A Competitive Analysis of the List Update Problem with Lookahead

Susanne Albers*

Abstract

We consider the question of lookahead in the list update problem: What improvement can be achieved in terms of competitiveness if an on-line algorithm sees not only the present request to be served but also some future requests? We introduce two different models of lookahead and study the list update problem using these models. We develop lower bounds on the competitiveness that can be achieved by deterministic on-line algorithms with lookahead. Furthermore we present on-line algorithms with lookahead that are competitive against static off-line algorithms.

1 Introduction

In recent years there has been tremendous interest in the competitive analysis of on-line algorithms. Many on-line problems have been studied in areas such as resource allocation, data structures, graph problems, scheduling and navigation. In the context of data structures, the *list update problem* is of fundamental importance. The problem is to maintain a set of items as an unsorted linear list. A list of n items is given. A list update algorithm is presented with a sequence of *requests*, where each request specifies an item of the list. In order to serve a request, a list update algorithm must *access* the requested item, i.e., it has to start at the front of the list and search linearly through the items until the desired item is found. Accessing the ith item in the list incurs a cost of i. Immediately after a request, the accessed item may be moved at no extra cost to any position closer to the front of the list. These exchanges are called *free exchanges*. All other exchanges of two consecutive items in the list cost 1 and are called *paid exchanges*. The goal is to serve the request sequence such that the total cost is as small as possible. A list update algorithm is *on-line* if it serves each request without the knowledge of future requests.

Competitive analysis [27] is a powerful means to analyze the performance of on-line algorithms for the list update problem. In a competitive analysis, an on-line algorithm is compared to an

*Max-Planck-Institut für Informatik, Im Stadtwald, D-66123 Saarbrücken, Germany. E-mail: albers@mpi-sb.mpg.de. This work was supported by a Graduiertenkolleg graduate fellowship of the Deutsche Forschungsgemeinschaft.

optimal off-line algorithm. An optimal off-line algorithm knows the entire request sequence in advance and can serve it with minimum cost. Given a request sequence σ, let $C_A(\sigma)$ denote the cost incurred by the online algorithm A in serving σ and let $C_{OPT}(\sigma)$ denote the cost incurred by the optimal off-line algorithm OPT on σ. Then the algorithm A is c-competitive if there exists a constant a such that

$$C_A(\sigma) \leq c \cdot C_{OPT}(\sigma) + a$$

for all request sequences σ. The *competitive factor* of A is the infimum of all c for which A is c-competitive.

The list update problem is of significant practical interest. List update techniques are often applied in practice when storing small dictionaries. Furthermore, they are efficient subroutines in algorithms related to data compression and computational geometry [7, 9, 12, 15]. Due to its structural simplicity and practical significance, the list update problem has been studied extensively [2, 3, 8, 10, 13, 18, 20, 25, 26, 27, 28]. In the following we mention the important results relevant to our work. Sleator and Tarjan [27] have shown that the MOVE-TO-FRONT algorithm, which simply moves an item to the front of the list each time it is requested, is 2-competitive. Karp and Raghavan [22] have observed that no deterministic on-line algorithm for the list update problem can be better than 2-competitive. Thus, the MOVE-TO-FRONT algorithm achieves the best possible competitive factor.

Recently there have been some attempts to beat the competitive factor of 2 using randomization. Irani [20] has described a randomized on-line algorithm for the list update problem that achieves a competitiveness of 1.935. Reingold *et al.* [25] have given a randomized algorithm that is $\sqrt{3}$-competitive. Albers [2] has presented a randomized algorithm whose competitiveness is equal to the Golden Ratio $\Phi = \frac{1+\sqrt{5}}{2}$. This performance ratio was further improved to 1.6 [3]. The best lower bound known for randomized on-line algorithms is due to Teia [28]. He shows that no randomized on-line algorithm for the list update problem can have a competitive factor which is less than 1.5. These bounds hold against the oblivious adversary, see [6] for details.

In this paper we study the problem of *lookahead* in on-line algorithms for the list update problem. An important question is what improvement can be achieved in terms of competitiveness, if an on-line algorithm knows not only the present request to be served, but also some future requests. This issue is interesting from the practical as well as the theoretical point of view. In practical applications requests do not necessarily arrive one after the other, but rather in blocks of possibly variable size. In addition, requests may be generated faster than they can be processed by a list update algorithm. Hence it is to be expected that some requests usually wait in line to be processed by an on-line algorithm. In some applications it may also be possible to delay the service of requests so as to wait for some incoming requests. In the theoretical context a natural question is: What is it worth to know a part of the future?

So far, only few on-line problems with lookahead have been studied in the literature. Previous research on lookahead in on-line algorithms has addressed paging problems [1, 5, 11, 23, 29, 30],

k-server problems [5], bin packing problems [16], dynamic location problems [14] and graph problems [17, 19, 21]. In particular, at present time, nothing is known about list update with lookahead. We begin our study of the influence of lookahead in the list update problem by introducing two different models of lookahead.

Let $\sigma = \sigma(1), \sigma(2), \ldots, \sigma(m)$ be a request sequence of length m. $\sigma(t)$ denotes the request at time t. For a given set S, $card(S)$ denotes the cardinality of S. Let $l \geq 1$ be an integer.

Weak lookahead of size l: The on-line algorithm sees the present request and the next l future requests. More specifically, when answering $\sigma(t)$, the on-line algorithm already knows $\sigma(t+1), \sigma(t+2), \ldots, \sigma(t+l)$. However, requests $\sigma(s)$, with $s \geq t+l+1$, are not seen by the on-line algorithm at time t.

Strong lookahead of size l: The on-line algorithm sees the present request and a sequence of future requests. This sequence contains l pairwise distinct items which also differ from the item requested by the present request. More precisely, when serving request $\sigma(t)$, the algorithm knows requests $\sigma(t+1), \sigma(t+2), \ldots, \sigma(t')$, where $t' = \min\{s > t | card(\{\sigma(t), \sigma(t+1), \ldots, \sigma(s)\}) = l+1\}$. The requests $\sigma(s)$, with $s \geq t'+1$, are not seen by the on-line algorithm at time t.

At first sight weak lookahead seems to be the natural model of lookahead. However, as we shall see later, weak lookahead is only of minor advantage in the list update problem. The reason is that an adversary that constructs a request sequence can replicate requests in the lookahead, thereby weakening the effect of the lookahead. In contrast, in the model of strong lookahead, we require an adversary to reveal some really significant information of future requests. Strong lookahead was first presented in [1], where on-line paging algorithms with lookahead are studied. Strong lookahead is a model of lookahead that can improve the competitive factors of on-line paging algorithms and has practical as well as theoretical importance. In the following, when we investigate on-line algorithms with lookahead, l always denotes the size of the lookahead. We always assume that an on-line algorithm has a lookahead of fixed size $l \geq 1$.

This paper represents an in-depth study of the deterministic list update problem with strong and weak lookahead. Section 2 is concerned with lower bounds. We show that an on-line algorithm requires a strong lookahead of size $\Omega(n)$ in order to be better than 2-competitive. Specifically, we prove that an on-line algorithm with strong lookahead l, where $l \leq n-1$, cannot be better than $(2 - \frac{l+2}{n+1})$-competitive. Again, n is the number of items in the list. If an on-line algorithm is given a weak lookahead, the situation is worse. We show that a lookahead of size $\Omega(n^2)$ is necessary to asymptotically beat the competitive factor of 2. This statement seems to imply that it would not be worthwhile to consider weak lookahead in the list update problem. However this might not be true, as more precise calculations show. We prove that if a weak lookahead of size l is given and $(l+1) = Kn^2$ for a positive real constant K, then an on-line list update algorithm cannot be better than c-competitive where $c = 2 - 2\sqrt{4K^2+2K} + 4K$. (Note that this expression goes to 1 as K tends to infinity.) Even for very small values of K, this bound gives values which are significantly below 2. For instance, if $K = \frac{1}{100}$, we obtain a lower bound

of $c = 1.75$. Recall that the list update problem is of practical interest for small lists consisting of only a few dozen items. For lists of lengths $n_1 = 12$ and $n_2 = 24$ the term $l = \frac{1}{100}n^2 - 1$ evaluates to a lookahead of size $l_1 = 1$ and $l_2 = 5$, respectively. If our lower bounds are relatively tight, a 1.75-competitive algorithm working on small lists would require a weak lookahead of reasonable size. For a more extensive discussion of the bound $c = 2 - 2\sqrt{4K^2 + 2K} + 4K$, we refer the reader to the table on page 9.

Section 3 addresses the development of on-line algorithms for the list update problem with lookahead. We present on-line algorithms that are competitive against *static off-line* algorithms. Static algorithms initially arrange the list in some order and make no other exchanges of items in the list while processing a request sequence. Given a request sequence σ, the *optimum static off-line* algorithm, which we call STAT, first sorts the items in non-increasing order of request frequencies and then does no further exchanges. Formally, an on-line algorithm A is c-competitive against static off-line algorithms if there exists a constant a such that $C_A(\sigma) \leq c \cdot C_{STAT}(\sigma) + a$ for all request sequences σ. Static off-line algorithms are weaker than dynamic off-line algorithms, which may rearrange the list after each request. However, static algorithms are valuable from the practical point of view since they can compute an optimal ordering of the list in $O(m)$ time, where m is the length of the request sequence. The best dynamic off-line algorithm currently known is due to Reingold and Westbrook [24] and takes $O(2^n n! m)$ time. There has also been work focused on analyzing list update algorithms against static off-line algorithms, e.g., Bentley and McGeoch [8] have shown that the MOVE-TO-FRONT algorithm is 2-competitive against static off-line algorithms. D'Amore *et al.* [4] have discussed a variant of the list update problem, called the weighted list update problem, with respect to static off-line algorithms. We develop a simple on-line algorithm for the list update problem that, given a strong lookahead of size $l \leq n-1$, is $(2 - \frac{2}{3} \cdot \frac{l+2}{2n-l})$-competitive. We also give an on-line algorithm with weak lookahead that has a competitiveness of $2 - \frac{2}{3}(\sqrt{K^2 + 2K} - K)$. We compare this performance to the corresponding lower bound we developed. We remark that our lower bounds hold against any off-line algorithm (static or dynamic), whereas our upper bounds hold against static off-line algorithms.

2 Lower bounds for list update with lookahead

We assume that the given list consists of n items, where $n \geq 2$. Furthermore, we generally assume that the size l of the given lookahead is constant or a function of n. We show that a deterministic on-line algorithm with strong lookahead l can only be better than 2-competitive (for all list lengths) if l is linear in n. Note that the size of a strong lookahead satisfies $l \leq n-1$.

Theorem 1 *Let A be a deterministic on-line algorithm with strong lookahead l for the list update problem. Then there exists a request sequence σ such that*

$$C_A(\sigma) \geq (2 - \frac{l+2}{n+1}) \cdot C_{OPT}(\sigma).$$

Proof: We construct a request sequence $\sigma = \sigma(1), \sigma(2), \ldots$ using the following algorithm.

Algorithm LIST-REQUEST: The first $l+1$ requests $\sigma(1), \sigma(2), \ldots, \sigma(l+1)$ are requests to the last $l+1$ items in the initial list. For $t \geq l+2$ the request $\sigma(t)$ is constructed as follows. After A has served $\sigma(t-l-1)$, determine the item x which has the highest position in the current list among items not contained in $\{\sigma(t-l), \sigma(t-l+1), \ldots, \sigma(t-1)\}$. Set $\sigma(t) = x$.

Given this request sequence σ, we compare the cost incurred by A to the cost incurred by the optimal algorithm OPT. It is not hard to see that OPT can process each request sequence such that its amortized cost on each request is at most $(n+1)/2$. OPT can simply use the optimum static algorithm STAT (which initially sorts the items in non-increasing order of request frequencies and makes no other exchanges). Hence OPT's amortized cost during $l+1$ successive requests in σ is at most $(l+1)(n+1)/2$.

We evaluate A's cost on request sequence σ. For simplicity, we handle paid exchanges made by A in the following way. Whenever A moves an item x closer to the front of the list using paid exchanges, we charge the cost of these paid exchanges to the next request to x. This charging scheme will be used in the remainder of this proof, including Lemma 1 and its proof. Lemma 1 shows that on any $l+1$ successive requests, A incurs a cost of at least $\sum_{i=0}^{l}(n-i)$. This implies that $C_A(\sigma) \geq c \cdot C_{OPT}(\sigma)$, where

$$c = \frac{\sum_{i=0}^{l}(n-i)}{(l+1)(n+1)/2} = \frac{(l+1)n - (l+1)l/2}{(l+1)(n+1)/2} = \frac{2n-l}{n+1} = 2 - \frac{l+2}{n+1}. \quad \square$$

Lemma 1 *Let $C_A(\tilde{\sigma}(t))$ be the cost incurred by A when processing the subsequence $\tilde{\sigma}(t) = \sigma(t), \sigma(t+1), \ldots, \sigma(t+l)$. Then $C_A(\tilde{\sigma}(t)) \geq \sum_{i=0}^{l}(n-i)$ for all $t \geq 1$.*

Proof: For $t \geq 1$, let $S(t) = \{t, t+1, \ldots, t+l\}$ and let $C_A(\sigma(t))$ be the cost incurred by A when processing request $\sigma(t)$. We prove by induction on t that for all $t \geq 1$ and for all k, where $n-l \leq k \leq n$, the inequality

$$card(\{s \in S(t) | C_A(\sigma(s)) \geq k\}) \geq n-k+1 \tag{1}$$

holds. This implies the lemma.

For an item x and $t \geq 1$, let $pos(x,t)$ denote x's position in the list immediately after A has served $\sigma(t-1)$. By the construction of σ, any $l+1$ successive requests in σ are pairwise distinct. Thus for any $s \in S(t)$, $C_A(\sigma(s)) \geq pos(\sigma(s), t)$ because paid exchanges applied to the item $\sigma(s)$ during the time interval $[t, s-1]$ are charged to request $\sigma(s)$.

We proceed with the inductive proof. Inequality (1) holds at time $t = 1$. By induction hypothesis it holds at time $t-1$. We show that the inequality is also satisfied at time t. When

making the transition from $S(t-1)$ to $S(t)$, we lose time $t-1$. Thus, the induction hypothesis implies that for all k, $n-l \leq k \leq n$,

$$card(\{s \in S(t) \setminus \{(t+l)\} | C_A(\sigma(s)) \geq k\}) \geq n-k. \tag{2}$$

If $pos(\sigma(t+l),t) = n$, then the inequality (1) obviously holds for all k, $n-l \leq k \leq n$. So suppose $pos(\sigma(t+l),t) < n$. After A has served $\sigma(t-1)$, the items $\sigma(s)$ with $s \in S(t) \setminus \{(t+l)\}$ occupy all positions $pos(\sigma(t+l),t)+1, pos(\sigma(t+l),t)+2, \ldots, n$. We observe that for $k > pos(\sigma(t+l),t)$

$$card(\{s \in S(t) \setminus \{(t+l)\} | C_A(\sigma(s)) \geq k\}) \geq n-k+1.$$

Since inequality (2) holds, inequality (1) must be satisfied for all k, $n-l \leq k \leq n$. □

Next we consider algorithms with weak lookahead.

Theorem 2 *Let A be a deterministic on-line algorithm with weak lookahead l for the list update problem.*

a) If $l = o(n^2)$ and A is c-competitive, then $c \geq 2$.

b) If $l+1 = Kn^2$ and A is c-competitive, then $c \geq 2 - 2\sqrt{4K^2+2K} + 4K$.

Proof: For integers j with $1 \leq j \leq \min\{l+1, n-1\}$, we construct request sequences σ_n^j and then use $\limsup_{n\to\infty} C_{\max}(n)$, where $C_{\max}(n) = \max\{C_A(\sigma_n^j)/C_{OPT}(\sigma_n^j) \mid 1 \leq j \leq \min\{l+1, n-1\}\}$, to bound A's competitive factor from below.

If $j = l+1$ then we generate a request sequence using the algorithm LIST-REQUEST proposed in the proof of Theorem 1. If $j < l+1$ we use a slightly different algorithm. Let x_1 be the first item in the initial list. The request sequence σ_n^j consists of a series of phases each of which contains exactly $l+1$ requests. In each phase, the first j requests are made to items $x \neq x_1$, while the remaining $l+1-j$ requests are made to item x_1. More precisely, the first j requests in the first phase are requests to the last j items in the initial list. If $\sigma(t)$ belongs to the first j requests in a given phase i, where $i \geq 2$, then $\sigma(t)$ is constructed as follows. After A has served $\sigma(t-l-1)$, determine the item x which is at the highest position in the current list among items not contained in $\{\sigma(t-l), \sigma(t-l+1), \ldots, \sigma(t-1)\}$. Set $\sigma(t) = x$.

We analyze A's and OPT's cost incurred on a given sequence σ_n^j. Again, whenever A moves an item x closer to the front of the list using paid exchanges, we charge the cost of these paid exchanges to the next request to x. We claim that in each phase, A incurs a cost of at least

$$jn - \frac{j(j-1)}{2} + (l+1-j).$$

The claim clearly holds if $j = l+1$ or if $j < l+1$ and A always stores x_1 at the first position of the list. In these two cases we can use the same analysis as in the proof of Lemma 1. If $j < l+1$ and if A does not always store x_1 at the front of the list, then consider the following algorithm

A'. The algorithm A' always maintains the items $x \neq x_1$ in the same order as A, but always stores x_1 at the first position of the list. It is easy to verify that in each phase, A' does not incur a higher cost than A.

We show that in each phase, OPT's amortized cost is at most

$$j(\frac{n(n+1)}{2(n-1)} - \frac{1}{n-1}) + (l+1-j).$$

This bound holds true if $j = l+1$ because $\frac{n(n+1)}{2(n-1)} - \frac{1}{n-1} \geq \frac{n+1}{2}$. If $j < l+1$ then OPT can apply the following static algorithm. Initially, the list is rearranged such that item x_1 occupies position 1 in the list and such that the remaining items are sorted in order of non-increasing request frequencies. While processing σ_n^j no exchanges are made. Using this static algorithm, OPT's amortized cost on a request to an item $x \neq x_1$ is at most $\frac{1}{n-1}(\sum_{k=2}^{n} k) = \frac{n(n+1)}{2(n-1)} - \frac{1}{n-1}$. This bound is tight if all items $x \neq x_1$ have the same request frequency.

For $j = 1, 2, \ldots, \min\{l+1, n-1\}$, let $C_n(j) = C_A(\sigma_n^j)/C_{OPT}(\sigma_n^j)$. We have

$$C_n(j) \geq \frac{jn - \frac{j(j-1)}{2} + (l+1-j)}{j(\frac{n(n+1)}{2(n-1)} - \frac{1}{n-1}) + (l+1-j)} = \frac{2jn - j^2 - j + 2l + 2}{jn + 2l + 2}. \tag{3}$$

Then A's competitive factor c satisfies $c \geq \limsup_{n\to\infty} C_{\max}(n)$, where $C_{\max}(n) = \max\{C_n(j) | 1 \leq j \leq \min\{l+1, n-1\}\}$. Now we prove the two parts of the theorem.

Part a): If $l = O(1)$, then consider the sequence $C_n(1)$, $n = 1, 2, 3, \ldots$. This sequence converges to 2 as n tends to infinity. Now suppose $l = \omega(1)$ and $l = O(n^2)$. We maximize the function

$$C_n(j) = \frac{2jn - j^2 - j + 2l + 2}{jn + 2l + 2} \tag{4}$$

subject to the constraint $0 < j \leq \min\{l+1, n-1\}$. Here we are also interested in possibly non-integral solutions for j. We determine j_n such that $\frac{dC_n(j_n)}{dj_n} = 0$.

$\frac{dC_n(j_n)}{dj_n} = 0$ is equivalent to

$$\begin{aligned}
& (2n - 2j_n - 1)(j_n n + 2l + 2) - (2j_n n - j_n^2 - j_n + 2l + 2)n &= 0 \\
\Leftrightarrow \quad & 2j_n n^2 + 4nl + 4n - 2j_n^2 n - 4j_n l - 4j_n - j_n n - 2l - 2 & \\
& -(2j_n n^2 - j_n^2 n - j_n n + 2nl + 2n) &= 0 \\
\Leftrightarrow \quad & j_n^2 n + 4j_n l + 4j_n - 2nl - 2n + 2l + 2 &= 0
\end{aligned}$$

This implies

$$(j_n + 2\frac{(l+1)}{n})^2 = 4\frac{(l+1)^2}{n^2} + 2(l+1) - 2\frac{(l+1)}{n}.$$

Since we require $j_n > 0$, only

$$j_n = \frac{1}{n}(\sqrt{4(l+1)^2 + 2(l+1)n(n-1)} - 2(l+1))$$

can be a solution to our maximization problem.

Defining $D = 4(l+1)^2 + 2(l+1)n(n-1)$, we have

$$\begin{aligned} C_n(j_n) &= \frac{1}{\sqrt{D}}(2\sqrt{D} - 4(l+1) - \frac{1}{n^2}(D - 4\sqrt{D}(l+1) + 4(l+1)^2) \\ &\quad -\frac{1}{n}(\sqrt{D} - 2(l+1)) + 2(l+1)) \\ &= \frac{1}{\sqrt{D}}(2\sqrt{D} - \frac{2}{n^2}D + \frac{1}{n^2}\sqrt{D}(4(l+1) - n)) \\ &= 2 - \frac{2}{n^2}\sqrt{D} + \frac{1}{n^2}(4(l+1) - n). \end{aligned}$$

Hence

$$C_n(j_n) = 2 - \frac{2}{n^2}\sqrt{4(l+1)^2 + 2(l+1)n(n-1)} + \frac{1}{n^2}(4(l+1) - n). \quad (5)$$

It is easy to verify that $C_n(j_n)$ is in fact a maximum of the function $C_n(j)$ and that $0 < j_n \leq \min\{l+1, n-1\}$.

Note that j_n might not be an integer. However, since $l = \omega(1)$, the sequence j_n, $n = 1, 2, 3, \ldots$, is $\omega(1)$. Thus, using equation (4), one can easily prove that the sequences $C_n(j_n)$ and $C_n(\lfloor j_n \rfloor)$ have the same lim sup as n tends to infinity. Taking the lim sup of the sequence $C_n(j_n)$, we obtain that A's competitive factor cannot be asymptotically better than 2, if $l = o(n^2)$. This proves part a) of the theorem.

Part b): If $(l+1) = Kn^2$, then by equation (5)

$$C_n(j_n) = 2 - \frac{2}{n^2}\sqrt{4K^2n^4 + 2Kn^4 - 2Kn^3} + 4K - \frac{1}{n} \geq 2 - 2\sqrt{4K^2 + 2K} + 4K - \frac{1}{n}$$

and this expression converges to $2 - 2\sqrt{4K^2 + 2K} + 4K$ as n tends to infinity. □

3 On-line algorithms with lookahead

In this section we present deterministic on-line algorithms with lookahead. These algorithms are competitive against static off-line algorithms. In the following we consider strong and weak lookahead in parallel because the algorithms and analyses are very similar for both kinds of lookahead. We assume that we are given a request sequence σ of length m. If a strong lookahead of size l is given, then for all $t \geq 1$ we define a value $\lambda(t)$. If $card(\{\sigma(t), \sigma(t+1), \ldots, \sigma(m)\}) < l+1$ then let $\lambda(t) = m$; otherwise let $\lambda(t) = \min\{t' > t | card(\{\sigma(t), \sigma(t+1), \ldots, \sigma(t')\}) = l+1\}$. The value $\lambda(t)$ is the time of the request farthest in the future that can be seen at time t. Note that if a strong lookahead l is provided, then $l \leq n-1$.

Algorithm FREQUENCY-COUNT(l): Serve the request sequence in a series of blocks $B(i)$. Each block is a subsequence of consecutive requests that will be served together. If a strong lookahead l is given, then $B(1) = \sigma(1), \sigma(2), \ldots, \sigma(\lambda(1))$ and $B(i) = \sigma(t^e_{i-1}+1), \sigma(t^e_{i-1}+2), \ldots, \sigma(\lambda(t^e_{i-1}+1))$ for $i \geq 2$. Here t^e_{i-1} denotes the end of block $B(i-1)$. If a weak lookahead l is provided, then $B(i) = \sigma((i-1)(l+1)+1), \sigma((i-1)(l+1)+2), \ldots, \sigma(\min\{i(l+1), m\})$ for $i \geq 1$. Each block is processed as follows. At the beginning of each block, sort the items

$l+1$	Competitive Factors		Value of $(l+1)$ for			
	Lower Bound	Upper Bound	$n=15$	$n=20$	$n=25$	$n=30$
$\frac{1}{500}n^2$	1.88	1.96	0.45	0.8	1.25	1.8
$\frac{1}{200}n^2$	1.82	1.94	1.125	2	3.125	4.5
$\frac{1}{100}n^2$	1.75	1.91	2.25	4	6.25	9
$\frac{1}{50}n^2$	1.67	1.88	4.5	8	12.5	18
$\frac{1}{20}n^2$	1.54	1.82	11.25	20	31.25	45
$\frac{1}{10}n^2$	1.42	1.76	22.5	40	62.5	90

Table 1. Competitive factors for list update with weak lookahead

in the list such that they are in non-increasing order of request frequencies with respect to the current block. Execute this step using as few exchanges as possible. (This restriction ensures that items with the same request frequency are not exchanged.) After this rearrangement, serve the requests in the current block without making any further exchanges.
Note that the sorting of the items can be implemented as follows. First determine the items with the highest request frequency in the current block, and move these items in an order preserving way to the front of the list. Then determine the items with the next lower request frequency and move these items (in an order preserving way) as close to the front of the list as possible, but without passing the items with the highest request frequency. Repeat this process for the other request frequencies. The sorting step is accomplished using paid exchanges that are counted in FREQUENCY-COUNT(l)'s cost.

We evaluate the performance of FREQUENCY-COUNT(l) for a fixed n.

Theorem 3 *Let $l \leq n-1$. The algorithm FREQUENCY-COUNT(l) with strong lookahead l is c-competitive against static off-line algorithms, where*

$$c \leq 2 - \frac{2}{3} \cdot \frac{l+2}{2n-l}.$$

Theorem 4 *Let $K > 0$ be a real constant. If a weak lookahead l is given with $(l+1) = Kn^2$, then FREQUENCY-COUNT(l) is c-competitive against static off-line algorithms, where*

$$c \leq 2 - \frac{2}{3}(\sqrt{K^2+2K} - K).$$

The terms subtracted from 2 in the bounds given in Theorems 3 and 4 are positive for all $l \leq n-1$ and $K > 0$, respectively. Notice that FREQUENCY-COUNT(l) can be $(4/3)$-competitive if a large lookahead is given. Table 1 compares, for various values of a weak lookahead l and various n, the performance of FREQUENCY-COUNT(l) to the lower bounds derived in Section 2. Note that the lower bounds hold asymptotically.

In order to prove the two theorems, we start with a general analysis of the algorithm FREQUENCY-COUNT(l) (also called FC) that applies to strong and weak lookahead. We

use a potential function Φ to analyze the performance of our on-line algorithm. Φ is the number of *inversions* in FC's list with respect to STAT's list. Given two lists containing the same items, an inversion is an unordered pair of items $\{x, y\}$ such that x occurs before y in one list while x occurs after y in the other list. We assume that FC and STAT start with the same list, so that the initial potential is zero.

Consider a request sequence σ. Initially, STAT rearranges the items in the list using paid exchanges. Each paid exchange incurs a cost of 1 and can increase the potential by 1. In the following we bound FC's amortized cost in each block of σ. We consider an arbitrary block B. Let $C_{FC}(B)$ be the actual cost FC incurs in processing B and let $\Delta\Phi$ be the change in the potential function between the beginning and the end of the given block. The sum $C_{FC}(B)+\Delta\Phi$ is FC's amortized cost in block B. Furthermore, let S be the set of items in the list, and let S_B be the set of items requested in block B. For an item $x \in S_B$ and $A \in \{\text{FC}, \text{STAT}\}$, let $C_A(x)$ be the cost that algorithm A incurs when serving a request to item x in block B. $f_B(x)$ denotes the request frequency of item x in block B, i.e., $f_B(x)$ is the number of times item x is requested in B. Finally, let $j = card(S_B)$ be the number of different items requested in B. Note that $j = l + 1$ if we deal with strong lookahead.

Lemma 2

$$C_{FC}(B) + \Delta\Phi \leq 2 \sum_{x \in S_B} C_{STAT}(x) + \frac{4}{3} \sum_{x \in S_B} (f_B(x) - 1) C_{STAT}(x) - \frac{1}{3} j(j+1)$$

Proof: For a subset $M \subseteq S$ we introduce the following definitions.

1. For $A \in \{\text{FC}, \text{STAT}\}$ and $x \in S_B$ let

   $$C_A(x, M) = card(\{y \in M \mid y = x \text{ or item } y \text{ precedes item } x \text{ in } A\text{'s list when } A \text{ serves a request to } x \text{ in block } B\}).$$

   $C_A(x, M)$ is the cost caused by M when A serves a request to item x.

2. Let $\Delta\Phi^+(M)$ be the number of inversions $\{x, y\}$ created between items $x \in S_B$ and $y \in M$ when B is served, and let $\Delta\Phi^-(M)$ be the number of inversions $\{x, y\}$ removed between items $x \in S_B$ and $y \in M$. Set $\Delta\Phi(M) = \Delta\Phi^+(M) - \Delta\Phi^-(M)$.
3. Let $p(M)$ be the number of paid exchanges FC incurs when swapping an item $x \in S_B$ with an item $y \in M$ at the beginning of the block.

Notice that for any $x \in S_B$ and $A \in \{\text{FC}, \text{STAT}\}$, $C_A(x) = C_A(x, S_B) + C_A(x, S \setminus S_B)$ and $\Delta\Phi = \Delta\Phi(S_B) + \Delta\Phi(S \setminus S_B)$. We have $C_{FC}(x, S \setminus S_B) = 0$ for all $x \in S_B$. Thus FC's amortized cost in block B satisfies

$$C_{FC}(B) + \Delta\Phi = \sum_{x \in S_B} f_B(x) C_{FC}(x, S_B) + p(S_B) + p(S \setminus S_B) + \Delta\Phi(S_B) + \Delta\Phi(S \setminus S_B).$$

Claim 1 $p(S \setminus S_B) + \Delta\Phi(S \setminus S_B) \leq 2 \sum_{x \in S_B} C_{STAT}(x, S \setminus S_B)$

Proof of Claim 1: We have

$$\sum_{x \in S_B} C_{STAT}(x, S \setminus S_B) = \sum_{x \in S_B} \sum_{y \in S \setminus S_B} C_{STAT}(x, \{y\}).$$

Suppose FC moves an item $x \in S_B$ closer to the front of the list using paid exchanges and swaps x with an item $y \in S \setminus S_B$. If an inversion is removed, then the potential decreases by 1. If an inversion is created, then the pair $\{x, y\}$ incurs a cost of 2 on the left hand side of the inequality in the claim. But $C_{STAT}(x, \{y\}) = 1$. This proves the claim. □

Claim 2 $\sum_{x \in S_B} f_B(x) C_{FC}(x, S_B) + p(S_B) - \Delta\Phi^-(S_B) \leq \sum_{x \in S_B} f_B(x) C_{STAT}(x, S_B)$

Proof of Claim 2: For any $x \in S_B$ and $A \in \{\text{FC}, \text{STAT}\}$ we have $C_A(x, \{x\}) = 1$. This implies that the inequality in the claim is equivalent to

$$\sum_{x \in S_B} \sum_{\substack{y \in S_B \\ y \neq x}} f_B(x) C_{FC}(x, \{y\}) + p(S_B) - \Delta\Phi^-(S_B) \leq \sum_{x \in S_B} \sum_{\substack{y \in S_B \\ y \neq x}} f_B(x) C_{STAT}(x, \{y\}). \quad (6)$$

Consider any pair $\{x, y\}$ with $x, y \in S_B$ and $x \neq y$. Suppose y is before x in FC's list after the rearrangement of the items in S_B. Note that FC orders the items x and y optimally.

Case 1: If FC does not swap x and y at the beginning of the block, then

$$f_B(x) C_{FC}(x, \{y\}) + f_B(y) C_{FC}(y, \{x\}) \leq f_B(x) C_{STAT}(x, \{y\}) + f_B(y) C_{STAT}(y, \{x\}).$$

Case 2: If FC swaps x and y and the potential decreases, then

$$f_B(x) C_{FC}(x, \{y\}) + f_B(y) C_{FC}(y, \{x\}) + 1 - 1 \leq f_B(x) C_{STAT}(x, \{y\}) + f_B(y) C_{STAT}(y, \{x\}).$$

Case 3: If FC swaps x and y and the potential increases, then

$$f_B(x) C_{FC}(x, \{y\}) + f_B(y) C_{FC}(y, \{x\}) + 1 \leq f_B(x) C_{STAT}(x, \{y\}) + f_B(y) C_{STAT}(y, \{x\}),$$

because $f_B(y) > f_B(x)$. Adding the appropriate inequalities for all such pairs, we obtain inequality (6). □

Claim 3 $\Delta\Phi^+(S_B) \leq \frac{1}{3} \sum_{x \in S_B} f_B(x) C_{STAT}(x, S_B)$

Proof of Claim 3: Suppose FC moves an item x closer to the front of the list and creates an inversion with an item $y \in S_B$. Notice that x must be requested at least twice in block B and that $C_{STAT}(x, \{y\}) = 1$. If x is requested three times, then we may charge a cost of 1/3 to each of these $f_B(x)$ requests.

We estimate the number J of inversions created between items requested twice and items requested once in B. Let S_B^1 be the set of items requested exactly once in B and let S_B^2 be the set of items requested exactly twice in B. Define $j_1 = card(S_B^1)$ and $j_2 = card(S_B^2)$. We prove

$$J \leq \frac{1}{3} \sum_{x \in S_B^1 \cup S_B^2} C_{STAT}(x, S_B^1 \cup S_B^2) + \frac{1}{3} \sum_{x \in S_B^2} C_{STAT}(x, S_B^1 \cup S_B^2). \tag{7}$$

This implies the claim. We have $\sum_{x \in S_B^1 \cup S_B^2} C_{STAT}(x, S_B^1 \cup S_B^2) = \frac{1}{2}(j_1 + j_2)(j_1 + j_2 + 1)$. First suppose that each of the j_2 items in S_B^2 causes j_1 new inversions. Then $J = j_1 j_2$ and $\sum_{x \in S_B^2} C_{STAT}(x, S_B^1 \cup S_B^2) = \frac{1}{2}((j_1 + j_2)(j_1 + j_2 + 1) - j_1(j_1 + 1))$. Now suppose that an item $x \in S_B^2$ causes only $j_1 - k_x$ inversions. Then, $J = j_1 j_2 - \sum_{x \in S_B^2} k_x$ and

$$\frac{1}{2}((j_1 + j_2)(j_1 + j_2 + 1) - j_1(j_1 + 1)) - \sum_{x \in S_B^2} k_x \leq \sum_{x \in S_B^2} C_{STAT}(x, S_B^1 \cup S_B^2).$$

Simple algebraic manipulations show that

$$j_1 j_2 \leq \frac{1}{3}(\frac{1}{2}(j_1 + j_2)(j_1 + j_2 + 1) + \frac{1}{2}((j_1 + j_2)(j_1 + j_2 + 1) - j_1(j_1 + 1))).$$

Using the last two inequalities, we can easily derive inequality (7). □
Summing up the inequalities in Claim 1, Claim 2 and Claim 3 we obtain, as desired,

$$\begin{aligned} C_{FC}(B) + \Delta\Phi &\leq 2 \sum_{x \in S_B} C_{STAT}(x, S \setminus S_B) + \frac{4}{3} \sum_{x \in S_B} f_B(x) C_{STAT}(x, S_B) \\ &\leq 2 \sum_{x \in S_B} C_{STAT}(x) + \frac{4}{3} \sum_{x \in S_B} (f_B(x) - 1) C_{STAT}(x) - \frac{2}{3} \cdot \frac{j(j+1)}{2}. \ \square \end{aligned}$$

Proof of Theorem 3: Suppose the request sequence consists of b blocks $B(1), B(2), \ldots, B(b)$. By Lemma 2, $C_{FC}(\sigma)/C_{STAT}(\sigma)$ is bounded from above by

$$\frac{\sum_{i=1}^{b} (2 \sum_{x \in S_{B(i)}} C_{STAT}(x) + \frac{4}{3} \sum_{x \in S_{B(i)}} (f_{B(i)}(x) - 1) C_{STAT}(x) - \frac{(l+1)(l+2)}{3})}{C_{STAT}(\sigma)}.$$

Here we may assume without loss of generality that the last block $B(b)$ contains $l + 1$ distinct requests. Hence,

$$\frac{C_{FC}(\sigma)}{C_{STAT}(\sigma)} \leq 2 - \frac{2}{3} \cdot \frac{\frac{b(l+1)(l+2)}{2} + \sum_{i=1}^{b} \sum_{x \in S_{B(i)}} (f_{B(i)}(x) - 1) C_{STAT}(x)}{\sum_{i=1}^{b} \sum_{x \in S_{B(i)}} f_{B(i)}(x) C_{STAT}(x)}.$$

We have $\sum_{i=1}^{b} \sum_{x \in S_{B(i)}} (f_{B(i)}(x) - 1) C_{STAT}(x) \geq 0$ and $\sum_{i=1}^{b} \sum_{x \in S_{B(i)}} C_{STAT}(x) \geq b \frac{(l+1)(l+2)}{2}$. Thus

$$\frac{C_{FC}(\sigma)}{C_{STAT}(\sigma)} \leq 2 - \frac{2}{3} \cdot \frac{\frac{b(l+1)(l+2)}{2}}{\sum_{i=1}^{b} \sum_{x \in S_{B(i)}} C_{STAT}(x)} \leq 2 - \frac{2}{3} \cdot \frac{(l+2)/2}{n - l/2} = 2 - \frac{2}{3} \cdot \frac{l+2}{2n - l},$$

where the second inequality follows from $\sum_{i=1}^{b} \sum_{x \in S_{B(i)}} C_{STAT}(x) \leq b \sum_{k=0}^{l} (n - k) = b((l + 1)n - l(l+1)/2)$. The above line implies the theorem. □

Proof of Theorem 4: Again, we assume that the request sequence σ consists of b blocks $B(1), B(2), \ldots, B(b)$. Let j_i be the number of different items requested in block $B(i)$. By Lemma 2, $C_{FC}(\sigma)/C_{STAT}(\sigma)$ is bounded from above by

$$\frac{\sum_{i=1}^{b}(2\sum_{x\in S_{B(i)}} C_{STAT}(x) + \frac{4}{3}\sum_{x\in S_{B(i)}}(f_{B(i)}(x)-1)C_{STAT}(x) - \frac{j_i(j_i+1)}{3})}{\sum_{i=1}^{b}(\sum_{x\in S_{B(i)}} C_{STAT}(x) + \sum_{x\in S_{B(i)}}(f_{B(i)}(x)-1)C_{STAT}(x))}.$$

Note that $\sum_{x\in S_{B(i)}} C_{STAT}(x) \leq j_i n$ and that $\sum_{i=1}^{b} j_i(j_i+1) \geq bj(j+1)$, where $j = \frac{1}{b}\sum_{i=1}^{b} j_i$. Hence,

$$\frac{C_{FC}(\sigma)}{C_{STAT}(\sigma)} \leq \frac{\sum_{i=1}^{b}(2jn - \frac{j(j+1)}{3} + \frac{4}{3}\sum_{x\in S_{B(i)}}(f_{B(i)}(x)-1)C_{STAT}(x))}{\sum_{i=1}^{b}(jn + \sum_{x\in S_{B(i)}}(f_{B(i)}(x)-1)C_{STAT}(x))}.$$

Since $\frac{2jn-\frac{1}{3}j(j+1)}{jn} \geq \frac{4}{3}$, we obtain

$$\frac{C_{FC}(\sigma)}{C_{STAT}(\sigma)} \leq \frac{\sum_{i=1}^{b}(2jn - \frac{1}{3}j(j+1) + \frac{4}{3}(l+1-j))}{\sum_{i=1}^{b}(jn + (l+1-j))} = \frac{2jn - \frac{1}{3}j^2 - \frac{5}{3}j + \frac{4}{3}(l+1)}{jn - j + (l+1)}.$$

We have $(l+1) = Kn^2$. We maximize the function

$$C_n(j) = \frac{2jn - \frac{1}{3}j^2 - \frac{5}{3}j + \frac{4}{3}Kn^2}{jn - j + Kn^2}$$

subject to the constraint $0 < j \leq \min\{Kn^2, n\}$. Using the same techniques as in the proof of Theorem 2 we can show that $j_n = \frac{1}{n-1}(\sqrt{K^2n^4 + Kn^2(2n-1)(n-1)} - Kn^2)$ is the solution to this maximization problem and that

$$C_n(j_n) = 2 - \frac{1}{3}(\frac{2}{(n-1)^2}\sqrt{K^2n^4 + Kn^2(2n-1)(n-1)} - \frac{2Kn^2}{(n-1)^2} - \frac{1}{n-1}).$$

The above expression goes to $c = 2 - \frac{2}{3}(\sqrt{K^2+2K} - K)$ as n tends to infinity.

We remark that it is possible to derive more precise but also more complicated bounds on the competitive factor if one takes into account that $\sum_{x\in S_{B(i)}} C_{STAT}(x) \leq j_i n - \frac{j_i(j_i-1)}{2}$. □

4 Conclusion and open problems

In this paper we have investigated the list update problem with lookahead. We have defined two different models of lookahead and developed lower and upper bounds on the competitiveness that can be achieved by deterministic on-line algorithms with lookahead. However, our bounds are not tight; we conjecture that the algorithms FREQUENCY-COUNT(l) perform better than we can actually prove. One open problem is to tighten the gaps between the lower and upper bounds. Our on-line algorithms with lookahead are competitive against static off-line algorithms. Another open problem is to develop algorithms that are competitive against dynamic off-line algorithms, too.

References

[1] S. Albers. The influence of lookahead in competitive paging algorithms. In *Proc. 1st Annual European Symposium on Algorithms*, Springer Lecture Notes in Computer Science, Volume 726, pages 1-12, 1993.

[2] S. Albers. Improved randomized on-line algorithms for the list update problem. In *Proc. 6th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 412–419, 1995.

[3] S. Albers, B. von Stengel and Ralph Werchner. A combined BIT and TIMESTAMP algorithm for the list update problem. *Information Processing Letters*, **56**:135–139, 1995.

[4] F. d'Amore, A. Marchetti-Spaccamela and U. Nanni. Competitive algorithms for the weighted list update problem. *Theoretical Computer Science*, **108**(2):371–384, 1993.

[5] S. Ben-David and A. Borodin. A new measure for the study of on-line algorithms. *Algorithmica*, **11**(1):73-91, 1994.

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

[7] J.L. Bentley, K.L. Clarkson and D.B. Levine. Fast linear expected-time algorithms for computing maxima and convex hulls. In *Proc. 1st ACM-SIAM Symposium on Discrete Algorithms*, pages 179-187, 1990.

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

[9] J.L. Bentley, D.D. Sleator, R.E. Tarjan and V. Wei. A locally adaptive data compression scheme. *Communications of the ACM*, **29(4)**:320-330, 1986.

[10] J.R. Bitner. Heuristics that dynamically organize data structures for representing sorted lists. *SIAM Journal on Computing*, **8**:82-110, 1979.

[11] D. Breslauer. On competitive on-line paging with lookahead. In *Proc. 13th Annual Symposium on Theoretical Aspects of Computer Science*, Springer Lecture Notes in Computer Science, Volume 1046, pages 593–603, 1996.

[12] M. Burrows and D.J. Wheeler. A block-sorting lossless data compression algorithm. DEC SRC Research Report 124, 1994.

[13] P.J. Burville and J.F.C. Kingman. On a model for storage and search. *Journal of Applied Probability*, **10**(3):697-701, 1973.

[14] F.K. Chung, R. Graham and M.E. Saks. A dynamic location problem for graphs. *Combinatorica*, **9**(2):111-131, 1989.

[15] M.J. Golin. *Probabilistic Analysis of Geometric Algorithms*. Ph.D. thesis, Princeton University, 1991. Available as Computer Science Department Technical Report CS-TR-266-90.

[16] E.F. Grove. Online bin packing with lookahead. In *Proc. 6th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 430–436, 1995.

[17] M.M. Halldórsson and M. Szegedy. Lower bounds for on-line graph coloring. In *Proc. 3rd Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 211-216, 1992.

[18] W.J. Hendricks. An extension of a theorem concerning an interesting Markov chain. *Journal of Applied Probability*, **10**(4):886-890, 1973.

[19] S. Irani. Coloring inductive graphs on-line. In *Proc. 31st Annual IEEE Symposium on Foundations of Computer Science*, pages 470-479, 1990.

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

[21] M.-Y. Kao and S.R. Tate. Online matching with blocked input. *Information Processing Letters*, **38**:113-116, May 1991.

[22] R. Karp and P. Raghavan. From a personal communication cited in [25].

[23] E. Koutsoupias and C.H. Papadimitriou. Beyond competitive analysis. In *Proc. 35th Annual IEEE Symposium on Foundations of Computer Science*, pages 394–400, 1994.

[24] N. Reingold and J. Westbrook. Optimum off-line algorithms for the list update problem. Technical Report YALEU/DCS/TR-805, August 1990.

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

[26] R. Rivest. On self-organizing sequential search heuristics. *Communications of the ACM*, **19**(2):63-67, 1976.

[27] D.D. Sleator and R.E. Tarjan. Amortized efficiency of list update and paging rules. *Communication of the ACM*, **28**:202-208, 1985.

[28] B. Teia. A lower bound for randomized list update algorithms. *Information Processing Letters*, **47**:5-9, 1993.

[29] E. Torng. A unified analysis of paging and caching. In *Proc. 36th Annual IEEE Symposium on Foundations of Computer Science*, pages 194–203, 1995.

[30] N. Young. *Competitive Paging and Dual-Guided On-Line Weighted Caching and Matching Algorithms*. Ph.D. thesis, Princeton University, 1991. Available as Computer Science Department Technical Report CS-TR-348-91.