On List Update and Work Function Algorithms

Eric J. Anderson[1], Kris Hildrum[2], Anna R. Karlin[1], April Rasala[3], and Michael Saks[4]

[1] Dept. of Computer Science, Univ. of Wash., {eric,karlin}@cs.washington.edu
[2] Computer Science Div., Univ. of Calif., Berkeley, hildrum@cs.berkeley.edu
[3] Laboratory of Computer Science, Massachusetts Inst. of Technology, arasala@theory.lcs.mit.edu
[4] Dept. of Mathematics, Rutgers Univ., saks@math.rutgers.edu

Abstract. The *list update* problem, a well-studied problem in dynamic data structures, can be described abstractly as a metrical task system. In this paper, we prove that a generic metrical task system algorithm, called the *work function algorithm*, has constant competitive ratio for list update. In the process, we present a new formulation of the well-known "list factoring" technique in terms of a partial order on the elements of the list. This approach leads to a new simple proof that a large class of online algorithms, including Move-To-Front, is $(2 - 1/k)$-competitive.

1 Introduction

1.1 Motivation

The *list accessing* or *list update* problem is one of the most well-studied problems in competitive analysis [1],[2],[3],[4],[5]. The problem consists of maintaining a set S of items in an unsorted linked list, for example as a data structure for implementation of a dictionary. The data structure must support three types of requests: ACCESS(x), INSERT(x) and DELETE(x), where x is the name, or "key", of an item stored in the list. We associate a cost with each of these operations as follows: accessing or deleting the i-th item on the list costs i; inserting a new item costs $j + 1$ where j is the number of items currently on the list before insertion. We also allow the list to be reorganized, at a cost measured in terms of the minimum number of transpositions of consecutive items needed for the reorganization. We consider the standard cost model in the literature: immediately after an access or an insertion, the requested item may be moved at no extra cost to a position closer to the front of the list. These exchanges are called *free exchanges.* Intuitively, using free exchanges, the algorithm can lower the cost on subsequent requests. In addition, at any time, two adjacent items in the list can be exchanged at a cost of 1. These exchanges are called *paid exchanges.* The list update problem is to devise an algorithm for reorganizing the list, by performing free and/or paid exchanges, that minimizes search and reorganization costs. As usual, the algorithm will be evaluated in terms of its competitive ratio.

As with much of the work on list accessing, we will focus on the *static list update problem*, where the list starts out with k elements in it, and all requests are accesses. The results described are easily extended to the dynamic case including insertions and deletions. Specifically, the cost of an insertion is the same for any algorithm; and the cost of a deletion is the same as the cost of an access. Some results for static list update are expressed in terms of the length k of the list. In the case of dynamic list update, the length k is no longer uniquely defined. However, for constant-competitive ratio results, it is enough for our purposes to interpret k as the *maximum* length of the list where necessary.

Many deterministic online algorithms have been proposed for the list update problem. Of these, perhaps the most well-known is the *Move-To-Front* algorithm: after accessing an item, move it to the front of the list, without changing the relative order of the other items. *Move-To-Front* is known to be $2 - \frac{2}{k+1}$ competitive, and this is best possible [2],[7].

We note that other cost models have also been considered for the list update problem [6], [?], [?]. Increasing the cost of exchanges to two (instead of one) makes *Move-To-Front* optimal; this provides an independent proof that *Move-To-Front* is two-competitive. Other alternatives analyzed in the literature include allowing free exchanges for other than the referenced element, and allowing free exchanges between elements that are not adjacent[?], [?]. These alternative cost models can lead to qualitatively different results.

1.2 Metrical task systems

The (static) list update problem can also be considered within the *metrical task system* framework introduced by Borodin, Linial and Saks [8]. Metrical task systems (MTS) are an abstract model for online computation that captures a wide variety of online problems (paging, list update and the k-server problem, to name a few) as special cases. A metrical task system is a system with n states, with a distance function d defined on the states: $d(i, j)$ is the distance between states i and j. The distances are assumed to form a metric. The MTS has a set $\mathcal{T}$ of allowable tasks; each task $\tau \in \mathcal{T}$ is a vector $(\tau(1), \tau(2), \ldots, \tau(n))$ where $\tau(i)$ is the (nonnegative) cost of processing task τ in state i. An online algorithm is given a starting state and a sequence of tasks to be processed online, and must decide in which state to process each task. The goal of the algorithm is to minimize the total distance moved plus the total processing costs.

The list update problem can be viewed as a metrical task system as follows. The states of the list update MTS are the $k!$ possible orderings of the k elements in the list, which we also call *list configurations*. There are k possible tasks, one corresponding to each list element x. The cost $\tau_x(\pi)$ of processing the task τ_x in a particular list configuration π, is equal to the depth of x in the list π. The distance between two states or list configurations is the number of inversions between the list orderings, considered as permutations.[1]

[1] In this formulation, "free exchanges" are treated as made at unit cost immediately before the item is referenced. Because the cost of these exchanges is precisely offset

One of the initial results about metrical task systems was that the *work function algorithm* (WFA) has competitive ratio $2n - 1$ for all MTS's, where n is the number of states in the metrical task system [8]. It was also shown that this is best possible, in the sense that there exist metrical task systems for which no online algorithm can achieve a competitive ratio lower than $2n - 1$. However, for many MTS's the upper bound of $2n-1$ is significantly higher than the best achievable competitive ratio. For example, there are known constant-competitive algorithms for list update, even though the MTS for a list of k elements has $k!$ states. Another example is the k-server problem on a finite metric space consisting of r points. For this problem, the metrical task system has $n = \binom{r}{k}$ states, but a celebrated result of Koutsoupias and Papadimitriou shows that in fact the *very same work function algorithm* is $2k-1$ competitive for this problem [9], nearly matching the known lower bound of k on the competitive ratio [10].

Unfortunately, our community understands very little at this point about how to design competitive algorithms that achieve close to the best possible competitive ratio for broad classes of metrical task systems. Indeed, one of the most intriguing open questions in this area is:

> For what metrical task systems is the work function algorithm strongly competitive? [2]

Burley and Irani have shown the existence of metrical task systems for which the work function algorithm is not strongly competitive [11]. However, these "bad" metrical task systems seem to be rather contrived, and it is widely believed that the work function algorithm is in fact strongly competitive for large classes of natural metrical task systems. The desire to make progress towards answering this broad question is the foremost motivation for the work described in this paper. We were specifically led to reconsider the list update problem when we observed the following curious fact:

> The *Move-To-Front* algorithm for list update is a work function algorithm. (Proposition 8, Section 4.)

This observation was intriguing for two reasons. First because it raised the question of whether work function algorithms generally (that is, those with more general tie-breaking rules than that used in *Move-To-Front*) are strongly competitive for list update. This would provide an example of a substantially different type of metrical task system for which the work function algorithm is strongly competitive than those considered in the past.

The second and perhaps more exciting reason for studying work functions as they relate to list update is the tantalizing possibility that insight gained from

by the lower reference cost, this model is identical to the standard model. See [6], Theorem 1. We continue to use the term "paid exchanges" to describe specifically those exchanges not involving the next-referenced element.

[2] We say an algorithm is *strongly competitive* if its competitive ratio is within a constant factor of the best competitive ratio achievable.

that study could be helpful in the study of dynamic optimality for self-adjusting binary search trees [1],[12]. It is a long-standing open question whether or not there is a strongly competitive algorithm for dynamically rearranging a binary search tree using rotations, in response to a sequence of accesses. The similarity between *Move-To-Front* as an algorithm for dynamically rearranging linked lists, and the splay tree algorithm of Sleator and Tarjan [12] for dynamically rearranging binary search trees, long conjectured to be strongly competitive, is appealing. Our hope is that the use of work function-like algorithms might help to resolve this question for self-adjusting binary search trees.

1.3 Results

The main result of this paper is a proof that a class of work function algorithms is $O(1)$ competitive for the list update problem.[3] Proving this theorem requires getting a handle on the work function values, the optimal offline costs of ending up in each state. This is tricky, as the offline problem is very poorly understood. At present it is even unknown whether the problem of computing the optimal cost of executing a request sequence is NP-hard. The fastest optimal off-line algorithm currently known runs in time $O(2^k k! m)$, where k is the size of the list and m is the length of the request sequence [6].

Using the framework that we have developed for studying work functions and list update, we also present a new simple and illustrative proof that *Move-To-Front* and a large class of other online algorithms are $(2 - 1/k)$-competitive.

The rest of the paper is organized as follows. In Section 2, we present background material on work functions and on the work function algorithm. In Section 3, we present a formulation of the list update work functions in terms of a partial order on the elements of the list and use this formulation to prove that a large class of list update algorithms are $(2 - 1/k)$-competitive. Finally, in Section 4 we present our main result, that work function algorithms are strongly competitive for list update.

2 Background

We begin with background material on work functions and work function algorithms.

2.1 Definitions

Consider an arbitrary metrical task system, with states $s \in S$ and tasks $\tau \in T$. Given a sequence of requests σ, denote the $t+1$st request in the sequence as σ_{t+1}. Let σ_{t+1} be the task τ. Let $\tau(s)$ denote the cost of executing task τ in state s.

[3] The proof does not achieve the best possible competitive ratio of 2.

Definition 1. *The* work function $\omega_t(s)$ *for any state s and index t is the lowest cost of satisfying the first t requests of σ and ending up in state s [13],[8].*

Because the states and task costs are time-independent, the work functions can be calculated through a dynamic programming formulation (which can equally be taken as the definition):

$$\omega_{t+1}(s') = \min_{s} \left(\omega_t(s) + \tau(s) + d(s, s')\right). \tag{1}$$

The work function algorithm is defined in terms of *fundamental* states:

Definition 2. *A state f is* fundamental at time t *if it satisfies* $\omega_{t+1}(f) = \omega_t(f) + \tau(f)$.

(Where the context is evident, we will simply say a state f is "fundamental".)

The *Work Function Algorithm* (WFA), [13],[8], defined for an arbitrary metrical task system, is the following:

Definition 3. *WFA: When in state s_t, service the request $\sigma_{t+1} = \tau$ in the state s_{t+1} such that*

$$s_{t+1} = argmin_s(\omega_{t+1}(s) + d(s_t, s)) \tag{2}$$

where the minimum is taken over states s that are fundamental at time t.

From the definition, we see that the work function algorithm chooses s_{t+1} so that

$$s_{t+1} = argmin_s \left(\omega_t(s) + \tau(s) + d(s_t, s)\right). \tag{3}$$

We consider a variant of this work function algorithm, differing only in the subscript of the work function:

Definition 4. *WFA': When in state s_t, service the request τ in the state s_{t+1} such that*

$$s_{t+1} = argmin_s \left(\omega_{t+1}(s) + \tau(s) + d(s_t, s)\right). \tag{4}$$

The minimum in the expression (4) may not be unique. Accordingly, we define the class of states to which the work function algorithm might move:

Definition 5. *Given state s_t at time t, a state s at time $t+1$ is* wfa-eligible *if it is one of the states that minimizes (4).*

Unless otherwise specified, the work function algorithm WFA' could move to any of these *wfa-eligible* states.

We next note several elementary identities, which hold at all times t and all states s and s'. As above, we let τ denote the $t+1$st task σ, and $\tau(s)$ its task cost in the state s.

2.2 Elementary identities

Proposition 1.

$$\omega_{t+1}(s) \geq \omega_t(s). \tag{5}$$

Proof. By the alternative definition above (1), for some s' we have $\omega_{t+1}(s) = \omega(s') + d(s,s') + \tau(s')$. By (7), $\omega_t(s) \leq \omega(s') + d(s,s')$. Since all task costs are nonnegative, $\tau(s') \geq 0$, and the result follows.

Proposition 2.

$$\omega_{t+1}(s) \leq \omega_t(s) + \tau(s). \tag{6}$$

Proof. By the definition (1), $\omega_{t+1}(s) = \min_{s'} (\omega_t(s') + \tau(s') + d(s,s'))$, so for all such s', $\omega_{t+1}(s) \leq \omega_t(s') + \tau(s') + d(s,s')$. Substituting s for s', and noting that $d(s,s) = 0$, the result follows.

Proposition 3.

$$\omega_t(s) \leq \omega_t(s') + d(s,s'). \tag{7}$$

Proof. For notational convenience, we show $\omega_{t+1}(s) \leq \omega_{t+1}(s') + d(s,s')$. From the definition (1), there is some $\hat{s}$ for which $\omega_{t+1}(s') = \omega_t(\hat{s}) + \tau(\hat{s}) + d(\hat{s},s')$. By the triangle inequality, $d(\hat{s},s') \leq d(\hat{s},s) + d(s,s')$. But $\omega_t(\hat{s}) + \tau(\hat{s}) + d(\hat{s},s)$ is a lower bound on $\omega_{t+1}(s)$ by (1), so $\omega_{t+1}(s') \leq \omega_{t+1}(s) + d(s,s')$.

Proposition 4. *For any s,*

$$\omega_{t+1}(s) = \omega_{t+1}(f) + d(f,s) \tag{8}$$

for some state f that is fundamental at time t. (The state s is derived from some *fundamental state.)*

Proof. By the definition (1), there is some f for which $\omega_{t+1}(s) = \omega_t(f) + \tau(f) + d(f,s)$. By (7), $\omega_{t+1}(s) \leq \omega_{t+1}(f) + d(f,s)$, so $\omega_t(f) + \tau(f) \leq \omega_{t+1}(f)$. But $\omega_t(f) + \tau(f) \geq \omega_{t+1}(f)$ by (6). Hence $\omega_t(f) + \tau(f) = \omega_{t+1}(f)$ and f is fundamental at time t. Then $\omega_{t+1}(s) = \omega_{t+1}(f) + d(f,s)$ by substitution.

Proposition 5. *Suppose WFA' is in state s_t at time t. Suppose s is wfa-eligible at time t, and suppose further that $\omega_{t+1}(s) = \omega_{t+1}(f) + d(f,s)$ where f is fundamental. (There is at least one such state f by (8).) Then f is also wfa-eligible at time t, and $x(f) = x(s)$. (The fundamental state f is wfa-eligible if s is.)*

Proof. Since s is wfa-eligible, it minimizes (4), $\omega_{t+1}(s) + \tau(s) + d(s_t,s) \leq \omega_{t+1}(s') + \tau(s') + d(s_t,s')$ for all s'. If we show that $\omega_{t+1}(f) + \tau(f) + d(s_t,f) \leq \omega_{t+1}(s) + \tau(s) + d(s_t,s)$, then f minimizes (4) as well, and f then must also be wfa-eligible.

We observe first that $\tau(f) \leq \tau(s)$. By (5) and (6), $\omega_{t+1}(s) \leq \omega_t(s) + \tau(s) \leq \omega_t(f) + \tau(s) + d(f,s)$. Then $\tau(s) < \tau(f)$ would imply $\omega_{t+1}(s) < \omega_t(f) + \tau(f) +$

$d(f, s) = \omega_{t+1}(f) + d(f, s)$. By hypothesis, however, we have $\omega_{t+1}(f) + d(f, s) = \omega_{t+1}(s)$.

Next, by the triangle inequality, $d(s_t, f) \leq d(s_t, s) + d(f, s)$. Then $\omega_{t+1}(f) + d(s_t, f) \leq \omega_{t+1}(f) + d(s_t, s) + d(f, s) = \omega_{t+1}(s) + d(s_t, s)$. Since $\tau(f) \leq \tau(s)$, we have $\omega_{t+1}(f) + d(s_t, f) + \tau(f) \leq \omega_{t+1}(s) + d(s_t, s) + \tau(s)$, and f is wfa-eligible.

Finally, since s is also wfa-eligible, the above inequality cannot be strict. It would be if $\tau(f) < \tau(s)$, so we must have $\tau(f) = \tau(s)$.

Proposition 6. *If s is wfa-eligible, then $\tau(s) \leq \tau(s_t)$.*

Proof. Suppose instead that $\tau(s) > \tau(s_t)$. Then the condition (4) for s to be wfa-eligible is $\omega_{t+1}(s) + \tau(s) + d(s, s_t) > \omega_{t+1}(s) + \tau(s_t) + d(s, s_t) \geq \omega_{t+1}(s_t) + \tau(s_t)$ by (7). But this last expression is (4) applied to the state s_t. If s_t satisfies (4) strictly more strongly than s, s cannot be wfa-eligible.

2.3 Observations

The work function algorithm can be viewed as a compromise between two very natural algorithms. First, a natural *greedy* algorithm tries to minimize the cost spent on the current step. It services the $(t + 1)$st request τ in a state s that minimizes $d(s_t, s) + \tau(s)$. Another natural algorithm is a *retrospective* algorithm, which tries to match the state chosen by the optimal offline algorithm. It services the $(t + 1)$st request τ in a state s that minimizes $\omega_{t+1}(s)$.

Each of these natural algorithms is known to be noncompetitive for many metrical task systems. WFA combines these approaches and, interestingly, this results in an algorithm which is known to be strongly competitive for a number of problems for which neither the *greedy* and *retrospective* algorithms are competitive.[4]

The difference between WFA and the variant, WFA', is in the subscript of the work function. We actually feel that WFA' is a slightly more natural algorithm, in light of the discussion above about combining a greedy approach and a retrospective approach. It is this latter work function algorithm WFA' that we will focus on in this paper. Our proof that WFA' is $O(1)$ competitive for list update can be extended to handle WFA as well. [WARNING – NOT YET SHOWN.] [5]

[4] Varying the relative weighting of the greedy and retrospective components of the work function algorithm was explored in [14].

[5] In addition, many prior results which hold for WFA also hold for WFA'. For example, for the k-server problem the work function values at t and $t + 1$ are identical for any states s that serve the $t + 1$st request, $\omega_{t+1}(s) = \omega_t(s)$. Hence WFA' and WFA define the same algorithm, and so WFA' is $2k - 1$ competitive for the k-server problem. The proof that WFA is $2n - 1$ competitive for any metrical task system with n states also holds for WFA' (using the same potential function), and so WFA' also is $2n - 1$ competitive for any metrical task system.

3 A different view on list factoring

A technique which has been used in the past to analyze list update algorithms is the *list factoring* technique, which reduces the competitive analysis of list accessing algorithms to lists of size two [3],[7],[15],[4],[16]. For example, this technique, in conjunction with phase partitioning, was used to prove that an algorithm called *TimeStamp* is 2-competitive [4],[16]. In this section, we repeat the development of this technique, but present it in a somewhat different way, in terms of a partial order on elements in the list.[6] This view leads us to a simple generalization of previous results and will assist us in our study of WFA'.

Consider the metrical task system corresponding to a list of length two. In this case there are two lists, (a, b) (a in front of b) and (b, a) (b in front of a), and the distance between these two states is 1. Since for all t we have $\omega_t((a, b)) - 1 \le \omega_t((b, a)) \le \omega_t((a, b)) + 1$, we can characterize the work functions at any given time t as having one of three distinct properties:

- $\omega_t((a, b)) < \omega_t((b, a))$, which we denote $a \succ b$,
- $\omega_t((a, b)) = \omega_t((b, a))$, which we denote $a \sim b$, or
- $\omega_t((a, b)) > \omega_t((b, a))$, which we denote $a \prec b$.

It is easy to verify directly from Equation (1) the transitions between these three properties as a result of references in the string σ.

$$\begin{tikzcd}
a \succ b \arrow[loop left, "a"] \arrow[r, bend right, "b"'] & a \sim b \arrow[l, bend right, "a"'] \arrow[r, bend right, "b"'] & a \prec b \arrow[l, bend right, "a"'] \arrow[loop right, "b"]
\end{tikzcd}$$

Fig. 1. The three-state DFA: the state $a \succ b$ corresponds to the case $\omega_t((a, b)) = \omega_t((b, a)) - 1$, the state $a \sim b$ corresponds to the case $\omega_t((a, b)) = \omega_t((b, a))$, and the state $a \prec b$ corresponds to the case $\omega_t((a, b)) = \omega_t((b, a)) + 1$

The resulting *three-state DFA* shown in Figure 1 can be used to completely characterize the work functions, the optimal offline list configuration, and the optimal cost to service a request sequence σ. The start state of the DFA is determined by the initial order of the elements in the list: it is $a \succ b$ if the initial list is (a, b) and $a \prec b$ if the initial list is (b, a). Each successive request in σ results in a change of state in accordance with the transitions of the DFA, reflecting the work function values after serving that request.

Notice that the number of times a is referenced when in the state $a \prec b$ plus the number of times that b is referenced when in the state $a \succ b$ is equal to the

[6] This partial order has apparently been considered by Albers, von Stengel and Werchner in the context of randomized list update, and was used as a basis for an optimal randomized online algorithm for lists of length 4. [17]

total number of transitions into the middle DFA-state. The optimal sequence cannot avoid incurring cost upon such references. Therefore, the optimal cost of satisfying a sequence of requests σ is given by the number of transitions into the middle state of the DFA, plus the length of the sequence. The corresponding optimal offline strategy is: Immediately before two or more references in a row to the same element, move that element to the front of the list.

Now consider list update for a list of length k. The cost of an optimal sequence can be written as the sum of the number of exchanges performed [7] and the reference costs at each state. For any pair of elements (a, b) we can identify a pairwise reference cost attributable to (a, b), adding one whenever b is referenced but a is in front of b in the list, or vice versa. The standard list factoring approach is to describe the cost of any optimal sequence for satisfying σ by decomposing it into $|\sigma|$ plus the sum over all pairs (a, b) of (i) this pairwise reference cost and (ii) all pairwise transpositions of a with b. For any pair (a, b), the sum of the pairwise transpositions and the pairwise reference cost describes a (possibly suboptimal) solution to the list of length two problem for the subsequence of σ consisting of references only to a and b. Therefore a lower bound on the optimal cost of satisfying σ is the sum of the costs of the optimal length-two solutions over all pairs (a, b), plus the length $|\sigma|$.

It is important to note that this "list factoring" lower bound is not tight.

Example 1. Consider a list of length five, initialized *abcde*, and the reference sequence $\sigma = ebddcceacde$. The sum of the length-two solutions, plus the length of σ, is 31; the optimal cost of satisfying σ is 32.

On the other hand, we do not know of any small examples where the optimal cost exceeds the list factoring lower bound by more than one, and we conjecture that the optimal cost does not exceed the lower bound by more than an additive constant related to the length of the list.

3.1 The partial order

We are thus led to consider the collection of $k(k-1)/2$ pairwise three-state DFAs, one for each pair a, b of elements in the list of length k. Consider the result of executing all these DFAs in parallel in response to requests in σ, starting from the states corresponding to the initial list. Figure 2 shows an example. Each DFA defines a pairwise relation, $a \prec b$, $a \succ b$, or $a \sim b$ as the case may be, on the elements a and b. It is easy to verify that at every time t the resulting collection of relations defines a valid *partial order* on the k elements of the list. In particular, the list configuration obtained by following *Move-To-Front* at every step is always consistent with this partial order.

This partial order at each time t is defined by the reference sequence σ, and does not depend on any choice of algorithm for list update. When we refer to the

[7] Recall that in our model we charge for each exchange, whether "paid" or "free"; each free exchange in the standard model precisely corresponds in our model to a reduced reference cost on the immediately following reference. See [6], Theorem 1.

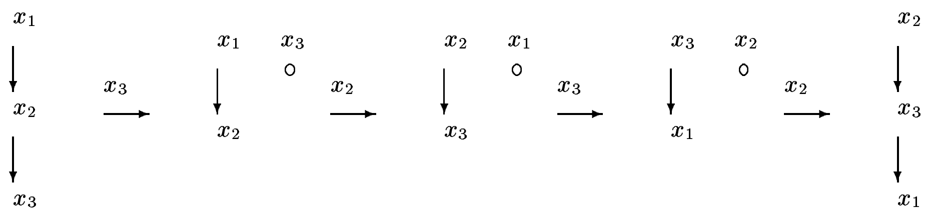

Fig. 2. Illustration of the evolution of the partial order on three elements in response to the request sequence $\sigma = x_3, x_2, x_3, x_2$ assuming the initial list is ordered x_1, x_2, x_3 from front to back. As usual, a directed edge from a to b indicates that $a \succ b$ in the partial order, whereas the absence of an edge indicates that $a \sim b$

"partial order", we mean this partial order as induced by a particular σ at a given time t. When we say that an algorithm is "consistent with the partial order", we mean that, when applied to a reference sequence σ, the list configuration visited by the algorithm at each time t, considered as a total order of the list elements, is consistent with the partial order induced by σ at that time t.

Define by G_t (respectively I_t) the number of elements greater than (respectively incomparable to) σ_t in this partial order immediately prior to its reference at time t. By the discussion above, the optimal cost of servicing a request sequence σ of length n and ending up in any state s is bounded below by the number of transitions into middle states of the DFAs, which at each step t is G_t. Hence for states s, $\omega_n(s) \geq n + \sum_{1 \leq t \leq n} G_t$.

An easy counting argument also shows:

Lemma 1. $\sum_t I_t \leq \sum_t G_t$.

Proof. Since we start with a total ordering on the elements, determined by the initial ordering of the list, each two element DFA begins either in state $a \prec b$ or $a \succ b$. For each DFA, each transition out of its middle state $a \sim b$ must therefore be preceded by a transition into the middle state. Taken together, this implies that, cumulatively, the number of transitions out of middle states cannot exceed the number of transitions into middle states. Since $\sum G_t$ is the cumulative number of transitions into middle states of the DFA's, and $\sum I_t$ the cumulative number of transitions out of middle states, the result follows.

Lemma 1 leads to a useful characterization of online algorithms:

Theorem 1. *Any online list update algorithm that performs only free exchanges and maintains the invariant that the list order is consistent with the partial order is* $(2 - 1/k)$*-competitive.*

Proof. Any online algorithm A that maintains a list order consistent with the partial order and performs no paid exchanges has a total cost $A(\sigma)$ satisfying $A(\sigma) \leq n + \sum_t (I_t + G_t)$, where $|\sigma| = n$.

By Lemma 1 and the fact that $OPT(\sigma) \leq kn$, we can conclude that $A(\sigma) \leq n + 2\sum_t G_t \leq (2 - 1/k)OPT(\sigma)$. □

3.2 Competitive analysis of online algorithms

Theorem 1 provides a new, simple proof that a collection of online algorithms (many already known to be competitive) are all $2 - 1/k$ competitive. These algorithms include *Move-To-Front*, *TimeStamp*, $MRI(k)$, and $SBR(\alpha)$ [4], [5], [?]. Each of these online algorithms moves only the referenced element. By Theorem 1, it is enough to show that these algorithms maintain lists consistent with the partial order.

We observed above that *Move-To-Front* maintains lists consistent with the partial order. Suppose the list is consistent with the partial order at time t, immediately before a reference to x. Then immediately after the reference (and after x is moved to the front), each element of the list is less than or incomparable to x, and is also behind x in the list. And because the respective pairwise order of other elements does not change, the list remains consistent with the partial order at time $t + 1$.

The *TimeStamp* algorithm (originally called *TimeStamp*(0)) due to Albers [4] is defined as follows:

> On a request for an item x, insert x in front of the first (from the front of the list) item y that precedes x on the list and was requested at most once since the last request for x. Do nothing if there is no such item y or if x is being requested for the first time.

The *TimeStamp* algorithm makes only free exchanges. Furthermore, by construction, after a reference to x, each item y that precedes it in the resulting list must have been requested at least twice since the last request for x. Therefore every element in front of x is incomparable to x (and not less than x) after the request. Each element behind x is less than or incomparable to x. Finally, the respective orders of other elements do not change as a result of the reference to x. Immediately prior to the initial reference to x, all elements in front of it are greater than it in the partial order. Hence *TimeStamp* maintains a list order consistent with the partial order.

Ran El-Yaniv has recently presented another family of algorithms, the $MRI(k)$ family [5]:

> On a request for an item x, move x forward to just behind the rearmost item y that precedes x on the list and was requested at least $k + 1$ times since the last request for x. If there is no such item y or if x is being requested for the first time, move x to the front.

El-Yaniv shows that $MRI(1)$ is equivalent (except for the first move of each element) to *TimeStamp*. Because any element that is requested more than twice since the last reference to x must be incomparable to x after the reference to x, the result follows for all k.

Schulz has recently presented the $SBR(\alpha)$ family [?]. From his Lemma 1 and the definition, the referenced element is moved forward at least as far as *TimeStamp*. Any such algorithm maintains a list order consistent with the partial order.

We have shown:

Corollary 1. *Move-To-Front, TimeStamp, RTS and $SBR(\alpha)$ are all $(2-1/k)$-competitive.*

4 On the performance of work function algorithms

4.1 Preliminaries

We begin with some definitions and facts. In what follows, the $(t+1)$st request σ_{t+1} is x. The task cost $\tau(s)$ is denoted $x(s)$, which is the depth of x in the list configuration s. As before, we denote by s_t the state visited by the work function algorithm at time t, immediately before the request to x.

We first define the $\uparrow_x$ binary relation on two states.

Definition 6. *$s \uparrow_x s'$ iff s and s' are identical, or if s' can be derived from s by moving x forward while leaving the relative positions of other elements undisturbed.*

Where x is understood from context, we write simply $s \uparrow s'$.

Proposition 7. *Suppose s is wfa-eligible, and $s \uparrow_x s'$. Then $\omega_{t+1}(s) \leq \omega_{t+1}(s')$. (Moving x forward cannot increase the work function.)*

[Is it true here that $\omega_{t+1}(s) = \omega_{t+1}(s')$? And is it therefore true that s wfa-eligible and $s \uparrow_x s'$ implies s' is wfa-eligible?]

Proof. In the case of list update, the "free exchange" cost model implies that whenever $s \uparrow_x s'$, $x(s) = x(s') + d(s, s')$. Suppose first that s is fundamental, $\omega_{t+1}(s) = \omega_t(s) + x(s)$. We have $\omega_{t+1}(s') \leq \omega_t(s') + x(s')$ by (6), and $\omega_t(s') \leq \omega_t(s) + d(s', s)$ by (7), so $\omega_{t+1}(s') \leq \omega_t(s) + d(s', s) + x(s')$. But $d(s', s) + x(s') = x(s)$ so $\omega_{t+1}(s') \leq \omega_t(s) + x(s) = \omega_{t+1}(s)$ as was to be shown.

Next suppose that s is wfa-eligible. By (7) we have $\omega_{t+1}(s) = \omega_{t+1}(f) + d(f, s)$ for some fundamental state f, for which also $x(f) = x(s)$. This means that $\omega_{t+1}(s) = \omega_t(f) + d(f, s) + x(f) = \omega_t(f) + d(f, s) + x(s)$. But $\omega_{t+1}(s') \leq \omega_t(s') + x(s') \leq \omega_t(f) + d(f, s') + x(s') \leq \omega_t(f) + d(f, s) + d(s, s') + x(s') = \omega_t(f) + d(f, s) + x(s)$, so $\omega_{t+1}(s') \leq \omega_{t+1}(s)$.

Recall from (6) that $\tau(s) \leq \tau(s_t)$, so the work function algorithm cannot move x backward.

We can now show that (a) there always exists a wfa-eligible state that requires no paid exchanges, and (b) that if WFA' is restricted to moving the referenced element only, it is equivalent to the following algorithm ("*Move-To-Min-ω*"):

> *$Mtm\omega$*: On a reference to x, move x forward (or not at all) to a state with lowest work function value immediately after the reference.

In other words, if s_t is the state the algorithm is in immediately before servicing the $t+1$-st request σ_{t+1}, then $Mtm\omega$ moves to a state s_{t+1} such that $s_{t+1} = argmin_{s\,:\,s_t \uparrow_x s} \omega_{t+1}(s)$ and satisfies σ_{t+1} there. Summarizing:

Proposition 8. *$Mtm\omega$ is a special case of WFA' and* Move-To-Front *is a special case of $Mtm\omega$.*

Proof. We first show that $Mtm\omega$ is a special case of WFA'. That is, we need to show that any state produced by $Mtm\omega$ is wfa-eligible. Suppose instead s is a state for which $s_t \uparrow s$, s minimizes $\omega_{t+1}(s)$ among all such, but s is not wfa-eligible. Then let s' be some wfa-eligible state for which $s_t \uparrow s'$. (The existence of such a state is demonstrated below.) Because both $s_t \uparrow s$ and $s_t \uparrow s'$, we have $d(s_t, s) + x(s) = x(s_t) = d(s_t, s') + x(s')$. Since by hypothesis $\omega_{t+1}(s) \leq \omega_{t+1}(s')$, then $\omega_{t+1}(s) + d(s_t, s) + x(s) \leq \omega_{t+1}(s') + d(s_t, s') + x(s')$; so s is wfa-eligible if s' is. Indeed, because s is a wfa-eligible state, the last inequality cannot be strict, and the last expression must be equality. The set of wfa-eligible states s for which $s_t \uparrow s$ thus all have the same value of $\omega_{t+1}(s)$. Since by (8) the work function value is non-increasing as x moves forward from s_t, if $s_t \uparrow s$ and s is wfa-eligible, then any s' for which $s \uparrow s'$ is also wfa-eligible.

(We note by the triangle inequality and (8) that this corollary is true for arbitrary wfa-eligible states: if $s \uparrow_x s'$, and s is wfa-eligible, then so is s'.)

For convenience in what follows, we denote generally by $\hat{s}$ the state formed from s by moving x to the front without changing the order of other elements, $s \uparrow_x \hat{s}$ and $x(\hat{s}) = 1$.

It remains to demonstrate that there is at least one wfa-eligible state s for which $s_t \uparrow s$. We show that the move-to-front state $\hat{s_t}$, the state which simultaneously satisfies $s_t \uparrow \hat{s_t}$ and $x(\hat{s_t}) = 1$, is wfa-eligible. By the above corollary, there must be some r wfa-eligible for which $x(r) = 1$ (for any wfa-eligible r', take $\hat{r'}$). It is a basic fact of permutation distance that $d(r, s_t) = d(r, \hat{s_t}) + d(\hat{s_t}, s_t)$, because the interchanges in $d(r, s_t)$ not involving x can all be resolved first, without moving x. Given this fact, then $\omega_{t+1}(r) + x(r) + d(r, s_t) = \omega_{t+1}(r) + x(\hat{s_t}) + d(r, \hat{s_t}) + d(\hat{s_t}, s_t)$. But $\omega_{t+1}(r) + d(r, \hat{s_t}) \geq \omega_{t+1}(\hat{s_t})$ by (7), hence $\omega_{t+1}(\hat{s_t}) + x(\hat{s_t}) + d(s_t, \hat{s_t}) \leq \omega_{t+1}(r) + x(r) + d(s_t, r)$, which was to be proved.

As a corollary, the move-to-front algorithm *Move-To-Front* is a special case of the work function algorithm.

4.2 WFA' is $O(1)$ competitive for list update.

In the preceding section, we characterized the work function algorithm in terms of the work function values of states formed by moving the referenced element forward. We noted that the work function value cannot increase as the referenced element is moved forward. In order to prove results about the work function algorithm, however, we must characterize all states states to which the work function algorithm could move; and thus we must characterize circumstances under which the work function value must *strictly decrease.* Our proof technique, then, supposes by hypothesis that the work function algorithm encounters a state of a particular undesired type; we consider the optimal sequence of interchanges and references that leads to the given work function value; then we must construct a new sequence, leading to a state identical to the first but for moving the referenced element forward, for which the total cost (of references and interchanges) is strictly lower.

The technically challenging part of the proof is the following lemma.

Lemma 2. *Consider $\sigma = \sigma_1, x, \sigma_2, x$, where in σ_2 there are no references to x, and $|\sigma| = t$. Let S be any fundamental state at the final time step t.*

Let $\mathcal{N}$ be the set of elements that are not referenced in σ_2 that are in front of x in S, and let $\mathcal{R}$ be the set of elements (not including x) that are referenced in σ_2. Also, let $\hat{S}$ be S with x moved forward just in front of the element in $\mathcal{N}$ closest to the front of the list. Then

$$\omega_t(\hat{S}) \leq \omega_t(S) + |\mathcal{R}| - |\mathcal{N}|. \tag{9}$$

Proof. Suppose O is an optimal sequence ending in S after satisfying σ_1, x, σ_2, x, so that the cost of O is the work function value $\omega_t(S)$. Let T denote the state in which O satisfies the penultimate reference to x (between $\sigma_1 and \sigma_2$). We note that, at the point immediately prior to the penultimate reference to x (at time k, say), the cost of O is $\omega_{k-1}(T)$. In this construction, we modify O between T and S so as to obtain a state $\hat{S}$, with $S \uparrow_x \hat{S}$ and $\omega_t(\hat{S}) \leq \omega_t(S) - |\mathcal{N}| + |\mathcal{R}|$.

Let N denote the total number of elements *not* referenced between $\sigma_k = x$ and $\sigma_t = x$. (This set specifically includes x, and is potentially much larger than $|\mathcal{N}|$, which is the number of such elements in front of x in S.) Order these non-referenced elements $p_1, \ldots, p_N$ in the order they occur in the state T.

The construction of the lower-cost state $\hat{S}$ proceeds in three stages (illustrated below):

1. Rearrange the respective order of the non-referenced elements within T to obtain some state T'. In T', x will occupy the location of the front-most non-referenced element in T. All other non-referenced elements p in T' will satisfy a *non-decreasing depth property*, that $p(T) \leq p(T')$.[8] All referenced elements remain at their original depths. (The specific definition of the state T' will emerge from the rest of the construction; the cost of the modified sequence can be bounded by using only the non-decreasing depth property.) Evaluate $\sigma_k = x$ in this state T'.

Denoting by $I[X, Y]$ the number of interchanges of non-referenced elements other than x between states X and Y, and using the non-decreasing depth property, we show (Proposition 9, proof deferred to the Appendix) that $x(T') + d(T, T') \leq x(T) + |\mathcal{R}| + \mathcal{I}[\mathcal{T}, \mathcal{T}']$.

2. Considering O as a sequence of transpositions and references transforming T to S, $O : T \to S$, apply a suitably chosen subsequence O', including all of the references and many of the transpositions, of O. This subsequence O' will transform T' to a state S'. In this state S', (i) each referenced element has the

[8] Recall that we denote the depth of an element p in the state X by $p(X)$.

same depth as it does in S; (ii) the element x occupies the position of the front-most non-referenced element in S; and (iii) all other non-referenced elements in S' are in their same respective pairwise order as in S. Evaluate x in S'.

We show (Proposition 11, proof deferred to the Appendix) that such a transformation from some T' with the non-decreasing depth property, to S' as so defined, can be achieved by a suitably chosen subsequence of O. We also show that $I[T,T']+I[T',S'] \leq I[T,S]$, by showing that all of the interchanges between non-referenced items from T to T' and from T' to S' are contained in O.

3. Transform S' to the state $\hat{S}$, where $\hat{S}$ is defined by (i) $S \uparrow_x \hat{S}$, and (ii) the depth of x in $\hat{S}$ is the depth of the front-most non-referenced element in S (which is also its depth in S').

We show (Proposition 10, proof deferred to the Appendix) that $x(S') + d(S',\hat{S}) + |\mathcal{N}| \leq x(S)$.

This process can be illustrated as follows, using $\rightarrow$ to denote a reference, and $\rightsquigarrow$ to denote pairwise interchanges between references. The original sequence O has:

$$\ldots \rightsquigarrow T \xrightarrow{\text{x}} T \rightsquigarrow \ldots \rightsquigarrow S \xrightarrow{\text{x}} S$$

(Recall that we assume that O satisfies $x = \sigma_t$ in S.)

After the above modifications (denoted 1, 2, 3), the sequence is:

$$\ldots \overset{\text{O}}{\rightsquigarrow} T \overset{1}{\rightsquigarrow} T' \xrightarrow{\text{x}} T' \ldots \overset{2}{\rightsquigarrow} S' \xrightarrow{\text{x}} S' \overset{3}{\rightsquigarrow} \hat{S}$$

The result now follows by comparing the cost of the modified sequence to the cost of the original sequence from and after $\omega_{k-1}(T)$. The cost attributable to the original sequence is the sum of

1. $x(T)$;
2. the cost of references in σ_2;
3. the cost of interchanges from T to S between referenced elements;
4. the cost of such interchanges between referenced and non-referenced elements;
5. the cost $I[T,S]$ of such interchanges between non-referenced elements; and
6. $x(S)$.

The cost attributable to the modified sequence is the sum of

1. the cost of interchanges leading from T to T';
2. $x(T')$;
3. the cost of references in σ_2;
4. the cost of interchanges from T' to S' between referenced elements;
5. the cost of such interchanges between referenced and non-referenced elements;
6. the cost $I[T',S']$ of interchanges between non-referenced elements;
7. $x(S')$; and
8. the cost of interchanges leading from S' to $\hat{S}$.

By construction, items two, three and four are identical for the two sequences. Thus we compare $x(T)+I[T,S]+x(S)$ for the first sequence to $d(T,T')+x(T')+I[T',S']+x(S')+d(S',\hat{S})$.

Given $x(T')+d(T,T') \leq x(T)+|\mathcal{R}|+\mathcal{I}[\mathcal{T},\mathcal{T}']$, $I[T,T']+I[T',S'] \leq I[T,S]$, and $x(S')+d(S',\hat{S})+|\mathcal{N}| \leq x(S)$, the result follows by substitution.

We obtain the following corollary to Lemma 2.

Corollary 2. *Consider a request sequence σ where the last request (the t-th request in σ) is to x. If s is wfa-eligible after executing σ, then the depth of x in s is at most $2|R|$, where R is the set of elements that have been referenced since the penultimate reference to x.*

Proof. Let f be a fundamental state such that $\omega_{t+1}(s)=\omega_{t+1}(f)+d(f,s)$. By Proposition 5, f is also wfa-eligible and $x(f)=x(s)$. Suppose $x(s)>2|R|$. Then $x(f)>2|R|$. Elements in front of x in f either have or have not been referenced since the penultimate reference to x; so $x(f)>2|R|$ implies $|N|>|R|$, where N is the set of elements in front of x in f that have not been referenced since the penultimate reference to x. Then by Lemma 2 there exists $\hat{f}$ with $\omega_t(\hat{f})<\omega_t(f)$ and $f \uparrow_x \hat{f}$, contradicting the assumption that f is wfa-eligible. □

Finally, we use the lemma to obtain the main theorem.

Theorem 2. *WFA' is O(1) competitive.*

Proof. We consider $Mtm\omega$ first. Consider an arbitrary element x, and let $\sigma=\sigma_0,x,\sigma_1,x,\sigma_2,x$, where in σ_1 and σ_2 there are no references to x. Then by Lemma 2 the depth of x in the $Mtm\omega$ state, immediately before the final reference to x, is at most $2r_1+r_2$, where r_1 is the number of distinct elements referenced in σ_1 and r_2 is the number of distinct elements referenced in σ_2, not referenced in σ_1, that are moved in front of x at some point during the subsequence σ_2.

As usual, let G be the number of elements greater than x immediately before its final reference and let I be the number of elements incomparable to x immediately before its final reference. In addition, let $L(0)$ be the number of elements less than x immediately before its final reference that were incomparable to x immediately before the penultimate reference to x. We have $r_1+r_2 \leq G+I+L(0)$.

Denote by t_1 the time of the penultimate reference to x, and by t_2 the time of the final reference. Since each element in $L(0)$ at time t_2 is incomparable to x at time t_1, we have $L(0)_{t_2} \leq I_{t_1}$. That is, for any t_2, there is some $t_1<t_2$ such that $L(0)_{t_2} \leq I_{t_1}$. Thus $\sum_t L(0)_t \leq \sum_t I_t$. But $\sum_t I_t \leq \sum_t G_t$ by the counting argument, Lemma 1. Summarizing, we have

$$WFA'(\sigma) \leq \sum_t (2r_1+r_2) \leq 2(r_1+r_2)$$
$$\leq 2\sum_t (G_t+I_t+L(0)_t) \leq 6\sum_t G_t \leq 6OPT(\sigma)$$

Now suppose that WFA' can introduce paid exchanges. To extend the proof from $Mtm\omega$ to WFA', we would demonstrate (1) that the paid exchanges do not affect the validity of Lemma 2, and (2) that the cost of each paid exchange can be allocated uniquely to an earlier reference of the rearward element of the pair. Therefore the total cost of paid exchanges cannot exceed the total reference cost, and WFA' is at least 12-competitive.

Note that, for list update, the algorithm WFA (without the subscript) can be less effective than WFA'. Consider the sequence $\sigma = bbb$ for a two-element list (a, b). After the second reference to b, the list configuration (b, a) has strictly lower work function value. But WFA does not (necessarily) move to that state until after the *third* reference to b. However, we believe it is possible to extend the above proof of $O(1)$-competitiveness to WFA.

It is fairly clear that the competitive ratios shown by our analyses of these algorithms are not tight. The above example shows that WFA, even without paid exchanges, is no better than 3-competitive.

5 Acknowledgments

We gratefully acknowledge discussions with Susanne Albers, Ran El-Yaniv, Sandy Irani, and Jayram Thathachar.

This work was supported in part by NSF grant EIA-9870740 and BSF grant 96-00247 (Karlin), the CRA Distributed Mentor Project (Hildrum and Rasala), and an IBM Research Fellowship (Anderson).

References

1. S. Albers and J. Westbrook. Self-organizing data structures. In *Online Algorithms: The State of the Art*, Fiat-Woeginger, Springer, 1998.
2. D.D. Sleator and R.E. Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28:202–208, 1985.
3. J.L. Bentley and C. McGeoch. Amortized analysis of self-organizing sequential search heuristics. *Communications of the ACM*, 28(4):404–411, 1985.
4. S. Albers. Improved randomized on-line algorithms for the list update problem. *SIAM Journal on Computing*, 27: 682–693, 1998.
5. R. El-Yaniv. There are infinitely many competitive-optimal online list accessing algorithms. Discussion paper from The Center for Rationality and Interactive Decision Making. Hebrew University.
6. N. Reingold and J. Westbrook. Off-line algorithms for the list update problem. *Information Processing Letters*, 60(2):75–80, 1996.
7. S. Irani. Two results on the list update problem. *Information Processing Letters*, 38(6):301–306, 1991.
8. A. Borodin, N. Linial, and M. Saks. An optimal online algorithm for metrical task systems. *Journal of the ACM*, 52:46–52, 1985.
9. E. Koutsoupias and C. Papadimitriou. On the k-server conjecture. *Journal of the ACM*, 42(5): 971–983, September 1995.
10. M. Manasse, L. McGeoch and D.D. Sleator. Competitive algorithms for server problems. *Journal of Algorithms*, 11:208–230, 1990.

11. W. Burley and S. Irani. On algorithm design for metrical task systems. In *Proceedings of ACM-SIAM Symposium on Discrete Algorithms*, 1995.
12. D.D. Sleator and R.E. Tarjan. Self-adjusting binary search trees. *Journal of the ACM*, 32: 652-686, 1985.
13. M. Chrobak, L. Larmore. The server problem and on-line games. In *On-Line Algorithms, Proceedings of a DIMACS Workshop*, Vol 7 of *DIMACS Series in Discrete Mathematics and Computer Science*, pp. 11 – 64, 1991.
14. W.R. Burley. Traversing layered graphs using the work function algorithm. *Journal of Algorithms*, 20(3):479–511, 1996.
15. B. Teia. A lower bound for randomized list update algorithms. *Information Processing Letters*, 47:5–9, 1993.
16. S. Albers, B. von Stengel and R. Werchner. A combined BIT and TIMESTAMP algorithm for the list update problem. *Information Processing Letters*; 56: 135–139, 1995.
17. S. Albers. Private communication.

In this Appendix, we address the proofs of the three propositions leading to Lemma 2. The most intricate part of the part of the construction is contained in Proposition 11; we save its proof for last. Then we present some concluding remarks about the construction and the prospect for improved bounds. [Omitted.]

Proposition 9. *Suppose T' is derived from T, such that (i) all referenced elements p have $p(T') = p(T)$, (ii) x occupies in T' the location of the front-most non-referenced element in T, and (iii) T' has the non-decreasing depth property, $p(T') \geq p(T)$ for all $p \neq x$. Then the number of exchanges required to transform T to T' is bounded by (i) the number of interchanges involving x, plus (ii) the number of referenced elements, plus (iii) the number of interchanges involving non-referenced elements. In particular, the cost of the reference to x in T is equal to the cost of the reference in T', plus the number of interchanges involving x.*

Proof. If the first non-referenced element in T is x itself, then in order to satisfy the non-decreasing depth property, $p(T') = p(T)$ for all non-referenced elements, and there is nothing to prove. Otherwise, denote by $p_1, \ldots, p_N$ the non-referenced elements in their order in T, with $p_z = x$. We note that for $i > z$, the non-decreasing depth property requires $p_i(T) = p_i(T')$. We start by moving x forward to the location of the first non-referenced element, p_1. These interchanges all involve x. Next, we move p_1 to location 2.[9] By the non-decreasing depth property, either p_1 or p_2 must occupy location 2. If p_2 is x, we are done. Otherwise, there may be one interchange, between p_1 and p_2. Inductively, at step i, for location i, each referenced element below location i has interchanged with at most one non-referenced element, and each referenced element above location i has not interchanged with any non-referenced elements; and some element $p_j, j < i$, is either (i) adjacent to p_i, or (ii) is in location i and p_i is x. If the latter, we are done. If the former, one or the other of p_i and p_j must occupy location i

[9] In what follows, by slight abuse of notation, we refer to the location of the i'th non-referenced element in T by the description "location i" or "position i".

by the non-decreasing depth property; swap them if necessary; and interchange the other with all referenced elements between location i and location $i+1$. By induction, each referenced element has interchanged with at most one non-referenced element, so the number of such interchanges is bounded by the number of referenced elements. The result follows.

We next prove Proposition 10, which is in some sense the obverse of the preceding one. Here x is already ahead of all other non-referenced elements; we move the non-referenced elements forward to their ending positions.

Proposition 10. *As above, let S' be derived from the state S as follows:*

- *All referenced elements are in the same locations and in the same order in S' as in S.*
- *x occupies in S' the position of the front-most non-referenced element in S.*
- *All other non-referenced elements are in the same pairwise order in S' as in S.*

As above, let $\hat{S}$ be derived from the state S by moving x forward to immediately in front of the front-most non-referenced element. Then the cost $x(S')$ of the reference to x in state S', plus the distance $d(S', \hat{S})$ from S' to $\hat{S}$, is less than the depth $x(S)$ of x in S by at least the number of non-referenced elements in front of x in S. That is, we have

$$x(S') + d(S', \hat{S}) + |\mathcal{N}| \leq x(S).$$

Proof. Suppose x occupies the i'th non-referenced location from the front in S. (That is, $|\mathcal{N}| = i$.) Denote the first i non-referenced elements of S in order by $q_1, \ldots, q_i = x$. In S', the element q_{i-1} occupies position i; q_{i-2}, position $i-1$; and so on; q_1 occupies position 2; and x occupies position 1. We transform S' to $\hat{S}$ by interchanging, for all $1 < j < i$, q_j with all referenced elements between it and q_{j-1}, and q_1 with all referenced elements between it and x. Each referenced element between x and q_{i-1} interchanges with at most one non-referenced element, and each such is in front of x in S. Thus the number of exchanges required to transform S' to $\hat{S}$, plus the number of referenced elements in front of x in S', plus the number of non-referenced elements in front of x in S, is no greater than the depth of x in S. The result follows.

Finally, we address the most intricate part of the construction:

Proposition 11. *Suppose S' is derived from S, such that (i) all referenced elements p have $p(S') = p(S)$, (ii) x occupies in S' the position of the front-most non-referenced element in S, and (iii) all other non-referenced elements are in their same respective order in S' as in S. Then there is a T' with the non-decreasing depth property, and a subsequence $O' \subseteq O$, such that (i) $O'(T') = S'$, and (ii) the cost of O is at least the cost of O' plus the cost of interchanges $I[T', T]$ of non-referenced elements (other than x) necessary to derive T' from T.*

Proof. As above, we denote by p_i the non-referenced element occupying the i'th non-referenced position in T. For convenience, let z denote the location of x as a non-referenced element in T, $p_z = x$.[10]

We proceed by iteratively constructing T' from the end of the list, beginning with $O^{-1}(S')$. The location of referenced elements remains fixed throughout the construction. As a result, we consider only the N positions of non-referenced elements. For convenience, we describe the iteration as proceeding from $i = N$ to $i = 1$. (The "base case" is denoted by "$i = N + 1$".) At each step, then, we define a map $O_i : T'_i \to S'$. The non-decreasing depth property is maintained for the elements (other than x) in $O_i^{-1}(S') = T'_i$ that occupy the locations i through N in T'_i. We show that any necessary interchanges of elements as we proceed from T'_i to T'_{i-1} correspond to transpositions in O'_i.

For each pair of elements $p, q \neq x$ at locations i and below in T'_i, we can determine whether these two elements are in the same or in the opposite order in T. We denote by $I_i[T'_i, T]$ the number of pairwise inversions of such elements (other than x). We denote by $|O|$ (respectively, $|O_i|$) the number of transpositions in the sequence O (respectively, O_i).

Formally, we show by induction that for each i:

1. $O_i(T'_i) = S'$ (and $O_i^{-1} : S' \to T'_i$)
2. $O_i \subseteq O$ in the sense of a subsequence of transpositions, and $|O| \geq |O_i| + I_i[T'_i, T]$ (all swaps and inversions are accounted for)
3. $x(T'_i) \leq p_i(T)$ (x is no deeper than position i)
4. $\forall p \neq x$ with $p(T) \geq p_i(T), p(T'_i) \geq p(T)$ (all elements other than x at position i or below in T have the non-decreasing depth property)
5. $\forall p, q \neq x$ with $p(T), q(T) < p_i(T)$:
   (a) $p(S) < x(S) \iff p(T'_i) \neq p(T)$, and $p(S) > x(S) \iff p(T'_i) = p(T)$
   (b) $p(T) = q(T'_i) \implies p(S) > q(S)$

To carry out the induction proof, we will start by demonstrating the hypotheses for an appropriate base case. For the induction step, we assume the five hypotheses for $i + 1$, derive a transformation O_i, and show the validity of the hypotheses for i. Then we define $T' = T_1$, and note that the non-increasing depth property is satisfied for all $p_i \neq x$. We define $O' = O_1$, and note all of the inversions between non-referenced elements in T' have been accounted for, i.e., $I[T, T'] + |O'| \leq |O|$. Finally, we repeat that because the only transpositions removed from O are between non-referenced elements, the depths, and thus the reference costs, of all referenced elements remains identical between O and O'.

□

The base case. For the base case, we define $O_b = O, T'_b = O^{-1}(T')$. (Notationally, b is $N + 1$.) Then (1) and (2) follow from our definition (I_b is zero). Items (3) and (4) are vacuous. We must show that items (5)(a) and (5)(b) are true for all non-referenced elements in T'_b. For item (5)(a), by construction of S', elements

[10] We use the terms "position" and "location" interchangeably to refer to the respective positions of non-referenced elements in T.

q deeper than x in S are unaffected by the shift, $q(S) > x(S) \Longrightarrow q(S) = q(S')$ so $q(T) = q(T'_b)$, while elements q closer to the front of S are "shifted down", $q(S) < x(S) \Longrightarrow q(S') \neq q(S)$ (indeed $q(S') > q(S)$), so $q(T) \neq q(T'_b)$.

Finally, for (5)(b), $p(T) = q(T'_b), p \neq q$ implies $p(T) \neq p(T'_b)$ implies (by (5)(a)) $p(S) < x(S)$, similarly $q(S) < x(S)$. By construction, $p(S') > p(S)$ by one non-referenced position, but $q(S') = p(S)$ since O^{-1} takes q to location $p(T)$ in T'_b. Hence $p(S') > q(S')$.

Induction step. Now suppose the entire hypothesis is true for $i + 1$ (including for example $b = N + 1$). We will construct an appropriate mapping O_i that satisfies the hypotheses for i. We describe three stages, depending on whether the element x has yet been considered. Denoting by z the location of x in T, so that $x = p_z$, we consider p_i for (i) $i > z$, (ii) $i = z$, (iii) $i < z$ in turn.

Case (i): $i > z$. We have the non-decreasing depth property for $j \in i+1, \ldots, N$, which because $i > z$ requires strictly that $p_j(T'_{i+1}) = p_j(T)$ for all locations j. In this case, the non-decreasing depth property applied to i will require strictly that $p_i(T) = p_i(T')$. Throughout this stage, in particular, the non-decreasing depth property for i implies $I_i[T'_i, T] = 0$.

We examine the current occupant of position i in T'_{i+1}. There are three possibilities to consider:

– The occupant is p_i itself, $p_i(T'_{i+1}) = p_i(T)$. In that case, set $O_i = O_{i+1}$. The non-decreasing depth property is (precisely) satisfied. Hypotheses (1) and (2) follow immediately from their validity for $i + 1$; hypothesis (3) and (4) follow from the depth property; and hypothesis (5) is more restrictive, hence valid.
– The occupant is x, $x(T'_{i+1}) = p_i(T)$. In this case, T'_i will be obtained by interchanging p_i with x. We observe that $p_i(T'_{i+1}) < x(T'_{i+1})$ (by the depth property at $i + 1$), and $x(S') < p_i(S')$ by construction (x is the front-most non-referenced element in S'), so x and p_i are inverted by O_{i+1}, and there is a transposition in O_{i+1} between them. Remove this transposition to get $O_i \subset O_{i+1}$. The non-decreasing depth property is again precisely satisfied, implying hypotheses (3) and (4); and hypothesis (5) is unchanged for $p_j, j < i$. (Hypothesis (5) does not apply to x.)
– The occupant is $p_j \neq x$, $p_j(T'_{i+1}) = p_i(T)$. In this case, T'_i will be obtained by interchanging p_j with p_i. We observe (again) that the depth property is precisely satisfied for $k > i$, so $p_i(T'_{i+1}) < p_j(T'_{i+1}) = p_i(T)$. Also, by hypothesis (5)(b), we have $p_j(T'_{i+1}) = p_i(T) \Rightarrow p_j(S) < p_i(S)$. Therefore, O_{i+1} inverts p_i and p_j. Remove this transposition to get $O_i \subset O_{i+1}$. The non-decreasing depth property is again precisely satisfied, implying hypotheses (3) and (4). We note that hypothesis (5) holds for p_j by transitivity: suppose p_i occupies location k in T'_{i+1}, $p_k(T) = p_i(T'_{i+1})$; and $p_i(T) = p_j(T'_{i+1})$; so by hypothesis (5)(b) at the previous step we have $p_j(S) < p_i(S) < p_k(S)$, implying (5)(b) at the current step (and in particular $j \neq k$, so (5)(a) is satisfied).

This concludes the analysis of the stage where the depth property is precisely satisfied, $p_i(T') = p_i(T) \forall i > z$.

Case (ii): $i = z$. In this case there is nothing to do, $O_i = O_{i+1}$. Since the depth property is satisfied for $i+1, \ldots, N$, and does not apply to x, it remains satisfied.

Case (iii): $i < z$. In this case, the elements occupying locations $i+1, \ldots, N$ are $p_{i+1}, \ldots, p_N$ other than x, together with (by the non-decreasing depth property) some single element p_j, which might be x.

We consider four cases: p_j is x; p_j is p_i; $p_j \neq x$, and p_i occupies location i in T'_{i+1}; $p_j \neq x, p_i$, and $p_i(T'_{i+1}) \neq p_i(T)$.

- p_j is x. In this case, by hypothesis (3), $p_j = x$ occupies location $i+1$ in T'_{i+1}. In this case, T'_i will be obtained by interchanging p_i and x. We know p_i is above x in T'_{i+1}, $p_i(T'_{i+1}) < x(T'_{i+1})$. But, as above, $x(S') < p_i(S')$ by construction (x is the front-most such in S'). Therefore there is a swap in O_{i+1} between x and p_i. Remove it to obtain $O_i \subset O_{i+1}$. We note that $I_i = I_{i+1}$, because x occupied location $i+1$ (hypothesis (3)), and all elements below location $i+i$ have index larger than i. The occupant of location i is left undetermined here; even if it is x, hypothesis (3) remains satisfied. Hypothesis (5) remains true, since only x's location has changed, and the hypothesis does not apply to it.
- p_j is p_i. In this case, we do nothing, $O_i = O_{i+1}$. We have by induction that p_i is the unique element in locations $i+1, \ldots, N$ whose index is less than $i+1$. This implies hypothesis (3). For the same reason, the depth property continues to be satisfied in T'_i. Also, $I_i = I_{i+1}$, because the inversions with respect to p_i were already counted in I_{i+1}, and the element occupying location i in T'_{i+1} (now T'_i, and considered in I_i for the first time) is either x or (by the depth property) has index less than i, so has no inversions with respect to any elements in locations $i+1, \ldots, N$. Hypothesis (5) is unchanged.
- p_j is neither x nor p_i, and $p_i(T'_{i+1}) = p_i(T)$. (That is, p_i occupies location i in T and in T'_{i+1}.) In this case, T'_i will be obtained by interchanging p_i with p_j. We have from hypothesis (5)(a) that $p_i(S) > x(S)$, and (since $j < i$, but $p_j(T'_{i+1}) > p_i(T)$) $p_j(T'_{i+1}) \neq p_j(T)$, so $p_j(S) < x(S)$. Hence there is an interchange in O_{i+1} between p_i and p_j. Remove it to obtain $O_i \subset O_{i+1}$. The element p_j now occupies location i, element p_i now satisfies the depth property, and elements $p_i, p_{i+1}, \ldots, p_N$ (other than x) occupy locations $i+1, \ldots, N$ (though not necessarily in that order), so hypothesis (3) is also satisfied. Furthermore, the inversions I_i are the same as in I_{i+1}, since p_j and p_i are the two elements with smallest indices among those occupying locations $i, i+1, \ldots, N$ (so that swapping p_i for p_j replaced inversions involving p_j with inversions involving p_i, but introduced no new inversions), and p_j is now in location i (so that there are no inversions involving p_j). Finally, we note that, even though the location of p_j has changed as a result of the swap, hypothesis (5)(a) remains satisfied because p_j now occupies location $i \neq j$, $p_j(T) \neq p_j(T'_i)$ as before; and hypothesis (5)(b) does not apply to location i.

- p_j is neither x nor p_i, and $p_i(T'_{i+1}) \neq p_i(T)$. In this case, $p_i(T'_{i+1}) < p_i(T)$ (that is, p_i occupies a position closer to the front of the list T'_{i+1} than position i). We swap p_i with the element (p_k, say) occupying position i in T'_{i+1}. We have from hypothesis (5)(b) that $p_k(S) < p_i(S)$, and here that $p_k(T'_{i+1}) = p_i(T) > p_i(T'_{i+1})$, so there is a swap between them. Remove it from O_{i+1} to obtain O_i. The depth property continues to be satisfied, as is hypothesis (3). The only element in locations $i+1, \ldots, N$ in T'_i that is not in locations $i+1, \ldots, N$ in T (that is, does not have index $\geq i+1$) is p_j. We have therefore introduced only one additional inversion (that between p_i and p_j) by reason of the progression from I_{i+1} to I_i. That additional inversion is offset by the swap between p_i and p_k that we have removed from O_{i+1} to obtain O_i. (This is the only case in which this construction requires this offset.) Thus hypothesis (2) remains valid. Finally, we show that hypothesis (5) remains valid for p_k, the element swapped with p_i. Suppose p_i occupied location l in T'_{i+1}. Then $p_i(S) > p_l(S)$ by hypothesis (5)(b) (induction), and $p_k(S) > p_i(S)$ by hypothesis (5)(b) (induction), so $p_k(S) > p_l(S)$, establishing (5)(b), and in particular $k \neq l$, establishing (5)(a).

This exhausts the possible cases for Proposition 11, and concludes the proof. □

Remarks. [Omitted.]