Theoretical Computer Science 130 (1994) 101–123
Elsevier

The list update problem and the retrieval of sets*

Fabrizio d'Amore

Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", via Salaria 113, 00198 Roma, Italy

Vincenzo Liberatore**

Department of Computer Science, Hill Center for the Mathematical Sciences, Rutgers University, New Brunswick, NJ 08903, USA

Abstract

d'Amore F. and V. Liberatore, The list update problem and the retrieval of sets, Theoretical Computer Science 130 (1994) 101–123.

We consider the list update problem under a sequence of requests for *sets* of items, and for this problem we investigate the competitiveness features of two algorithms. We prove that algorithm Move-Set-to-Front (MSF) is $(1+\beta)$-competitive, where β is the size of the largest requested set, and that a lower bound is roughly 2. We also provide an upper bound to the MSF competitive ratio by relating it to the size n of the list, showing that it is $(1+n/4)$-competitive in general, and $O(\sqrt{n})$-competitive with a small constraint to the size of the requested sets.

Moreover, we prove that the randomized algorithm BIT-for-Sets is $(1+\frac{3}{4}\beta)$-competitive against an oblivious adversary.

Also, we study two extensions. The first one generalizes the list update problem under a sequence of requests of sets by considering weighted lists, where a weight representing a visiting cost is associated with each item. For this case we give a competitiveness result as well.

The second one is a variant, where the list is searched to retrieve whichever element of the currently requested set (the first that can be found in the list). For this problem we provide negative results.

Correspondence to: F. d'Amore, Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", via Salaria 113, 00198 Roma, Italy. Email: damore@dis.uniroma1.it.

* Work partially supported by the ESPRIT II Basic Research Actions Program Project no. 3075 "ALCOM", and by the Italian MURST National Project "Algoritmi, Modelli di Calcolo e Strutture Informative". A preliminary version of this paper appeared in [4].

** This research has been carried out while this author was at Unversità di Roma "La Sapienza", Roma, Italy.

SSDI 0304-3975(94)00010-G

1. Introduction

In the last two decades the list update problem has been studied under several formulations and different aspects. First, the problem of studying the performance of linear search under self-adjusting algorithms was faced (a wide bibliography has been given by Hester and Hirschberg in [7]). Then, Sleator and Tarjan formalized such issues into the list update problem [16] by providing a suitable cost model. In that article the authors studied the problem by means of amortized analysis. Furthermore, they made use of a "comparative" technique, actually one of the first competitive analyses, even if at that time the term "competitive" [10] had not been introduced yet. Such a technique results of practical use in many cases, and has been often exploited in more recent papers dealing with on-line algorithms.

Given a sequentially accessed list, the traditional list update problem consists of rearranging its items during the processing of a sequence of operations on the list itself, in order to minimize the processing costs of subsequent operations. So, the very question is finding on-line rearrangement rules (also called permutation algorithms [7]) with high performance.

For the list update problem, it has been shown [16] that algorithm Move-to-Front (MSF) (after accessing an item, move it to the front of the list without changing the relative ordering of the other items) is 2-competitive against the optimum off-line algorithm, and that no deterministic on-line algorithm can achieve a better competitive ratio [11, 8]. For the same problem randomized algorithms have been presented too [9]. For example BIT (a randomly initialized bit is associated with any item: complement this bit whenever the item is accessed, and if an access causes the bit to be changed to 1, then move the item to the front of the list) has been shown to be 1.75-competitive. In [19] it has been recently proved that no randomzed on-line algorithm can achieve a competitive ratio better than 1.5.

Some generalizations of the traditional model [2, 16] have been considered, such as the weighted list [5] and the paid exchange (P^d) models [14, 9]. Such models differ from the traditional one for the cost of the operations that can be performed on the list. In the weighted list model, the cost of inspecting a list item is a positive value that depends on the item itself. In contrast, in the P^d model the access cost is the same as in the standard model, but there are no free exchanges and the cost of rearrangement is scaled up by some arbitrarily large value d; in practice each paid exchange has cost d. For these generalization several algorithms, with different competitiveness features, have been provided.

In this paper we are concerned with the generalization to the case where the list is searched to retrieve *sets* of elements rather than just one item at a time. We study this problem under two different (but related) cost models: the *standard* one [16], and the *wasted work* one [2, 9]. While in the latter the cost of finding the ith item of the list is given by that of accessing the $i-1$ preceding items (i.e., only costs of unsuccessful comparisons are charged), in the former all costs are computed, including that of examining the searched item. To the best of our knowledge, this generalization has not been investigated till now.

Also in this case we are mainly interested in designing efficient permutation algorithms, with the goal of minimizing the overall cost of processing a sequence of requests.

Our research is motivated by two main reasons: the importance of the generalization itself; and the relation among the list update problem, its generalizations, and several on-line problems.

First, broadly speaking list update problems are tools that can be used to analyze algorithms for several on-line problems, such as the scheduling of the operations of a sequential machine. Weighted lists can model problems where different operations have different costs, such as in the case of visiting a tree by depth first searches [5]. Lists with search for sets are useful for modeling situations where several operations can be executed in an unordered fashion (without constraints that force a certain operation to precede some others).

Second, the list update problem is the simplification and the archetype of several important problems on dynamic data structures. Consider the problem of repeatedly accessing a data structure to find some items. To give an efficient solution to the access problems, the requested elements should be placed in positions of the structure where they can be located by means of few operations.

For example, consider the problem of maintaining a binary search tree under a sequence of accessess to the items in the tree; the cost is proportional to the sum of the lengths of the paths between the root and the accessed nodes. To solve the problem of dynamic search trees, the splay tree structure has ben proposed and analyzed by Sleator and Tarjan [17]: a splay tree moves each accessed item to the root by means of a specific set of "splaying steps".

Several striking similarities give insight on the relation between list update and binary search trees: in both structure there is an entry point (the front of the list and the root of the tree, respectively), and the costs are given by the distance from the entry point and the location of the accessed item, plus the rearrangement costs. There are somewhat similar competitive algorithm for the two problems, as both Move-to-Front and Splay move the accessed item to the entry point. However, despite the similarities, there is a major difference about what is known of lists and search trees: Move-to-Front is 2-competitive against any adversary, whereas Splay has been proved to be competitive only when compared with static algorithms.

In general, there is still a remarkable gap between known results for lists and known results for more complex structures for which competitive algorithms are known only in a particular setting. A motivation for our work is to consider a problem that appears to be more complex than the list update and simpler than several others. We have chosen to augment the traditional list update problem in a very particular way, that is, by allowing requests for sets; then, we have examined to what extent the known results could be generalized.

In this paper we prove that the deterministic algorithm Move-Set-to-Front is $(1+\beta)$-competitive, where β is the maximum size among all the requested sets. The proving technique provides innovative tools that can be useful in a wide range of

applications. Some similar ideas were used by Grove [6] and by Chrobak and Larmore [3] while dealing with the k-server problem [13].

Moreover, we show that the randomized algorithm BIT-for-Sets is $(1+\frac{3}{4}\beta)$-competitive

Two variants of the problem are also studied. In the first one the list is weighted: for this case, under some restrictive hypothesis, algorithm Move-Set-to-Front is still competitive. In the second one, the list is searched to retrieve any item of the requested set: for this case we prove that all the algorithms belonging to a general class are not competitive.

The rest of the paper is organized as follows. In the next section some basic concepts and notations are introduced. In Section 3 we provide upper and lower bounds. In Section 4 we study algorithms Move-Set-to-Front and BIT-for-Sets, proving their competitiveness. In Section 5, we consider two variants: first, weighted lists, and, second, the case where the list is searched to retrieve whichever element of the requested set (the first that is found). For the last problem we provide negative results. Finally, in Section 6 we draw some conclusions and address future work and open questions.

2. Preliminaries

The *list update problem with retrieval of sets* consists of storing the items of a set $\mathscr{S}$ as an unsorted list $\mathscr{L}$ which only supports sequential access, and of on-line updating the list while serving a sequence $\underline{r}=(r_1, \ldots, r_m)$ of requests. Each request consists of finding, inserting or deleting a *set* of items, i.e. a subset of $\mathscr{S}$.

We are interested in developing on-line permutation algorithms for updating the list in order to process $\underline{r}$ efficiently [7]. Update operations consist of inserting a new item, deleting an item, or rearranging the list after any access.

We denote by $\mathscr{L}^j$ the list after jth request has been served and possible consequent rearrangements have been carried out. If x and y are items of $\mathscr{L}^j$ we write that $x \prec^j y$ (x *precedes* y) to denote that x is stored in $\mathscr{L}^j$ before y (however we omit the superscript j when ambiguities do not arise).

Since both insertions and deletions can be handled as special cases of access operations [14] we consider sequences $\underline{r}$ that only consist of accesses to a list of a fixed size n. Consequently, the generic request r_j can be expressed as a subset of $\mathscr{S}$. In order to serve request r_j, list $\mathscr{L}^{j-1}$ is sequentially searched to find *all* the items belonging to r_j.

Henceforward, we denote by β_j the size of r_j. A fundamental quantity is β, defined as $\max_{1 \leq j \leq m}\{\beta_j\}$. β is a "physical" constant of our system and characterizes the real world process we are modeling. Another interesting value is the *average* (averaged on the sequence) value of the sizes, defined as $\beta_{\text{av}}=\sum_{j=1}^{m} \beta_j/m$.

For the list update problem with retrieval of sets we consider two cost models. With regard to the special case in which r_j is a singleton, say the ith item of the list, the cost

of finding r_j is i in the *standard model* and $i-1$ in the *wasted work model*. So, in the former we consider all the cost paid to answer a request, while in the latter we do not take into account "useful" costs.

If r_j is not a singleton, then the cost of retrieving all its elements is *not* the sum of all the costs which we would pay by separately searching all these elements as singletons. In fact, we assume that a special function is available:

$$\theta_{\mathscr{L}}: \mathscr{P}(\mathscr{S}) \times \mathscr{S} \to \{\text{no, yes\_continue, yes\_stop}\},$$

where $\mathscr{P}(\mathscr{S})$ is the power set of $\mathscr{S}$. $\theta_{\mathscr{L}}$ is the function that permutation algorithms use to test whether the item x currently under inspection belongs to the answer. If x is the item currently inspected in the list $\mathscr{L}$ we have:

- $\theta_{\mathscr{L}}(r_j, x) = \text{yes\_continue}$ if $x \in r_j$ and there is some other item in r_j to be found in $\mathscr{L}$;
- $\theta_{\mathscr{L}}(r_j, x) = \text{yes\_stop}$ if $x \in r_j$ and all the elements in r_j have been already found; and, finally,
- $\theta_{\mathscr{L}}(r_j, x) = \text{no}$ if $x \notin r_j$ (obviusly, in this case the search continues).

For now we assume that the cost of computing function $\theta_{\mathscr{L}}$ is 1, but later on we renounce this hypothesis and examine two more general cases.

Hence, in the standard model, the cost of retrieving r_j is the cost l_j of finding the element of r_j that in $\mathscr{L}$ is preceded by all the other elements of r_j. In the wasted work model, the cost is $l_j - \beta_j$, according to the spirit of not considering useful costs.

Permutation algorithms are allowed to exchange adjacent items. There are two kinds of exchanges, free and paid ones. The elements recognized as belonging to r_j can be moved by mens of *free* exchanges closer to the front of the list (at no cost) without changing their relative ordering. All other exchanges are *paid* and cost 1.

The generalized list update problem reduces to the well-known list update problem [16] when all requests are singletons. In this case, the cost models match those presented in [16] (standard) and [2, 9] (wasted work).

In the case of *weighted* lists [5] there is a cost function $w: \mathscr{S} \to \mathbb{R}^+$ which expresses the cost $w(x)$ that $\theta_{\mathscr{L}}$ is charged for the inspection of item x. However, in what follows, if a precise specification is missing, we refer to unweighted lists.

Typically, the performance of an on-line update rule is analyzed in comparison to that of an off-line algorithm, i.e. an algorithm that is able to perform some optimizations by virtue of its knowledge about future requests. In practice, in order to make the comparison, it is convenient to introduce the concept of *adversary*, a person that has the power of *creating* the sequence $\underline{r}$ to be served by the on-line algorithm. The adversary can also make use of an off-line algorithm H which can take advantage of the knowledge of $\underline{r}$. Of course, the optimal off-line algorithm is the one which minimizes the overall cost of processing $\underline{r}$.[1]

Let α be a function $\alpha: \mathbb{R} \to \mathbb{R}$ of the type $\alpha(x) = c \cdot x + e$. A deterministic on-line algorithm G is said to be α-competitive against the adversary that uses algorithm H if

[1] In the rest of the paper, for simplicity, we often identify the adversary with its own algorithm.

for any sequence $\underline{r}$ $G(\underline{r}) \leqslant \alpha(H(\underline{r}))$, where $G(\underline{r})$ and $H(\underline{r})$ are the costs respectively paid by G and H for serving $\underline{r}$ [10, 1]. The constant e does not depend on the sequence $\underline{r}$ but only on the handled list [8].

If H is the static algorithm, namely the one that does not actually rearrange the list (so it never changes the initial arrangement), then the adversary that uses H is *static*. Among all the initial orderings for the items of the list there is the *optimal static* one, i.e. the one that minimizes the overall cost of processing $\underline{r}$ without rearranging the list.

If G is a randomized algorithm, in order to analyze its performance it makes sense to refer to different kinds of adversaries, that differ on the basis of their actual power [1]. The *oblivious* adversary is required to generate $\underline{r}$ before G begins working: this is the *weak* adversary since it cannot see the random choices made by G. Thus, intuitively, randomization can help against the oblivious adversary.

Two more kinds of adversaries are *adaptive* and are more powerful since they are allowed to generate the next request after G has served the current one: in this way, they can force G to pay a higher cost. If an adaptive adversary uses an on-line algorithm it is said to be *medium*; if it is allowed to use the optimum off-line algorithm it is said to be *strong*.

G is said to be α-competitive against an oblivious adversary if $\mathrm{E}\{G(\underline{r})\} \leqslant \alpha(H(\underline{r}))$, where $\underline{r}$ is generated by the adversary and $\mathrm{E}\{G(\underline{r})\}$ is the cost of G averaged over all the random choices that G makes while processing $\underline{r}$ [12, 1]. The definition of competitiveness against the other kinds of adversaries is similar, but requires the expectation of the cost of H and a careful specification of the way of generating $\underline{r}$.

In practice, both deterministic and randomized α-competitive algorithms are said to have competitive ratio α or to be α-competitive [12].

Randomization cannot help against strong adversaries [1], and, for the list update problem, not even against medium adversaries [9].

In order to compare the performance of two given algorithms a common technique consists of using a *potential function* [15] whose value at any time measures a suitable quantity which reflects the difference between the states of the list handled by the two algorithms [9, 16].

Finally, concerning the cost models we adopted, it is worth observing that if we supposed that free exchanges could be used to modify the relative ordering of the items belonging to r_j, the adversary would unfairly profit by this. In fact, while in the model we adopted the adversary is charged $O(n^2)$ for any total reorder, and on-line algorithms have no cost, in the model with free rearrangements, the adversary, whenever he wishes to completely reorganize the list at no extra cost, can request $r_j = \mathscr{S}$, and for this request both the adversary and the on-line algorithm are charged the same cost. In addition, we should have consequently assumed null the rearrangement cost paid by the adversary, even for very large requested sets.

However, in Section 4 (Corollary 4.3) we show that there exists an on-line algorithm that, in the standard model, is so powerful that one can allow the adversary to reorganize the list by free exchange without worsening the competitive ratio.

3. Lower and upper bounds

In this section we provide a lower and an upper bound to the competitive ratio, which hold for any on-line deterministic algorithm under no restrictive hypothesis.

It is well-known that for the traditional list update problem the optimal static ordering is any ordering according to nonincreasing frequencies (NIF) of access [2, 7]. However, this result does not hold in the case we are studying, as the following counterexample shows.

Let $\mathscr{S}$ be $\{a, b, c, d\}$, and $\underline{r}=(\{a\}, \{a, c\}, \{a, c\}, \{b\}, \{b, d\}, \{b, d\})$. According to a NIF ordering a and b precede c and d and then a total cost of 17 is charged (in the standard model) for processing $\underline{r}$. On the other side, the (non-NIF) static ordering $(a\,c\,b\,d)$ has a total processing cost equal to 16.

The lower bound against an adversary that uses a static NIF ordering algorithm is roughly 2, as the following theorem states.

Theorem 3.1. *For any list $\mathscr{L}$ of n items, for any β_{av}, and for any deterministic on-line algorithm G, the competitive ratio of G cannot be less than $2n/(n+\beta_{\mathrm{av}})$ in the standard model and cannot be less than 2 in the wasted work one.*

Proof. The proof follows the scheme provided by Karp and Raghavan [11]. We refer to a static adversary.

Of course, $1 \leqslant \beta_{\mathrm{av}} \leqslant n$. If $\beta_{\mathrm{av}}=1$, then the proof reduces to that in [11]. Now suppose that $\beta_{\mathrm{av}}>1$.

Let $\mathscr{L}_G$ and $\mathscr{L}_H$ be the list maintained by G and H, respectively.

Let $\beta'=\lfloor \beta_{\mathrm{av}} \rfloor$, and choose the nonnegative integers m, h_1 and h_2 so that $m=h_1+h_2$. The adversary builds a sequence of m requests by concatenating h_1 requests of size β' and h_2 requests of size $\beta'+1$. It is easily seen that h_1 and h_2 can always be chosen so that for any given β_{av} it holds $\sum_{j=1}^{m} \beta_j/m=\beta_{\mathrm{av}}$. If β_{av} is an integer then h_2 is set to 0.

In order to create the sequence the adversary selects in whatever fashion a set B of $\beta'-1$ items and an item y not belonging to B. Let x_{j-1} be the last item of $\mathscr{L}_G^{j-1}$. Request r_j, for $1 \leqslant j \leqslant h_1$, is given by

$$r_j=\begin{cases} B \cup \{x_{j-1}\} & \text{if } x_{j-1} \notin B, \\ B \cup \{y\} & \text{if } x_{j-1} \in B. \end{cases}$$

Thus the first h_1 requests have size β'.

Let w be the most frequently requested item not belonging to B among the first h_1 requests. Now, in order to generate the subsequent requests, the adversary arbitrarily chooses an item $z \notin B \cup \{w\}$. Request r_j, for $h_1+1 \leqslant j \leqslant m$, is

$$r_j=\begin{cases} B \cup \{w, x_{j-1}\} & \text{if } x_{j-1} \notin B \cup \{w\}, \\ B \cup \{w, z\} & \text{if } x_{j-1} \in B \cup \{w\}. \end{cases}$$

Hence the last h_2 requests have size $\beta'+1$.

The list of the adversary is arranged according to the NIF of access to the items. Therefore in the first $\beta' - 1$ positions there are the elements of B, at position β' there is item w, and in the rest of the list there are the remaining items, according to the NIF.

Now, in the spirit of [11], we claim that to serve the sequence the adversary pays, in the standard model, at most $\sum_{j=1}^{m}(n+\beta_j)/2$. In this expression the quantity $(n+\beta_j)/2$ is the mean of the maximum and minimum costs the adversary can incur to serve r_j. Analogously, the cost in the wasted work model is $\sum_{j=1}^{m}(n-\beta_j)/2$.

The cost of G is patently nm in the standard model and $\sum_{j=1}^{m}(n-\beta_j)$ in the wasted work one.

The cost ratios can be directly computed and they turn out to be at least

$$\frac{2nm}{\sum_{j=1}^{m}(n+\beta_j)/2},$$

and 2 in the standard and in the wasted work model, respectively. □

The lower bound in Theorem 3.1 reduces to previous results [11] if $\beta_{av}=1$.

Concerning the upper bound, the following theorem gives a simple but general result.

Theorem 3.2. *Let G be a deterministic on-line algorithm that never makes paid exchanges. Then in the standard model G is ζ-competitive where $\zeta \leqslant n/\beta_{av}$.*

Proof. In the worst case for G, the adversary requests at each step a set in which one element is in the last position in $\mathscr{L}_G$, while the whole set is located in the first β_j positions in the adversary's list.

Since G does not make paid exchanges, its global cost is nm; on the other hand the cost of the adversary is $\sum_{j=1}^{m}\beta_j$. □

All the bounds above are illustrated in Figs. 1 and 2.

4. Algorithms

In this section, we study two algorithms for updating lists. The algorithms are simple to define and work on-line; in particular, they do not require the knowledge of the maximum size β of the sets that will be requested. Nonetheless, the competitive ratio of both the algorithms is actually a function of β.

The first algorithm generalizes the well-known Move-to-Front, and we name it Move-Set-to-Front (MSF). It consists of moving to the front of the list any accessed set of items, without changing either their relative ordering or that of the other items.

We prove that MSF is $(1+\beta)$-competitive against the optimum off-line algorithm, and make the comparison on the basis of a potential function that depends on the number of inversions between the lists handled by the two algorithms.

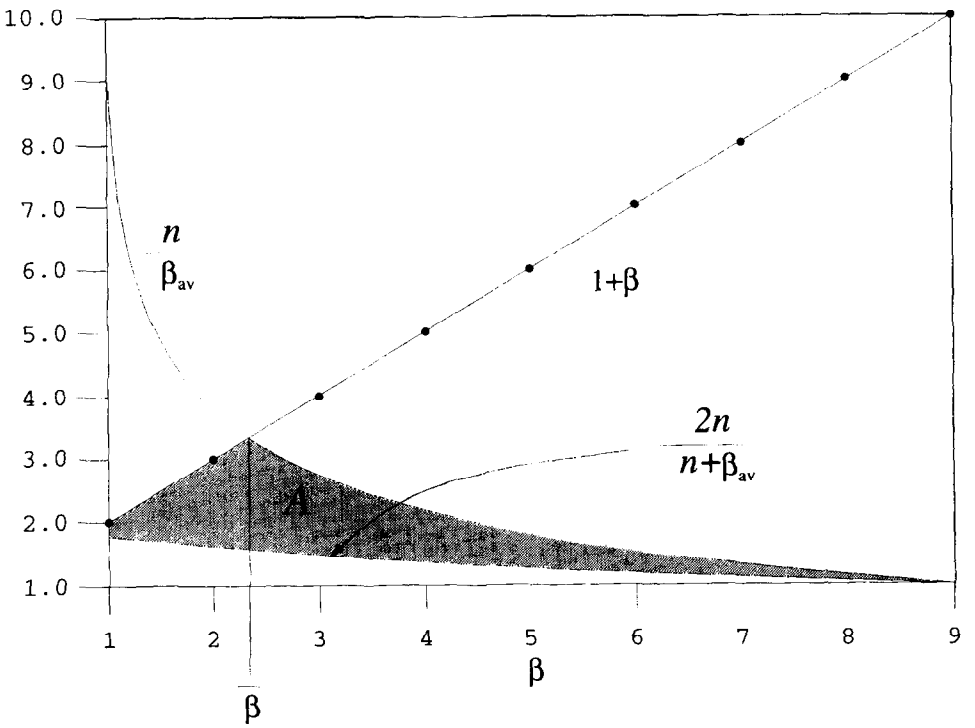

Fig. 1. Upper and lower bounds in the standard model. Curves drawn for $n=9$ and $\beta_j=\beta_{av}$, for any j.

Let τ be a permutation of $\mathscr{S}$, and let ι be the identity permutation, i.e. the one such that $\iota(x)=x$, for any $x\in\mathscr{S}$.

Now we give a definition of *inversion* which generalizes that provided by Sleator and Tarjan [16]. Given a permutation τ and two lists $\mathscr{L}_H$ and $\mathscr{L}_G$, an inversion is an ordered pair $(y, x)\in\mathscr{S}^2$ such that $\tau(x)\prec\tau(y)$ in $\mathscr{L}_H$ and $y\prec x$ in $\mathscr{L}_G$. When $\tau=\iota$, the usual definition of inversion is obtained. Note that the number of inversions depends on the order the two lists are considered and on the used permutation, and two different permutations τ_1 and τ_2 generally yield different results.

A similar idea is used by Grove [6] and by Chrobak and Larmore [3] in the context of the k-server problem. In fact, they make use of a function that maps the servers of the on-line algorithm to those of the adversary. Such a function is then used while computing changes in the value of the potential due to movements of the servers. In a way, this is the same use we make of τ, since both the mapping functions allow to take into account "spatial" properties that fully characterize the dynamics of the events that occur in the systems under inspection.

Throughout our analysis we make use of the concept of *current* permutation, i.e. the permutation which we are referring inversions to. While changing the current permutation from τ_1 to τ_2, a variation in the number of inversions occurs. Let $h(x)$ be the number of items that precede x in $\mathscr{L}_G$ and are located between $\tau_1(x)$ and $\tau_2(x)$ in $\mathscr{L}_H$.

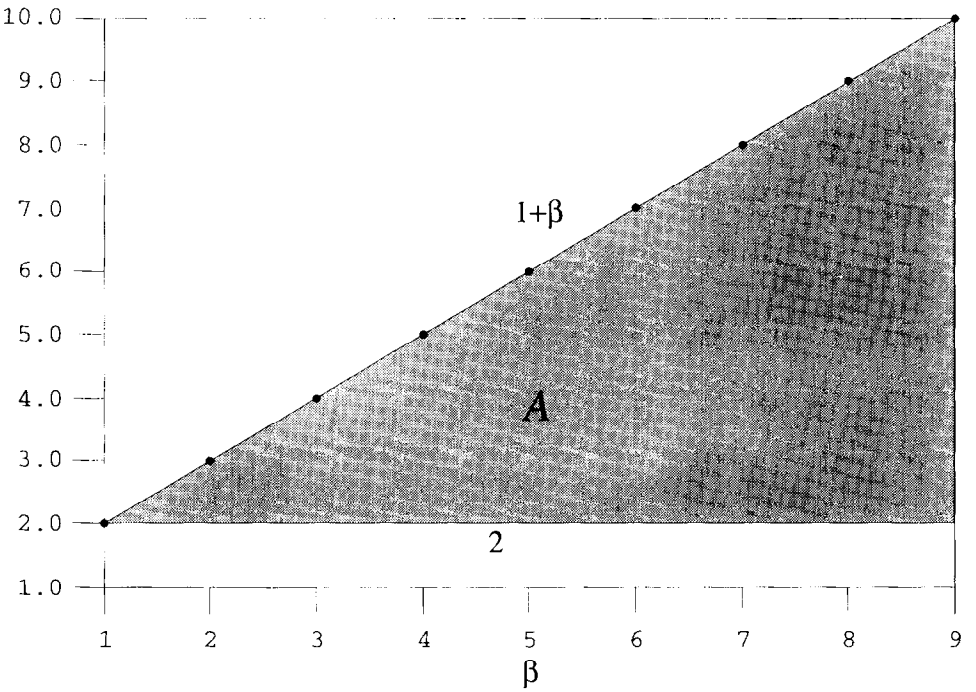

Fig. 2. Upper and lower bounds in the wasted work model. Curves drawn for $n=9$ and $\beta_j=\beta_{av}$, for any j.

In a similar way, let $t(x)$ the number of items that follow x in $\mathscr{L}_G$ and are located between $\tau_1(x)$ and $\tau_2(x)$ in $\mathscr{L}_H$. When the current permutation passes from τ_1 to τ_2, x is said to move *forward* if $\tau_2(x) \prec \tau_1(x)$, or *backward* if $\tau_1(x) \prec \tau_2(x)$.

It is easily seen that, for any item x, if we only consider its movement while the rest of the list remains fixed, the following holds:

Lemma 4.1. *For any fixed x, the increase in the number of inversions of type (x, y) plus those of type (z, x), for any y and z belonging to $\mathscr{S}$, is given by*

$$\begin{cases} h(x)-t(x) & \text{if } x \text{ moves forward,} \\ t(x)-h(x) & \text{if } x \text{ moves backward.} \end{cases}$$

Proof. Suppose that x moves forward. If some items precede x in $\mathscr{L}_G$ and follow $\tau_2(x)$ in $\mathscr{L}_H$, by definition, after the movement of x, they give rise to one inversion each; those items can be divided among those that follow $\tau_1(x)$, and those that are located between $\tau_2(x)$ and $\tau_1(x)$. The items that follow $\tau_1(x)$ brought about one inversion each also before x moved forward; so, they do not cause any change of the number of inversions. The items that are between $\tau_2(x)$ and $\tau_1(x)$ give rise to one new inversion each: since by definition there are $h(x)$ such items, after x has been moved there are $h(x)$ inversions.

However, when x moves, some items cease causing an inversion. Consider the $t(x)$ items that followed x in $\mathscr{L}_G$ and are placed between $\tau_2(x)$ and $\tau_1(x)$ in $\mathscr{L}_H$: they gave rise to inversions that do not exist any longer when x is moved forward.

On the whole, the net increase in the number of inversions is $h(x)-t(x)$.

The proof for x moving backward is similar. □

Now, regarding to the problem of updating a list under sequences of requests of sets, we prove that MSF is competitive with respect to any algorithm H.

Theorem 4.2. MSF *is* $(1+\beta)$*-competitive both in the standard and in the wasted work model.*

Proof. First of all, we introduce some symbols. Let $\mathscr{L}_H$ and $\mathscr{L}_{\text{MSF}}$ be the lists updated by H and MSF, respectively. Let us assume that the jth request consists of set r_j, and let us denote its elements by $a_1, \ldots, a_{\beta_j}$, ordered according to the ordering of $\mathscr{L}_H$. Furthermore, for our convenience, we denote by $b_1, \ldots, b_{\beta_j}$, the same elements ordered according to the ordering of $\mathscr{L}_{\text{MSF}}$.

Let τ_j be the following permutation:

$$\begin{cases} \tau_j(x)=a_l & \text{if } x\in r_j \text{ and } x=b_l, \\ \tau_j(x)=x & \text{if } x\notin r_j. \end{cases}$$

Permutation τ_j leaves in their place items not belonging to r_j and rearranges those belonging to r_j in such a way that inversions with respect to ι between two items both belonging to r_j are not inversions with respect to τ_j. In other words, the ordering of r_j obtained by applying τ_j to $\mathscr{L}_{\text{MSF}}$ is the same as that in $\mathscr{L}_H$.

At any time the current permutation is an element in $\{\tau_j \mid 1\leqslant j\leqslant m\}\cup\{\iota\}$. In what follows an inversion is always meant with respect to the current permutation, $\mathscr{L}_{\text{MSF}}$ and $\mathscr{L}_H$. At the very beginning the current permutation is ι.

The proof makes use of a potential function, which is defined as the number of inversions between $\mathscr{L}_{\text{MSF}}$ and $\mathscr{L}_H$. The proving technique basically consists of considering a current permutation, which we modify in order to more easily evaluate the amortized costs associated with the various operations made by the two algorithms.

So, we calculate the variations in the number of inversions due to changes in $\mathscr{L}_{\text{MSF}}$, in $\mathscr{L}_H$ (as usual, see [16, 9]), and in the current permutation.

In order to analyze and compare the behavior of the two algorithms, we consider the sequence of operations, made by two algorithms, to serve the jth request. We may assume they are made in the following order:

(1) H makes its paid exchanges;

(2) MSF and H access the elements in r_j by means of function $\theta_{\mathscr{L}}$ and MSF makes its free exchanges.

(3) H makes its free exchanges.

In order to compute the amortized cost of the steps above we consider two additional steps, the first one between step 1 and step 2, the second one between step 2 and step 3. In the first additional step we modify the current permutation, while in the other one we resume the earlier one. Thus, in our analysis, we have actually five phases and we refer to them as A, B, C, D, and E, where phases A, C, and E respectively correspond to steps 1, 2, and 3 above, and phases B and D are the following:

B: the current permutation changes from ι to τ_j;

D: the current permutation changes from τ_j to ι.

An amortized cost (that can be due only to a change in the potential) is associated with each of the five phases. The total amortized cost to serve the jth request is the sum of the amortized costs of each phase:

$$a_j = t_j + \Phi_j - \Phi_{j-1} = t_A + t_B + t_C + t_D + t_E + (\Phi_j - \Phi_D) + (\Phi_D - \Phi_C) + (\Phi_C - \Phi_B) + (\Phi_B - \Phi_A) + (\Phi_A - \Phi_{j-1}),$$

where t_j is the actual total time for MSF to process the jth request, t_A, t_B, t_C, t_D and t_E are the actual costs associated with the five phases, Φ_{j-1} and Φ_j are the potential respectively at the beginning and at the end of the processing of the jth request, Φ_A, Φ_B, Φ_C and Φ_D are the potentials at the end of the corresponding phases. It is easily seen that $t_A = t_B = t_D = t_E = 0$.

In phase A, a paid exchange can increase the number of inversions at most by 1, thus $\Phi_A - \Phi_{j-1}$ (the amortized cost of MSF in this phase), is less than or equal to the cost paid by H for making its exchanges.

Now we show that the new inversions created in phase B by the modification of the current permutation do not give rise to a growth of the potential. First, inversions between elements in r_j are got rid of, those between elements of $\mathscr{S} - r_j$ are not modified.

It remains to show that neither the inversions in which one item belongs to $\mathscr{S} - r_j$ and the other to r_j give rise to an increase in the potential.

Any portion of the sublist of $\mathscr{L}_H$ between a_1 and a_{β_j} (which is the largest list portion involved in item movements) is spanned the same number of times by forward and backward movements. The sublist may be divided up into stretches. Let x be an item that moves backward as the current permutation passes from ι to τ_j (recall that if x moves backward, or forward, then it belongs to r_j, by definition of τ_j). The *stretch* determined by x is the sublist of $\mathscr{L}_H$ that extends between $\tau_j(x)$ and x. Thus, a stretch is any maximal portion of the list $\mathscr{L}_H$ that completely lies within the range of action of one backward movement.

Let σ be the stretch spanned by x, and $y_1, \ldots, y_p$ the items that by their forward movements cover σ. Fig. 3 illustrates an example.

Some of these stretches may overlap. This implies that for the items in the overlapping parts, the following piece of proof must be repeated once per each stretch which the items belong to.

Let $\mathscr{L}_{H,\sigma,r_j}$ be the list obtained from $\mathscr{L}_H$ by considering only the items outside r_j belonging to a certain stretch σ. Let t_σ and h_σ be two function defined as t and h above

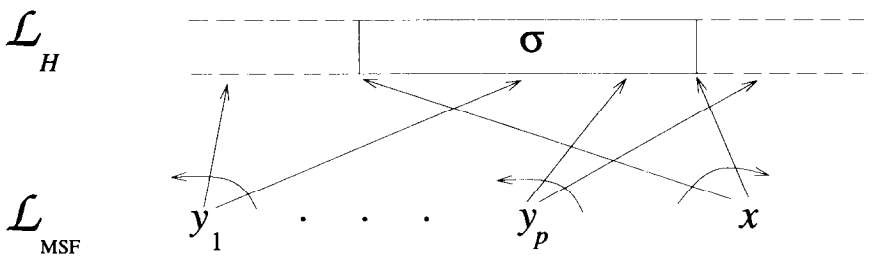

Fig. 3. Spanning of a stretch.

except for the fact that they are now referred to the two lists $\mathcal{L}_{MSF}$ and $\mathcal{L}_{H,\sigma,r_j}$. So, $t_\sigma(x)$ is the number of items that follow x in $\mathcal{L}_G$ and are located between x and $\tau_j(x)$ in $\mathcal{L}_{H,\sigma,r_j}$. Analogously, $h_\sigma(x)$ is the number of items that precede x in $\mathcal{L}_G$ and are located between x and $\tau_j(x)$ in $\mathcal{L}_{H,\sigma,r_j}$. The number of inversions in which one item belongs to $\mathcal{S}-r_j$ and the other belongs to $\{x, y_1, \ldots, y_p\}$ is (from Lemma 4.1):

$$t_\sigma(x)-t_\sigma(y_1)-\cdots-t_\sigma(y_p)+h_\sigma(y_1)+\cdots+h_\sigma(y_p)-h_\sigma(x).$$

Now, $y_1, \ldots, y_p$ precede x in $\mathcal{L}_{MSF}$, otherwise τ_j, by its definition, would have assigned to each of them an item that follows x rather than one that precedes x. Since the spanned stretch is the same (namely, σ), it holds:

$$t_\sigma(x) \leqslant \sum_{q=1}^{p} t_\sigma(y_q), \tag{1}$$

$$h_\sigma(x) \geqslant \sum_{q=1}^{p} h_\sigma(y_q). \tag{2}$$

Summarizing, in phase B, inversions between items in r_j are deleted because τ_j makes $\mathcal{L}_{MSF}$ and $\mathcal{L}_H$ more similar. Other inversions do not give rise to any trouble since they are at least counterbalanced.

A similar reasoning can be made in order to analyze phase D, in which the current permutation is reset to the identity. In fact, in this phase exactly the inversions (between elements of r_j) that were destroyed in phase B are re-introduced, since MSF does not change the relative ordering among the elements of r_j. The inversions between elements of $\mathcal{S}-r_j$ remain the same. The inversions in which one item belongs to $\mathcal{S}-r_j$ and the other to r_j globally give a null contribution to the potential for the following reason. First, note that $h_\sigma(\cdot)$ is null and that $t_\sigma(x)=\sum_{q=1}^{p} t_\sigma(y_q)$. Therefore, repeating the reasoning made for phase B, we draw that the potential increase due to inversions of this kind is null. In other words, backward and forward movements exactly balance, and, in the part of the list each movement spans, the movement itself involves every item in $\mathcal{S}-r_j$. That is, after a permutation change, a modification in the ordering of the first β_j items of the list (where r_j's elements now lie) does not increase the number of inversions between that portion and the remaining part of the list. So, $(\Phi_D-\Phi_C)+(\Phi_B-\Phi_A)\leqslant 0$.

During phase C, let i be the position of $a_{\beta_j}=\tau_j(b_{\beta_j})$ in $\mathscr{L}_H$, k the position of b_{β_j} in $\mathscr{L}_{\text{MSF}}$, v_j the number of items that precede b_{β_j} in $\mathscr{L}_{\text{MSF}}$ and follow a_{β_j} in $\mathscr{L}_H$. As in [16], we have that $t_C=k$ in the standard model, the change in the potential due to free exchanges on b_{β_j} only is $\Delta\Phi|_{\beta_j}\leqslant(k-\beta_j-v_j)-v_j$, and $k-v_j\leqslant i$. It follows that $t_C+\Delta\Phi|_{\beta_j}\leqslant 2i-\beta_j$. The movement of the other β_j-1 items to the front of the list gives rise to at most $(\beta_j-1)(i-\beta_j)$ inversions.

During phase E, each free exchange made by H gives rise to a unit decrease of the potential.

On the whole, disregarding changes in the potential due to the paid exchanges made by H, we obtain

$$a_j\leqslant(1+\beta_j)(i-\beta_j)+\beta_j. \tag{3}$$

Since H is charged i in the standard model and $i-\beta_j$ in the wasted work one, the theorem follows. □

Corollary 4.3. MSF *is* $(1+\beta)$*-competitive in the standard model even if the adversary is allowed to change the relative ordering of the requested set items by means of free exchanges.*

Proof. The proof is the same of that of the previous theorem except for phase E. H creates at most

$$(\beta_j-1)+(\beta_j-2)+\cdots+1=\frac{\beta_j^2-\beta_j}{2},$$

inversions between elements of r_j. Thus

$$a_j\leqslant(1+\beta_j)(i-\beta_j)+\beta_j+\frac{\beta_j^2-\beta_j}{2}=(1+\beta_j)i-\frac{1}{2}(\beta_j^2+\beta_j). \qquad \square$$

As a remark, we wish to spend a few words on a property that has been widely used to analyze the performance of permutation rules. In [2] the *pairwise independence property* for the traditional list update problem is defined as follows. An on-line algorithm G is said to satisfy the pairwise independence property if for any sequence $\underline{r}$ and any pair of different items x and y, the number of unsuccessful comparisons between x and y made while searching for x only depends on the initial list ordering and on the relative ordering of the requests for x and y. So, the number of comparisons between x and y does not depend on the other items.

Till now, the property has been used to analyze the competitiveness features of Move-To-Front and some variations of it [2, 8, 9, 5]. In fact, it has been exploited to break down the cost due to an access to item x into the costs due the comparisons between x and each other item. So, the cost of a single request can be traced back to the costs on $n-1$ lists of size 2 instead of a single list of size n.

In the case of retrieval of sets, the definition above needs to be modified. Instead of referring to the requests for x and y, we should refer to the requests for sets that contain x and y.

Although the pairwise independence property can be easily seen to hold for MSF, it cannot be used in the same way to analyze its competitiveness properties. In fact, the total cost depends not on the relative position of each pair of items, but also on the current position of the other items of the searched set. The straightforward use of the pairwise independence property would make us count many times the same unsuccessful comparison, once per each element of r_j that follows the non-searched item.

The results of Theorems 3.1, 3.2 and 4.2 are compared in Figs. 1 and 2. If $\beta = \beta_{av}$, i.e., the cardinality of the requested sets is always the same, MSF's infimum competitive ratio, in the standard model, is located inside area A.

Now we relate the competitive ratio of MSF to n, the size of the list.

In the first place, we show by an example that the ratio depends on n, even against a static algorithm.

Assume that $\sqrt{2n}$ is an integer. The adversary H serves the sequence without changing the ordering of the initial list $\mathscr{L}^0$. H builds $\underline{r}$ by requesting at first the last $n-\sqrt{2n}$ (we have assumed $\sqrt{2n}$ is an integer) items of $\mathscr{L}^0$, then one at a time the first $\sqrt{2n}$ items of $\mathscr{L}^0$, as singletons. It is easy to verify that this is possible if and only if $\sqrt{2n} \leqslant n$, namely, $n \geqslant 2$, that is to say, in every meaningful case.

If we denote $\sqrt{n/2}$ by q, the costs of the two algorithms over sequence $\underline{r}$ are:

$$H(\underline{r}) = n + \frac{n}{q}\frac{n/q+1}{2} = \frac{n(n+q+2q^2)}{2q^2}, \qquad \mathrm{MSF}(\underline{r}) = n + \frac{n}{q}n = \frac{n(n+q)}{q}. \tag{4}$$

Hence, in the standard model, the cost of ratio of MSF is

$$\frac{2q(n+q)}{n+q+2q^2} > \sqrt{n/2}.$$

The sequence $\underline{r}$ yields an example in which not every NIF ordering is an optimal static one. The NIF ordering in which the items in r_1 precede the other ones incurs in a cost greater than (4).

Now, with reference to Fig. 1, denoting by $\bar{\beta}$ the abscissa (which is roughly $\sqrt{n}$) of the intersection between the curves which represent the two upper bounds to the competitive ratio, the following theorem states that if β and β_{av} lie on the same part with respect to $\bar{\beta}$ then MSF is $O(\sqrt{n})$-competitive.

Theorem 4.4. *Let* $\bar{\beta} = 2n/(1+\sqrt{4n+1})$ *and assume the standard model. If* $(\beta - \bar{\beta})(\beta_{av} - \bar{\beta}) \geqslant 0$ *then the competitive ratio of* MSF *is*

$$1 + 2n/(1+\sqrt{4n+1}) = O(\sqrt{n}),$$

otherwise it is $(1+n/4)$.

Proof. If $(\beta-\bar{\beta})(\beta_{av}-\bar{\beta})\geqslant 0$ then $\min\{1+\beta, n/\beta_{av}\}$ is a competitive ratio for MSF because of Theorem 3.2. The worst case occurs when $\beta_{av}=\beta$ and

$$1+\beta=n/\beta, \tag{5}$$

(see Fig. 1), that is when $\beta=\bar{\beta}$, and this implies $c=1+\bar{\beta}$.

If $(\beta-\bar{\beta})(\beta_{av}-\bar{\beta})<0$, the cost ratio is (from (3))

$$1+\beta_j-\beta_j^2/i.$$

The maximum is in correspondence of $\beta_j=i/2$. Thus, the competitive ratio is at most $1+i/4\leqslant 1+n/4$. □

So far, we have supposed that the inspection of one item had unit cost. Suppose now that the requested set r_j is represented by means of a sequential list, or a heap, or another data structure. In this case, the cost charged to one computation of $\theta_{\mathscr{S}}$ depends on the chosen representation of r_j and on the list state. Let γ be an upper bound to such a cost. For example, for a linear list $\gamma=\beta$. Now suppose that the adversary pays 1 per item test. On any sequence of requests, MSF and the adversary H carry out the same operations they do when unit cost is charged. Now MSF pays at most γ times the cost it would pay in case of unit cost, while H pays the same cost. Hence, MSF is $(1+\beta)\gamma$-competitive. Furthermore, the $O(\sqrt{n})$ upper bound given in Theorem 4.4 still holds under the same hypotheses, since now the worst case equation (5) becomes $(1+\beta)\gamma=(n/\beta)\gamma$, that is in practice the same as that given in the proof of the theorem.

Till now, we have been concerned only with deterministic algorithms. The rest of this section is devoted to the study of a randomized one, that is devised from BIT [9]. We name it "BITS" (BIT-for-Sets). It associates a bit with each element in the list, and the n bits are initialized uniformly and independently at random. Whenever one accesses set r_j, the bit of the last element of r_j in BITS' list is complemented, and if it changes to 1, the accessed set is moved to the front of the list, otherwise it remains unchanged.

We denote by $b(x)$ the value of the bit associated with item x (note that $b(x)$ depends on the time). The following lemma holds.

Lemma 4.5. *At any time $b(x)$ is 0 or 1 with even probability.*

Proof. The proof is given in [9] for algorithm BIT. Identical arguments can be used for proving this lemma. □

Algorithm BITS is competitive, as shown in the following theorem.

Theorem 4.6. BITS *is $(1+\frac{3}{4}\beta)$-competitive against an oblivious adversary both in the standard and in the wasted work model.*

Proof. The proof uses components of that of Theorem 4.2.

As in [9], we define an inversion (y, x) to be of type 1 if $b(x)=0$, and of type 2 otherwise. Recall that the type of the inversion (y, x) is the number of accesses to x before x next moves to front. We define a potential function that depends both on the number of inversions with respect to the current permutation (as in Theorem 4.2) and on the type of the inversions. Let ϕ_1 be the number of inversions of type 1 and ϕ_2 the number of inversions of type 2. The potential function is:

$$\Phi = 2\phi_2 + \phi_1.$$

As in the proof of Theorem 4.2, the events are subdivided into five phases, which we denote by A, B, C, D and E.

We separately consider the amortized costs due to each of the five phases. Since we are now concerned with a randomized algorithm expected costs have to be reckoned.

In phase A, the adversary H pays a unit cost to carry out each of its paid exchanges. Because of Lemma 4.5, half the time H creates a type 1 inversion, and half the time a type 2 inversion. So, the expected value of the increase in the potential is 1.5 times the cost of H.

In phase B inversions between elements in r_j are eliminated. Consider a stretch σ that is spanned by the backward movement of item x. We denote the items whose forward movement spans σ as $y_1, \ldots, y_p$. The contribution to the potential change due to σ is (by Lemma 4.1 and 4.5):

$$\frac{1}{2}(t_\sigma(x)+2t_\sigma(x)) - \sum_{i=1}^{p} \frac{1}{2}(t_\sigma(y_i)+2t_\sigma(y_i)) + \sum_{i=1}^{p} \frac{3}{2} h_\sigma(y_i) - \frac{3}{2} h_\sigma(x)$$

$$= \frac{3}{2}\left(t_\sigma(x) - \sum_{i=1}^{p} t_\sigma(y_i) + \sum_{i=1}^{p} h_\sigma(y_i) - h_\sigma(x) \right) \leqslant 0,$$

where the inequality above is justified by inequalities (1) and (2).

During phase D inversions between elements of r_j are resumed. Let z be the last element of r_j in the BITS' list. Two cases can occur. First, r_j is not moved to the front (hence, $b(z)=0$). In this case the contribution of D to the variation in the potential at most counterbalances the corresponding one occurred during B becaue now $b(z)=0$.

Second, r_j moves to the front, and in this case z either moves forward or remains in the same place (z cannot move backward during phase d). If $z \notin \{y_1, \ldots, y_p\}$ the expected contribution to the potential change is: $\frac{3}{2} t_\sigma(x) - \frac{3}{2} \sum_{i=1}^{p} t_\sigma(y_i) = 0$. Otherwise, if $z \in \{y_1, \ldots, y_p\}$, the expected contribution is

$$\frac{3}{2} t_\sigma(x) - 2t_\sigma(z) - \frac{3}{2} \sum_{\substack{i=1 \\ y_i \neq z}}^{p} t_\sigma(y_i) \leqslant 0.$$

So, $(\Phi_D - \Phi_C) + (\Phi_B - \Phi_A) \leqslant 0$.

Let v_j be the number of inversions (w, z) at time j. Again, as in [16], the actual cost t_C of BITS is at most $i + v_j$ in the standard model, where i is the access cost of the adversary.

The following part of the proof is analogous to the one in [9]. Let A_j, B_j and C_j be the increases of the potential respectively due to new inversions created when r_j is moved to the front, to old inversions removed because of the movement, and to old inversions that change type.

We now compute the expected values of A_j, B_j and C_j during phase C. When r_j is moved to the front, $\mathrm{E}\{B_j\} = -v_j$ and $\mathrm{E}\{C_j\} = 0$. If r_j is not moved, then $\mathrm{E}\{B_j\} = 0$ and $\mathrm{E}\{C_j\} = -v_j$. In both cases, the amortized cost of BITS during C is at most $i + \mathrm{E}\{C_j\}$. The next step is to calculate $\mathrm{E}\{C_j\}$.

For each element of r_j at most $i - \beta_j$ inversions can be generated if r_j is moved to the front. So, the expected value of the potential increase per inversion is $\frac{1}{2}(\frac{1}{2} \cdot 2 + \frac{1}{2} \cdot 1)$ because r_j, for Lemma 4.5, is moved with probability $\frac{1}{2}$. $\mathrm{E}\{C_j\}$ is given by the sum of each contributions, resulting at most $3/4(i - \beta_j)\beta_j$.

During phase E, a free exchange by H half the time destroys an inversion, and half the time creates one. By averaging on the randomization bits, it can be seen that the potential increase cannot be positive.

By summing the amortized costs of each phase, we obtain the theorem. □

5. Extensions

In this section we address two extensions. In the first we suppose that the list is weighted, while in the second we study a variant where a request can be satisfied by retrieving just one element of the requested set.

5.1. *Weighted lists*

We extend the weighted model by assuming that a paid exchange between z and y, with $y \prec x$ just before the exchange, costs $w(y)$. This is a "minimum cost" assumption, similar to those in [16] for nondecreasing cost functions. In fact, if a smaller cost were charged for a paid exchange, the adversary would always move the requested items to the front of the list just before accessing them. In this way, it would pay less than the sum of the weights of the items it would have to inspect otherwise. That is, a scarcely interesting way it can profit by its knowledge about the future.

Let

$$W = \max_{1 \leqslant j \leqslant m} \{\textstyle\sum_{x \in r_j} w(x)\},$$

$$w_{\max} = \max_{x \in \mathscr{S}} \{w(x)\},$$

$$w_{\min} = \min_{x \in \mathscr{S}} \{w(x)\}.$$

With reference to the notation introduced in Section 4 we define the following function. Let $h^{(w)}(x)$ be the sum of the weights of the items that precede x in $\mathscr{L}_G$ and are located between $\tau_1(x)$ and $\tau_2(x)$ in $\mathscr{L}_H$.

First, we give an obvious inequality that correlates the cardinality of a set of items B to their total weight. If i denotes the size of B and $i^{(w)}$ its total weight, then

$$i \leqslant i^{(w)}/w_{\min}. \tag{6}$$

Let the weight of the inversion (y, x) be $w(y)$. For the case of weighted lists, Lemma 4.1 generalizes to the following.

Lemma 5.1. *For any fixed x, the increase in the sum of the weights of inversions of type (x, y) plus those of type (z, x), for any y and z belonging to $\mathscr{S}$, is given by*

$$\begin{cases} h^{(w)}(x) - w(x)t(x) & \text{if } x \text{ moves forward,} \\ w(x)t(x) - h^{(w)}(x) & \text{if } x \text{ moves backward.} \end{cases}$$

Proof. It directly derives from that of Lemma 4.1. □

The competitiveness properties of MSF are stated by the following theorem.

Theorem 5.2. *If $w_{\max}/w_{\min} \leqslant 2$,* MSF *is $(1 + W/\omega_{\min})$-competitive both in the standard and in the wasted work model.*

Proof. The proof is similar to that of Theorem 4.2, but requires a different choice for the potential function. This time its value depends on the weights associated with the items involved in the inversions.

More precisely, let (y, x) be an inversion. Its contribution to the potential is $w(y)$. So, the potential is defined as the sum over all such contributions.

In the proof we examine the amortized costs due to each of the five phases defined in the proof of Theorem 4.2.

During A the adversary can create an inversion (y, x) paying $w(y)$. The potential increase is at most $w(y)$. Therefore, the ratio between the amortized cost of MSF and the cost of H is at most 1.

During B, the change in the potential due to stretch σ is

$$w(x)t_\sigma(x) - h_\sigma^{(w)}(x) + \sum_{i=1}^{p} h_\sigma^{(w)}(y_i) - \sum_{i=1}^{p} w(y_i)t_\sigma(y_i).$$

During D, the new quantities $h_\sigma^{(w)}(\cdot)$ are null, and the number of items not belonging to r_j that follow x in $\mathscr{L}_{\text{MSF}}$ and lie in stretch σ are $t_\sigma(x) + h_\sigma(x)$, in terms of t_σ and h_σ of phase b. The potential change due to stretch σ is

$$-w(x)t_\sigma(x) + \sum_{i=1}^{p} w(y_i)t_\sigma(y_i) - w(x)h_\sigma(x) + \sum_{i=1}^{p} w(y_i)h_\sigma(y_i).$$

Let $k^{(w)}$ and $i^{(w)}$ be the cost paid by MSF and the adversary respectively during the access. Let $v_j^{(w)}$ be the overall weight of the items that precede b_{β_j} in $\mathscr{L}_{\text{MSF}}$ and follows a_{β_j} in $\mathscr{L}_H$. Thus, $k^{(w)} - v_j^{(w)} \leqslant i^{(w)}$.

During C, $t_C + \Delta\Phi|_{\beta_j} \leqslant k^{(w)} + w(b_{\beta_j})(k - \beta_j - v_j) - v_j^{(w)} \leqslant i^{(w)} + w(b_{\beta_j})(i - \beta_j)$. The change due to the other items is at most

$$(W - w(b_{\beta_j}))(i - \beta_j) - \sum_{y \in r_j - \{b_{\beta_j}\}} h^{(w)}(y).$$

On the whole, since a weighted version of inequality (2) holds, and $b_{\beta_j} \neq y_i$ for $1 \leqslant i \leqslant p$, the potential change due to stretch σ is

$$\begin{aligned}
\Delta\Phi_\sigma &\leqslant \sum_{i=1}^{p} w(y_i)h_\sigma(y_i) - w(x)h_\sigma(x) - \sum_{y \in r_j - \{b_{\beta_j}\}} h_\sigma^{(w)}(y) \\
&\leqslant \sum_{i=1}^{p} w(y_i)h_\sigma(y_i) - w(x)h_\sigma(x) - \sum_{i=1}^{p} w_{\min} h_\sigma(y_i) \\
&\leqslant w_{\min}\left(\sum_{i=1}^{p} h_\sigma(y_i) - h_\sigma(x) \right) \leqslant 0.
\end{aligned}$$

In phase E, each free exchange of the adversary brings about a decrease of the potential.

The total amortized cost a_j is reckoned by adding the contribution due to the five phases. Hence,

$$a_j \leqslant i^{(w)} + W(i - \beta_j) \leqslant i^{(w)} + W\frac{i^{(w)} - W}{w_{\min}},$$

because of inequality (6).

The cost of the adversary is $i^{(w)}$ or $i^{(w)} - W$ in the standard and in the wasted work respectively. Hence, the theorem follows. □

5.2. *When the first found is good*

An interesting variation is the retrieval of one element (whichever) of the currently requested set r_j. In this case, the list maintenance algorithm can answer to the request returning the first element of r_j that it finds in the list.

The cost models are the same as those introduced in Section 2, except for the fact that only the (first) found item may be moved at any distance forward in the list. Any other exchange costs 1. For this problem, the proving scheme in [11] yields a lower bound of 2.

Theorem 5.3. *For any list $\mathscr{L}$ of n items and for any deterministic on-line algorithm G, the competitive ratio of G cannot be less than $2 - 2/n$ in the standard model and cannot be less than 2 in the wasted work one.*

Proof. Let x be the last item of the initial list. The adversary turns out the jth request by asking the subset of the last two items in $\mathscr{L}_G^{j-1}$. It eventually reorders $\mathscr{L}_H$ as follows. x is left at rear. The other items are rearranged according to a NIF ordering.

To compute the cost of the adversary we follow the same strategy as that in [11] and in the proof of Theorem 3.1. In the worst case for H, x is all the time the last element of $\mathscr{L}_G$ and the frequencies of the other items are equal. Hence, the cost of the adversary is at most $mn/2$ in the standard model and at most $m(n-2)/2$ in the wasted work one. The cost of G is $m(n-1)$ and $m(n-2)$, respectively. □

On the other hand, for this problem, not only no competitive on-line algorithm is known, but in addition we claim that all the algorithms of a general class are not c-competitive for any fixed c, even if $\beta_j=2$ for any j.

Theorem 5.4. *For any on-line deterministic algorithm G that never makes a paid exchange on the last item after the first request has been processed, and for any fixed constant c, there exists a list, a sequence of requests $\underline{r}$ such that $\beta_j=2$ for any j, and an algorithm H such that $G(\underline{r})>c\cdot H(\underline{r})$.*

Proof. Let us denote by $\mathscr{L}_G^0$ the list of G after G has done its initialization operations, e.g., paid exchanges that modify the initial ordering of the list. The initial operations of G can be thoroughly forecast because it is deterministic. In such a way we also keep into account the differences between the initial list of G and that of the adversary.

Then, the adversary H moves the last item in $\mathscr{L}_G^0$ to the front of $\mathscr{L}_H$. It subsequently requests the last two items in $\mathscr{L}_G$. Note that the last item of $\mathscr{L}_G$ is always the same because G has no way of moving it.

H pays 1 per request in the standard model and 0 in the wasted work one. The corresponding quantities for G are $n-1$ and $n-2$. □

From the proof of the previous theorem it is clear that in the wasted work model its statement can be strengthened.

Theorem 5.5. *For any list (whose size is denoted by n), any on-line deterministic algorithm G that never makes a paid exchange on the item in position $v\leqslant n$ after the first request has been processed, and for any fixed constant c, there exists a sequence of requests $\underline{r}$ such that $\beta_j=2$ for any j and an algorithm H such that in the wasted work model $G(\underline{r})>c\cdot H(\underline{r})$.*

6. Final remarks

In this paper we have considered a few generalizations and extensions to the traditional list update problem.

The first consists of updating an unweighted list of items while searching for sets of items. For this problem we have provided a lower bound to the competitive ratio of any on-line deterministic algorithm, and two algorithms, one deterministic and the other randomized, for which the competitive ratio is a linear function of β, the size of

the largest requested set. Moreover, we have given an upper bound to the competitive ratio of MSF algorithm by relating β to the size of the list.

These results well extend to the case of weighted lists, where the cost model generalizes the traditional one [5] because we also consider pair exchanges.

Furthermore, we have studied the particular case where we are only interested in finding whichever element belonging to the set currently specified by the sequence of requests. However, for this case, we have been only able to provide negative results.

Currently, we are working to define on-line maintenance algorithms for AND–OR trees and DAGs under several visiting algorithms, by exploiting the results obtained in this paper. This work should lead to the design of competitive algorithms.

There are some open questions concerning the list update problem under searches for sequence of sets. Of course, the first one is either to devise an algorithm whose upper bound matches the lower bound stated in Theorem 3.1 or to prove the MSF achieves the best possible competitive ratio (i.e. MSF is strongly competitive [10]). It is known that one can use dynamic programming to develop an optimum algorithm [12]. On the other hand, some properties of the optimum algorithm for the traditional list update problem (see [14]), no longer hold.

The properties of the optimal static ordering deserve study as well.

Other open questions are related to the variant where one item has to be retrieved. First, we wonder whether competitive algorithms exist for this problem or the results stated in Theorems 5.4 and 5.5 can be strengthened. Second, do there exist randomized algorithms that allow us to overcome the difficulties in designing well-performing deterministic ones?

Acknowledgment

We wish to thank the anonymous referees for their criticism and helpful comments that allowed the authors to substantially improve the presentation of the paper.

References

[1] S. Ben-David, A. Borodin, R. Karp, G. Tardos and A. Wigderson, On the power of randomization in on-line algorithms, in: *Proc. 22nd Ann. ACM Symp. on Theory Comput. (STOC '90)* (May 1990) 379–386.

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

[3] M. Chrobak and L.L. Larmorre, An optimal on-line algorithm for k-servers on trees, *SIAM J. Comput.* **20** (1991) 144–148.

[4] F. d'Amore and V. Liberatore, The list update problem and the retrieval of sets, in: O. Nurmi and E. Ukkonen, eds., *Proc. 3rd Scandinavian Workshop on Algorithm Theory (SWAT '92)*, Lecture Notes in Computer Science, Vol. 621 (Springer, Berlin, 1992) 178–191.

[5] F. d'Amore, A. Marchetti Spaccamela and U. Nanni, The weighted list update problem and the lazy adversary, *Theoret. Comput. Sci.* **108** (1993) 371–384.

[6] E.F. Grove, The harmonic on-line K-server algorithm is competitive, in: *Proc. 23rd Annual ACM Symp. on Theory Comput. (STOC '91)* (1991) 260–266.
[7] J.H. Hester and D.S. Hirschberg, Self-organizing linear search, *ACM Comput. Surveys* **17** (1985) 295–311.
[8] S. Irani, Two results on the list update problem, Techn. Report TR-90-037, Computer Science Division, UCB, Berkeley, CA, 1990.
[9] S. Irani, N. Reingold, J. Westbrook and D.D. Sleator, Randomized competitive algorithms for the list udate problem, in: *Proc. 2nd ACM-SIAM Annual. Symp. on Disc. Algorithms (SODA '91)* (1991) 251–260.
[10] A.R. Karlin, M.S. Manasse, L. Rudolph and D.D. Sleator, Competitive snoopy caching, *Algorithmica* **3** (1988) 79–119.
[11] R. Karp and P. Raghavan, Private communication reported in [8].
[12] M.S. Manasse, L.A. McGeoch and D.D. Sleator, Competitive algorithms for on-line problems, in: *Proc. 20th Annual ACM Symp. Theory Comput. (STOC '88)* (1988) 322–333.
[13] M.S. Manasse, L.A. McGeoch and D.D. Sleator, Competitive algorithms for server problems, *J. Algorithms* **11** (1990) 208–230.
[14] N. Reingold and J. Westbrook, Optimum off-line algorithms for the list update problem, Techn. Report YALEU/DCS/TR-805, Department of Computer Science, Yale University, New Haven, CT, 1990.
[15] D.D. Sleator, Private communication reported in [8].
[16] D.D. Sleator and R.E. Tarjan, Amortized efficiency of list update and paging rules, *Comm. ACM* **28** (1985) 202–208.
[17] D.D. Sleator and R.E. Tarjan, Self-adjusting binary search trees, *J. ACM* **32** (1985) 652–686.
[18] R.E. Tarjan, Amortized computational complexity, *SIAM J. Algebraic Discrete Methods* **6** (1985) 306–318.
[19] B.A. Teia, Lower bound for randomized list update algorithms, *Inform. Process. Lett.* **47** (1993) 5–9.