Algorithmica (1994) 11: 15–32

Algorithmica

Randomized Competitive Algorithms for the List Update Problem[1]

Nick Reingold,[2] Jeffery Westbrook,[3] and Daniel D. Sleator[4]

Abstract. We prove upper and lower bounds on the competitiveness of randomized algorithms for the list update problem of Sleator and Tarjan. We give a simple and elegant randomized algorithm that is more competitive than the best previous randomized algorithm due to Irani. Our algorithm uses randomness only during an initialization phase, and from then on runs completely deterministically. It is the first randomized competitive algorithm with this property to beat the deterministic lower bound. We generalize our approach to a model in which access costs are fixed but update costs are scaled by an arbitrary constant *d*. We prove lower bounds for deterministic list update algorithms and for randomized algorithms against oblivious and adaptive on-line adversaries. In particular, we show that for this problem adaptive on-line and adaptive off-line adversaries are equally powerful.

Key Words. Sequential search, List-update, On-line algorithms, Competitive analysis, Randomized algorithms.

1. Introduction. Recently much attention has been given to *competitive analysis* of on-line algorithms [7], [20], [22], [25]. Roughly speaking, an on-line algorithm is c-competitive if, for any request sequence, its cost is no more than c times the cost of the optimum off-line algorithm for that sequence. In their seminal work on competitive analysis [25], Sleator and Tarjan studied heuristics commonly used in system software to maintain a set of items as an unsorted linear list. This problem is called the *list update* or *sequential search* problem. The cost of accessing an item is equal to its distance from the front of the list, and the list may be rearranged (at a cost of one per swap of adjacent elements) during the processing of a sequence of requests so that later accesses will be cheaper; for example, a commonly requested item may be moved closer to the front.

Maintaining a dictionary as a linear list is frequently used in practice because of its great simplicity. Furthermore, self-adjusting rules are effective because they take advantage of the locality of reference found in real systems. List update techniques have also been used to develop data compression algorithms [5], as

---

[1] A preliminary version of these results appeared in a joint paper with S. Irani in the *Proceedings of the 2nd Symposium on Discrete Algorithms*, 1991 [17].

[2] AT&T Bell Laboratories, 600 Mountain Avenue, Murray Hill, NJ 07974-0636, USA.This research was partially supported by NSF Grants CCR-8808949 and CCR-8958528.

[3] Department of Computer Science, Yale University, New Haven, CT 06520–2158, USA. This research was partially supported by NSF Grant CCR-9009753.

[4] Carnegie-Mellon University, Pittsburgh, PA, USA. This research was supported in part by the National Science Foundation under Grant CCR-8658139, by DIMACS, a National Science Foundation Science and Technology center, Grant No. NSF-STC88-09648.

Received September 18, 1990; revised November 22, 1991. Communicated by Prabhakar Raghavan.

$b \geq 1/2$. As in the theorem above, the adversary can arrange to pay arbitrarily close to $m(n+1)/2$. Since D is by assumption at least $mn/2$, A cannot be more than $3n/(n+1)$-competitive. On the other hand, suppose $b < \frac{1}{2}$. Now the adversary adopts the following strategy: at all times it maintains its list in the exact reverse order of A's list. That is, the adversary initially reverses its list and then does the reverse of every exchange A makes. The total cost to the adversary is at most $m + D + dn(n-1)/2$, since each access costs it 1. Ignoring the initial cost of reversal, the ratio of A's cost to the adversary's cost is $n(1+b)/(1+bn)$, which is at least $3n/(n+2)$ for $b \leq \frac{1}{2}$. Since one of the two strategies is applicable for all n, A cannot be better than 3-competitive. □

Notice that if we fix the size of the list to be n we get a lower bound of $3n/(n+2)$. Theorem 4.2 can be extended to the model in which an access to item i has cost $i-1$. In this situation the lower bound is exactly 3 regardless of n. The technique used in the proof of Theorem 4.2 can be extended to handle randomized algorithms against adaptive on-line adversaries.

THEOREM 4.3. *Let A be an on-line algorithm for list update in a P^d model. If A is randomized and c-competitive against adaptive on-line adversaries, then $c \geq 3$.*

PROOF. Suppose that A is an on-line algorithm. Consider $\hat{A}$, an adaptive on-line adversary, which behaves as follows. $\hat{A}$ first simulates A on all possible choices of A's random bits, and for each such choice of random bits, creates a request sequence of length m such that each request is to the last item in A's list. Consider the collection of all request sequences constructed in this way. The choice of A's random bits induces a probability distribution on these (indeed, all) sequences of length m. That is, the probability of a sequence is the probability that $\hat{A}$ will generate that sequence when it runs A and uses the strategy of always accessing the last item in A's list. $\hat{A}$ can compute the expected value of $b = D/mn$. Based on the expected value of b, $\hat{A}$ chooses one or the other strategy as in the deterministic case, and achieves the same lower bounds. □

The results of this section show that no randomized on-line algorithm can beat the deterministic lower bound against an adaptive on-line algorithm. It is known that the analogous theorem to Theorem 4.1 holds for other types of on-line situations such as the caching, but it is not known to hold in more general settings. The relationship between on-line adaptive and off-line adaptive adversaries in more general settings is an interesting open problem.

5. Lower Bounds Against Oblivous Adversaries. In this section we discuss a technique for deriving lower bounds for randomized on-line algorithms against oblivious adversaries. In this section when we refer to competitive ratios we mean competitive ratio against an oblivious adversary.

Karp and Raghavan (private communication, 1990) use the following strategy:

find an off-line algorithm with a small average cost per request assuming that the requests are drawn uniformly at random from $\{1, 2, \ldots, n\}$. If the average cost per request is no more than d, then no on-line algorithm can be better than $(n + 1)/2d$-competitive. Karp used this technique to obtain a lower bound of 9/8 for the competitive ratio of any on-line algorithm for lists of two items, and Raghavan (private communication, 1990) showed a lower bound of 1.18 for three-item lists. We extend these results by considering larger lists, and more complicated off-line algorithms. Fix the size of the list, n, and a *lookahead* number, k. Suppose we have an off-line algorithm that bases its decision on only the next $k + 1$ requests. Following Raghavan's approach, we can model the behavior of such an algorithm for randomly generated request sequences as follows. Consider a Markov chain whose states are k-tuples of request positions. State $\langle r_1, r_2, \ldots, r_k\rangle$ corresponds to the situation where the current *positions* of the next k requests are, respectively, r_1, $r_2, \ldots, r_k$. The transitions are conditioned on the current position of the $(k + 1)$st request. If we know the algorithm, we can construct the transition matrix for the chain.

Each state transition has a cost (the cost of any paid exchanges made just prior to the access plus the cost of the access), so for each state we can compute an expected cost. Let the vector of these expected costs be $\mathbf{c}$. Suppose the chain has stationary distribution $\boldsymbol{\pi}$. Then the expected cost per access is $\mathbf{c} \cdot \boldsymbol{\pi}$, if the chain is in steady state.

For $n \geq 3$ there is no obvious strategy for a k-lookahead algorithm, since no bounded lookahead algorithm can be optimum [23]. Furthermore, the size of the transition matrix is n^k, so for values of n and k even a little bigger than 3 it is infeasible to try different strategies by hand and compute the steady-state vector symbolically. Thus there are two problems: determining a good way to find off-line algorithms and generate transition rules for large lists, and finding the steady-state distribution of the resultant transition matrix.

Our approach is to write a program that generates the entries of the matrix corresponding to a particular class of good off-line algorithms, MARKOV(n, k), and then compute the steady-state vector numerically. Suppose we have a program OFF(n, k) that, given a sequence of exactly $k + 1$ accesses to an n-item list, generates a sequence of moves to service that request sequence. MARKOV(n, k) works as follows: start with the next k requests; look at the $(k + 1)$st request; service the next request in the same way that OFF(n, k) would service that request given the same sequence of $k + 1$ requests on the same list. Given an implementation of OFF(n, k), the entries of the transition matrix and cost vector for MARKOV(n, k) are filled in as follows. If the current state is $\langle r_1, r_2, \ldots, r_k\rangle$ (that is, if the next k requests are to the items in positions r_1, $r_2, \ldots, r_k$), and the next request is to the item in position r_{k+1}, transit to whatever state corresponds to the list that results from OFF(n, k) servicing request r_1 in the sequence $\langle r_1, r_2, \ldots, r_{k+1}\rangle$, assuming its list is $1, 2, \ldots, n$. The probability of that transition is $1/n$, since the request sequence is uniformly random. The cost of that state is $1/n$ times the sum over all choices of r_{k+1} of the cost of servicing the first request in $\langle r_1, r_2, \ldots, r_{k+1}\rangle$. Thus to generate the matrix we must generate all possible sequences of $k + 1$ requests on n items and run OFF(n, k) on each of them.

Any choice of an off-line algorithm gives a valid MARKOV(n, k) algorithm and transition matrix, but some choices are better than others. Our best results were achieved using a program to compute an optimum off-line algorithm for the $k + 1$ requests that has the following properties:

1. It only does paid exchanges.
2. It services a request to item x by choosing some subset of the items preceding x in the list and moving them in an order-preserving way to immediately after x. Then it pays for the access to x and goes on to the next request.
3. Whenever an item is requested twice in a row that item is moved to the front on its first access. This means that MARKOV(n, k) has the same property and ensures that the Markov chain has at most one stationary distribution (see Lemma 5.1).
4. Among optimum algorithms that have the above three properties, our algorithm services the first request with the lowest cost. This tends to make MARKOV(n, k)'s cost per access smaller, so we get a better lower bound.

It is shown in [23] that there is an optimum algorithm with the first three properties. There is no known way to compute the optimum algorithm for a given request sequence that is polynomial in both n and k, however, so the time and space to generate MARKOV(n, k) grow exponentially.

LEMMA 5.1. *For any n and k, the Markov chain corresponding to MARKOV(n, k) is irreducible. That is, for any two states, the probability of transiting from one to the other in some finite time is positive.*

PROOF. Suppose we want to get to state $\langle r_1, r_2, \ldots, r_k \rangle$. Consider MARKOV($n$, k)'s action on request sequence $\langle n, n, n-1, n-1, \ldots, 2, 2, 1, 1 \rangle$. MARKOV($n$, k) will move each item to the front, since each item is requested twice in a row. After that, MARKOV(n, k)'s list must be $1, 2, \ldots, n$, so if the next k requests are $\langle r_1, r_2, \ldots, r_k \rangle$, MARKOV($n$, k) will be in state $\langle r_1, r_2, \ldots, r_k \rangle$. □

By standard probability theory, Lemma 5.1 implies that the steady-state distribution is unique, and it is given by the (unique) eigenvector of the transition matrix corresponding to the eigenvalue 1. We compute this eigenvector from the matrix using the power method [11]. Table 2 shows the best results we have obtained for $n = 3, 4, 5$, and 6. In all cases, the number of iterations in the power method necessary to get the distance between successive iterates less than 10^{-7}

Table 2. Lower bound results.

n	k	Lower bound
3	10	1.1998
4	8	1.2467
5	7	1.2728
6	5	1.268

was no more than 30. As shown in the table, the largest lower bound we have been able to compute is approximately 1.27. We were unable to achieve higher results due to limitations on computational resources. From various simulations using nonoptimum off-line algorithms we believe the true lower bound to be at least 1.4.

6. Remarks and Open Problems. For the standard model, we have given an extremely simple algorithm that is 1.75-competitive against an oblivious adversary, and constructed a slightly more complicated one which is $\sqrt{3}$-competitive. We have shown that no algorithm can be better than 1.27-competitive against such an adversary. (Recently, we have improved the lower bounds for three- and four-item lists to 1.2 and 1.25, respectively.) This leaves a substantial gap. It is possible that our COUNTER algorithms are better than we can currently prove, since we do not know of an instance in which the upper bound is tight.

OPEN QUESTION 1. What is the best competitive ratio a randomized list update algorithm can achieve against oblivious adversaries in the standard model?

In the P^d models we have given a lower bound of 3 for the competitiveness of deterministic algorithms, and for randomized algorithms against adaptive adversaries. We have constructed randomized algorithms with smaller competitive ratios against oblivious adversaries for these models.

OPEN QUESTION 2. What is the best competitive ratio a randomized list update algorithm can achieve against oblivious adversaries in these models?

Our BIT and COUNTER algorithms use only a bounded number of random bits regardless of the number of requests serviced, yet still beat the deterministic lower bound. Recently barely random algorithms have also been found for the migration problem [10], [26].

OPEN QUESTION 3. For which other on-line problems do such algorithms exist?

Our results in Section 4 give evidence for the conjecture that, for a large class of applications, adaptive on-line and adaptive off-line adversaries are equally powerful. Similar results have been obtained for page caching [22] and metrical task systems [12]. On the other hand, the results of [10] and [26] show that this does not hold in general.

OPEN QUESTION 4. For what other classes of on-line problems are these two adversaries equivalent?

Acknowledgments. We thank Richard Beigel, Sandy Irani, Prabhakar Raghavan, and Neal Young for useful discussions.

References

[1] N. Alon, R. M. Karp, D. Peleg, and D. West. A graph-theoretic game and its application to the *k*-server problem. *Proc. DIMACS Workshop on On-line Algorithms*, pages 1–10. American Mathematical Society, Providence, RI, 1991.

[2] S. Ben-David, A. Borodin, R. M. Karp, G. Tárdos, and A. Wigderson. On the power of randomization in on-line algorithms. *Proc. 20th ACM Symp. on Theory of Computing*, pages 379–386, 1990.

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

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

[5] J. L. Bentley, D. D. Sleator, R. E. Tarjan, and V. Wei. A locally adaptive data compression scheme. *Comm. ACM*, 29(4):320–330, 1986.

[6] D. L. Black and D. D. Sleator. Competitive algorithms for replication and migration problems. Technical Report CMU-CS-89-201, Department of Computer Science, Carnegie-Mellon University, 1989.

[7] A. Borodin, N. Linial, and M. Saks. An optimal on-line algorithm for metrical task systems. *Proc. 19th ACM Symp. on Theory of Computing*, pages 373–382, 1987.

[8] P. J. Burville and J. F. C. Kingman. On a model for storage and search. *J. Appl. Probab.*, 10:697–701, 1973.

[9] M. Chrobak and L. Larmore. On fast algorithms for two servers. *J. Algorithms*, 12:607–614, 1991.

[10] M. Chrobak, L. L. Larmore, N. Reingold, and J. Westbrook. Optimal multiprocessor migration algorithms using work functions. Technical Report YALEU/DCS/TR-897, Department of Computer Science, Yale University, 1991.

[11] S. D. Conte and C. de Boor. *Elementary Numerical Analysis, An Algorithmic Approach*, 3rd edn. McGraw-Hill, New York, 1980.

[12] D. Coppersmith, P. Doyle, P. Raghavan, and M. Snir. Random walks on weighted graphs, and applications to on-line algorithms. *Proc. 20th ACM Symp. on Theory of Computing*, pages 369–377, 1990.

[13] A. Fiat, R. Karp, M. Luby, L. McGeoch, D. D. Sleator, and N. Young. On competitive algorithms for paging problems. *J. Algorithms*, 12:685–699, 1991.

[14] M. J. Golin. Ph.D. thesis, Department of Computer Science, Princeton University, 1990. Technical Report CS-TR-266-90.

[15] W. J. Hendricks. An account of self-organizing systems. *SIAM J. Comput.*, 5(4):715–723, 1976.

[16] S. Irani. Two results on the list update problem. *Inform. Process. Lett.*, 38:301–306, 1991.

[17] S. Irani, N. Reingold, J. Westbrook, and D. D. Sleator. Randomized algorithms for the list update problem. *Proc. 2nd ACM–SIAM Symp. on Discrete Algorithms*, pages 251–260, 1991.

[18] A. R. Karlin, M. S. Manasse, L. A. McGeoch, and S. Owicki. Competitive randomized algorithms for non-uniform problems. *Proc. 1st ACM–SIAM Symp. on Discrete Algorithms*, 1990.

[19] A. Karlin, M. Manasse, L. Rudolph, and D. Sleator. Competitive snoopy caching. *Algorithmica*, 3(1):79–119, 1988.

[20] M. Manasse, L. A. McGeoch, and D. Sleator. Competitive algorithms for on-line problems. *Proc. 20th ACM Symp. on Theory of Computing*, pages 322–333, 1988.

[21] J. McCabe. On serial files with relocatable records. *Oper. Res.*, 13:609–618, 1965.

[22] P. Raghavan and M. Snir. Memory versus randomization in on-line algorithms. Research Report RC 15622 (No. 69444), IBM T. J. Watson Reseach Center, 1990.

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

[24] R. Rivest. On self-organizing sequential search heuristics. *Comm. ACM*, 19(2):63–67, 1976.

[25] D. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *Comm. ACM*, 28(2):202–208, 1985.

[26] J. Westbrook. Randomized algorithms for multiprocessor page migration. *Proc. DIMACS Workshop on On-Line Algorithms*, pages 135–150. American Mathematical Society, Providence, RI, 1991.