Revisiting the COUNTER Algorithms for List Update

Susanne Albers* Michael Mitzenmacher†

Abstract

COUNTER algorithms, a family of randomized algorithms for the list update problem, were introduced by Reingold, Westbrook and Sleator [7]. They showed that for any $\epsilon > 0$, there exist COUNTER algorithms that achieve a competitive ratio of $\sqrt{3} + \epsilon$. In this paper we use a mixture of two COUNTER algorithms to achieve a competitiveness of 12/7, which is less than $\sqrt{3}$. Furthermore, we demonstrate that it is impossible to prove a competitive ratio smaller than 12/7 for any mixture of COUNTER algorithms using the type of potential function argument that has been used so far. We also provide new lower bounds for the competitiveness of COUNTER algorithms in the standard cost model, including a 1.625 lower bound for the variant BIT and a matching 12/7 lower bound for our algorithm.

1 Introduction

The *list update problem*, a fundamental and extensively studied on-line problem, is to maintain an unsorted linear linked list so as to minimize the total cost of accesses on a sequence of requests. (The formal definition of the problem is given in the next section.) List update algorithms are useful for maintaining small dictionaries and can be used as subroutines in adaptive data compression schemes. For more information, see, for example, [4] or [2].

The best competitive ratio that can be achieved by deterministic on-line algorithms is 2. Sleator and Tarjan [8] proved that the Move-To-Front rule is 2-competitive, and Karp and Raghavan [5] observed that this is the best achievable competitive ratio for any deterministic on-line algorithm for the problem. More recent work has focused on randomized list update algorithms. Here we consider algorithms against the oblivious adversary, see [3]. Against adaptive adversaries, no randomized on-line algorithm for list update can be better than 2-competitive.

Reingold *et al.* [7] developed an elegant family of so-called COUNTER algorithms. On each request to an item in the list, these algorithms either move the item to the front of the list or leave it where it is. The decision whether to move or not depends on the value of a counter that is initialized randomly. For any positive ϵ, there are algorithms in this family that achieve a competitive ratio of $\sqrt{3} + \epsilon \approx 1.73 + \epsilon$. Randomized algorithms achieving a better competitive ratio were presented in [1]. A drawback of these algorithms is that, in a straightforward implementation, a second pass through the list is required after each request to an item. In some applications, such as data compression, this may not be of concern; however, in other applications, the simplicity and ease of implementation of COUNTER algorithms may be preferable.

*Max-Planck-Institut für Informatik, Im Stadtwald, 66123 Saarbrücken, Germany. E-mail: albers@mpi-sb.mpg.de

†Digital Equipment Corporation, Systems Research Center, 130 Lytton Ave, Palo Alto, CA 94301. E-mail: michaelm@pa.dec.com

This paper is motivated by the goal of finding improved simple randomized list update algorithms. In particular we reconsider the COUNTER algorithms and show that, by properly mixing variants of these algorithms, one can achieve a competitive ratio of $\frac{12}{7}$, which is less than $\sqrt{3}$. Furthermore, we demonstrate that it is impossible to prove a competitive ratio smaller than $\frac{12}{7}$ for any mixture of COUNTER algorithms using the type of potential function argument that has been used so far.

Next, we develop new lower bounds on the competitive ratio of COUNTER algorithms in the standard model that are very close to the upper bounds developed by Reingold *et al.* [7]. Our results demonstrate that BIT, the simplest and most well-known member of the COUTNER family, is not better than 1.625-competitive. Also, it demonstrates that our analysis of the $\frac{12}{7}$-competitive mixed algorithm is tight.

Finally, we briefly describe how the technique of mixing COUNTER algorithms can be also used to improve similar counter-based on-line algorithms for page migration from [9].

2 The list update problem and COUNTER algorithms

We formally define the list update problem. Consider n items stored in an unsorted linear linked list. A list update algorithm receives a sequence of *requests*, where each request specifies one item in the list. To *serve* a request, the algorithm must *access* the requested item, i.e. it starts at the front of the list and proceeds linearly through the items until the desired item is found. In the standard model, serving an access to the item at position i in the list incurs a cost of i. Immediately after an access, 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*. At any time two adjacent items in the list may also be exchanged at a cost of 1; these exchanges are called *paid exchanges*. The goal is to serve a *sequence of requests* so that the total cost is as small as possible. A list update algorithm typically works *on-line*, i.e.. when serving a present request, the algorithm has no knowledge of future requests.

A randomized list update algorithm A is *c-competitive* against any oblivious adversary if there exists a constant a such that, for all list sizes and all request sequences σ,

$$E[C_A(\sigma)] \leq c \cdot C_{OPT}(\sigma) + a.$$

Here $E[C_A(\sigma)]$ denotes the expected cost incurred by A, and $C_{OPT}(\sigma)$ denotes the cost incurred by an *optimal offline algorithm* on σ. An optimal offline algorithm knows the entire request sequence in advance and can serve it with minimum cost.

In [7], Reingold *et al.* first presented an elegant randomized algorithm, called BIT. For each item x in the list, BIT maintains a bit $b(x)$. These bits are initialized independently and uniformly at random. Whenever an item x is accessed, its bit is complemented. If the bit changes to 0, the item is moved to the front of the list; otherwise the position of the item remains unchanged. Reingold *et al.* [7] showed that BIT is 1.75-competitive.

The COUNTER algorithms are a generalization of the BIT algorithm. Let s be a positive integer, and let S be a nonempty subset of $\{0, \ldots, s-1\}$. A COUNTER(s, S) algorithm maintains a counter modulo s, which we shall call the *counter size*, for each item in the list. The counters are initialized

independently and uniformly at random to a value in $\{0, \ldots, s-1\}$. On each access to an item x, the counter of x is decremented by 1, and x is moved to the front of the list if the counter value is in S. Reingold *et al.* [7] showed that for any $\epsilon > 0$, there is a $(\sqrt{3}+\epsilon)$-competitive COUNTER algorithm. Note that COUNTER algorithms are *barely random*, which means that they use only a constant number of random bits (for the initialization of the counters) regardless of the number of requests. Reingold *et al.* [7] also demonstrate a $\sqrt{3}$-competitive algorithm that is not barely random, based on the similar family of RANDOM RESET algorithms.

3 Improved COUNTER algorithms

In this section, we show how to mix COUNTER algorithms to achieve a better competitive ratio of $\frac{12}{7}$. We first extend the notation to include mixtures of COUNTER algorithms. If $\sum_{i=1}^{k} q_i = 1$, we let COUNTER$(s_1, S_1, q_1; \ldots; s_k, S_k, q_k)$ be the randomized on-line algorithm that uses the algorithm COUNTER(s_i, S_i) with probability q_i. Note that this is equivalent to choosing the counter size of all items randomly.

Given a particular COUNTER algorithm and a sequence $\sigma = \sigma(1), \sigma(2), \ldots, \sigma(m)$ of accesses, let $C_{CT}(\sigma(t))$ and $C_{OPT}(\sigma(t))$ denote the actual costs incurred by COUNTER and OPT on $\sigma(t)$, $1 \leq t \leq m$. In [6], it was shown that there exists an optimal offline algorithm that only uses paid exchanges to move items in the list; we will implicitly use this in what follows. Reingold *et al.* [7] analyzed list update algorithms using potential functions.

Definition 1 Given a sequences of accesses σ and a non-negative potential function Φ, a COUNTER$(s_1, S_1, q_1; \ldots; s_k, S_k, q_k)$ algorithm is called

a) c_1-*competitive on accesses* if for every access $\sigma(t)$ in σ, $E[C_{CT}(\sigma(t))] + E[\Delta\Phi] \leq c_1 C_{OPT}(\sigma(t))$. Here $E[\Delta\Phi]$ is the expected change in potential during the operation.

b) c_2-*competitive on paid exchanges* if $E[\Delta\Phi] \leq c_2$ for any paid exchange made by OPT.

The bounds of [7] are obtained by noting that if a COUNTER(s, S) algorithm is c_1-competitive on accesses and c_2-competitive on paid exchanges, then it is $\max\{c_1, c_2\}$-competitive. We improve on this by taking mixtures of COUNTER algorithms.

We introduce a key proposition, taken from Theorem 3.5 in [7], that describes the competitiveness of COUNTER(s, S) algorithms on accesses and paid exchanges. For a COUNTER(s, S) algorithm, let p_j be the probability that an item will next move to the front after j accesses. (For example, in the BIT algorithm, $p_1 = p_2 = \frac{1}{2}$.)

Proposition 2 *COUNTER(s, S) is $\left(1 + p_1 \sum_{j=1}^{s} j p_j\right)$-competitive on accesses and $\left(\sum_{j=1}^{s} j p_j\right)$-competitive on paid exchanges.*

The following theorems show that a simple mixture of COUNTER algorithms yields a smaller competitive ratio than any individual COUNTER algorithm under current methods of analysis, and that the mixture we present is currently the best possible:

Theorem 3 *The algorithm COUNTER*$(2, \{0\}, \frac{4}{7}; 3, \{0\}, \frac{3}{7})$ *is* $\frac{12}{7}$*-competitive.*

Note that $\frac{12}{7} = 1.714... < \sqrt{3} = 1.732...$. Moreover, this mixture of COUNTER algorithms is barely random and uses a very small number of random bits and bits of memory per item.

Proof: By Proposition 2, COUNTER$(2, \{0\})$ is $\frac{7}{4}$-competitive on accesses and $\frac{3}{2}$-competitive on paid exchanges. Similarly, COUNTER$(3, \{0\})$ is $\frac{5}{3}$-competitive on accesses and 2-competitive on paid exchanges. It is straightforward to check that COUNTER$(2, \{0\}, \frac{4}{7}; 3, \{0\}, \frac{3}{7})$ is then $\frac{12}{7}$-competitive on accesses as well as on paid exchanges and therefore $\frac{12}{7}$-competitive. ■

Theorem 4 *It is impossible to prove a competitive ratio smaller than* $\frac{12}{7}$ *for any COUNTER algorithm using only Proposition 2.*

Proof: The proof is a standard game-theoretical argument; here we use geometry to simplify matters. If a COUNTER algorithm is c_1-competitive on accesses and c_2-competitive on paid exchanges by Proposition 2, let us plot its location in the (x, y) plane by (c_2, c_1). The possible mixtures of COUNTER algorithms can be represented as convex combinations of these points, and hence lie on the convex hull determined by these points.

Any point obtained by Proposition 2 has the form $(z, 1 + p_1 z)$, where $z = \sum_{j=1}^{s} j p_j$. Also note that $p_1 \geq p_2 \geq \ldots \geq p_s$ for any COUNTER algorithm. We now take cases for p_1 to show that it is impossible for any point determined by Proposition 2 to lie below the line determined by $(\frac{3}{2}, \frac{7}{4})$ and $(2, \frac{5}{3})$ (given by $y = 2 - \frac{x}{6}$). This suffices to prove the claim.

Note that all points we consider are of the form $y = 1 + p_1 x$. This intersects the line $y = 2 - \frac{x}{6}$ at $x = \frac{6}{1+6p_1}$, and hence, if $z \geq \frac{6}{1+6p_1}$, the corresponding point lies above the line. We therefore only need to show that $\sum_{j=1}^{s} j p_j = z \geq \frac{6}{1+6p_1}$, with equality only when $p_1 = \frac{1}{2}$ or $p_1 = \frac{1}{3}$.

Note that the minimum value of z for a given value of p_1 is obtained by successively setting all $p_2 = p_1, p_3 = p_1, \ldots$, as far as possible, and setting the last p_j as large as possible so as to satisfy $\sum_{j=1}^{s} p_j = 1$. Using this fact, we now take cases.

If $0 \leq p_1 \leq \frac{1}{6}$, then

$$z = \sum_{j=1}^{s} j p_j \geq \sum_{j=1}^{6} j p_1 + 7(1 - 6p_1) = 7 - 21 p_1.$$

One may now easily check that $z \geq \frac{6}{1+6p_1}$ over this interval. Similarly, for $\frac{1}{6} \leq p_1 \leq \frac{1}{3}$, $z \geq \sum_{j=1}^{3} j p_1 + 4(1 - 3p_1) = 4 - 6p_1$, which is at least $\frac{6}{1+6p_1}$ over the interval. For $\frac{1}{3} \leq p_1 \leq \frac{1}{2}$, $z \geq 3 - 3p_1$; and for $\frac{1}{2} \leq p_1 \leq 1$, $z \geq 2 - p_1$. Again, these are both at least $\frac{6}{1+6p_1}$ over the appropriate intervals. The proof follows. ■

We note that the lower bound argument of Theorem 4 applies to the RANDOM RESET algorithms of [7].

Choosing the counter size randomly can be used to obtain algorithms with marginally better competitive ratios in the P^d model for the list update problem described in [7] as well.

4 Lower bounds

We now consider lower bounds on the competitive ratio of COUNTER algorithms in the standard model. First we give a general lower bound for COUNTER(s, S) algorithms that is very close to the upper bounds presented in Proposition 2. Using this general lower bound, we are able to derive a lower bound for BIT. Previous lower bounds for BIT were given by Reingold *et al.* [7] in the $i-1$ cost model, in which an access to the i-th item in the list incurs a cost if $i-1$ rather than i. In fact, they showed that in this model, BIT is exactly 1.75-competitive. We prove that BIT is no better than 1.625-competitive in the standard model. As a second corollary of our general lower bound, we find that COUNTER$(2, \{0\}, \frac{4}{7}; 3, \{0\}, \frac{3}{7})$ is exactly $\frac{12}{7}$-competitive.

Theorem 5 *If s is independent of the list size n, then the competitive ratio achieved by COUNTER(s, S) in the standard model is at least $\frac{1}{2} \cdot \left(1 + p_1 \sum_{j=1}^{s} jp_j + \sum_{j=1}^{s} jp_j\right)$.*

Note that the lower bound given in Theorem 5 is composed of the average of the terms from Proposition 2.

Proof: Consider a list of n items. We assume without loss of generality that COUNTER(s, S) and an optimal off-line algorithm OPT start with the same initial list, with the items in order $1, 2, \ldots, n$. Let $k \geq \max\{2, s\}$ be a constant and let i^k denote a sequence of k consecutive requests to i. The request sequence generated by an adversary consists of a sequence of *rounds*. Each round is a concatenation of two *subrounds* R_1 and R_2, where

$$R_1 = 1, 2, 3, \ldots, n, 1^k, 2^k, 3^k, \ldots, n^k$$

and R_2 is the reverse sequence,

$$R_2 = n, n-1, \ldots, 2, 1, n^k, (n-1)^k, \ldots, 2^k, 1^k.$$

That is, in each subround, every item in the list is first requested exactly once and then requested k times in a row. By generating rounds in this manner, the adversary can construct an infinitely long request sequence.

We assume without loss of generality that whenever OPT serves at least two consecutive requests to an item i, it moves i to the front of the list on the first of these requests. This cannot incur a higher cost on the remaining request sequence than moving i part-way to the front or leaving it where is was. Thus, at the beginning of each round the items in OPT's list are arranged in the order $1, 2, 3, \ldots, n$. The same is true for COUNTER(s, S)'s list because after $k \geq s$ consecutive requests to the same item, COUNTER(s, S) must have that item at the front of its list.

We analyze an arbitrary round and first show that OPT's cost in each round is $2n(n+k)$. Consider the first subround R_1. When serving the first n requests $R_{11} = 1, 2, \ldots, n$, OPT does not move the items. On the next kn requests $R_{12} = 1^k, 2^k \ldots, n^k$, OPT always moves the requested item to the front of the list on its first appearance, giving a total cost of $\frac{1}{2}n(n+1) + \frac{1}{2}n(n+1) + (k-1)n = n(n+k)$ for the processing of R_1. At the beginning of R_2, the items in OPT's list are arranged in the order $n, n-1, \ldots, 2, 1$. Thus, processing R_2 incurs again a cost of $n(n+k)$ to OPT.

Next we evaluate COUNTER(s,S)'s cost. Recall again that at the beginning of each round the items in COUNTER(s,S)'s list are arranged in the order $1,2,3,\ldots,n$. Within R_1, serving the first n requests $R_{11} = 1,2,\ldots,n$ incurs a cost of $\frac{1}{2}n(n+1)$. We have to analyze the expected cost on the following requests $R_{12} = 1^k, 2^k, \ldots, n^k$. Within R_{12}, consider k requests i^k, $1 \leq i \leq n$. At the first of these requests, i's expected position in the list is $i + p_1(n-i)$. This is because (a) all items j with $j < i$ were requested k times since the last request to i and thus precede i in COUNTER(s,S)'s list; (b) the expected number of items j with $j > i$ that were moved to the front of the list during the processing of R_{11} is $p_1(n-i)$. Moreover, from this argument, one may deduce that i's position is in fact *independent* of how many accesses to i are necessary to move it to the front. Thus, the expected number of accesses until i is moved to the front is just $\sum_{j=1}^{s} jp_j$. Hence, the expected cost to COUNTER(s,S) on R_1 is

$$\begin{aligned}
\frac{1}{2}n(n+1) + \sum_{i=1}^{n}\left[(i + p_1(n-i))\left(\sum_{j=1}^{s} jp_j\right) + k - \sum_{j=1}^{s} jp_j\right] &= \\
\frac{1}{2}n(n+1) + kn + \left(\sum_{j=1}^{s} jp_j\right)\left(\sum_{i=1}^{n}(i - 1 + p_1(n-i))\right) &= \\
\frac{1}{2}n(n+1) + kn + \frac{1}{2}n(n-1)(\sum_{j=1}^{s} jp_j) + \frac{1}{2}n(n-1)(p_1\sum_{j=1}^{s} jp_j).
\end{aligned}$$

COUNTER(s,S)'s expected cost on R_2 is the same because, at the beginning of R_2, the items in COUNTER(s,S)'s list are arranged in the order $n, n-1, \ldots, 2, 1$. The dominant terms in the cost to both OPT and COUNTER(s,S) over one round are clearly proportional to n^2. Since we may take n to be arbitrarily large, the theorem now follows by taking the appropriate ratio of these terms. ■

Corollary 6 *The competitive ratio achieved by BIT in the standard model is at least* $\frac{13}{8} = 1.625$.

Proof: The corollary follows immediately from Theorem 5, because in the BIT algorithm $p_1 = p_2 = \frac{1}{2}$. ■

Based on Theorem 5, we can also show that COUNTER$(2, \{0\}, \frac{4}{7}; 3, \{0\}, \frac{3}{7})$ is exactly $\frac{12}{7}$-competitive.

Corollary 7 *The competitive ratio achieved by COUNTER*$(2, \{0\}, \frac{4}{7}; 3, \{0\}, \frac{3}{7})$ *in the standard model is exactly* $\frac{12}{7}$.

Proof: We consider the request sequence used in the proof of Theorem 5, where each round consists of subrounds

$$R_1 = 1,2,3,\ldots,n,1^3,2^3,3^3,\ldots,n^3$$

and

$$R_2 = n, n-1, \ldots, 2, 1, n^3, (n-1)^3, \ldots, 2^3, 1^3.$$

The proof of Theorem 5 shows that, for every $\epsilon > 0$, there exists an n_0 such that for every $n \geq n_0$, the cost incurred by COUNTER$(2, \{0\})$ on this sequence is at least $\frac{13}{8} - \epsilon$ times the optimum offline cost. A similar statement holds for COUNTER$(3, \{0\})$ with a ratio of $\frac{11}{6}$. Hence the combination is at best $\frac{12}{7}$-competitive. The corresponding upper bound is proven in Theorem 3. ■

5 Page migration algorithms

Choosing a counter size randomly to obtain algorithms with improved competitive ratios can also be applied to the randomized page migration algorithms found in [9], which are similarly based on simple counters. For example, in the notation of [9], for uniform graphs where the page size $D = 1$, a 2.75-competitive randomized algorithm is given using a modulo 2 counter. This can be improved to a $\frac{29}{11}$-competitive algorithm by using an algorithm that randomly chooses an algorithm with either a modulo 2 counter (with probability $\frac{8}{11}$) or a modulo 3 counter (with probability $\frac{3}{11}$). As D grows to infinity, however, the improvement obtainable using this method falls to 0.

It is interesting to note for uniform graphs the analysis of [9] uses different potential functions for each possible counter size. This poses no barrier to analyzing mixtures of algorithms, as analyzing mixtures requires knowing how competitive each algorithm is on a request or a move by the optimal algorithm; it does not matter what potential function is used to obtain the result.

We point out that the random resetting and deterministic resetting algorithms suggested in [9] for the case of general graphs can be improved similarly by choosing the size of the counter randomly with appropriate weights.

6 Conclusion

We have improved bounds for counter-based algorithms on the list update problem, by randomizing the choice of the counter size. We have also explained that this paradigm can similarly be used to improve counter based page migration algorithms. Our work thus emphasizes the general principle that one can often achieve better competitive ratios by choosing a random algorithm within a family of online algorithms.

With our lower bounds, we have achieved nearly tight bounds on members of the the COUNTER family. We conclude that beating the 1.6-competitive algorithm for list update from [1] will require different types of algorithms, although the simple COUNTER algorithms may be more useful in practice.

References

[1] 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.

[2] R. Bachrach and R. El-Yaniv. Online list accessing algorithms and their applications: Recent Empirical Evidence. In *Proceedings of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 53-62, 1997.

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

[4] J.L. Bentley, D.S. Sleator, R.E. Tarjan, and V.K. Wei. A locally adaptive data compression scheme. *Communication of the ACM*, 29:320–330, 1986.

[5] R. Karp and P. Raghavan. From a personal communication cited in [7].

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

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

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

[9] J. Westbrook. Randomized algorithms for multiprocessor page migration. *SIAM Journal of Computing*, 23:951–965, 1994.