Two Results on the List Update Problem

Sandy Irani *
Computer Science Division
U.C. Berkeley
Berkeley, California 94720

September 4, 1995

Abstract

In this paper we give a randomized on-line algorithm for the list update problem. Sleator and Tarjan show a deterministic algorithm, Move-to-Front, that achieves competitive ratio of $(2L-1)/L$ for lists of length L. Karp an Raghavan show that no deterministic algorithm can beat $2L/(L+1)$. We show that Move-to-Front in fact achieves an optimal competitive ratio of $2L/(L+1)$. We show a randomized algorithm that achieves a competitive ratio of $(31L+1)/16(L+1)$ against an oblivious adversary. This is the first randomized strategy whose competitive factor beats a constant less than 2.

1. Introduction

In this paper, we examine the problem of updating a list of items so that requests to access items can be performed efficiently. The problem is formulated as follows:

We are given a list of L items. We receive as input a sequence of requests. Each request is the name of an item. The cost of servicing a request is one plus the number of items that precede it in the list. After an item is accessed, it can be moved anywhere closer to the front of the list at no extra cost. The idea is to model a linked list storing unordered data, where in order to access an item, one has to begin at the front and search linearly through the list until the item is found. In searching for the item, one can maintain a pointer to the place where the item is to be placed once found. Then the item can be moved to the new spot in constant time. We also allow *paid exchanges*, where any item may also be moved anywhere in the list, but the cost of moving the item is the distance it is moved.

An algorithm is *on-line* if it decides where to place each requested item without knowledge of future requests. An algorithm is *off-line* if it can see the entire request sequence before servicing the requests. The best known optimal off-line algorithm uses $O(m2^L L!)$ steps where L is the number of items and m is the number of requests [RW]. We call the optimal off-line algorithm OPT.

*This research was supported by an IBM Graduate Fellowship. Part of this work was done at IBM T.J. Watson Research Center, Yorktown Heights, NY.

We are interested in evaluating the worst case over all request sequences, σ, of the ratio of the algorithm's cost on σ to OPT's cost on σ. Formally, let $A(\sigma)$ be the cost of algorithm A on σ and $OPT(\sigma)$ the cost of OPT on σ. We say that A is α-competitive if there exists a β such that for all σ,

$$A(\sigma) \leq \alpha \cdot OPT(\sigma) + \beta.$$

Notice that the constant β can depend on the number of items but not on the request sequence. α is an upper bound on the *competitive ratio* for algorithm A. The competitive ratio was first introduced by Sleator and Tarjan in analyzing on-line algorithms for the list update problem [ST].

Sleator and Tarjan give a deterministic algorithm, MTF (Move-to-Front) which simply moves each requested item to the front of the list after it is requested. Let $MTF(\sigma)$ be the cost of MTF on request sequence σ. They show that for all σ,

$$MTF(\sigma) \leq 2 \cdot OPT(\sigma).$$

More precisely, the analysis in [ST] shows that $MTF(\sigma) \leq (2L-1)/L \cdot OPT(\sigma)$ on a list of length L. Since then, it has been shown by [KR] that for any deterministic algorithm A for the list update problem on a list of length L, there exists a σ such that

$$A(\sigma) \geq 2L/(L+1) \cdot OPT(\sigma).$$

To see this fact, suppose the adversary at each step requests the last item in A's list. Thus A is charged L per request. To service the sequence the adversary orders the list according to decreasing frequency accessed. Ordering the list costs some fixed amount depending only on L. Once the list is ordered, the list remains static. The adversary then incurs a cost of at most $(L+1)/2$ per request on average. The lower bound indicates that MTF achieves the asymptotically best competitive ratio. We show that MTF, in fact, achieves the best possible ratio for every list length L.

Theorem 1: *For all σ,*

$$MTF(\sigma) \leq 2L/(L+1) \cdot OPT(\sigma).$$

The proof of Theorem 1 is in section 2.
The question then remains whether randomization can help. It is important to distinguish between the type of adversary the randomized algorithm is working against. An *oblivious* adversary must choose the request sequence without knowledge of the algorithm's random choices [BBKTW]. An *adaptive on-line* adversary can see the algorithm's random choices in chosing the request sequence, but must also service the requests on-line [BBKTW]. We show a randomized strategy $SPLIT$ and prove the following theorem:

Theorem 2: *Against an oblivious adversary, for all σ,*

$$E[SPLIT(\sigma)] \leq 31/16 \cdot OPT(\sigma)$$

The proof for Theorem 2 is in section 3.
Both theorems are obtained by examining a pair of items and analyzing how much the on-line algorithm spends on that pair compared to how much OPT spends on the pair. For the proof of Theorem 1, the costs are distributed among the pairs such that when we sum the cost for each pair, we obtain exactly the cost of MTF in servicing σ.

There is no hope of beating a competitive ratio of $2L/(L+1)$ against an adaptive adversary because because it has been shown that there is a lower bound of $2L/(L+1)$ on the competitive ratio of any algorithm against an adaptive on-line adversary [RW2].

There is a lower bound of $9/8$ for any algorithm against an oblivious adversary [KR]. This is achieved on a list of length two, where the adversary feeds the algorithm a random sequence of requests of 1's and 2's. The cost of any on-line algorithm is then $3/2$ per request. The analysis for the off-line algorithm involves a Markov chain whose states consist of the configuration of the list and some number of requests into the future. The transitions are then dictated by the next request in the sequence. This technique has also been used by Reingold and Westbrook and independently by Chrobak and Larmore on lists of length three to obtain a lower bound of about 1.27 [CL], [RW2]. The difficulty in extending this technique to longer lists is that there is no known simple rule for the optimal off-line algorithm.

Subsequent to the work presented here, Reingold and Westbrook have devised a randomized on-line algorithm that achieves 1.75 [RW2]. The technique of analyzing pairs of items was also applied to the memoryless algorithm that moves the requested item to the front of the list with probability 1/2 to yield a competitive ratio of $2-(\log L/L)$ for lists of length L [IRWY]. Although this bound approaches 2 as the size of the list grows, it is significant because it is a memoryless algorithm that beats the lower bound against adaptive on-line adversaries for a list of fixed length.

2. Move To Front

Proof of Theorem 1: Fix a sequence $\sigma = (r_1, r_2, \ldots, r_m)$. We examine the relative position of two items, i, and j, in the list. Their relative position changes only on a request to i or j. Notice that if we were to run MTF on a list with just the two items i and j using σ as a request sequence but ignoring requests to items other than i and j, then their relative position in the list of two items is exactly their relative position in the list of L items. This idea of "factoring" the list into lists of length two was used independently in [BM] We define the following cost function for each pair of items.

$$MTF_{ij}(t,\sigma) = \begin{cases} 1+\frac{1}{(L-1)} & \text{i appears before j in MTF's list and item j is requested at time t} \\ \frac{1}{(L-1)} & \text{j appears before i in MTF's list and item j is requested at time t} \\ 0 & \text{otherwise} \end{cases}$$

$OPT_{ij}(t,\sigma)$ is defined similarly with respect to OPT's list. $MTF_{ij}(t,\sigma) = 1 + 1/(L-1)$ if MTF has to step over item i to reach a request to j at time t.

$$MTF(\sigma) = \sum_{t=1}^{|\sigma|} \sum_{1 \le i,j \le L, i \ne j} \left[MTF_{ij}(t,\sigma) + MTF_{ji}(t,\sigma)\right].$$

We prove that for all i and j,

$$\sum_{t=1}^{|\sigma|} [MTF_{ij}(t,\sigma) + MTF_{ji}(t,\sigma)]$$

$$\le 2L/(L+1) \sum_{t=1}^{|\sigma|} [OPT_{ij}(t,\sigma) + OPT_{ji}(t,\sigma)]$$

Thus we have reduced the problem to lists of length two. If the optimal algorithm ever uses a paid exchange between items i and j, the algorithm can be altered so that the cost on i and j remains the same and no paid exchange is. We alter the strategy as follows: used as follows: Suppose item j appears after i in the list before the exchange. The the optimal algorithm just waits for the next request to item j and then moves j ahead of i, paying only the cost to access j. So now, we can assume that the optimal algorithm does not use paid exchanges. We say that $MTF_{ij}(t,\sigma) + MTF_{ji}(t,\sigma)$ and $OPT_{ij}(t,\sigma) + OPT_{ji}(t,\sigma)$ are the cost at time t on the pair $\{i,j\}$ for MTF and OPT, respectively.

The theorem follows from the following lemma:

Lemma 3: *If at time t_1 and t_2, where $t_1 < t_2$, the cost of MTF on $\{i,j\}$ is $1 + 1/(L-1)$ and the cost of OPT on $\{i,j\}$ is $1/(L+1)$, then there is a time t such that $t_1 < t < t_2$ and both MTF and OPT have a cost of $1 + 1/(L-1)$ on $\{i,j\}$ at time t.*

Lemma 3 implies that

$$MTF(\sigma)/OPT(\sigma) \le \frac{2 + 2/(L-1)}{1 + 2/(L-1)} = \frac{2L}{(L+1)}.$$

Proof of Lemma 3: Suppose at time t_1, i precedes j in MTF's list. This means that j was the requested item, since $MTF_{ij}(t_1,\sigma) = 1 + 1/(L-1)$. It also means that j precedes i in OPT's list because $OPT_{ij}(t_1,\sigma) = 1/(L-1)$. After time t_1, MTF moves j before i, so j precedes i in both MTF and OPT's lists. Any request to j will cost both algorithms $1/(L-1)$ and will not change the relative position of i and j in either list. Any request to i will cost both algorithms $1 + 1/(L-1)$. ■

3. The SPLIT Algorithm

The following is a description of the randomized strategy, called $SPLIT$. Each item maintains a pointer that points to some other item in the list. We maintain the invariant that the pointer for an item either points to the item itself or to some item that precedes it in the list. Let *i.split* denote the item then item i points to.

The algorithm works as follows:

Initialization:
For $i \leftarrow 1$ to L
    $i.split \leftarrow i$;

If item i is requested:

With probability 1/2:
    Move item i to the front

With probability 1/2:
    Insert item i before item $i.split$

Set i.split to the first item in the list.

Theorem 4: *Let σ be any request sequence and OPT be the optimal deterministic off-line algorithm that services σ. Then if OPT and SPLIT start with their lists in the same configuration,*

$$E[SPLIT(\sigma)] \leq (31/16)OPT(\sigma)$$

In the analysis of SPLIT we use a slightly non-standard cost function; The cost of accessing the i^{th} item in the list is $i-1$ instead of i. Any upper bound that we obtain on the competitive ratio of an algorithm counting the cost as $i-1$ is also an upper bound on the competitive ratio counting the cost as i. For the remainder of this section, we use the $i-1$ cost function.

We examine a pair of items, $\{i, j\}$. The pair can be in one of four states depending on the relative position of i, j, $i.split$, and $j.split$ in the list. Given a starting configuration and an input sequence, the probability distribution of the states for that pair is well defined. If item r is requested, the cost of servicing that request is the sum over all items i of the probability that i precedes r in the list. We use the symbol $\prec$ to indicate precedence in the list, i.e. $i \prec j$ if i precedes j in the list. The probabilities for the four states are denoted as follows:

$$\begin{aligned} X_{ij}(t) &= Pr[i \prec j.split] \\ Y_{ij}(t) &= Pr[j.split \preceq i \prec j] \end{aligned}$$

$$\begin{aligned} X_{ji}(t) &= Pr[j \prec i.split] \\ Y_{ji}(t) &= Pr[i.split \preceq j \prec i] \end{aligned}$$

Let $P_{ij}(t)$ be the probability that at time t, item i precedes item j in the list. Then $P_{ij}(t) = X_{ij}(t) + Y_{ij}(t)$. Fix a sequence $\sigma = (r_1, r_2, \ldots)$. Let

$$\sigma_i(t) = \begin{cases} 1 & \text{if item } i \text{ is requested at time } t \\ 0 & \text{otherwise} \end{cases}$$

Then the expected cost of servicing the sequence σ is

$$E[SPLIT(\sigma)] = \sum_{t=1}^{|\sigma|} \sum_{1 \leq i,j \leq L, i \neq j} [(\sigma_j(t) P_{ij}(t)) \quad + \quad (\sigma_i(t) P_{ji}(t))]$$

If $r_t \notin \{i,j\}$ then the state of the pair $\{i,j\}$ does not change. If $r_t = i$, then

$$\begin{aligned} X_{ji}(t+1) &= 0 \\ Y_{ij}(t+1) &\leq (Y_{ij}(t) + Y_{ji}(t))/2 \\ Y_{ji}(t+1) &= X_{ji}(t)/2 \end{aligned}$$

are the transition probabilities for the pair $\{i,j\}$. Let

$$OPT_{ij}(t,\sigma) = \begin{cases} 1 & \text{if } r_t = j \text{ and item } i \text{ appears} \\ & \text{before item } j \text{ in } OPT\text{'s list} \\ & \text{at time } t. \\ 0 & \text{otherwise} \end{cases}$$

$OPT_{ij}(t,\sigma) = 1$ if OPT has to step over item i to access j at time t. The cost of OPT is then

$$OPT(\sigma) = \sum_{t=1}^{|\sigma|} \sum_{1 \leq i,j \leq L, i \neq j} [OPT_{ij}(t,\sigma) + OPT_{ji}(t,\sigma)]$$

Claim: For any pair $\{i,j\}$,

$$31/16 \sum_{t=1}^{|\sigma|} [OPT_{ij}(t,\sigma) + OPT_{ji}(t,\sigma)] \geq \sum_{t=1}^{|\sigma|} [(\sigma_i(t) P_{ij}(t)) + (\sigma_j(t) P_{ij}(t))]$$

We prove something slightly stronger. Let σ_{ij} be an input sequence for a list containing only i and j. σ_{ij} is obtained as follows: If the t^{th} request in σ is to an item other than i or j, then no request is made at time t. If the t^{th} request in σ is to i or j, then that item is requested at time t. We prove that

$$(31/16) OPT(\sigma_{ij}) \geq \sum_{t=1}^{|\sigma|} [(\sigma_i(t) P_{ij}(t)) + (\sigma_j(t) P_{ij}(t))]$$

In other words, we allow the optimal algorithm to service requests to every pair of items in a separate list of two items, even if the optimal algorithm on each pair does not merge to form a consistent algorithm on L elements. For example, the adversary can choose to have i ahead of j and j ahead of k, but k ahead of i. Since we compare the cost of $SPLIT$ cost to the cost of the stronger adversary, we do not have to account for paid exchanges. The optimal algorithm when operating on a list of length 2 will never use paid exchanges.

We divide the sequence into intervals, such that a new interval starts every time OPT incurs a cost of one on σ_{ij}. Recall that since we are using the $i-1$ cost function in this analysis, the cost of accessing the first item is 0 and the cost of accessing the second item is 1. Let t_1 and t_m be the first request in two consecutive intervals; that is, OPT incurs a cost at time t_1 and t_m, but does not incur a cost for any t where $t_1 < t < t_m$. Let $t_1, t_2, \ldots t_m$ be the sequence of request times in the interval t_1 through t_m where item i or j are requested. Without loss of generality, assume item i is requested at time t_1. Then $(r_{t_1}, \ldots r_{t_m}) = (i)^{m-1}j$ or $i(j)^{m-2}i$. Note that we can assume that every interval is longer than one request because the optimal algorithm on a list of length two will incur a cost of at most one on two consecutive requests.

To bound the cost of SPLIT in an interval, we maintain an inductive hypothesis on the distribution of states at the beginning of each interval. Using the inductive hypothesis, we bound the cost of SPLIT and then prove that the inductive hypothesis still holds at the beginning of the next interval. The inductive hypothesis for an interval depends on which item is requested first in the interval. If item i is the first request in the interval, then the inductive hypothesis is $X_{ji}(t_1)/4 + Y_{ji}(t_1) \leq 7/16$. If item j is the first request in the interval, then the inductive hypothesis is $X_{ij}(t_1)/4 + Y_{ij}(t_1) \leq 7/16$.

Case 1: $(r_{t_1}, \ldots, r_{t_m}) = (i)^{m-1}j$. $m > 2$.
If $X_{ji}(t_1)/4 + Y_{ji}(t_1) \leq 7/16$, then

$$\begin{aligned} \sum_{k=1}^{m-1} P_{ji}(t_k) &\leq 31/16 \text{ and} \\ X_{ij}(t_m)/4 + Y_{ij}(t_m) &\leq 7/16 \end{aligned}$$

It is sufficient to prove that

$$P_{ji}(t_1) + P_{ji}(t_2) \leq 31/16$$

because $P_{ji}(t_k) = 0$ for $k > 2$. It is also sufficient to prove that

$$X_{ij}(t_3)/4 + Y_{ij}(t_3) \leq 7/16$$

because $X_{ij}(t_k)/4 + Y_{ij}(t_k)$ decreases as k increases for $k \geq 3$. The proof for case 1 then follows from the inductive hypothesis and the transition probabilities.

Case 2: $(r_{t_1}, \ldots r_{t_m}) = i(j)^{m-2}i$. $m > 2$.
If $X_{ji}(t_1)/4 + Y_{ji}(t_1) \leq 7/16$, then

$$\begin{aligned} P_{ji}(t_1) + \sum_{k=2}^{m-1} P_{ij}(t_k) &\leq 31/16 \text{ and} \\ X_{ji}(t_m)/4 + Y_{ji}(t_m) &\leq 7/16 \end{aligned}$$

It is sufficient to prove that

$$P_{ji}(t_1) + P_{ij}(t_2) + P_{ij}(t_3) \leq 31/16$$

because $P_{ij}(t_k) = 0$ for $k \geq 4$. It is also sufficient to prove that

$$X_{ij}(t_3)/4 + Y_{ij}(t_3) \leq 7/16$$

because $X_{ij}(t_k)/4 + Y_{ij}(t_k)$ decreases as k increases for $k \geq 3$. The proof for case 2 then follows from the inductive hypothesis and the transition probabilities.

The stronger bound of $(31 \cdot L + 1)/16(L + 1)$ on the competitive ratio of $SPLIT$ can be obtained by using the cost function used in the proof of Theorem 2.

Note that SPLIT's competitive ratio is at least 1.75 for the $i - 1$ cost function. On a list consisting of two elements i and j, the sequence that repeats the subsequence $(i, j, j, j, \ldots)$ will force SPLIT to have an expected cost arbitrarily close to 1.75 per repetition. The cost of SPLIT will depend on the number of requests to item j in between each request to item i. SPLIT's cost converges to 1.75 as the number of requests to item j increases. An optimal algorithm will only incur a cost of 1 per request to i.

4. Conclusion

The main open question here is to determine the best possible competitive ratio for a randomized on-line algorithm for the list update problem. Theorem 1 settles this question for the deterministic case. It is still unknown whether the best bound can be achieved by dividing the analysis into pairs; do we give away too much in allowing the adversary to service every pair of items optimally? Presumably if this is the case, then it will be important to better understand the optimal off-line algorithm.

Another question is to examine variations of this model that allow items other than the requested item to be moved for free. In a linked list, any item that precedes the requested item in the list could be moved in constant time. How would the results change under this new model?

The question of lookahead is also interesting for the list update problem. How does the performance of an on-line algorithm improve if it is allowed to see some number of requests in advance?

References

[BBKTW] Ben-David, S., Borodin, A., Karp, R., Tárdos, G. ,Wigderson, A. ,'On The Power of On-line Algorithms', *Proc. 22nd Annual ACM Symposium on the Theory of Computing*, 1990, pp.379-388. To appear in *Algorithmica.*

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

[CL] Chrobak, M., Larmore, L., *Personal Communication*, 1990.

[IRWY] Irani, S., Reingold, N.,Westbrook, J., Young, N., *Personal Communication*, 1990.

[KR] Karp, R. Raghavan, P., *Personal Communication*, 1990.

[RW] Reingold,N., Westbrook, J., 'Optimum Off-line Algorithms foe The List Update Problem', *Technical Report*, YALEU/DCS/TR-805, August 1990.

[RW2] Reingold, N., Westbrook, J., 'Randomized Algorithms for the List Update Problem', *Technical Report*, YALEU/DCS/TR-804, June 1990.

[ST] Sleator, D. D., Tarjan, R. E., 'Amortized Efficiency of List Update and Paging Rules',*Communications of the ACM*, 28:202-208, February 1985.