List Update Class Notes

Alejandro López-Ortiz[1]

Cheriton School of Computer Science, University of Waterloo
Waterloo, Ont., N2L 3G1, Canada
{alopez-o}@uwaterloo.ca

1 Online Algorithms

Consider the scenario of scheduling customer appointments at a bank branch. Customers can be served by a subset of bank tellers depending on their needs. Some wish to perform a simple transaction and can be served by anyone, while others, say, wish to apply for a mortgage, a personal loan, a business loan or open a retirement fund and as such can only be served by the employees who have the training to perform these transactions. If we knew the customer needs ahead of time we could book a full roster of appointments in such a way that no customer is left waiting. However, as we know, in practice we have customers arriving unannounced at various times during the day which makes the scheduling of appointments substantially more difficult and waiting times inevitable.

This example sets out an important and broad classification of computational problems into two classes. One consisting of (classical) problems for which the entire input to the algorithm is given at the beginning of the computation and another consisting of problems in which the input is revealed over time to the algorithm which must now make (irrevocable) decisions as it goes along. Algorithms of the latter type are known as *online algorithms*. In contrast algorithms of the former type are known as *offline algorithms*.

2 Motivation

In this lecture we study one important instance of an online algorithm, namely list update. Recall that the *Dictionary* abstract data type holds a set of items $S = \{x_1, x_2, \ldots, x_n\}$ supporting the operations

- insert(x)
- find(x) or access(x)
- delete(x)

There are many different data structure realizations of the dictionary abstract data type such as binary trees, hash tables and balanced binary trees. Depending on the properties of the set S some realizations might be preferable to others. In our particular case we consider a scenario in which (1) we know the number of elements to be small enough to not justify the overhead of an $O(\log n)$ access data structure, (2) the keys cannot be compared and/or (3) we know that accesses to a given item appear "runs", meaning that once an item has been accessed it will likely be accessed again in the near future. This property has been observed in practice and it is known as the *locality of reference principle*.

A natural choice of data structure for this problem is to store the set as a linked list, with the cost of access being the number of items examined from the beginning of the list until the item requested is found. Given that we expect a certain amount of locality of reference it is sensible to keep recently accessed elements near the front of the list so that future accesses take less time. If on the other hand accesses are distributed over a wider set of elements or indeed the entire range, we expect performance to degrade and in the worst case be $\Theta(n)$ per element.

3 List Update

More formally, consider an unsorted list of ℓ items. The input consists of a sequence $X = \langle x_1, x_2, \ldots, x_n \rangle$ of n requests that must be served in an online fashion. The items are stored as a linked list. Let $\mathcal{A}$ be an arbitrary online list update algorithm. To serve a request for an item x, $\mathcal{A}$ linearly searches the list until it finds x at a position j in the list. The cost of $\mathcal{A}$ to access x is thus j. Immediately after this access, $\mathcal{A}$ can move x to any position closer to the front of the list at no extra cost. This is called a *free exchange*. $\mathcal{A}$ can also exchange any two consecutive items at a cost of 1. These are called *paid exchanges*. An efficient algorithm can use free and paid exchanges to rearrange the list so as to minimize the overall cost of serving a sequence.

4 Free versus Paid Exchanges

A natural question to ask is what are the roles of free and paid exchanges in an optimal offline algorithm. In their original paper, Sleator and Tarjan incorrectly claimed that an optimal offline algorithm would never perform paid exchanges. The following example by Nick Reingold and Jeff Westbrook[1] shows that paid exchanges are sometimes necessary.

Example 1. Consider the initial list configuration $(1, 2, 3)$ and the request sequence $\langle 3, 2, 2, 3 \rangle$. It is easy to verify by enumeration that any algorithm relying solely on free exchanges must incur a cost of at least 9. On the other hand, if we apply two paid exchanges (at a cost of 2) before any access is made and refashion the list into $(2, 3, 1)$ the accesses $\langle 3, 2, 2, 3 \rangle$ can be served on that list at a cost of $2, 1, 1, 2$, for an access cost of 6 and an overall cost of $2 + 6 = 8$ once we include the cost of the paid accesses.

Observe that this example is not restricted to lists of size three or with that particular initial configuration. The same holds for any consecutive run of three elements in an arbitrary list configuration. This can be seen by a simple relabeling argument: call the first element in the run 1, the second 2, and the third 3 and then request them in the order $\langle 3, 2, 2, 3 \rangle$ described above.

Even more interestingly, the following theorem shows that any offline algorithm can do without *free exchanges* at all. In other words, for offline algorithms we have the exact opposite of the situation as claimed by Sleator and Tarjan: it is free exchanges that are not needed.

Theorem 1. *Let A be an offline algorithm performing free and paid exchanges. Then there exist an algorithm A' performing only paid exchanges with the same cost as A.*

Proof. Consider the scenario shown in Figure 1. The top two rows show the configuration of A's list at times $i-1$ and i, while the bottom two rows show the configuration of A' also at times $i-1$ and i on the same request sequence. A moves x_i forward using free exchanges from position k to position ℓ in the list. In contrast, the modified algorithm A' moves x_i during the $(i-1)$th access. This is done using paid exchanges.

Now let us compare the costs of the two algorithms A and A'. Table 1 shows that the total cost of A for the $(i-1)$ and ith accesses is $j + k$, which exactly matches the cost of A' given by $j + (k - \ell) + \ell = j + k$, thus proving the theorem. □

[1] Trivia: Jeff Westbrook is now a writer and co-producer of the animated TV show "The Simpsons".

Operation	Cost A	Cost A'
access(x_{i-1})	j	j
exchanges	0	$k-\ell$
access(x_i)	k	ℓ
exchanges	0	0
TOTAL	$j+k$	$j+k$

Table 1. Cost comparison between A and A'

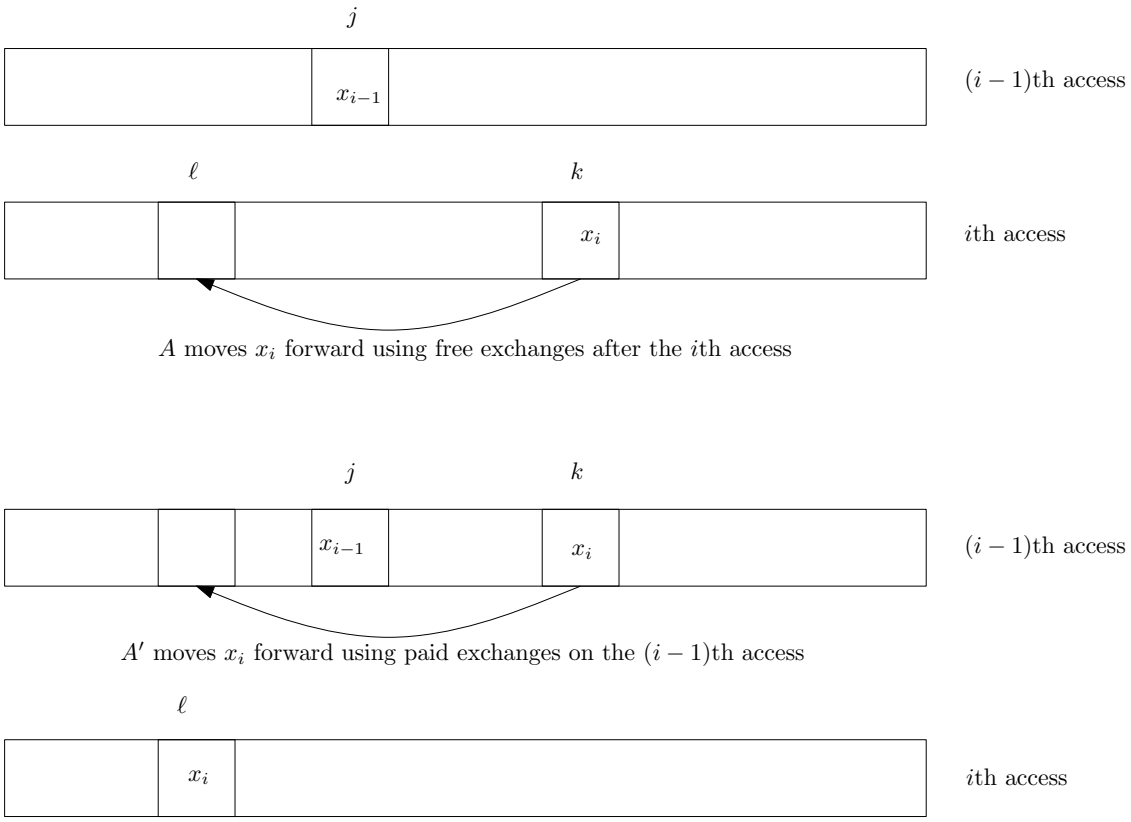

Fig. 1. Free and paid exchanges

5 Move-To-Front

A sensible strategy to reorder the list after each access is Move-To-Front (MTF) heuristic. MTF scans the list looking for the requested item and upon finding it moves it to the first position (or front) of the list. In real life, items on a desk, kitchen counter or work area are often organized in a move-to-front like-manner in that items which are presently being used are located nearby (front of the list) and items which are not currently being used are stowed further away (back of the list). This intuitive observation is the basis of the Move-To-Front heuristic.

Recall that a heuristic is a sensible strategy or technique which seems to work in some instances but lacks theoretical guarantees. In what follows we will introduce a framework which allows us to provide solid guarantees on the quality of the behaviour of MTF.

6 Offline Optimum

List update was first studied by McCabe [11] in the context of maintaining a sequential file. In the early stages, list update algorithms were analyzed using the distributional or average-case model (e.g. [11, 8]). In this model, the request sequences are generated according to a probability distribution and the efficiency of an algorithm is related to the expected cost it incurs. A drawback is that in practice the input distribution is often unknown, difficult to determine and furthermore it may evolve during runtime. For example, if we are using list update as part of a WWW server, users arriving during the morning likely exhibit different request patterns from those arriving at lunch time or after hours.

One way to measure the quality of MTF or in general of any offline algorithm is to compare its performance on an input sequence X against that of the best offline algorithm which is given the same sequence X in advance. This optimal offline algorithm is usually referred to as OPT.

Observe that OPT sets a rather high standard, and as such is a harsh taskmaster. In general, even the best online algorithms will perform substantially worse than OPT as they do not have access to the input before hand.

To complicate issues further, it can be quite difficult to determine the exact set of exchanges required by OPT to serve the problem offline optimally. For the case of list update its exact complexity remained open until relatively recently when in 2000 Christoph Ambühl, a second year PhD student in Zurich showed that computing OPT is NP-hard.

Open Question: What is the complexity of OPT_F, that is the optimum algorithm in the case where we only allow free exchanges for OPT?

7 Performance of MTF

The following theorem shows that MTF compares rather well to the offline optimum OPT, all things considered.

Note that the fact that the comparison can be performed at all is, at first, rather startling as we have just stated above that the exact shape of OPT on a given request sequence is NP-hard to compute. This might suggest that we have no hope of ever performing the comparison if we cannot materialize OPT in practice. However, this difficulty can be surmounted by observing that we do not need the precise cost and exact nature of OPT and that relying on properties, general shape and/or an approximation of its configuration we can still perform meaningful comparisons.

Theorem 2. *Let $X = \langle x_1, x_2, \ldots, x_n \rangle$ be an arbitrary request sequence. Let $C_{MTF}(X)$ and $C_{OPT}(X)$ denote the cost—including paid exchanges—of servicing the sequence X under MTF and* OPT*, respectively, starting from the same initial list configuration. Then $C_{MTF}(X) \leq 2\, C_{OPT}(X)$*

Proof. Intuitively, if the lists as maintained by MTF and OPT are rather similar then their costs must match. How similar are the two given lists? One way to measure their relative differences is to compare the number of items (x, y) such that x appears before y in MTF's list while x appears somewhere after y in OPT's list. This is called the *number of inversions* between the two list configurations. Observe that in the worst case the number of inversions between two arbitrary given lists is $\binom{n}{2} = n(n-1)/2 = \Theta(n^2)$.

We sill show using a potential function argument that the lists remain relatively close. Let $\phi(i)$ denote the number of inversions after servicing the ith request. Consider the cost $c_{MTF}(i)$ and $c_{OPT}(i)$ of servicing the ith item in the sequence by MTF and OPT respectively. We will show first that

$$c_{MTF}(i) + \phi(i) - \phi(i-1) \leq 2\, c_{OPT}(i) - 1.$$

Suppose we were to run the two algorithms in parallel, each on their own copy of the list, starting from a common configuration. Figure 2 shows a schematic of the two list configurations at access time i.

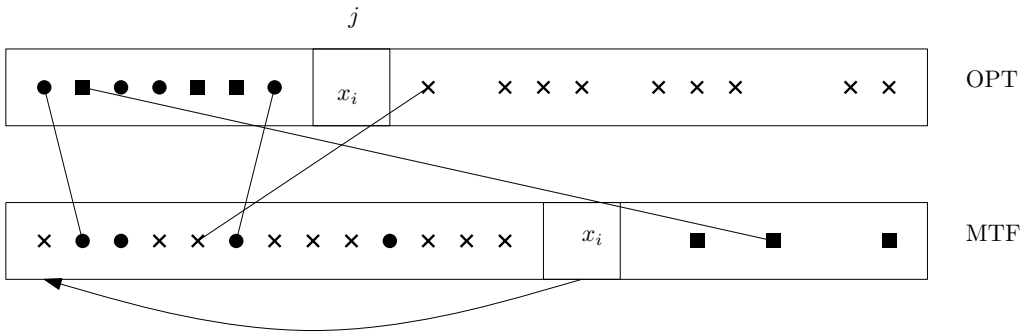

Fig. 2. OPT and MTF configurations at time i.

Let k denote the number of elements that precede x_i in both MTF and OPT's lists at time i (see Figure 2 where these elements are shown as solid circles). Let ℓ denote the number of elements that appear after x_i in OPT's list but precede x_i in MTF's list also at time i (shown as crosses in Figure 2). Then clearly we have

$$c_{MTF}(i) = k + \ell + 1.$$

Turning our attention to the terms $\phi(i) - \phi(i-1)$ we see that after MTF moves x_i to the front of the list, the solid disks now follow x_i in MTF's list thus creating k new inversions while the crosses now follow x_i in both MTF and OPT's list thus removing ℓ inversions. Hence the number of inversions so far has changed by $k - \ell$. Least we forget, OPT is also allowed to rearrange its list. From Theorem 1 we know that OPT performs paid exchanges only. Each paid exchange creates one inversion thus boosting by one the value of $\phi(i) - \phi(i-1)$ while $c_{OPT}(i)$ goes up by the same amount as paid exchanges cost one each. Let P denote the number of paid exchanges, hence

$$c_{OPT}(i) = j + P \geq k + 1 + P$$

where j is the position of x_i in OPT's list at time of access i as shown in Figure 2. Putting it all together we get

$$c_{MTF}(i) + [\phi(i) - \phi(i-1)] = k + \ell + 1 + [k - \ell + P] = 2k + 1 + P \leq 2\, c_{OPT}(i) - 1$$

Now, having shown that

$$c_{MTF}(i) + \phi(i) - \phi(i-1) \leq 2c_{OPT}(i) - 1$$

we observe that $C_{MTF}(X) = \sum_{i=1}^{n} c_{MTF}(i)$ and $C_{OPT}(X) = \sum_{i=1}^{n} c_{OPT}(i)$ so we add up over all requests i on both sides of the inequality above to obtain

$$C_{MTF}(X) + \sum_{i=1}^{n} [\phi(i) - \phi(i-1)] \leq 2\, C_{OPT}(X) - n \leq 2\, C_{OPT}(X).$$

Lastly note that the summation above telescopes giving $\sum_{i=1}^{n} [\phi(i) - \phi(i-1)] = \phi(n) - \phi(0) > 0$. since $\phi(0) = 0$ and $\phi(n) > 0$. This completes the proof that

$$C_{MTF}(X) \leq 2\, C_{OPT}(X).$$

□

In the next section we introduce a framework to design and analyse online algorithms whose performance differ from that of the offline optimum by a given factor.

8 Competitive Ratio

The competitive ratio was first introduced formally by Sleator and Tarjan [13]. An algorithm A is said to be α-competitive (assuming a cost-minimization problem) if the cost of serving any specific request sequence never exceeds α times the optimal cost (up to some additive constant) of an *offline* algorithm which knows the entire request sequence, i.e.

$$cost_A(X) \leq \alpha\, cost_{OPT}(X) + c,$$

where c is a constant and α is either a constant or a function of n as needed—hence we can talk, for example, about algorithms which are $\log n$-competitive. The term competitive ratio comes from the fact that asymptotically we have

$$\frac{cost_A(X)}{cost_{OPT}(X)} \leq \alpha.$$

The competitive ratio has served as a practical measure for the study and classification of online algorithms in general and list-update algorithms in particular. List update algorithms were among the first algorithms studied using competitive analysis. Other well-known deterministic online algorithms are *Transpose* and *Frequency-Count* (FC). Transpose exchanges the requested item with the item that immediately precedes it. FC maintains an access count for each item ensuring that the list always contains items in non-increasing order of frequency count. Sleator and Tarjan showed that MTF is 2-competitive, while Transpose and FC do not have constant competitive ratios [13]. Since then, several other deterministic and randomized online algorithms have been studied using competitive analysis. (See [9, 1, 2, 7] for some representative results.)

Notwithstanding its wide applicability, competitive analysis has some drawbacks. For certain problems, it gives unrealistically pessimistic performance ratios and fails to distinguish between algorithms that have vastly differing performance in practice. As such competitive results should be used with caution when applied to real life scenarios. In some cases they seem to capture the essence of the problem, in others they give unrealistic and pessimistic evaluations of what are, in practice, rather good strategies.

As well, a common objection to competitive analysis is that it relies on an optimal offline algorithm, OPT, as a baseline for comparing online algorithms. While this may be convenient, it is rather indirect: one could argue that in comparing two online algorithms $\mathcal{A}$ and $\mathcal{B}$ all the information we should need is the cost incurred by the algorithms on each request sequence. For example, for some problems OPT is too powerful, causing all online algorithms to seem equally bad. Certain alternative measures allow direct comparison of online algorithms, for example the *Max-Max Ratio* [4], *Relative Worst Order Ratio* [5], Bijective Analysis and Average Analysis [3]. These measures have been applied mostly to the paging problem as well as some other online problems.

Such anomalies have led to the introduction of many alternatives to competitive analysis of online algorithms (see [6] for a comprehensive survey).

9 Alternate List Update Models

While list update algorithms with better competitive ratio tend to have better performance in practice the validity of the update model itself has been debated. More precisely, Martínez and Roura [10] and Munro [12], independently addressed the drawbacks of the standard cost model. Let $(a_1, a_2, \ldots, a_l)$ be the list currently maintained by an algorithm $\mathcal{A}$. Martínez and Roura argued that in a realistic setting a complete rearrangement of all items in the list which precede item a_i would in practice require time proportional to i, while this has cost proportional to i^2 in the standard cost model. Munro provided the example of accessing the last item of the list and then reversing the entire list. The real cost of this operation in an array or a linear link list should be $O(l)$, while it costs about $l^2/2$ in the standard cost model. As a consequence, their main objection to the standard model is that it prevents online algorithms from using their true power. They instead proposed a new model in which the cost of accessing the i^{th} item of the list plus the cost of reorganizing the first i items is linear in i. We will refer to this model as the *modified cost model* or MRM model. Surprisingly, it turns out that the offline optimum benefits substantially more from this realistic adjustment than the online algorithms do. Indeed, under this model, every online algorithm has amortized cost of $\Theta(l)$ per access for some arbitrary long sequences, while an optimal offline algorithm incurs a cost of $\Theta(\log l)$ on every sequence and hence all online list update algorithm have a constant competitive ratio of $\Omega(l/\log l)$. One may be tempted to argue that this is proof that the new model makes the offline optimum too powerful and hence this power should be removed, however this is not correct as in real life online algorithms can rearrange items at the cost indicated.

Interestingly enough, in the MRM model the offline optimum on a request sequence of length n can be computed in $O(n^2)$ time.

References

1. S. Albers. Improved randomized on-line algorithms for the list update problem. *SIAM Journal on Computing*, 27(3):682–693, June 1998.
2. S. Albers, B. von Stengel, and R. Werchner. A combined BIT and TIMESTAMP algorithm for the list update problem. *Information Processing Letters*, 56:135–139, 1995.

3. S. Angelopoulos, R. Dorrigiv, and A. López-Ortiz. On the separation and equivalence of paging strategies. In *Proceedings of the 18th ACM-SIAM Symposium on Discrete Algorithms (SODA '07)*, pages 229–237, 2007.
4. S. Ben-David and A. Borodin. A new measure for the study of on-line algorithms. *Algorithmica*, 11:73–91, 1994.
5. J. Boyar and L. M. Favrholdt. The relative worst order ratio for on-line algorithms. In *Proceedings of the 5th Italian Conference on Algorithms and Complexity*, 2003.
6. R. Dorrigiv and A. López-Ortiz. A survey of performance measures for on-line algorithms. *SIGACT News (ACM Special Interest Group on Automata and Computability Theory)*, 36(3):67–81, September 2005.
7. R. El-Yaniv. There are infinitely many competitive-optimal online list accessing algorithms. Manuscript, 1996.
8. G. H. Gonnet, J. I. Munro, and H. Suwanda. Towards self-organizing linear search. In *Proceedings of the 20th Annual IEEE Symposium on Foundations of Computer Science (FOCS '79)*, pages 169–174. IEEE, 1979.
9. S. Irani. Two results on the list update problem. *Information Processing Letters*, 38:301–306, 1991.
10. C. Martínez and S. Roura. On the competitiveness of the move-to-front rule. *Theoretical Computer Science*, 242(1–2):313–325, July 2000.
11. J. McCabe. On serial files with relocatable records. *Op. Res.*, 12:609–618, 1965.
12. J. I. Munro. On the competitiveness of linear search. In *Proceedings of the 8th Annual European Symposium on Algorithms (ESA '00)*, volume 1879 of *Lecture Notes in Computer Science*, pages 338–345, 2000.
13. D. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28:202–208, 1985.