On the Competitiveness of the Move-to-Front Rule*

Salvador Roura and Conrado Martínez†

January 14, 1997

Abstract

We consider the list access problem and show that one questionable assumption in the original cost model presented by Sleator and Tarjan [13] and subsequent literature allowed for several competitiveness results of the move-to-front rule (MTF). We present an off-line algorithm for the list access problem and prove that, under a more realistic cost model, no on-line algorithm can be c-competitive for any constant c, MTF included.

1. Introduction

For the last years there has been a growing interest in the competitive analysis of on-line algorithms. Good examples are the study of the list access and update problems, the k-server problem, the memory paging and dynamic allocation problems, etc. (see for instance [13, 1, 3, 4, 5, 6, 10])

Competitive analysis measures the quality of an on-line algorithm by comparing it against an optimal off-line algorithm. Recall that the goal of an *on-line algorithm* is to serve a sequence of requests (for instance, to retry information from given data structure), attending the requests one at each time, and without prior knowledge of the sequence of requests. In contrast, an *off-line algorithm* is one provided with full knowledge of the sequence of requests to be served. The performance of both on-line and off-line algorithms when attending a request depends not only on the request itself, but also on some sort of internal state of the algorithm that can be changed (at a certain cost) after the request has been attended. Off-line algorithms exploit full knowledge of the sequence of requests to evolve through a sequence of internal states while serving each request. An off-line algorithm is said to be *optimal* if it serves any sequence of request with minimal total cost. On the other hand, on-line algorithms are deprived of prior knowledge of future requests and should try to do their best to guess possible

*This research was supported by the ESPRIT BRA Program of the EC under contract no. 20244 (Project ALCOM-IT) and by a grant from CIRIT (Comissió Interdepartamental de Recerca i Innovació Tecnològica). Part of this work was done while the first author visited Princeton University.

†Departament de Llenguatges i Sistemes Informàtics, Universitat Politècnica de Catalunya, E-08028 Barcelona, Catalonia, Spain. E-mail: {roura,conrado}@goliat.upc.es

future requests and spot regularities and patterns that appear in the sequence, to keep the total cost to a minimum. An on-line algorithm is defined to be c-competitive if the total cost it incurs to serve any (sufficiently long) sequence of requests is, at most, c times the cost of serving the same sequence with an optimal off-line algorithm. The least such constant is called the *competitiveness ratio* of the on-line algorithm.

Another approach to the study of on-line algorithms is the so-called distributional analysis (also known as average-case analysis). Its goal is to compute the expected time to serve a request from a sequence of requests not known in advance to the on-line algorithm, under some distributional hypothesis. For instance, it is often assumed that requests are independently generated, so that a request of type i is generated with probability p_i irrespective of the time the request is made, and irrespective of previous or future requests. Typically, the on-line algorithm is compared against an off-line algorithm which knows the probabilistic properties of the source that generates the sequence of requests, but not the actual sequence. Many authors have claimed that competitive analysis provides stronger and more valuable results that distributional analysis since the former does not rely upon previous assumptions about the sequences of requests.

One of the earliest and most often cited results in this subject is due to Sleator and Tarjan [13]. One of the on-line algorithms that they studied is the move-to-front rule (MTF) for the list access and update problem in linked lists. The list access problem is to access each of the requested items in an unsorted linked list. An algorithm for the list access problem attends each request in the sequence by traversing the list until it finds the requested item. In order to minimize the total cost of attending the requests, the algorithm is allowed to reorganize the linked list, exchanging its items.

The MTF rule for the list access moves the last accessed (the last inserted) item to the front of the linked list; the rationale behind this rule is that recently accessed items will be probably requested in the near future. Several previous studies had shown that MTF performed very well in practice and in theory [2, 7, 12]. The work of Bentley and McGeoch [2] is an early attempt to explain the excellent performance of MTF. There, the authors show that the cost of attending any sequence of requests with MTF will cost at most twice the cost of an off-line algorithm that, before attending any of the requests, sorted the list according to its knowledge of the sequence of requests and made no reorganization afterwards.

The seminal paper of Sleator and Tarjan [13] provided an strong argument in favor of MTF's case, following an original approach. Under their model of costs, they proved that MTF is 2-competitive (or 4-competitive, depending on the initial assumptions). That means that MTF cannot be beaten by any other algorithm, even an off-line one, by more than a constant factor.

Sleator and Tarjan's work opened a vast new area, namely, the competitive analysis of on-line algorithms. A great number of related papers considered variants of the original list access and update problems, as well as improvements of the competitiveness ratio. Particularly successful is the use of randomization, giving 1.75-competitiveness (and even smaller competitiveness ratios) thus beating the lower bound of 2 for the competitiveness ratio of deterministic on-line algorithms [1, 8, 11].

However, as we will discuss afterwards, we think that the result of Sleator and Tarjan and its limitations are more often than not misunderstood. To the best of our knowledge, nobody has provided sound arguments for (nor against) the model of costs for the list update problem proposed by Sleator and Tarjan. A throughly justification for this model (we will call it the *basic cost model*, from now on) is missing in [13] and the subsequent literature. There have been several attempts to generalize some of the assumptions in the basic cost model, but all them maintain its fundamental “structure” and limitations.

Thus, a big misunderstanding has gone unnoticed for these past years. The statement “MTF is competitive under the basic cost model” has been loosely and inconciously replaced by “MTF is competitive”. It seems that it has been implicitly assumed that no off-line algorithm can beat MTF by more than a constant factor, for any reasonable cost model. Many times this idea is reflected by claims such as “knowledge of the future does not help”, in the context of the list access and update problems. However, as we shall show hereafter, it turns out that under a reasonable and realistic cost model, there cannot exist an on-line algorithm for the list access and update problems which is c-competitive, for any constant c. Furthermore, we claim that the basic cost model is interesting and helps explaining the properties of MTF, but it is rather unrealistic.

Our main objection to the basic model of costs is that the seemingly omnipotent off-line algorithm has to pay a prohibitive cost if it has to take advantage of its knowledge. In the basic cost model, it is not worth the effort to do anything but the same type of operations as MTF does, once a convenient ordering of the list has been achieved. Hence, the conclusion that MTF is competitive follows —basically, this is the result of Bentley and McGeoch— but we should be aware that MTF is competitive against a rather weakened off-line algorithm.

In other contexts, for instance, in the study of competitive on-line paging algorithms, restrictions on the almighty off-line algorithm/adversary are achieved by means of the so-called access graphs (see for instance [9, 3]), and the strenght and limitations of the results are fairly well-understood.

In next section we point out two reasons that make the basic cost model rather unrealistic, and prove a trivial $\Theta(n)$ lower bound over the worst-case amortized cost per access of any on-line algorithm (either deterministic or randomized). We prove the lower bound by considering sequences of n different accesses to a linked list of n items. In Section 3 we present an off-line algorithm that achieves $\Theta(\log n)$ amortized cost per access, when serving any sequence of n different requests and a more realistic model of costs is assumed. This implies that no on-line algorithm can be c-competitive for any constant c, including MTF.

2. THE COST MODEL

The basic cost model presented in [13] is as follows (we will only consider the static list model, i.e. when no updates are allowed). Accessing the i-th item costs i. Immediately after an access to the item i, we are allowed to move it at no cost to any position closer to the front of the list. This is called a *free exchange*. Any other update in the list must consist of exchanges of consecutive items, where each one costs 1 and is called a *paid exchange*.

The first thing one could argue is that the cost of traversing a link can differ from the cost of an exchange. This generalization has been already considered and it is not difficult to see that it does not yield significantly different conclusions.

On the other hand, it is clear that there cannot be free exchanges under a realistic model. However, while traversing the list in order to find the requested item it is simple to keep a pointer at some nearer position to the beginning of the list, and update the list at small constant cost. Again, considering that free exchanges have in fact some constant cost (say 1) does not affect the conclusions of [13] and many other papers on the subject. So this is not the point, either.

But there are two reasons that make the basic cost model not utterly realistic.

First of all, in practice, there is no way of exchanging two consecutive items if they have not been reached before. Hence, we cannot consider that exchanging two consecutive items has cost 1, unless they appear before the accessed item in the list. Otherwise, we should pay for the cost of accessing the items to be exchanged, plus one for the exchange itself.

Secondly, after reaching the currently requested item, any item before it in the list could be moved closer to the beginning of the list by means of a free (or constant time) exchange, for the same reason that we are allowed to use free exchanges with the requested item and move it to the front if we wish so. In other terms, the cost of moving any item before the accessed one j positions closer to the front of the list should not cost j units, but a constant amount of time.

The first of these two reasons does not affect the main conclusion about MFT's competitiviness. Considering more realistic costs for paid exchanges, as explained above, actually means that the off-line algorithm would have to pay more than it was charged by the basic cost model, while MTF's cost would remain the same as it does not use paid exchanges.

The second reason above turns out to change things dramatically. In next section we will see how a simple off-line algorithm can profit by these generalized free (constant time) exchanges to defeat any on-line algorithm (MTF included).

One simple way to state our objection to the basic cost model is that $\Theta(n^2)$ inversions can be removed in practice in linear time, once we allow for usual operations on linked lists and their costs are accounted in a realistic manner.

We end up this section proving a simple linear lower bound on the cost of any on-line algorithm (either deterministic or randomized). To prove this lower bound we will only consider sequences of n different requests over a list with n elements. This restriction will be enough for our purposes, and there is no loss of generality. In particular, the arguments that we give here can be easily generalized to sequences of length m consisting on m/n blocks of requests, each one a permutation of the n elements of the list.

Let $\mathcal{A}$ be any on-line algorithm that deals with a sequence of n distinct requests to access the n elements of a linked list. Consider the initial state from the point of view of an adversary trying to induce worst-case behaviour. Even if we only know the probability that each permutation has of being the current configuration of the list, it is very easy to see that we can always force, at least, $(n+1)/2$ expected comparisons/pointers traversed to serve a request — this happens when every item has the same expected search time; for instance, when all the permutations are equally likely. After that, the best thing to happen from $\mathcal{A}$'s point of view is that it moves the requested item to the end of the list, since this item will not be accessed any more. Now we are in the same situation as before, except that, from the point of view of the adversary, the list has $n-1$ elements. This argument shows that it is always possible to find a sequence of requests whose expected cost is $(n+1)/2+n/2+\cdots+1 = n(n+3)/4$. Therefore, $\mathcal{A}$ requires $\Theta(n)$ expected time per access to serve this sequence of requests.

3. THE OFF-LINE ALGORITHM

Roughly speaking, our off-line algorithm works as follows when given a sequence of n distinct requests to the elements of a list of size n. To serve the first request, the algorithm visits all the nodes of the list, irrespective of the requested item. Although the algorithm pays n at this step, it can save much of the future work by means of a simple strategy, which consists in splitting the original list into three sublists, one with the requested item, and two additional sublists with $(n-1)/2$ elements each. The first one of these sublists (*first*) contains the items that will be requested first, the other (*second*) the items that will be requested afterwards. The new list is build by joining *first*, *second* and the requested item in this order. This ensures that the following $(n-1)/2$ requests will hit elements in the first half of the list. Therefore, these requests can be recursively served by the same algorithm, but now over a sublist half the size of the original one. This is the point of the algorithm.

When all the elements in the first sublist have been already accessed, the next requests will be for the items in the second sublist. The first of these requests is served by

1. skipping *previous*, the sublist with the $(n-1)/2$ previously accessed items;
2. acting exactly as described for the general case but over the next $(n-1)/2$ items (that is, splitting this sublist into three sublists and joining them), and
3. joining the result of the step above with *previous*.

Again, the new list has at its front a sublist with the $(n-1)/2$ elements that are going to be immediately requested (except for the farthest of them, which

Stage	State of the list	Cost
1	10, 8, 3, 6, 15, 5, 1, 14, 11, 2, 9, 4, 7, 13, 12*	15
	8, 3, 6, 5, 2, 4, 7*, 10, 15, 14, 11, 9, 13, 12, 1	7
	3, 5, 4*, 8, 6, 7, 2, 10, 15, 14, 11, 9, 13, 12, 1	3
	4*, 5, 3, 8, 6, 7, 2, 10, 15, 14, 11, 9, 13, 12, 1	1
5	4, 5*, 3, 8, 6, 7, 2, 10, 15, 14, 11, 9, 13, 12, 1	2
	5, 4, 3, 8, 6, 7*, 2, 10, 15, 14, 11, 9, 13, 12, 1	6
	7*, 8, 6, 5, 4, 3, 2, 10, 15, 14, 11, 9, 13, 12, 1	1
	7, 8*, 6, 5, 4, 3, 2, 10, 15, 14, 11, 9, 13, 12, 1	2
	8, 7, 6, 5, 4, 3, 2, 10, 15, 14, 11, 9, 13, 12*, 1	14
10	10, 11, 12*, 15, 14, 13, 9, 8, 7, 6, 5, 4, 3, 2, 1	3
	11*, 12, 10, 15, 14, 13, 9, 8, 7, 6, 5, 4, 3, 2, 1	1
	11, 12*, 10, 15, 14, 13, 9, 8, 7, 6, 5, 4, 3, 2, 1	2
	12, 11, 10, 15, 14, 13*, 9, 8, 7, 6, 5, 4, 3, 2, 1	6
	14*, 15, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1	1
15	14, 15*, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1	2
	15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1	

Figure 1: An execution of the off-line algorithm. An asterisk (*) marks the farthest element to the right reached during the corresponding stage. The sequence of requests is $\sigma = 1, 2, \ldots, n$ with $n = 15$.

is the one just accessed). Hence, we can also use the algorithm recursively to serve the second sublist and end up servicing all the requests.

Let L be the original list to be accessed. For the sake of simplicity, let us assume that $n = 2^m - 1$ for some $m \geq 1$.

We give a recursive version of the off-line algorithm $\mathsf{serve}(\mathit{list},\ sz,\ \mathit{previous})$. The meaning of the parameters is as follows: *list* is a pointer to the beginning of the linked list, $sz = 2^k - 1$ is the number of items to be serviced by the current recursive call to serve, for some $1 \leq k \leq m$, *previous* is a pointer to a list of already serviced items that have been skipped and must be added after the first sz elements in *list*. The first call to serve is thus $\mathsf{serve}(L,\ n,\ \mathbf{NIL})$.

We assume that the sequence of requests is globally accesible to all the procedures below. Requests are attended one at a time and the global variable t, which is initialized to 1, indicates the next request to be attended.

A possible execution of this algorithm is given in Figure 3. Each row reflects the order of the items in the linked list in each stage, the initial row being the initial state of the list and each successive row showing a step of the process. In the example, we assume that $\sigma = 1, 2, 3, \ldots, n$; with $n = 15$. The asterisks mark the farthest reached item in each step, and the last column keeps track of the cost of each stage, in this case, the number of visited items.

The analysis of the cost of this algorithm is quite simple. Let $T(k)$ be the number of visited items while serving a sequence with $2^k - 1$ requests. By definition, $T(1) = 1$. Furthermore, for every $k \geq 2$ we have that

$$T(k) = (2^k - 1) + T(k-1) + (2^{k-1} - 1) + T(k-1) = 3 \cdot 2^{k-1} - 2 + 2T(k-1).$$

A simple proof by induction yields $T(k) = 3k \cdot 2^{k-1} - 2^{k+1} + 2$. We can use it to compute the amortized time to serve a request in the whole list as

$$\frac{T(m)}{n} = \frac{3m}{2} - 2 + \frac{3m}{2^{m+1} - 2} = \frac{3}{2}\log_2 n + \mathcal{O}(1) = \Theta(\log n).$$

Finally and to be completely rigorous, we should mention that we have not shown which is the cost of deciding whether an item must be attached to *first* or *second*, while we are classifying the list. But we have not to, since internal computations made by off-line algorithms are not usually taken into account, only the cost of actually serving requests and reorganizing the list have to be considered. But even if we took these costs into account, it is not very difficult to show that the amortized cost per access is still $\mathcal{O}(\log n)$ for our off-line algorithm.

4. Conclusions

We have considered the list access problem and proved that, under a realistic cost model, no on-line algorithm can be c-competitive. In particular, we have shown that the result about the 2-competitiveness of MTF follows from the high costs associated in the basic cost model to the operations that enable an off-line algorithm to take advantadge of its full knowledge of the sequence of requests. Informally stated, Sleator and Tarjan's result is that MTF is competitive against

Algorithm 1 The off-line algorithm.

```
serve(list, sz, previous) {

  classify(list, sz, first, second, current, rest);
  // Traverse the first sz items in list.
  // Build three new lists by adding elements at their ends:
  //      first:   contains the (sz - 1) / 2 items which,
  //               according to σ, will be accessed first;
  //      second:  contains the (sz - 1) / 2 items that
  //               will be accessed afterwards;
  //      current: contains the currently requested item.
  // Also, return rest, a pointer to the sz + 1-th item in list.

  attend(current); t = t + 1;
  // Service the request to the item pointed to by current.

  join(first, second, current, previous, rest, list);
  // Concatenate the lists first, second, current, previous and rest,
  // in this order, and return list, a pointer to the head of the so
  // constructed linked list.

  if (sz == 1) return;

  serve(list, (sz - 1) / 2, NIL);

  chop(list, (sz - 1)/2 , previous);
  // Traverse (sz - 1)/ 2 items in list.
  // These items form a new linked list, called previous.
  // After the call, list points to the beginning of the remaining
  // elements.

  serve(list, (sz - 1)/2, previous);
}
```

off-line algorithms that are only allowed to sort the list at the very beginning and eventually make local reorganizations of the list on-the-fly, not extensive reorganizations that would incur in extremely high costs according to the basic cost model.

We think that our contribution is twofold. On the one hand, we can conclude that the knowledge of the future sequence of requests indeed improves the average time per access. On the other, although competitive analysis is a worthwhile technique for the study of on-line problems, more attention should be paid to the assumptions upon which competitiviness results are based.

The new open problem is thus if there exists any on-line algorithm that using non-local exchanges would do much better than MTF. Our conjecture is that such an algorithm does not exist. Basically, we conjecture that no on-line algorithm could take much advantage of moving non-requested items far away from their positions towards the front of the list. Essentially, there would exist sequences such that MTF is much better than the hypothetical on-line using non-local exchanges and viceversa, making them incomparable from the point of view of competitiveness.

ACKNOWLEDGEMENTS

We would like to thank Josep Díaz and Robert Tarjan for useful comments and suggestions on the preliminary versions of this work.

REFERENCES

[1] S. Ben-David, A. Borodin, R. Karp, G. Tardos, and A. Wigderson. On the power of randomization in on-line algorithms. *Algorithmica*, 11(1):2–14, 1994.

[2] J.L. Bentley and C. C. McGeoch. Amortized analyses of self-organizing sequential search heuristics. *Communications of the ACM*, 28(4):404–411, 1985.

[3] A. Borodin, S. Irani, Raghavan, and Schieber. Competitive paging with locality of reference. In *ACM Symposium on Theory of Computing (STOC)*, 1991.

[4] A. Borodin, N. Linial, and M. Saks. An optimal on-line algorithm for metrical task systems. *Journal of the ACM*, 39(4):743–763, 1992.

[5] A. Fiat, R.M. Karp, M. Luby, L.A. McGeoch, D.D. Sleator, and N.E. Young. Competitive paging algorithms. *Journal of Algorithms*, 12:685–699, 1991.

[6] A. Fiat, Rabani, and Ravid. Competitive k-server algorithms. In *IEEE Symposium on Foundations of Computer Science (FOCS)*, 1990.

[7] G. Gonnet, J.I. Munro, and Suwanda. Exegesis of self-organizing linear search. *SIAM Journal on Computing*, 1981.

[8] S. Irani. Two results on the list update problem. *Information Processing Letters*, 38:301–306, 1991.

[9] S. Irani, A.R. Karlin, and S. Phillips. Strongly competitive algorithms for paging with locality of reference. *SIAM Journal on Computing*, 25(3):477–497, 1996.

[10] M.S. Manasse, L.A. McGeoch, and D.D. Sleator. Competitive algorithms for server problems. *Journal of Algorithms*, 11:208–230, 1990.

[11] N. Reingold, J. Westbrook, and D.D. Sleator. Randomized competitive algorithms for the list update problem. *Algorithmica*, 11(1):15–32, 1994.

[12] Rivest. On self-organizing sequential search heuristics. *Communications of the ACM*, 19, 1976.

[13] Sleator and Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985.