Comp 7720 - Online Algorithms

Assignment 1: Introduction, Searching & List Update

University of Manitoba - Fall 2019

Due: October 1st at 11:59 pm

“Travel doesn’t become adventure until you leave yourself behind ...” Marty Rubin

Please pay attention to the followings when preparing/submitting your assignment:

• All problems are written problems. There are six problems with a total of 70 marks.

• If you feel the assignment is too hard (or too simple), do not panic! To some extent, this assignment serves to indicate how easy/hard the future assignments/exam should be. More importantly, the assignment is designed to raise a group discussion. I would like to see a discussion on Piazza and when you are confused drop a hint about a problem. Think of the assignment as a project that you need to work on together on Piazza. If you are not active on Piazza, you will miss the hints that I ‘plan’ to drop.

• You are welcome to discuss the problems with your friends (or enemies). But you should write your answers individually. You might be interviewed about your answers. Be careful not to accidentally copy.

• Let me be redundant: if you have any question related to the assignment, you are encouraged to post it on Piazza. Note that you can submit anonymously. Also, instead of emailing me, you can always write a private note in Piazza. It is likely that I drop hints when a question is posted publicly on Piazza (because all students can benefit from it). It is not the case when you ask questions in emails or during office hours.

• Submit your answers electronically using Crowdmark.
Problem 1  Ski-rental & Randomization [4+6 = 10 marks]

I) Consider the following algorithm for the ski-rental problem: flip two fair coins at the beginning, if both are tails, buy the equipment at the beginning; if any of the coins is a head, always rent and never buy. Assume the cost of buying is \( b \) and the cost of renting is \( 1 \) per day; let \( x \) denote the number of days that the the player goes skiing.

a) What is the expected cost of the algorithm in terms of \( b \) and \( x \)?

Answer: a) The cost of the algorithm is \( b \) when both coins are tails and \( x \) otherwise. On expectation, it is \( b/4 + 3x/4 = (b + 3x)/4 \).

b) Despite using randomization, the algorithm is not competitive. Consider the following adversarial input: the adversary selects a large \( x = 2^b \), where \( b \) is also a large number. The expected cost of the algorithm is \( (b+3x)/4 = (b+3\cdot 2^b)/4 > 2^{b-1} \). The cost of \( \text{OPT} \) is \( b \) (it knows \( x \) is pretty large and buys at the beginning). The competitive ratio will be at least \( \frac{2^{b+1}}{b} \) which is unbounded for large values of \( b \).

II) Consider another randomized algorithm that works as follows. It rents equipment for the first \( b - 1 \) days. For each day that follows, it buys the equipment with a chance of \( 3/4 \) and rents otherwise (note that if the algorithm buys the equipment at some day, it obviously does not make a decision in the days that follows).

What is the competitive ratio of the algorithm? To answer this question, put yourself in the shoes of an adversary who wants to make a worst-case scenario for the algorithm.

Answer: For sufficiently large values of \( x \), the algorithms rents for the first \( b - 1 \) days, and, on average, it rents for another \( A = \frac{3}{4} \cdot 0 + \frac{3}{4} \cdot 1 + \frac{3}{4} \cdot 2 + \ldots \) days. We have \( 4A = 3 \cdot 0 + \frac{3}{4} \cdot 1 + \frac{3}{4} \cdot 2 + \frac{3}{4} \cdot 3 + \ldots \). Subtracting the two terms, we get \( 3A = \frac{3}{4} + \frac{3}{4^2} + \ldots = 3 \), i.e., \( A = 1 \). So, the adversary chooses a large value for \( x \), e.g., \( x \geq 2^b \). The cost of \( \text{OPT} \) will be \( b \) and the expected cost of the algorithm will be \( b \) (for renting) plus \( b \) (for buying) which gives \( 2b \). The competitive ratio will be \( \frac{2^b}{b} \).

Note that the adversary does choose \( x \) to be arbitrary large. In case \( x \) is smaller than \( b \), the algorithm is optimal and when it is larger than or equal to \( b \), increasing \( x \) will make the algorithm to pay a larger expected cost while the cost of \( \text{OPT} \) won’t change.

Problem 2  Path-cow Problem [6+6+8 = 20 marks]

I) Consider the following algorithm for path-cow problem. The cow starts at the origin, moves \( x = 1 \) unit to the right. If the target is not found, the cow comes back to the origin and repeats this procedure with \( x \) the target is not found, the cow comes back to the origin and goes

a) Assume the cow finds the hole at distance \( u \) from the origin on its left. Assume the largest power of 2 which is smaller than \( u \) is \( 2^k \). What is the total distance moved by the cow?

b) Where does the adversary place the hole in order to harm the algorithm?

c) What is the competitive ratio of this algorithm?

Answer: a) The total moved distance would be:
\[
4(1 + 2 + \ldots + 2^k) + 2 \cdot 2^{k+1} + u = 6 \cdot 2^{k+1} + u - 4
\]

b) Adversary places the hole at distance \( u = 2^k + \epsilon \) on the left. The cost of \( \text{OPT} \) will be \( u = 2^k + \epsilon \).

c) The competitive ratio would be: \( \frac{6 \cdot 2^{k+1} + u - 4}{u} = 1 + \frac{6 \cdot 2^{k+1}}{u} \approx 1 + \frac{6}{2^k} \approx 13 \) (to get this ratio, the adversary chooses small \( \epsilon \) and large \( u \)).

II) In part I we assumed the algorithm is deterministic and the first move is to the right. Consider the same algorithm in which the first move is randomly selected to be to the right or left (each with a chance of 1/2). What is the competitive ratio of this randomized algorithm?
The competitive ratio will be $(3k \text{ round accordingly.})$

What is the competitive ratio of this algorithm? To answer, assume the target is at depth $i$. The total cost of the two algorithms is $\text{Alg}_1 + \text{Alg}_2 = 4(1 + 2 + \ldots + 2^k) + 2 \cdot 2^{k+1} + u + 4(1 + 2 + \ldots + 2^k) + u = 8 \cdot 2^{k+1} - 8 + 2u + 2 \cdot 2^{k+1}$

So, the expected cost of the randomized algorithm is $4 \cdot 2^{k+1} + 2^k + u - 4 = 2^{k+3} + 2^k + u - 4$. As before, the adversary places the target at distance $2^k + \epsilon$ (this time it does not matter on which side). The competitive ratio will be:

$$\frac{2^{k+3} + 2^{k+1} + u - 4}{u} = 1 + \frac{2^{k+3} + 2^{k+1} - 4}{2^k} \approx 11$$

III) Assume instead of a path, we have a ternary tree with each edge having a length of 1. Originally, the cow is located at the root. First, she moves to the left child; if the target is not found, she goes back to the root and then moves to the middle child; if the target is not found, goes back to the root and then the right child. If the target is still not found, she returns to the root and goes to visit nodes at depth 2 of the tree (i.e., at distance 2 from the root on the left). After checking these nodes, the cow returns to the root and repeats the same for nodes of depth 3. This procedure is repeated until at some point the target is found. In a nutshell, the algorithm works in rounds, where at round $i$ it visits all vertices of depth $i$.

What is the competitive ratio of this algorithm? To answer, assume the target is at depth $k$ and write the competitive ratio in terms of $k$. As before, you have to indicate where the adversary places the target and deduce the competitive ratio accordingly.

Answer: At step $i$, nodes of depth $i$ are visited. The induced ternary subtree is formed by $1 + 3 + \ldots + 3^i = \frac{3^{i+1} - 1}{2} - 1$ vertices and hence $\frac{3^{i+1} - 1}{2} - 1$ edges. Each of these edges is visited 2 times in a round. So, the cost of the algorithm at round $i$ is $3^{i+1} - 3$. In the worst case, the target is located at depth $k$ and is the last vertex checked by the algorithm at round $k$. In this case, the distance moved by round $k$ is $3^{k+1} - 3 - k$ (the last deduction is because the algorithm does not return to the root after finding the target). On the other hand, the cost of Opt is $k$. The cost of the algorithm will be

$$(3^{k+1} - 3) + (3^{k+2} - 3) + \ldots + (3^{k(k-1)} - 3) + (3^{k+1} - 3 - k) = (3^{k+2} - 1)/2 - 4k - 4$$

The competitive ratio will be $\frac{(3^{k+2} - 1)/2 - 4k - 4}{k}$.

Problem 3 Online Bidding & Advice [4 + 6 = 10 marks]

We saw in the class that a simple doubling approach gives the best competitive ratio that a deterministic online bidding algorithm can achieve. That ratio was 4. In this problem, we examine the power of advice and randomization for this problem; basically we want to show that advice can be stronger than randomization. Consider two deterministic algorithms Alg1 and Alg2, where Alg1 guesses are $1, 4, 16, \ldots 4^i$ and Alg2 guesses are $2, 8, 32, \ldots, 2 \cdot 4^i$.

I) Consider an algorithm that flips a fair coin at the beginning and randomly chooses between Alg1 and Alg2, and uses the guesses of the selected algorithm. What is the competitive ratio of this randomized algorithm?

Answer: The adversary chooses $u = 2^k + \epsilon$. The sum of the costs of the two algorithms is $1 + 2 + 4 + \ldots + 2^k + 2^{k+1} + 2^{k+2} = 2^{k+3} - 1$. The expected cost is therefore $2^{k+2} - 1/2$ and the competitive ratio is $\frac{2^{k+2} - 1}{2^{k+1} - 1} \approx 4$. Note that this no better than the deterministic algorithm.

II) Assume an algorithm that receives 1 bit of advice as follows. For each instance of the problem the advice bit indicates the algorithm which has smaller cost between Alg1 and Alg2. What is the competitive ratio of the algorithm with 1 bit of advice?

Answer: As before, assume the adversary chooses $u = 2^k + \epsilon$. If $k$ is even, Alg2 has cost $2 + 8 + 32 + \ldots + 2^{k-1} + 2^{k+1} = 2(1 + 4 + 16 + \ldots + 4^{(k-2)/2} + 4^{k/2}) = 2 \cdot \frac{4^{k/2+1} - 1}{3} \approx 8/3 \cdot 2^k$. The ratio between the cost of Alg2 and Opt will converge to $8/3$ for large values of $k$.

If $k$ is odd, Alg1 has cost $1 + 4 + 16 + \ldots + 2^{k-1} + 2^{k+1} = 1 + 4 + 16 + \ldots + 4^{(k-1)/2} + 4^{(k+1)/2} = \frac{4^{(k+1)/2} - 1}{3} \approx \frac{5}{3} 2^k$. Like before the ratio between the cost of the algorithm and Opt approaches to $8/3$ for large values of $k$. Note that only one bit of advice resulted in an algorithm with better competitive ratio than any purely-online algorithm.
Problem 4  Clustering & Advice \( [4 + 6 = 10 \text{ marks}] \)

Consider the online clustering problem. Recall that in this problem, we need to partition a sequence of online points into \( k \) clusters so that the maximum diameter of clusters is minimized. Assume the length of the input sequence (i.e., the number of points) is \( n \).

a) Show that \( O(n \log k) \) bits of advice are sufficient to achieve an optimal clustering. To get the full mark, you need to precisely indicate i) what the advice encodes? ii) how the algorithm works? iii) why the algorithm is optimal?

**Answer:** Consider an optimal clustering \( \text{OPT} \). Let \( c(x) \) indicates the cluster at which \( x \) belongs to in the final clustering of \( \text{OPT} \). The four steps for an algorithm with advice are as follows:

- what is the advice? for each point \( x \), the advice indicates \( c(x) \).
- what is the size of advice? the value of \( c(x) \) can be encoded in \( O(\log k) \). This is because there are at most \( k \) clusters in the final packing of \( \text{OPT} \). So, the total advice size is \( O(n \log k) \).
- how the algorithm uses advice? the algorithm places each input point \( x \) in cluster \( c(x) \) (indicated by advice).
- why the algorithm is optimal? the algorithm creates the same clustering as \( \text{OPT} \).

b) Consider a version of the problem in which no three points are located at the same line. In particular, there is a value \( \epsilon > 0 \) such that for any three points \( x, y, \) and \( z \), we have \( d(x, z) \leq (1 - \epsilon)(d(x, y) + d(y, z)) \). Follow the steps that we took in the analysis of the algorithm we saw in the class and indicate its competitive ratio as a function of \( \epsilon \). Show your work.

**Answer:** As we saw in the class, let \( d_1, d_2, \ldots, d_n \) indicate the parameters that the algorithm select. The max diameter of clusters in an optimal solution in phase \( i + 1 \) is at least \( d_i \). We have to be more careful for the cost of online algorithm. Let \( r_{i+1} \) denote the maximum radius of any new (merged) cluster at phase \( i + 1 \). We can write

\[
r_{i+1} \leq (1 - \epsilon)(r_i + d_{i+1})
\]

The above inequality follows from the triangle inequality on the triangle formed by centers of the merged clusters and the new point (see Slide 21). Applying a bit of boring algebra (substituting the recursions) we get:

\[
r_{i+1} \leq (1 - \epsilon)(r_i + d_{i+1})
\]

\[
\leq (1 - \epsilon)((1 - \epsilon)(r_{i-1} + d_i) + d_{i+1}) = (1 - \epsilon)d_{i+1} + (1 - \epsilon)^2d_i + r_{i-1}
\]

\[
\leq \ldots
\]

\[
\leq (1 - \epsilon)d_{i+1} + (1 - \epsilon)^2d_i + (1 - \epsilon)^3d_{i-1} + \ldots + (1 - \epsilon)^{i+2}d_0
\]

The above formula is for the radius at round \( i + 1 \). The diameter at that round is no more than \( 2(1 - \epsilon)r_{i+1} \). For the competitive ratio of the algorithm at any round \( i + 1 \) we get:

\[
\text{competitive ratio} \leq \frac{2(1 - \epsilon)r_{i+1}}{d_i} \leq 2 \frac{(1 - \epsilon)^2d_{i+1} + (1 - \epsilon)^3d_i + (1 - \epsilon)^4d_{i-1} + \ldots + (1 - \epsilon)^{i+3}d_0}{d_i}
\]

\[
< 2(1 - \epsilon)^2d_{i+1} + d_i + \ldots + d_0
\]

So, the competitive ratio is improved by a fraction of at least \( (1 - \epsilon)^2 \).

For the case of doubling, the competitive ratio becomes at most \( 4(1 - \epsilon)^2 + 2(1 - \epsilon)^3 + (1 - \epsilon)^4 + \frac{1}{2}(1 - \epsilon)^5 + \ldots \)

Problem 5  List-Update Algorithms \( [10 \text{ marks}] \)

Consider the Move-By-Bit algorithm for the list update problem. Here, each item has a bit associated with it. At the beginning, all bits are 0. After an access to an item \( x \), Move-By-Bit moves it to the front if the bit of \( x \) is 1; otherwise it keeps \( x \) at its position. In addition, after each access, the bit of the accessed item is flipped.

Use a potential function argument to show the competitive ratio of Move-By-Bit is at most 3. You should indicate what your potential function is, what the amortized cost of the algorithm for different scenario is, and provide an upper bound
for the ratio between the amortized cost of the algorithm and $\text{OPT}$.

[in case you wonder, the actual competitive ratio of Move-By-Bit is 2.5]

**Answer:** Define inversions as we did for the analysis of MTF. Consider a pair of items which appear as $x \rightarrow y$ in the list of Move-By-Bit and $y \rightarrow x$ in the list of Opt (hence they form an inversion). Define the weight of the inversion between $x$ and $y$ as 2 if the bit of $y$ is 0 and 1 if the bit of $y$ is 1. Define the potential as the total weight of all inversions. Assume there is a request to an item $a$. In the analysis of MTF, assume $a$ is at index $i$ of the algorithm and index $j$ of $\text{OPT}$, and assume $\text{OPT}$ makes $k$ paid exchanges before serving $a$. Note that the cost of $\text{OPT}$ is $i + k$. Any paid exchange of $\text{OPT}$ creates at most one inversion and increases the potential by at most 2. So, the increase in potential because of $\text{OPT}$’s actions is $\Delta_{\text{OPT}} \Phi \leq 2k$. For the algorithm, we need to consider two events:

- Assume Move-by-Bit moves $a$ to the front (it means the bit of $a$ was 1 before the access). In this case, at least $i - j$ inversions are removed (all having weight 1 since bit of $a$ is 1), and at most $j$ inversions are added (each having a weight of at most 2). So, the difference in potential because of the actions of the algorithm is $\Delta_{\text{Alg}} \Phi \leq -(i - j) + 2j = -i + 3j$. The amortized cost will be $\text{actual cost} + \Delta_{\text{Alg}} \Phi + \Delta_{\text{OPT}} \Phi \leq (i) + (-i + 3j) + 2k = 3j + 2k$. Comparing with the cost $j + k$ of $\text{OPT}$, we conclude that amortized cost is less than 3 times the cost of opt.

- Assume Move-by-Bit does not move $a$ to the front (it means the bit of $a$ was 0 before the access). In this case, the number of inversions does not change; but the weight of all inversions including $a$ will decrease from 2 to 1. As before, the number of such inversions is at least $i - j$. Hence for the increase of potential for actions of the algorithm we have $\Delta_{\text{Alg}} \Phi \leq -(i - j) = -i + j$. The amortized cost will be $\text{actual cost} + \Delta_{\text{Alg}} \Phi + \Delta_{\text{OPT}} \Phi \leq (i) + (-i + j) + 2k = j + 2k$. Comparing with the cost $j + k$ of $\text{OPT}$, we conclude that amortized cost is at most twice the cost of opt in this case.

In both cases, the amortized cost is less than 3 times the cost of $\text{OPT}$. Consequently, the competitive ratio of the algorithm is at most 3.

**Problem 6  List Update Algorithms [10 marks]**

Consider a variant of the list update where accessing an item at position $i$ has a cost of $2i$ (paid exchanges have a cost of 1 as before). What is the competitive ratio of Move-To-Front under this new cost model? Justify your answer.

**Answer:** Lower bound: Let $C$ denote the cost of an optimal algorithm for a sequence $\sigma$ under the standard (original) cost model. An offline algorithm that maintains the list in the same way as $\text{OPT}$ does for the original model, has cost at most $2C$ (the access costs are doubled and the paid exchanges are not changed). We conclude the cost of an optimal algorithm under the new model is no more than $2C$. On the other hand, the cost of is doubles (it pays only for access costs, which now have double the cost). In summary, under the new model, the cost of is doubled and the cost of $\text{OPT}$ is not more than doubled, and consequently the competitive ratio under the new model is not smaller than the original model, that is no better than 2.

**Upper bound:** First, you should note that all free exchanges can be replaced by paid exchanges in an optimal algorithm without increasing its cost. To see that, if an optimal algorithm $\text{OPT}$ accesses an item $x$ at index $p$ and moves it closer to front to index $q < p$ using a free exchange, we can modify $\text{OPT}$ so that it uses paid exchanges to moves $x$ to index $q$ before the access. The cost of $\text{OPT}$ decreases from $2p$ to $p - q + 2q = p + q$ (note that $q < p$). So, we can assume $\text{OPT}$ does not make any free exchange.

Let’s do the steps required for a potential function argument:

- Define the potential at any time twice the number of inversions, i.e., each inversions adds 2 units to the potential.
- Assume $\text{OPT}$ accesses an item $x$ at position $j$ and makes $k$ paid exchanges (recall that it does not use free exchanges). For the cost of $\text{OPT}$, we have $\text{cost}_{\text{OPT}} = 2j + k$. For each paid exchange, the number of inversions increase by at most 1, so the actions of $\text{OPT}$ increase the potential by at most 2, i.e., $\Delta_{\text{OPT}} \Phi \leq 2k$.
- Assume MTF accesses $x$ at index $i$. We have $\text{actual cost}(\text{MTF}) = 2i$. Moving $x$ to front removes at least $i - j$ inversions and adds at most $j$ new inversions. So, the increase in the number of inversions is at most $-i + 2j$. As a result, the potential is increased by $\Delta_{\text{MTF}} \Phi \leq -2i + 4j$.
- So, for serving a request $x$ at time $t$ we have $\text{cost}_{\text{OPT}} = k + 2j$, $\text{actual cost}(\text{MTF}) = 2i$, $\Delta_{\text{OPT}} \Phi \leq 2k$, and $\Delta_{\text{MTF}} \Phi \leq -2i + 4j$. The difference in potential will be $\Delta(\Phi) = \Delta_{\text{OPT}} \Phi + \Delta_{\text{MTF}} \Phi \leq 2k - 2i + 4j$. The amortized cost of the algorithm will be $\text{amortized cost} = \text{actual cost} + \Delta(\Phi) \leq 2i + (2k - 2i + 4j) = 2k + 4j = 2(k + 2j) = 2\text{cost}_{\text{OPT}}$. 
