(k-1)$ intervals. Then, a_j was covered in the first $(k-1)$ intervals, which contradicts our statement above. Hence, we cannot have I_a to the right of I_b.

Therefore I_a must be to the left of I_b. Since both algorithms agreed up to the k^{th} step, we know that all points in A to the left of a_j have already been covered by the first $(k-1)$ intervals of both $OPT(L)$ and $LMF(L)$ (and therefore $OPT'(L)$). Otherwise, I_b would have been chosen further to the left by the definition of LMF. Thus, if I_a started before interval I_b, we know there could be no a_l not covered by the first $(k-1)$ intervals such that $a_l \in A \cap I_a$ and $a_l \notin I_b$ as this would require $a_l < a_j$. Therefore, I_b will cover all points in $A \cap I_a$ not covered by the first $(k-1)$ intervals. Thus, $OPT'(L)$ is still an optimal solution and agrees with $LMF(L)$ for one more step. Therefore, we can continue with this process to create an $OPT^m(L)$ that is still optimal and agrees with $LMF(L)$ for every step. By the exchange argument, we contradict our assumption that $LMF(L)$ is not an optimal solution, so the LMF algorithm must be correct/optimal.

7. Consider the following problem. The input consists of the lengthst $l_1, \ldots, l_n$, and access probabilities $p_1, \ldots, p_n$, for n files $F_1, \ldots, F_n$. The problem is to order these files on a tape so as to minimize the expected access time. If the files are placed in order $F_{s(1)}, \ldots, F_{s(n)}$ then the expected access time is

$$\sum_{i=0}^{n} p_{s(i)} \sum_{j=0}^{i} l_{s(i)}$$

For each of the below algorithms, either give a proof that the algorithm is correct, or a proof that the algorithm is incorrect.

(a) Order the files from shortest to longest on the tape. That is, $l_i < l_j$ implies that $s(i) < s(j)$.

Solution: Suppose there are only 2 files, F_1 with length $l_1 = 20$ and access probability $p_1 = 0.5$ and F_2 with length $l_2 = 25$ and access probability $p_2 = 0.9$. Then, ordering the files from shortest to longest would give $s(1) = 1$ and $s(2) = 2$. Then, the expected access time for this ordering is $0.5*(20)+0.9*(20+25) = 50.5$. However, if we examine the only other ordering for these two files, where $s(1) = 2$ and $s(2) = 1$, we see that the expected access time is $0.9*(25)+0.5*(25+20) = 45$. Since $45 < 50.5$, there exists a case where the shortest first algorithm produces incorrect output. Hence, the shortest first algorithm is wrong.

(b) Order the files from most likely to be accessed to least likely to be accessed. That is, $p_i < p_j$ implies that $s(i) > s(j)$.

Solution: Again, suppose there are only 2 files. This time F_1 has length $l_1 = 50$ and access probability $p_1 = 0.6$. F_2 has length $l_2 = 5$ and access probability $p_2 = 0.5$. Then, ordering the files from most likely to least likely to be accessed gives $s(1) = 1$ and $s(2) = 2$. The expected access time for this ordering is $0.6*(50)+0.5*(50+5) = 57.5$. However, by switching the order so that $s(1) = 2$ and $s(2) = 1$, we get an expected access time of $0.5*(5)+0.6*(5+50) = 35.5$. Since $35.5 < 57.5$, there exists a case where the highest probability first algorithm produces incorrect output. Hence, the highest probability first algorithm is incorrect.

(c) Order the files from smallest ratio of length over access probability to largest ratio of length over access probability. That is $\frac{l_i}{p_i} < \frac{l_j}{p_j}$ implies that $s(i) < s(j)$.

Solution: Suppose the smallest ratio first algorithm (SRF) is incorrect. Then, there exists some input L of files such that $SRF(L)$ is not an optimal file ordering. Select an optimal ordering for input L, which we will call $OPT(L)$. Then, the expected access time for $OPT(L)$ must be less than the expected access time for $SRF(L)$. Examine the first file in order where $SRF(L)$ and $OPT(L)$ disagree (say at position k). Suppose $OPT(L)$ chooses file F_a (with length l_a and access probability p_a) at the k^{th} step, and $SRF(L)$ chooses file F_b (with length l_b and access probability p_b) at the k^{th} step. Observe that $SRF(L)$ could not have chosen F_a before the k^{th} step because then $OPT(L)$ would have chosen F_a before the k^{th} step since $OPT(L)$ and $SRF(L)$ agree for the first $(k-1)$ steps. Then, $SRF(L)$ must put file F_a after file F_b. (Note that we can assume $a \neq b$ since the two solutions differ at the k^{th} step.) By the definition of $SRF(L)$, we must then have $\frac{l_b}{p_b} \leq \frac{l_a}{p_a}$. Rearranging this inequality, we get $l_b * p_a \leq l_a * p_b$.

Define $OPT'(L)$ to be $OPT(L)$ with F_b inserted before F_a (so that all files that were originally between F_a and F_b are shifted to the right by 1). We can see that $OPT'(L)$ will then agree with $SRF(L)$ for one more step. We must show that $OPT'(L)$ is still optimal. Suppose F_b is originally at the j^{th} location in $OPT(L)$ (so $j > k$ from above). Observe that we can rewrite the original expected access time formula as follows:

$$\sum_{i=0}^{n} p_{s(i)} \sum_{j=0}^{i} l_{s(i)} = \sum_{i=0}^{k-1} p_{s(i)} \sum_{j=0}^{i} l_{s(i)} + \sum_{i=k}^{j} p_{s(i)} \sum_{j=0}^{i} l_{s(i)} + \sum_{i=j+1}^{n} p_{s(i)} \sum_{j=0}^{i} l_{s(i)}$$

Observe that the total length of the preceding files (i.e., the inner sum) does not change for the files before k or after j when we convert $OPT(L)$ to $OPT'(L)$, and each position in those regions contains the same file in $OPT(L)$ and $OPT'(L)$. Therefore, the first and last sums are the same for $OPT(L)$ and $OPT'(L)$. Therefore, it only remains to show that

$$\sum_{i=k}^{j} p_{s(i)} \sum_{j=0}^{i} l_{s(i)} \ (for\ OPT'(L)) \leq \sum_{i=k}^{j} p_{s(i)} \sum_{j=0}^{i} l_{s(i)} \ (for\ OPT(L))$$

Let $u(x)$ be the file index of the file that $OPT(L)$ puts in the x^{th} position. Let $v(x)$ be the file index of the file that $OPT'(L)$ puts in the x^{th} position. Then, for $x \in [k+1, j-1]$, $u(x) = v(x+1)$.

Observe that, for $OPT(L)$, $\sum_{i=k}^{j} p_{u(i)} \sum_{j=0}^{i} l_{u(i)} =$

$p_a*(l_a)+p_{u(k+1)}*(l_a+l_{u(k+1)})+\ldots+p_{u(j-1)}*(l_a+l_{u(k+1)}+\ldots+l_{u(j-1)})+p_b*(l_a+l_{u(k+1)}+\ldots+l_{u(j-1)}+l_b)$.

Similarly, for $OPT'(L)$, $\sum_{i=k}^{j} p_{v(i)} \sum_{j=0}^{i} l_{v(i)} =$

$p_b * (l_b) + p_a * (l_b + l_a) + p_{v(k+2)} * (l_b + l_a + l_{v(k+2)}) + \ldots + p_{v(j)} * (l_b + l_a + l_{v(k+2)} + \ldots + l_{v(j)})$

$= p_b * (l_b) + p_a * (l_b + l_a) + p_{u(k+1)} * (l_b + l_a + l_{u(k+1)}) + \ldots + p_{u(j-1)} * (l_b + l_a + l_{u(k+1)} + \ldots + l_{u(j-1)})$.

Then, we can see that (expected access time of $OPT(L)$)$-$(expected access time of $OPT'(L)$) $= \sum_{i=k}^{j} p_{u(i)} \sum_{j=0}^{i} l_{u(i)} - \sum_{i=k}^{j} p_{v(i)} \sum_{j=0}^{i} l_{v(i)} = (p_b * l_a - l_b * p_a) + (p_b * l_{u(k+1)} - l_b * p_{u(k+1)}) + \ldots + (p_b * l_{u(j-1)} - l_b * p_{u(j-1)})$. For each $i > k$, we know that $\frac{l_b}{p_b} \leq \frac{l_{u(i)}}{p_{u(i)}}$ because F_b was chosen before $F_{u(i)}$ in $SRF(L)$. Therefore, $l_b * p_{u(i)} \leq l_{u(i)} * p_b$ for any $i > k$, or $p_b * l_{u(i)} - l_b * p_{u(i)} \geq 0$. Also, from above, $p_b * l_a - l_b * p_a \geq 0$. Hence, (expected access time of $OPT(L)$)$-$(expected access time of $OPT'(L)$) is the sum of non-negative parts and is therefore non-negative. This gives (expected access time of $OPT'(L)$)$\leq$(expected access time of $OPT(L)$). Thus, $OPT'(L)$ is still an optimal solution that agrees with $SRF(L)$ for one more step. We can continue by this method to construct $OPT''(L)$, $OPT'''(L)$,... until we reach a solution $OPT^m(L)$ that is still optimal and matches $SRF(L)$ for every step. Therefore, we contradict the supposition that $SRF(L)$ is not optimal, so the SRF algorithm must be correct.