

CMSC 39600: Online Algorithms

Lecture 2

Course Instructor: Adam Kalai

Date: September 29, 2004

Frequency Count Accounting

1 Introduction

Recall the setup. We have a list of n items: $1, 2, \dots, n$. We receive a sequence of lookups. When we have a lookup to item i , we must pay a cost of $\text{depth}(i)$ (the depth of the first item is defined to be 1, the last is n). After the lookup, we are allowed to rearrange the list as follows. We can move the requested item forward anywhere in the list, between its current position and the front, at no cost. We can then perform arbitrary swaps of adjacent items at a cost of 1 per swap.

This simple problem was one of the motivating examples of online algorithms, given by Sleator and Tarjan, and has many of the elements of more complicated online problems.

Recall that we discussed two algorithms for this problem. The first was the Move-to-front (MTF) algorithm, which always moved the requested item directly to the front of the list. We argued that, for any sequence of requests and any offline algorithm A (one that knows the request sequence in advance) starting from the same list order,

$$\text{cost of MTF} \leq 2(\text{cost of } A).$$

That's pretty darn good. In other words, even if you knew the entire sequence in advance and could perform arbitrary swaps accordingly, you couldn't beat the cost of MTF by more than a factor of two. That's the kind of guarantee that makes one want to use competitive analysis.

If the algorithm A doesn't have to start from the same list order, we have the following guarantee:

$$\text{cost of MTF} \leq 2(\text{cost of } A) + O(n^2).$$

This is because we could always transform A into an algorithm which does start with the same permutation but then immediately rearranges the list (which can always be done at a cost of $O(n^2)$). Here n is the length of the list.

In fact, 2 is nearly the best competitive ratio you can have for any *deterministic* algorithm. For, imagine any deterministic online algorithm A . Consider the sequence of requests that always requests the item at the end of A . Thus A pays at least n every request. On the other hand, there exists a simple algorithm which pays at most $(n+1)/2$ on average. Namely, use the optimal static list, i.e. the best fixed list to use, the one with items reverse sorted according to how many times total they are looked up. This list has an average cost of at most $(n+1)/2$ because there are at least as many requests to the i th item as to the $n+1-i$ th item, for $1 \leq i \leq n/2$. Thus the competitive ratio cannot be better than $\frac{n}{(n+1)/2} = 2 \left(1 - \frac{1}{n+1}\right)$.

1.1 Frequency count

The Frequency Count (FC) algorithm orders items according to the number of times each item was looked up (in reverse order). That is, it maintains a count of the number of lookups of each

item. When an item is looked up, if necessary, it moves it forward in the list until its count is no larger than the previous item. In other words, the list is reverse sorted by counts, breaking ties by laziness.

While FC doesn't have a good competitive ratio (see Homework 1), it is an appealing algorithm that sure seems like it's adaptive. How can we theoretically describe the sense in which it is adaptive?

Let the *min-static-cost* be defined to be the cost of the best static list in hindsight. That is, after seeing the request sequence, take the list which has minimal cost, and compute it's cost on the sequence. Next, let F_t be the list used by FC for the t th lookup. Thus min-static-cost for a sequence of length T is equal to the cost of the list F_{T+1} on the entire sequence, because F_{T+1} is the best static list to use if you had to use the same list the whole time.

Theorem 1 *Suppose examples are drawn independently from some fixed distribution \mathcal{D} . Then, after T requests,*

$$E[\text{cost of FC}] \leq E[\text{min-static-cost}] + n^2\sqrt{T}.$$

Notice that this is actually quite good performance, considering that the *additive regret* (the cost - min-cost), grows only like $O(n^2\sqrt{T})$ over T requests. To better understand this, notice that over T requests, min-static-cost $\geq T$ because the cost of any list is at least T (each lookup costs at least 1). Equivalently, $n^2\sqrt{T} \leq \frac{n^2}{\sqrt{T}}$ min-static-cost. Combining this with the theorem gives,

$$E[\text{cost of FC}] \leq \left(1 + \frac{n^2}{\sqrt{T}}\right) E[\text{min-static-cost}].$$

Proof. We break it into two cases:

Case 1. $n = 2$. In a length-2 list, the cost of each lookup is 1 or 2. Now, observe that if FC never changed lists, the additive regret (cost - min-static-cost) would be 0. Moreover, in general,

$$\text{cost} \leq \text{min-static-cost} + \# \text{ changes to FC.} \tag{1}$$

That is, our regret is at most the number of times FC changes the list. To argue this rigorously, notice that FC uses the lists F_1, F_2, \dots, F_T , while min-static-cost uses $F_{T+1}, F_{T+1}, \dots, F_{T+1}$.

First, we want to argue that if we used the sequence of lists F_2, F_3, \dots, F_{T+1} , we would have no additive regret. This can be seen by a series of lists used:

$$\begin{aligned} \text{cost of } F_{T+1}, \dots, F_{T+1}, F_{T+1}, F_{T+1} &\geq \\ \text{cost of } F_T, \dots, F_T, F_T, F_{T+1} &\geq \\ \text{cost of } F_{T-1}, \dots, F_{T-1}, F_T, F_{T+1} &\geq \\ &\vdots \\ \text{cost of } F_2, \dots, F_{T-1}, F_T, F_{T+1} &\geq \end{aligned}$$

To get from the first line to the second, we replace the first $T - 1$ lists with F_T . By definition, F_T is the best list to use on the first T requests, so this could only have reduced the total cost. Similarly for the rest. So, if we used the lists the sequence of lists F_2, F_3, \dots, F_{T+1} , we would have no additive regret. The problem is that we are using the sequence F_1, F_2, \dots, F_T instead. The cost for using F_t rather than F_{t+1} is at most 1, and is 0 when $F_t = F_{t+1}$. This implies equation (1).

Now, how many times, in expectation, would there be a change, i.e. $F_{t+1} \neq F_t$? From Homework 1, the expected number of changes is at most \sqrt{T} . This implies that, for $n = 2$,

$$E[\text{cost of FC}] \leq E[\text{min-static-cost}] + \sqrt{T}.$$

Case 2. $n > 2$. Notice that the relative order of two items in frequency count doesn't depend on lookups to other items. That is, if all we care about is the order of two elements i and j , it suffices to consider the subsequence of lookups to only items i and j . (The same holds for determining the best static list in hindsight.) The key observation for this case is that the cost of a sequence of lookups on n items can be assigned to pairs of items. In particular, when item i is looked up, its depth is equal to 1+ the number of items that precede it. So, when i is requested, we can assign a cost of 1 to each pair i, j such that j precedes i in the list, and a cost of 1 to the pair i, i as well. Now our total cost is the sum over all pairs i, j of the cost assigned to that pair.

Again, notice that the cost assigned to a pair doesn't depend on the requests to other items. Let S be the sequence of lookups and $S_{i,j}$ be the subsequence of lookups to items i, j . In fact, the above observation tells us that,

$$\text{cost on } S = \sum_{i \leq j} \text{cost on } S_{i,j}.$$

Moreover, we can now see that regrets add as well,

$$\text{add. regret of FC on } S = \sum_{i < j} \text{add. regret of FC}_2 \text{ on } S_{i,j}.$$

In the above, FC_2 is the FC algorithm run with only two items. Combining this with the previous case over the $n(n+1)/2 \leq n^2$ pairs gives the theorem. \square