The Weighted List Update Problem and the Lazy Adversary*

Fabrizio d'Amore† Alberto Marchetti-Spaccamela†‡

Umberto Nanni‡§

TR-92-011

February 1992

Abstract

The *List Update Problem* consists in maintaining a dictionary as an unsorted linear list. Any request specifies an item to be found by sequential scanning through the list. After an item has been found, the list may be rearranged in order to reduce the cost of processing a *sequence* of requests.

Several kind of adversaries can be considered to analyze the behavior of heuristics for this problem. The *Move-To-Front* (MTF) heuristic is 2-competitive against a *strong* adversary, matching the deterministic lower bound for this problem [21].

But, for this problem, moving elements does not help the adversary. A *lazy* adversary has the limitation that he can use only a static arrangement of the list to process (off-line) the sequence of requests: still, no algorithm can be better than 2-competitive against the lazy adversary [3].

In this paper we consider the *Weighted List Update Problem* (WLUP) where the cost of accessing an item depends on the item itself. It is shown that MTF is not competitive by any constant factor for this problem against a lazy adversary. Two heuristics, based on the MTF strategy, are presented for WLUP: *Random Move-To-Front* is randomized and uses biased coins; *Counting Move-To-Front* is deterministic, and replaces coins by counters. Both are shown to be 2-competitive against a lazy adversary. This is optimal for the deterministic case.

We apply this approach for searching items in a tree, proving that any c-competitive heuristic for the weighted list update problem provides a c-competitive heuristic for the *Tree Update Problem*.

*Work supported by the ESPRIT II Basic Research Actions Program Project no. 3075 ("ALCOM") and by the Italian National Project "Algoritmi e Strutture di Calcolo", Ministero dell'Università e della Ricerca Scientifica e Tecnologica.

†Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", via Salaria 113, I-00198 Roma, Italia.

‡Dipartimento di Matematica Pura ed Applicata, Università di L'Aquila, via Vetoio, Coppito I-67100 L'Aquila, Italia.

§Currently visiting the International Computer Science Institute, 1947 Center St., Berkeley, CA 94704.

1 Introduction

The *List Update Problem* (LUP) has been extensively studied in the literature (see, for example, [20, 4, 9, 3, 21, 13]). It consists in maintaining a dictionary as an unsorted linear list.

While processing a sequence of requests, the list may be rearranged in order to minimize the access cost of subsequent operations. For any request, the cost of accessing the searched item depends on its position in the list.

In the *Weighted List Update Problem* (WLUP) any item has an associated cost due for its "visit" and therefore the cost of searching an item within the list depends on the sum of the costs of the preceding items. Also in this case the problem consists in minimizing the total cost incurred in processing a sequence of requests.

This is an example of on-line problem, where each request in a sequence has to be processed before the subsequent ones are made known, and some decision taken will affect the cost of answering the subsequent requests. A usual framework to analyze the behavior of heuristics for on-line problems is the technique based on adversaries. An adversary is in charge to generate the sequence of requests in order to maximize the ratio between the cost incurred by the heuristics to be analyzed and the cost of an optimal algorithm to handle the sequence. The heuristic is c-competitive if this ratio is asymptotically not greater then c [14]. Several heuristics for the LUP have been considered, such as the *Frequency Count*, the *Transpose*, the *Move-To-Front* (overviews can be found, for example, in [20, 9]). Sleator and Tarjan proved that Move-To-Front (MTF) is 2-competitive against any strategy of the adversary, that is what is called a *strong* adversary. MTF has been proved to be optimal among all the deterministic heuristics for this problem [15, 11] with a competitivity $\theta(\frac{2L}{L+1})$. Several kinds of adversaries can be considered for the LUP: for example in [3] Bentley and McGeoch showed that MTF is 2-competitive against an adversary using the optimal static ordering and answering requests without moving any item. Indeed this has been also proven to be optimal [15, 11] and quite surprisingly it turns out that moving items does not help the adversary for this problem. A *lazy* adversary for an on-line problem is one that moves as few as possible to service requests [16, 7]: in the case of list update problem a lazy adversary does not move anything. In this paper we consider heuristics for the WLUP and we show that MTF is not c-competitive against a lazy adversary by any constant factor in the weighted case. We propose two heuristics for this problem, both derived from MTF:

- the *Counting Move-To-Front* (CMTF), which is a deterministic heuristic which uses n real counters (one counter per item) in order to decide whether moving to the front the accessed items;
- the *Random Move-To-Front* (RMTF), which is a randomized heuristic, obtained by CMTF by substituting counters by biased coins.

Both are shown to be 2-competitive against a lazy adversary. Moreover we consider the *Tree Update Problem*, where items are to be found in a tree instead of in a sequential list. The tree is represented by means of lists of successors and is searched by a leftist depth first search. The only possible update operation consists in rearranging the lists of children of the vertices. If we weigh each vertex by means of the size of its subtree, we can handle the list of the children of any vertex by using any weighted list update heuristic in order to

reduce the overall cost of processing a sequence of searches over the tree. We prove that, given a heuristic for the WLUP which is c-competitive against an adversary, it is possible to devise a heuristic for the Tree Update Problem which is c-competitive against the same adversary. For example AND-OR trees, problem solving [18, 8] and diagnosis [19] are some of the areas which could exploit efficient solutions for the WLUP [6].

The paper is organized as follows. In the next section the concepts of adversary and c-competitivity are discussed, the weighted list update problem is formally stated and two cost functions for this problem are considered.

In section 3 we present two algorithms, CMTF and RMTF, and prove that both of them are 2-competitive against a static algorithm. CMTF is a deterministic algorithm which makes use of auxiliary memory (one counter per item), while RMTF is a randomized variant which uses no extra space and has expected cost equal to that of CMTF. We also prove that MTF is not c-competitive in the same situation.

Section 4 describes the application of the weighted list update problem to the Tree Update Problem.

2 Preliminaries

An *on-line* problem requires to serve a sequence of requests so that each request has to be answered before the following one is known.

The general framework to analyze the behavior of an on-line algorithm A requires an *adversary* who generates a sequence σ of requests and is charged the cost of processing the sequence by means of his own algorithm B.

An algorithm A is *c-competitive* against an adversary using the algorithm B if for each weighted list $\mathcal{L}$ and for each sequence σ of requests:

$$A(\sigma) \leq c \cdot B(\sigma) + f(\mathcal{L}) \,,$$

where f does not depend on the sequence chosen by the adversary but only on the handled list $\mathcal{L}$ [14]. This definition has been generalized to randomized algorithms [16], and in that case the first member of the above inequality is the *expected* cost of the algorithm, taken with respect to the random choices made by the algorithm.

Adversaries can be classified from various points of view.

An *off-line* adversary has the advantage to process all the requests after the end of the sequence, while an *on-line* adversary must process any request on line.

Furthermore, an adversary is *static* if he is not allowed to rearrange the list after each request.

An *oblivious* adversary must generate the sequence without knowing possible random choices of the algorithm A, while the *adaptive* adversary can choose the next request on the basis of the previous behavior of A. The distinction between oblivious and adaptive becomes meaningful in the case of a randomized algorithm. In fact randomization does not help against an adaptive adversary, as shown by Ben-David *et al.* [2]. Therefore any competitivity result about randomized algorithms refers to oblivious adversaries and, of course, it is intended to provide an estimation of the expected behavior of the randomized algorithm, where the expectation is taken with respect to its random choices.

Hence adversaries are classified on the basis of both the knowledge he has about the behavior of the algorithm A and the features of his own algorithm B.

Several adversaries have been considered in the literature to analyze performance of heuristics for on-line problems [14, 2]: the oblivious off-line (*week*) adversary, the adaptive on-line (*medium*) adversary, the adaptive off-line (*strong*) adversary.

In the context of *server* problems a *lazy* strategy for the adversary consists in moving a server only when it is strictly required to serve a request [16, 7]. In the case of the list update problem, since it is not strictly required to move anything to serve a request, we call *lazy* adversary one that uses a static arrangement of the list, without resorting the list after each request. Of course a lazy adversary is supposed to use the optimal static ordering, which can be computed off-line.

Using the terminology introduced so far, Bentley and McGeoch [3] proved that MTF for the unweighted list update problem is 2-competitive against the lazy adversary. Sleator and Tarjan in [21] proved MTF to be 2-competitive against the strong adversary as well. Indeed MTF has been proved to be be optimal among all the deterministic heuristics for this problem [15, 11] with a tight competitivity bound of $\theta(\frac{2L}{L+1})$. The lower bound is stated showing an example where the adversary uses a static arrangement of the list: it turns out that moving items does not help the adversary for this problem.

In general it is not known whether an adaptive on-line adversary is actually weaker than the strong one: for the unweighted list update problem the answer is no [13].

In [11, 13] randomized algorithms for the list update problem are presented with a competitive ratio less than two. The best competitivity factor is $\sqrt{3}$ (about 1.73). The lower bound for randomized algorithms is about 1.27 [15, 13].

3 Algorithms for Weighted Lists

In this section we show that in the weighted case the competitivity of the Move-To-Front heuristic against the lazy adversary cannot be made independent from the distribution of weights. We propose two 2-competitive heuristics for the weighted list problem, consisting of suitable extensions of MTF.

In the weighted version of the problem the cost of visiting an item of the list depends on the element itself. To the best of our knowledge this problem has never been studied. In [21] a cost model was considered in which the cost of an element depends on its position.

Suppose we are given a pair $\langle \mathcal{S}, w \rangle$, where $\mathcal{S}$ is a set $\mathcal{S} = \{e_1, e_2, \ldots, e_n\}$ of elements and $w : \mathcal{S} \to R^+$ is a total function mapping the elements of $\mathcal{S}$ into the set of positive reals R^+. In what follows we will denote by w_i the value of $w(e_i)$.

Two cost models can be considered to compute the work done by an algorithm to answer one request: they are analogous to the $i-1$ and i cost functions used in [13]. In the *wasted* work model the cost of accessing the i-th item of $\mathcal{L}^t$ (where $\mathcal{L}^i$ denotes the list *after* the i-th request has been served) is given by

$$\sum_{e_j \prec^t e_i} w_j \ ,$$

where $e_i \prec^t e_j$ means that e_i is stored in $\mathcal{L}^t$ before e_j. In the *total* work model the summation is extended to the elements $e_j \preceq^t e_i$ (i.e. including e_i itself).

The lazy adversary is supposed to use the *optimal static ordering* that, for any sequence of requests and any weighted lists, is obtained by sorting the items by non increasing values of n_i/w_i, where n_i is the number of occurrences of e_i in σ. Such an ordering meets the one pointed out in [3] for unweighted sequential lists.

Let $\mathrm{LAZY}^{\mathcal{L}}(\sigma)$ the cost of processing the sequence σ by a lazy adversary using the static arrangement $\mathcal{L}$. If $\mathcal{L}_{\mathrm{OPT}}$ is the optimal static ordering and does not respect the above condition, then there exist two consecutive items e_i, e_{i+1} such that:

$$\frac{n_i}{w_i} < \frac{n_{i+1}}{w_{i+1}}.$$

Let $\mathcal{L}'$ be the list obtained by swapping the two consecutive items e_i, e_{i+1}. Since swapping e_i and e_{i+1} only affects the cost of searching for e_i and e_{i+1}, we have, for both the considered cost models:

$$\mathrm{LAZY}^{\mathcal{L}'}(\sigma) - \mathrm{LAZY}^{\mathcal{L}_{\mathrm{OPT}}}(\sigma) = n_i w_{i+1} - n_{i+1} w_i < 0 \ ;$$

hence the arrangement $\mathcal{L}'$ would be cheaper for the lazy adversary.

The *Counting Move-To-Front* (CMTF) heuristic is deterministic and uses counters to decide when moving items. Any element e_k has an associated counter c_k with real values in $[0, 1 + d/w_k)$ (which, in all practical cases, is contained in $[0, 2)$. After a searched item e_k has been found, c_k is increased by d/w_k and if it reaches 1 or more, e_k is moved to the front of the list and the counter is decreased by 1. The best choice for the constant d, as it will be seen later, is to take $d = w_{\min}$. This is also the maximum value for d to make the frequency of move to front of any item proportional to the ratio between its frequency in the sequence and cost.

Note that the number of occurrences of a generic item e_k in σ between two consecutive "move to front" of item e_k is not more than $\lceil w_k/d \rceil$ and not less than $\lfloor w_k/d \rfloor$. Due to the way the counter is handled, over many requests for e_k, this actually occurs w_k/d times in the average.

The *Random Move-To-Front* (RMTF) heuristic is randomized and uses biased coins instead of counters. After a searched item e_k has been found, a biased coin is tossed in order to decide whether moving e_k to the front of the list or not. Such a coin has a probability $p_k = d/w_k$ to give a positive answer, where $d = \min_j w_j$. In this case the expected number of occurrences of an item e_i in σ between consecutive positive answer of the coin (and consequent movement to the front) is w_i/d.

Comparing the two proposed algorithms we have that in the general case CMTF requires real valued counters (and arithmetics), but they can be truncated down to a reasonable number of bits without affecting the substantial behavior of the heuristic in most practical cases. Hofri and Shachnai [10] present a study of the error introduced by truncating integer counters in the context of the *frequency count* approach for the list update problem. On the other side RMTF replaces counters by random resources, but pseudo-randomness should not introduce substantial error against a non truly malicious adversary.

In the following we prove that both CMTF and RMTF are 2-competitive against a lazy adversary, while MTF does not share this property in the case of weighted lists.

In the next theorems we use the *pairwise independence property* pointed out, for example, in [3]. It can be described as follows. The number of times e_i is examined while searching for e_j starting from any given initial arrangement of the list depends only on the relative position of the occurrences of e_i and e_j in σ. In order to compute such contribution to the total cost, we can consider, for any i and j, the compressed sequence $\sigma_{i,j}$ obtained from σ by deleting all the elements out of e_i and e_j, and a list containing only these two items. The wasted cost for a generic algorithm *MTF to answer a sequence σ of requests can be rewritten as:

$$*MTF(w,\sigma) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} *MTF(w_{i,j}, \sigma_{i,j})$$

where $w_{i,j}$ denotes the weight function w restricted to the domain $\{e_i, e_j\}$.

The following theorems prove that RMTF and CMTF are 2-competitive against the lazy adversary.

Theorem 3.1 *In the wasted cost model, for any weight function w and any sequence σ:*

$$CMTF(w,\sigma) \leq 2 \cdot LAZY(w,\sigma) + f(w),$$

where $f(w)$ has a value not depending on σ.

Proof. The lazy adversary is supposed to use use the optimal static arrangement. Without loss of generality we assume that the elements in $\mathcal{S}$ are numbered according to the optimal static ordering, that is $i < j$ if and only if $e_i \prec e_j$ in the static list.

In order to prove the theorem, we will separately examine (by virtue of the pairwise independence property) the contribution of any pair e_i, e_j to the global cost while processing the entire sequence σ.

Hence the theorem can be restated as

$$\sum_{1 \leq i < j \leq n} CMTF(w_{i,j}, \sigma_{i,j}) \leq 2 \cdot \sum_{1 \leq i < j \leq n} LAZY(w_{i,j}, \sigma_{i,j}) + f(w). \qquad (1)$$

Therefore we separately consider the cost of the two algorithms while processing the sequence $\sigma_{i,j}$ and handling a two-elements list containing only the items e_i and e_j. Let $\mathcal{L}_{i,j}$ be the list handled by CMTF. The lazy adversary keeps the position of the two elements fixed according the optimal static ordering (say $e_i \prec e_j$). CMTF instead repeatedly exchanges the position of the two elements in $\mathcal{L}_{i,j}$, according to the described strategy.

The compressed sequence $\sigma_{i,j}$ can be subdivided into *fragments* on the basis of the behavior of CMTF, namely on the basis of the presence of e_i before e_j (and vice versa) in $\mathcal{L}_{i,j}$. Each fragment begins when e_j is moved to precede e_i (in opposite arrangement with respect to the LAZY's list): note that this can happen only if e_j is moved to the front by CMTF. After some requests the two elements are switched again, and in this second portion of the fragment e_i precedes e_j, just as in the optimal static list used by the adversary. We remark that at the beginning and at the end of $\sigma_{i,j}$ there are partial fragments, and that the number $s_{i,j}$ of fragments in $\sigma_{i,j}$ depends on the indices i, j. Now we compute the cost paid by either algorithms to process a generic single fragment within $\sigma_{i,j}$.

If $s_{i,j}$ is the number of fragments of $\sigma_{i,j}$, and $\sigma_{i,j}^k$ denotes the k-th fragment of $\sigma_{i,j}$, inequality (1) can be rewritten as:

$$\sum_{1\le i<j\le n}\sum_{k=1}^{s_{i,j}} CMTF(w_{i,j},\sigma_{i,j}^k) \le 2\cdot \sum_{1\le i<j\le n}\sum_{k=1}^{s_{i,j}} LAZY(w_{i,j},\sigma_{i,j}^k) + f(w).$$

and the theorem can be proved by showing that it holds

$$CMTF(w_{i,j},\sigma_{i,j}^k) \le 2\cdot LAZY(w_{i,j},\sigma_{i,j}^k)\ ,$$

for any $k = 2,3,\ldots,s_{i,j}-1$, and that the processing cost of the first and the last fragment of $\sigma_{i,j}$ to CMTF can be bounded by a function of the weights in the list.

Let us define the following quantities, relative to the particular fragment $\sigma_{i,j}^k$:

- a_i^k is the number of occurrences of e_i in the first portion of the fragment (i.e. while $e_j \prec e_i$);
- a_j^k is the number of occurrences of e_j in the first portion of the fragment;
- b_j^k is the number of occurrences of e_j in the second portion of the fragment (i.e. while $e_i \prec e_j$, just as in LAZY's list).

Due to the behavior of CMTF, for any fragment $\sigma_{i,j}^k$ we have that e_i (resp. e_j) is not moved to front until the first (second) portion of the fragment ends, therefore the following inequalities hold:

$$a_i^k \le \frac{w_i}{c} \quad \text{and} \quad b_j^k \le \frac{w_j}{c}\ , \tag{2}$$

and, moreover:

$$a_j^k + b_j^k \ge \frac{w_j}{c}\ , \tag{3}$$

because the first member represents the number of occurrences of e_j between two (not necessarily consecutive) move to front of item e_j.

Since $e_i \prec e_j$ in the optimal static list, in the wasted cost model the lazy adversary will be charged the quantity w_i any time that e_j is requested. The cost to CMTF is w_j for any request of e_i in the first part of the fragment, and w_i for any request of e_j in the second part.

Hence, if $\sigma_{i,j}^k$ is the generic k-th fragment of $\sigma_{i,j}$ we can bound the processing cost of the two algorithms as follows:

$$CMTF(w_{i,j},\sigma_{i,j}^k) = a_i^k w_j + b_j^k w_i \le 2\ \frac{w_i w_j}{c}$$

$$LAZY(w_{i,j},\sigma_{i,j}^k) = a_j^k w_i + b_j^k w_i \ge \frac{w_i w_j}{c}.$$

As far as the partial fragments is concerned, we have that the first and last partial fragments cannot cost to CMTF more that one fragment. On the other hand, the initial partial fragment of $\sigma_{i,j}$ has the same cost to CMTF and LAZY, if the two lists have the same initial arrangement.

Hence inequality (1) holds, with:

$$f(w) = 2 \sum_{1 \le i < j \le n} \frac{w_i w_j}{c}$$

if the two lists have the same initial arrangement, or the double of this quantity, with any initial arrangement. □

Note that the theorem is true for any choice of the value of the constant c not greater than $w_{\min}$ (in order to make inequalities (2) and (3) true). The choice $c = w_{\min}$ minimizes the additive term $f(w)$ in inequality (1).

Theorem 3.2 *In the wasted cost model, for any weight function w and any sequence σ:*

$$E\{RMTF(w,\sigma)\} \le 2 \cdot LAZY(w,\sigma) + f(w),$$

where $f(w)$ has a value not depending on σ.

Proof. The proof is very similar to that of theorem 3.1. The statement to be proved, due to the pairwise independence property, and to the same kind of fragmentation, is the following:

$$\sum_{1 \le i < j \le n} \sum_{k=1}^{s_{i,j}} E\{RMTF(w_{i,j}, \sigma_{i,j}^k)\} \le 2 \cdot \sum_{1 \le i < j \le n} \sum_{k=1}^{s_{i,j}} E\{LAZY(w_{i,j}, \sigma_{i,j}^k)\} + f(w).$$

Note that here $s_{i,j}$ and the length of any fragment are random variables, but the sum in the right hand side is deterministically determined by $\sigma_{i,j}$. Also in this case we prove a sufficient condition, relative to the single fragment $\sigma_{i,j}^k$:

$$E\{RMTF(w_{i,j}, \sigma_{i,j}^k)\} \le 2 \cdot E\{LAZY(w_{i,j}, \sigma_{i,j}^k)\}.$$

for any $k = 2, 3, \ldots, s_{i,j} - 1$. Note that the cost to the lazy adversary of a single fragment is a random variable, too.

The quantities a_i^k, a_j^k, b_j^k are defined as in the proof of the previous theorem, and their expected values are subject to the following inequalities, analogous to the (2) and (3):

$$E\{a_i^k\} \le \frac{w_i}{c}, \quad E\{b_j^k\} \le \frac{w_j}{c}, \quad E\{a_j^k + b_j^k\} \ge \frac{w_j}{c}\ .$$

The expected cost to process a single fragment $\sigma_{i,j}$ is for the two algorithms:

$$E\{RMTF(w_{i,j}, \sigma_{i,j}^k)\} = E\{a_i^k w_j + b_j^k w_i\} \le 2\ \frac{w_i w_j}{c}$$

$$E\{LAZY(w_{i,j}, \sigma_{i,j}^k)\} = E\{a_j^k w_i + b_j^k w_i\} \ge \frac{w_i w_j}{c}.$$

The value of the term $f(w)$ is due to the cost of the partial fragments and is bounded by the expected cost of a single fragment. □

Also in this case it is worth observing that any choice of p_h proportional to $1/w_h$ would satisfy the theorem. So, we could choose $p_h = w_{\min}/w_h$ for maximizing the probability of moving items to the front of their lists, in order to obtain the minimum value for the additive term $f(w)$.

Unfortunately, in the wasted work model, CMTF and RMTF are not c-competitive, for any constant c, against an oblivious adversary that uses the optimal off-line algorithm. In order to show this, it suffices to show that CMTF is not c-competitive against an adversary who uses MTF (which is not better than the optimal off-line algorithm).

Consider the case in which there are only three items and both CMTF and MTF start working on two initially identical lists, containing the items in the following order: $\langle e_1, e_2, e_3 \rangle$. Moreover suppose that $w_1 \geq w_2 \geq w_3$. Thus, counters c_1, c_2 and c_3 will be respectively incremented by the quantities w_3/w_1, w_3/w_2 and 1. For the sequence $\sigma = e_3 \cdot e_2^{\lceil w_2/w_3 \rceil} \cdot e_1^{\lceil w_1/w_3 \rceil}$, we have the following costs:

$$CMTF(\sigma) = (w_1 + w_2) + (w_3 + w_1)\left\lceil \frac{w_2}{w_3} \right\rceil + (w_2 + w_3)\left\lceil \frac{w_1}{w_3} \right\rceil$$

$$MTF(\sigma) = 2(w_1 + w_2 + w_3) \; .$$

It is immediate to verify that the ratio $CMTF(\sigma)/MTF(\sigma)$ can be made greater than any fixed c.

It is possible to show that CMTF is not c-competitive against the lazy adversary in the wasted cost model as well [1].

In the next theorem we prove that MTF is not c-competitive for any constant value c with respect to the lazy adversary for the weighted list update problem.

Theorem 3.3 *For each list of $n \geq 2$ items and for any c there exist a weight function w and a sequence σ of requests such that, in the wasted cost model:*

$$MTF(w, \sigma) > c \cdot LAZY(w, \sigma) \; .$$

Proof. Let us consider the set $\mathcal{S} = \{e_1, e_2\}$, with $w_1 < w_2$, and the sequence $\sigma = (e_2 e_1)^c$. The optimal static ordering is $\langle e_1, e_2 \rangle$ and the wasted cost to the lazy adversary consists in visiting the item e_1 while looking for e_2, that is:

$$LAZY(w, \sigma) = c w_1 \; .$$

The Move-To-Front heuristic swaps the two items for any request and then:

$$MTF(w, \sigma) = c(w_1 + w_2) \; .$$

The theorem holds for any choice for the weight function such that $\frac{w_2}{w_1} > c - 1$. □

In general, that is for longer lists, we have that the expression of the processing cost in the wasted cost model to the lazy adversary $LAZY(w, \sigma)$ does not depend on w_n (the weight of the last element in the optimal static ordering), while $MTF(w, \sigma)$ does. Therefore the competitivity factor can be made greater than any constant value by choosing a suitable value for w_n.

4 Algorithms for Non-Modifiable Trees

A tree T is said to be non-modifiable if the parent-child relationships cannot be modified. Such a structure is interesting when the father-child relationships capture relevant aspects of reality. For example, in decision trees the father-child relationships have a precise meaning and the order of the children of a vertex can affect only the complexity of discovering something, not the result of the search, in the case that there is only one vertex to be found.

The *Tree Update Problem* is the following. A sequence $\sigma = \langle r_1, r_2, r_3, \ldots \rangle$ of requests has to be answered on-line, and each request consists in specifying a vertex to be found within a non-modifiable tree by means of a leftist depth first search. The only allowed update operation consists in changing the order of the children of a vertex, i.e. changing the order by which its subtrees are searched. The goal, again, is to minimize the total cost of answering all the requests. The *wasted cost* is assumed to be the number of vertices unsuccessfully "visited" while searching for some other vertex. Of course this simply generalizes to the case where each vertex is weighted, that is it has its own visiting cost.

Our basic idea consists in using a heuristic for weighted lists to modify the relative ordering among siblings, taking into account both the frequence of requested vertices in any subtree and the size of the subtree itself. This section describes this approach and provides a general analysis of the competitivity of heuristics for the tree update problem using a weighted list heuristics.

Any requested vertex r_t univocally characterizes a path from the root of the tree to r_t itself. Let $\pi(r_t) = \langle x_0, x_1, \ldots, x_{L-1}$ be the vertexes preceding $r_t = x_L$ along this path.

Moreover, for any vertex u, let $S(u)$ be the subtree rooted at u and $|S(u)|$ be its size. The wasted cost for searching for r_t by a leftist depth first search is given by:

$$\sum_{x_k \in \pi(r_t)} \sum_{u \prec^{t-1} x_{k+1}} |S(u)| \, ,$$

where $v_1 \prec^i v_2$ means that v_1 precedes its sibling v_2 after the i-th request has been served ($\prec^i$ is a partial order only defined between pairs of siblings). In other words the cost incurred by the depth first search for r_t can be charged to the vertices belonging to $\pi(r_t)$: more precisely, each $x \in \pi(r_t)$ is charged the sum of the sizes of the subtrees whose roots are children of $x_k \in \pi(r_t)$ and are visited without finding r_t.

The tree can be managed by means of any algorithm for the weighted list update problem. For example, in the case of RMTF, we perform the following steps. For any request r_t in σ we search the tree by means of a left dfs. Once r_t has been found we consider the path $\pi(r_t)$. For any $x_k \in \pi(r_t)$ we toss a coin, whose probability of a positive answer is proportional to $1/|S(x_k)|$ in order to decide whether x must be moved before all its siblings in the list of children of x_{k-1}. In the case of CMTF, for any $x_k \in \pi(r_t)$ we increment its counter by a quantity proportional to $1/|S(x_k)|$, and if the counter reaches or exceeds 1 then we subtract 1 to it and move x_1 before all its siblings.

In general, after the t-th requested item r_t has been found, for each $x_k \in \pi(r_t)$, we consider the list $\mathcal{L}^{t-1}(x_k)$ of its children before the request r_t is served. So, we have $|\pi(r_t)|$

lists, corresponding to as many weighted lists to update. We can apply one heuristic for the WLUP to update each of these lists.

In the following, A_T denotes the algorithm for the tree update problem obtained by applying the WLUP algorithm A_L to each of the $|\pi(r_t)|$ lists, $t = 1, 2, \ldots$, according to the proposed strategy, and $A_T(\sigma)$ denotes its wasted cost to process a sequence σ of requests on the tree T.

The next theorem motivates this approach.

Theorem 4.1 *Given an algorithm A_L for the weighted list update problem, for any non-modifiable tree T, and any sequence σ of requests:*

$$A_T(\sigma) = \sum_{v_k \in T} A_L(\mathcal{L}(v_k), w^{(k)}, \sigma_k)$$

in the wasted cost model, where:

- *$A_L(\mathcal{L}(u), w, \sigma)$ denotes the cost incurred by the WLUP algorithm A_L applied to the list $\mathcal{L}(u)$ of the children of u, with a weight function w and a sequence of requests σ,*

and, for any vertex $v_k \in T$:

- *$w^{(k)}$ is the weight function such that, for any child u of v_k, $w^{(k)}(u) = |S(u)|$, that is the size of the subtree rooted at u;*
- *σ_k is the sequence of requests restricted to the items belonging to $S(v_k)$.*

Proof. The theorem follows from the fact that rearranging the children of a vertex v can only affect the cost of finding vertices in $S(v)$.

Let us introduce the following quantities:

$$q_{k,t} = \begin{cases} 1 & \text{if } v_k \in \pi(r_t) \\ 0 & \text{otherwise} \end{cases}$$

Let us denote as $W_{k,t}$ the wasted work charged to any vertex $x_k \in \pi(r_t)$ while searching T for r_t, due to the wrong choices made before finding the right child x_{k+1}:

$$W_{k,t} = \sum_{u \prec^{t-1} x_{k+1}} w^{(k)}(u).$$

Using the introduced notation, the theorem can be obtained as follows:

$$A_T(\sigma) = \sum_{r_t \in \sigma} \sum_{v_k \in \pi(r_t)} W_{k,t} = \sum_{r_t \in \sigma} \sum_{v_k \in T} q_{k,t} W_{k,t} = \sum_{v_k \in T} \sum_{r_t \in \sigma} q_{k,t} W_{k,t} = \sum_{v_k \in T} \sum_{r_t \in \sigma_k} W_{k,t}$$

□

Theorem 4.1 allows to extend to the tree update problem any result regarding the competitiveness of heuristics for the weighted list update problem.

For example it is easy to obtain the performance of RMTF and CMTF against the lazy adversary, using the optimal static arrangement of the tree.

Let $n(v)$ be the number of occurrences of the descendants of v (including v) in σ. The optimal static arrangement for a non-modifiable tree is such that for any two siblings u and v we have that $u \prec v$ in the adjacency list of their parent only if $\frac{n(u)}{w(S(u))} \geq \frac{n(v)}{w(S(v))}$.

Theorem 4.2 *In the wasted cost model, for any non-modifiable tree T and any sequence σ of requests:*

$$CMTF_T(\sigma) \leq 2 \cdot LAZY(T, \sigma) ,$$

$$E\{RMTF_T(\sigma)\} \leq 2 \cdot LAZY_T(\sigma) .$$

Proof. What asserted follows by theorems 3.1, 3.2 and 4.1. □

On the base of what seen we can extend the result on the non-competivity of MTF.

Theorem 4.3 *In the wasted cost model, for any $c \geq 0$ there exists a non-modifiable tree T and a sequence σ of requests such that:*

$$MTF_T(\sigma) > c \cdot LAZY_T(\sigma) .$$

Proof. The thesis holds by virtue of theorem 3.3 and the observation that any instance of weighted list update problem can be transformed into an instance of tree update problem (using a tree of depth 1). □

5 Conclusions

The question of whether c-competitive algorithms exist for the weighted list update problem against a strong adversary is still open. Another basic problem is finding some randomized algorithms better than 2-competitive against an oblivious adversary.

Many possible generalizations of the proposed algorithms for the weighted list update problem might be considered to deal with some of the open problems. In order to gain advantage against an oblivious adversary, CMTF may be modified by introducing a random initialization of the counters: this makes it similar to the COUNTER algorithm [13]. The random resources are required only in the initialization of the data structures, and not for the whole length of the sequence. Moreover, we observe that RMTF can be used for unweighted lists by moving accessed items to the front of the list with constant probability p. Using $p = 1/2$ we conjecture this algorithm to be as good as the BIT algorithm, which is 1.75-competitive against any oblivious off-line adversary [13].

Some of the current research trends in on-line algorithms (and in particular the *paging* problem) are based on the *access graph* [5, 12] to restrict the arbitrariety of the adversary in generating sequence of requests. An important issue would be to extend these approaches toward weighted versions of the problem to make them better suited for the requirements of applicative environments. Weighted counters and biased coins are candidates to tackle with this kind of problems.

Acknowledgements

We like to thank Mike Luby for constructive discussions helpful to improve an earlier version of the paper.

References

[1] Y. Azar, U. Nanni, *personal communication*, 1991.

[2] S. Ben-David, A. Borodin, R. Karp, G. Tardos, and A. Wigderson, On the Power of Randomization in On-Line Algorithms, in *Proceedings of the 2Oth ACM Annual Symposium on Theory of Computing*, May 1990, 379–386.

[3] J. L. Bentley, and C. McGeogh, Amortized Analyses of Self-Organizing Sequential Search Heuristics, *Communications of the ACM* **28**, 4 (April 1985), 404–411.

[4] J. R. Bitner, Heuristics that Dynamically Organize Data Structures, *SIAM J. of Computing* **8**, 1 (February 1979), 82–110.

[5] A. Borodin, S. Irani, P. Raghavan, B. Schieber, Competitive Paging with Locality of Reference, in *Proceedings of the 21th ACM Annual Symposium on Theory of Computing*, 1991, 249–259.

[6] F. d'Amore, U. Nanni, and A. Marchetti-Spaccamela, Robust Algorithms for Diagnosis, Technical Report, Dipartimento di Informatica e Sistemistica, Univ. of Roma "La Sapienza", 1991.

[7] A. Fiat, R. M. Karp, M. Luby, L. A. McGeoch, D. D. Sleator, N. E. Young, Competitive Paging Algorithms, *Journal of Algorithms* **12**, (1991), 685–699.

[8] S. Gnesi, U. Montanari, and A. Martelli, Dynamic programming as graph searching: An algebraic approach, *Journal of ACM* **28**, (1981), 737–751.

[9] J. H. Hester, and D. S. Hirschberg, Self-Organizing Linear Search, *ACM Computing Surveys* **17**, 3 (September 1985), 295–311.

[10] M. Hofri, H. Shachnai, *On the limited utility of auxiliary information in the list update problem*, manuscript, 1991.

[11] S. Irani, Two Results on the List Update Problem, *Technical Report TR-90-037*, Computer Science Division, U. C. Berkeley, California, August 1990.

[12] S. Irani, A. R. Karlin, S. Phillips, Strongly Competitive Algorithms for Paging with Locality of Reference, to appear in *Proceedings of the 3rd ACM-SIAM Annual Symposium on Discrete Algorithms*, Orlando, January 1992.

[13] S. Irani, N. Reingold, J. Westbrook, and D. D. Sleator, Randomized Competitive Algorithms for the List Update Problem, *Proceedings of the 2nd ACM-SIAM Annual Symposium on Discrete Algorithms*, San Francisco, CA, January 1991, 251–260.

[14] A. R. Karlin, M. S. Manasse, L. Rudolph, and D. D. Sleator, Competitive Snoopy Catching, *Algorithmica* **3**, 1, 1988, 79–119.

[15] R. Karp, P. Raghavan, *unpublished result*, 1990.

[16] M. S. Manasse, L. A. McGeoch, and D. D. Sleator, Competitive Algorithms for On-line Problems, *Proceedings of the 18th ACM Annual Symposium on Theory of Computing*, May 1988, 322–333.

[17] M. S. Manasse, L. A. McGeoch, and D. D. Sleator, Competitive Algorithms for Server Problems, *Journal of Algorithms* **11**, 2, 1990, 208–230.

[18] N. J. Nilsson, *Principles of Artificial Intelligence*, Springer Verlag, (1982).

[19] R. Reiter, A Theory of Diagnosis from First Principles, *Artificial Intelligence* **32**, (1987), 57–95.

[20] R. Rivest, On Self-Organizing Sequential Search Heuristics, *Communications of the ACM* **19**, 2 (February 1976), 63–67.

[21] D. D. Sleator, and R. E. Tarjan, Amortized Efficiency of List Update and Paging Rules, *Communications of the ACM* **28**, 2 (February 1985), 202–208.

[22] R. E. Tarjan, Amortized Computational Complexity, *SIAM J. Alg. Disc. Meth.* **6**, 2 (April 1985), 306–318.