Two Results on the List Update Problem

Sandy Irani\(^1,2\)
TR-90-037
August 2, 1990

Abstract

In this paper we give a randomized on-line algorithm for the list update problem introduced by Sleator and Tarjan in [ST]. Sleator and Tarjan show a deterministic algorithm, Move-to-Front, for the list update problem. Their analysis yields a competitive ratio of \((2L-1)/L\) for lists of length \(L\). No deterministic algorithm can beat \(2L/(L+1)\) on a list of length \(L\) [KR]. We show that Move-to-Front in fact achieves an optimal competitive ratio of \(2L/(L+1)\). This is even the best that any randomized strategy can achieve against an adaptive on-line adversary[RW]. We show a randomized algorithm that achieves a competitive ratio of \((15 \cdot L +1)/8(L + 1)\) against a non-adaptive adversary. This is the first randomized strategy whose competitive factor beats a constant less than 2.

\(^1\)This research was supported by an IBM Graduate Fellowship.
\(^2\)Computer Science Division, U.C. Berkeley, Berkeley, California 94720.
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. This model was first introduced by Sleator and Tarjan in [ST]. 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 the distance from the requested item to the front of 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 to be placed once found. Then the item can be moved to the new spot in constant time. Sleator and Tarjan also have paid exchanges in their model, where any item may also be moved anywhere in the list, but the cost of moving the item is the distance it is moved. The results in this paper also hold when paid exchanges are possible, but we will only provide proofs for the model without paid exchanges.

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 optimal off-line algorithm can be determined using dynamic programming in $O(mL \cdot L!)$ steps where $L$ is the number of items and $m$ is the number of requests. We call the optimal off-line algorithm $OPT$.

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

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

Notice that the constant $\beta$ can depend on the number of items but not on the request sequence. $\alpha$ is an upper bound on the competitive ratio for algorithm $A$.

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 $\sigma$. They show that $\forall \sigma$, $\quad MTF(\sigma) \leq 2 \cdot OPT(\sigma)$. 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 $L$, $\exists \sigma$ such that $\quad A(\sigma) \geq 2L/(L + 1) \cdot OPT(\sigma)$.

To see this fact, let the adversary request 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 $L$. 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:** $\forall \sigma$, $\quad MTF(\sigma) \leq \frac{2L}{L + 1} \cdot OPT(\sigma)$.

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. An adaptive on-line adversary can see the algorithm’s random choices in choosing the request sequence, but must also service the requests on-line. We show a randomized strategy $SPLIT$ and prove the following theorem:

**Theorem 2:** Against an oblivious adversary, $\forall \sigma$, $\quad E[SPLIT(\sigma)] \leq \frac{15}{8} \cdot OPT(\sigma)$

The proof for Theorem 2 is in section 3.

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 [RW].

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 $\sigma$.

There is a lower bound of 9/8 for any algorithm against a non-adaptive 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],[RW]. The difficulty in extending this technique to longer lists is that the only known optimal off-line algorithm is dynamic programming which maintains $L!$ states and requires $L \cdot L!$ time per request.

Subsequent to the work presented here, Reingold and Westbrook have devised a randomized on-line algorithm that achieves 1.75 [RW]. 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 $\sigma$ 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.
Let

\[ \sigma_i(t) = \begin{cases} 
1 & \text{if } r_t = i \\
0 & \text{otherwise}
\end{cases} \]

\[ MTF_{ij}(t) = \begin{cases} 
1 + 1/(L - 1) & \text{if } i \text{ appears before } j \text{ in } MTF\text{'s list} \\
1/(L - 1) & \text{if } j \text{ appears before } i \text{ in } MTF\text{'s list}
\end{cases} \]

Similarly for \( OPT_{ij}(t) \).

\[ [\sigma_j(t) \cdot MTF_{ij}(t)] = 1 + 1/(L - 1) \]

if \( MTF \) has to step over item \( i \) to reach a request to \( j \) at time \( t \). Then

\[ MTF(\sigma) = \sum_{t=1}^{\sigma} \sum_{i \geq j \geq n} [\sigma_j(t) \cdot MTF_{ij}(t) + \sigma_i(t) \cdot MTF_{ji}(t)]. \]

We prove that for all \( i \) and \( j \),

\[ \sum_{i=1}^{\sigma} [\sigma_j(t) \cdot MTF_{ij}(t) + \sigma_i(t) \cdot MTF_{ji}(t)] \]

\[ \leq 2L/(L + 1) \sum_{t=1}^{\sigma} [\sigma_j(t) \cdot OPT_{ij}(t) + \sigma_i(t) \cdot OPT_{ji}(t)] \]

Thus we have reduced the problem to lists of length two. The theorem follows from the following claim:

Claim: If \( t_1 \) and \( t_2 \) are such that

\[ [\sigma_j(t_1) \cdot MTF_{ij}(t_1) + \sigma_i(t_1) \cdot MTF_{ji}(t_1)] \]

\[ = [\sigma_j(t_2) \cdot MTF_{ij}(t_2) + \sigma_i(t_2) \cdot MTF_{ji}(t_2)] = 1 + 1/(L - 1) \]

and

\[ [\sigma_j(t_1) \cdot OPT_{ij}(t_1) + \sigma_i(t_1) \cdot OPT_{ji}(t_1)] \]

\[ = [\sigma_j(t_2) \cdot OPT_{ij}(t_2) + \sigma_i(t_2) \cdot OPT_{ji}(t_2)] = 1/(L - 1) \]

then there is a \( t \) such that \( t_1 < t < t_2 \)

\[ [\sigma_j(t) \cdot OPT_{ij}(t) + \sigma_i(t) \cdot OPT_{ji}(t)] \]
\[ = [\sigma_j(t) \cdot MTF_{i_j}(t) + \sigma_i(t) \cdot MTF_{j_i}(t)] = 1 + 1/(L - 1) \]

This implies that

\[
MTF(\sigma)/OPT(\sigma) \leq \frac{2 + 2/(L - 1)}{1 + 2/(L - 1)} = \frac{2L}{L + 1}.
\]

**Proof of Claim:** Suppose at time \( t_1 \), \( i \) precedes \( j \) in \( MTF \)'s list. This means that \( j \) was the requested item, since \( \sigma_j(t_1) \cdot MTF_{i_j}(t_1) = 1 + 1/(L - 1) \). It also means that \( OPT \) has \( j \) before \( i \), because \( \sigma_j(t_1) \cdot OPT_{i_j}(t_1) = 1/(L - 1) \). 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 will maintain a pointer into the list. The pointer for item \( i \) will be called \( i.split \). We maintain the invariant that the pointer always points forward, i.e. \( i.split \) precedes \( i \) in the list. If item \( i \) is requested, with probability \( 1/2 \) it is moved to the front of the list. With probability \( 1/2 \), it is moved just in front of \( i.split \). In either case \( i.split \) is set to point to the first item in the list. If item \( i \) is requested and \( j \neq i \), then \( j.split \) does not change unless \( i \) is inserted just before \( j.split \). In this case \( j.split \) is set to point to \( i \) if \( j \) preceded \( i \) in the list before the request.

The algorithm works as follows:

### Initialization:

For \( i \leftarrow 1 \) to \( L \)

\[
i.split \leftarrow i;
\]

Suppose item \( i \) requested:

With probability \( 1/2 \):

Move item \( i \) to the front
With probability $1/2$:

Insert item $i$ before item $i$.split

If $j$ precedes $i$ in the list, and $i$.split = $j$.split, then $j$.split $\leftarrow$ $i$.

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

**Proof of Theorem 2:** For the sake of analysis, we introduce a new cost function. With the new cost function, the cost of accessing the $i$th item in the list is $i - 1$ instead of $i$. This just subtracts 1 from the cost of every request for both the on-line and off-line algorithms. Any upper bound that we obtain on the competitive ratio of an algorithm with the $i - 1$ cost function is also an upper bound on the competitive ratio with the $i$ cost function. From now on, we will assume the $i - 1$ cost function.

Fix a sequence $\sigma = (r_1, r_2, \ldots)$. We use the symbol $\prec$ to indicate precedence in the list, i.e. $i \prec j$ if $i$ precedes $j$ in the list. $\sigma_i(t)$ is defined as in the previous section. To determine the cost of algorithm SPLIT on $\sigma$, we examine the probability distribution of states at each point in the request sequence. Define the following functions:

\[ W_{ij}(t) = Pr[i \prec j$.split \quad \text{and} \quad i$.split is first at time $t] \]

\[ X_{ij}(t) = Pr[i \prec j$.split \quad \text{and} \quad i$.split is not first at time $t] \]

\[ Y_{ij}(t) = Pr[i$.split $\leq j$.split $\prec i$ $\prec j$ at time $t] \]

\[ Z_{ij}(t) = Pr[j$.split $\prec i$.split $\leq i$ $\prec j$ at time $t] \]

Let $P_{ij}(t)$ be the probability that at time $t$, item $i$ precedes item $j$ in the list. Then, $P_{ij}(t) = W_{ij}(t) + X_{ij}(t) + Y_{ij}(t) + Z_{ij}(t)$. Then the expected cost of servicing the sequence $\sigma$ is

\[ \text{SPLIT}(\sigma) = \sum_{i=1}^{n} \sum_{i=1}^{n} P_{ri}(t) \]

7
\[
\sum_{t=1}^{n} \sum_{i \geq j \geq n} [\sigma_j(t) \cdot P_{ij}(t) + \sigma_i(t) \cdot P_{ji}(t)]
\]

The transition probabilities on distribution of states for \textit{SPLIT} are well defined by the algorithm and they are as follows:

If \( r_t \neq i, j \) then
\[
W_{ij}(t) + X_{ij}(t) = W_{ij}(t + 1) + X_{ij}(t + 1)
\]
\[
Y_{ij}(t) = Y_{ij}(t + 1)
\]
\[
Z_{ij}(t) = Z_{ij}(t + 1)
\]

If \( r_t = i \) then
\[
Y_{ij}(t + 1) \leftarrow 1/2[Y_{ji}(t) + Z_{ij}(t)]
\]
\[
Z_{ji}(t + 1) \leftarrow 1/2 \cdot X_{ji}(t)
\]
\[
Y_{ji}(t + 1) \leftarrow 1/2 \cdot W_{ji}(t)
\]
\[
W_{ij}(t + 1) = 1 - Y_{ij}(t + 1) - Y_{ji}(t + 1) - Z_{ji}(t + 1)
\]

Now let \( OPT \) be the optimal off-line algorithm that services \( \sigma \). Let
\[
OPT_{ij}(t) = \begin{cases} 
1 & \text{if item } i \text{ appears before item } j \text{ in } OPT\text{'s list at time } t. \\
0 & \text{otherwise}
\end{cases}
\]

Then if \( OPT(\sigma) \) is the cost of algorithm \( OPT \) on the request sequence \( \sigma \),
\[
OPT(\sigma) = \sum_{t=1}^{n} \sum_{i = 1}^{n} OPT_{ir}(t)
\]
\[
= 1/2 \sum_{t=1}^{n} \sum_{i \geq j \geq n} [\sigma_j(t) \cdot OPT_{ij}(t) + \sigma_i(t) \cdot OPT_{ji}(t)]
\]

Claim: For any \( i \) and \( j \),
\[
15/8 \sum_{t=1}^{n} [\sigma_j(t) \cdot OPT_{ij}(t) + \sigma_i(t) \cdot OPT_{ji}(t)] \geq \\
\sum_{t=1}^{n} [\sigma_i(t) \cdot P_{ij}(t) + \sigma_j(t) \cdot P_{ji}(t)]
\]

Proof:
We prove something slightly stronger: let \( \sigma_{ij} \) be the sequence obtained by deleting from \( \sigma \) all requests to items other than \( i \) or \( j \). We prove that
\[ 15/8 \cdot OPT(\sigma_{ij}) \geq \sum_{i=1}^{m} \sigma_i(t) \cdot P_{ij}(t) + \sigma_j(t) \cdot P_{ij}(t) \]

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 \( i \).

We divide the sequence into intervals, such that a new interval starts every time \( OPT \) incurs a cost of one on the list with \( i \) and \( j \). Recall that since we are using the \( i - 1 \) cost function, the cost of accessing the first item is 0 and the cost of accessing the second item is 1.

So let \( t_1 \) and \( t_m \) be the first request in two consecutive intervals:

\[
\begin{align*}
\sigma_j(t_1) \cdot OPT_{ij}(t_1) + \sigma_i(t_1) \cdot OPT_{ji}(t_1) &= 1 \\
\sigma_j(t_m) \cdot OPT_{ij}(t_m) + \sigma_i(t_m) \cdot OPT_{ji}(t_m) &= 1 \\
\sigma_j(t) \cdot OPT_{ij}(t) + \sigma_i(t) \cdot OPT_{ji}(t) &= 0 & \text{for } t_1 < t < t_m
\end{align*}
\]

Let \( t_1, \ldots, t_m \) 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 \( \sigma_i(t_1) = 1 \).

Then \((r_{i_1}, \ldots, r_{i_m}) = (i)^{m-1} j \) or \( (i)^{m-2} i \). Note that we can assume that every interval is longer than one request because the optimal algorithm on a list of length two will never incur a cost of one on two consecutive requests.

**Case 1:** \((r_{i_1}, \ldots, r_{i_m}) = (i)^{m-1} j \). \( m > 1 \).

We prove that if \( (X_{ji}(t_1) + W_{ji}(t_1))/4 + 3Y_{ji}(t_1)/4 + Z_{ji}(t_1) \leq 3/8 \) then

\[ \sum_{k=1}^{m-1} P_{ji}(t_k) \leq 15/8 \]

and

\[ (X_{ij}(t_m) + W_{ij}(t_m))/4 + 3Y_{ij}(t_m)/4 + Z_{ij}(t_m) \leq 3/8 \]

**Proof:** It is sufficient to prove that

\[ P_{ji}(t_1) + P_{ji}(t_2) \leq 15/8 \]
because $P_{ji}(t_k) = 0$ for $k > 2$.

It is also sufficient to prove that

$$(X_{ij}(t_3) + W_{ij}(t_3))/4 + 3Y_{ij}(t_3)/4 + Z_{ij}(t_3) \leq 3/8$$

because $(X_{ij}(t_k) + W_{ij}(t_k))/4 + 3Y_{ij}(t_k)/4 + Z_{ij}(t_k)$ decrease as $k$ increases for $k \geq 3$.

The proof for case 1 then follows from the inductive hypothesis and the transition probabilities.

**Case 2:** $(r_1, \ldots , r_m) = i(j)^{m-1}i$. $m \geq 1$.

We prove that if $(X_{ji}(t_1) + W_{ji}(t_1))/4 + 3Y_{ji}(t_1)/4 + Z_{ji}(t_1) \leq 3/8$ then

$$P_{ji}(t_1) + \sum_{k=2}^{m-1} P_{ij}(t_k) \leq 15/8$$

and

$$(X_{ji}(t_m) + W_{ji}(t_m))/4 + 3Y_{ji}(t_m)/4 + Z_{ji}(t_m) \leq 3/8$$

**Proof:** It is sufficient to prove that

$$P_{ji}(t_1) + P_{ij}(t_2) + P_{ij}(t_3) \leq 15/8$$

because $P_{ij}(t_k) = 0$ for $k \geq 4$.

It is also sufficient to prove that

$$(X_{ji}(t_3) + W_{ji}(t_3))/4 + 3Y_{ji}(t_3)/4 + Z_{ji}(t_3) \leq 3/8$$

because $(X_{ij}(t_k) + W_{ij}(t_k))/4 + 3Y_{ij}(t_k)/4 + Z_{ij}(t_k)$ decrease as $k$ increases for $k \geq 3$.

The proof for case 2 then follows from the inductive hypothesis and the transition probabilities.

The competitive ratio can be improved slightly by changing the probability that an item moves to the front to $1/3$. In this case the competitive ratio is bounded by 1.852.
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. So far the only technique is brute-force dynamic programming.

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


