CS294-1 On-line Computation & Network Algorithms **Spring 1997**

Lecture 24: April 29

Lecturer: Yair Bartal *Scribe: Zhendong Su*

This lecture is a guest lecture by Susanne Albers on the List Update Problem.

24.1 The List Update Problem

We have an unsorted list L, for example

$$L : y \to w \to z \to x \to v \to u$$

As in other on-line problems, we have a sequence of requests to the items in the list, say $\sigma = \sigma_1\sigma_2\ldots\sigma_m$. There are three operations that need to be supported:

- access (x)
- insert (x)
- delete (x)

The standard cost model for this problem is:

- Access to the i-th item costs i.
- Deletion of the i-th item costs i.
- Insertion of a new item costs $n+1$, where n is the length of the list before this insertion.
- Paid exchange costs 1.

Here are some example applications of the problem:

- Data Structures with small data sets and Computational Geometry
- Data Compression

24.2 Deterministic Algorithms for the Problem

Below are a few deterministic algorithms for this problem:

- MOVE-TO-FRONT (MTF): move requested item to the front of the list
- TRANSPOSE: move requested item one position ahead
- FREQUENCY-COUNT (FC): maintain a counter c_x, for each item x. Whenever x is accessed, increase its counter by 1. The items are sorted by their counters in decreasing order.

The following theorem is due to Sleator and Tarjan [ST85].

Theorem 24.1 *MOVE-TO-FRONT is 2-competitive.*

Proof: Consider a request sequence $\sigma = \sigma_1\sigma_2\ldots\sigma_m$. We want to show for all t,

$$C_{MTF}(\sigma_t) + \phi(t) - \phi(t-1) \leq 2 \cdot C_{OPT}(\sigma_t),$$

where ϕ is a potential function. Let's define the potential function ϕ as

$$\phi = \# \text{ of inversions in MTF's list w.r.t. OPT's list.}$$

An inversion is an unordered pair $\{x, y\}$ such that $x < y$ in OPT's list, but $y < x$ in MTF's list, where $x < y$ means that in the list x proceeds y. Consider $\sigma_t = x$. Let

$$k = \# \text{ of items that proceed } x \text{ in MTF's and OPT's lists}$$

and

$$l = \# \text{ of items that proceed } x \text{ only in MTF's list.}$$

We have that

$$\begin{array}{rcl} C_{OPT} & \geq & k+1, \\ C_{MTF} & = & k+l+1, and \\ \delta\phi & \leq & -l+k. \end{array}$$

Therefore,

$$\begin{array}{rcl} C_{MTF}(\sigma_t) + \delta\phi & = & k+l+1-l+k \\ & = & 2k+1 \\ & \leq & 2 \cdot C_{OPT}(\sigma_t) \end{array}$$

Hence, MTF is 2-competitive. ∎

Theorem 24.2 *TRANSPOSE and FC are not c-competitive for any constant c.*

Proof: To see that TRANSPOSE is not competitive, consider the following list:

$$x_1x_2\ldots x_{n-1}x_n$$

and the following sequence of requests:

$$(x_nx_{n-1})^+,$$

i.e., only requests the last two items. It's clear that for this sequence TRANSPOSE is not competitive. To see that FC is not competitive, we can also consider a sequence of requests. It's left as an exercise. ∎

Theorem 24.3 *If a deterministic on-line algorithm A is c-competitive, then $c \geq 2$. This implies that MTF achievs the optimal competitive ratio.*

Proof: Assume that A has a list n items. The adversary always asks for the last item of the on-line list. For a sequence σ of m requests, we have

$$C_A(\sigma) = m \cdot n.$$

We'll show that

$$C_{OPT}(\sigma) = m \cdot \frac{n+1}{2}.$$

Consider two algorithms:

- STAT 1: the configuration of the algorithm is always:

$$x_1 \rightarrow x_2 \rightarrow \ldots \rightarrow x_{n-1} \rightarrow x_n.$$

- STAT 2: the configuration of the algorithm is always the reverse of that of STAT 1, i.e.:

$$x_n \rightarrow x_{n-1} \rightarrow \ldots \rightarrow x_2 \rightarrow x_1.$$

For every request, the total cost of STAT 1 and STAT 2 is clearly $n+1$. Thus, the total cost of STAT 1 and STAT 2 on σ is $m \cdot (n+1)$. The preprocessing of getting the lists to the two desired orders taks at most $\frac{n(n+1)}{2}$ exchanges. However, as $m \rightarrow \infty$, $\frac{\frac{n(n+1)}{2}}{mn}$ tends to 0. Thus, we have that MTF is optimal. ∎

24.3 Randomized Algorithm for the Problem

Theorem 24.4 *If a randomized on-line algorithm A is c-competitive against an adaptive on-line adversary, then $c \geq 2$.*

Proof: The same argument for the lower bound in the deterministic case can be applied here to get the lower bound for the randomized case. ∎

How about randomized algorithms against oblivious on-line adversaries? Consider the following algorithm (BIT):
At the beginning of the algorithm, we pick an assignment of 0 or 1 to the items uniformly at random from all the possible 2^n assignements. Every time a request is made to an item x, the bit that is assoicated with x $b(x)$ is flipped and if the bit is flipped from 0 to 1, the item is moved to the first of the list.

The following theorem was proved in [IRSW91].

Theorem 24.5 *BIT is 1.75-competitive.*

Proof: To prove the theorem, we first prove a lemma.

Lemma 24.6 *For every item x and every t, just after the t-th request $b(x)$ is 0 or 1 with equal probability. The value is independent of x's position in the list and independent of bit values of the other items.*

Proof: Initially all 2^n bit configurations are equally likely to be assigned to the n items. $b(x)$ is the sum of its intially value and the number of requests modular 2. So, after the t-th request all 2^n configurations of bits are again equally likely. ■

W.l.o.g., OPT only does paid exchanges. To see this assumption is valid, consider the

Consider a sequence of requests $\sigma = \sigma_1\sigma_2\ldots\sigma_m$. We want to show that for any t,

$$C_{BIT}(\sigma_t) + E(\delta\phi) \leq 1.75 C_{OPT}(\sigma_t),$$

for some potential function ϕ.

We first give a slightly different defintion of inversions. An inversion is an ordered pair items (x, y), such that $x < y$ in OPT's list and $y < x$ in BIT's list. An inversion is of type 1 if $b(x) = 0$ and is of type 2 is $b(x) = 1$.

Define

$$\phi_i = \# \text{ of type } i \text{ inversions}, i = 1, 2.$$

We define the potential function ϕ as

$$\phi = \phi_1 + 2\phi_2.$$

Again, we consider a request to $\sigma_t = x$. Let

$$k = \# \text{ of items that proceeds } x \text{ in BIT's and OPT's lists}$$

and

$$l = \# \text{ of items that proceeds } x \text{ only in BIT's list.}$$

1. change in potential due to inversions that are destroied or changed type. We have two cases:
   - $b(x) = 0$ before request: in this case, l inversions are destroied and all of them are of type 1, i.e., potential drops by l.
   - $b(x) = 1$ before request: in this case, l inversions changed type. Again, potential drops by l.
2. change in potential due to new inversions created. Again, here there are two cases:
   - $b(x) = 0$ before the request: k new inversions can be created. For each inversion, the expected change in potential is $\frac{1}{2} \cdot 1 + \frac{1}{2} \cdot 2 = 1.5$, considering whether $b(y)$ is 0 or 1 for the y that x moved ahead of.
   - $b(x) = 1$ before the request: new new inversions are created.

Putting everything together, we have:

$$\begin{aligned} C_{BIT}(\sigma_t) + E[\delta\phi] &= k + l + 1 - l + \frac{1.5k}{2} \\ &= 1.75k + 1 \\ &\leq 1.75 C_{OPT}(\sigma_t) \end{aligned}$$

If the OPT also has paid exchanges, then the cost of OPT is 1. What's the expected change in potential?

$$E(\delta\phi) = \frac{1}{2} \cdot 1 + \frac{1}{2} \cdot 2 = 1.5.$$

Therefore, we have shown that BIT is 1.75-competitive against oblivious on-line adversaries. ■

We can generalize the BIT algorithm to get the COUNTER algorithm. Here is the description of COUNTER(s, S) where s is non-negative integer and $S \subseteq \{0, 1, \ldots, s-1\}$. For every item x of the list L, the algorithm maintains a mod s counter $c(x)$. On each request to item x, decrement $c(x)$ by 1 and move x to the front of the list if $c(x) \in S$.

Let P_j be the probability that an item is next moved to the front after j accesses, for $j = 1, 2, \ldots, s-1$. We can show that COUNTER(s,S) is $max\{1 + p_1 \sum_{j=1}^{s-1} jP_j, \sum_{j=1}^{s-1} jP_j\}$.

We can show the algorithm that execute COUNTER(2,{0}) with probability $\frac{4}{7}$ and execute COUNTER(3,{0}) with probability $\frac{3}{7}$ is 12/7 competitive against oblivous on-line adversaries. This is best result that we can get using combinations of the COUNTER algorithms.

STATUS: The current best upper bound is 1.6 [Albers95] and the current best lower bound is 1.5 [Teia93].

24.4 The Current Best Upper Bound of 1.6

First we give the TIMESTAMP algorithm. In the TIMESTAMP algorithm, we move the requested item x in front of the first item in the list that was requested at most once since the last request to x.

We then have the following algorithm:
COMBINATION (COMB): with probability $\frac{4}{5}$ serve σ using BIT and with probability $\frac{1}{5}$ serve σ using TIMESTAMP(σ).

Theorem 24.7 *COMB is 1.6 competitive.*

Proof: Let $\sigma = \sigma_1\sigma_2\ldots\sigma_m$ be a sequence of requests. We'll analyze the cost in the $(i-1)$-cost model, i.e., an access to the i-th item in the list costs $i-1$ rather than i.

It's clear that

$$C_A^{i-1}(\sigma) \leq c \cdot C_{OPT}^{i-1}(\sigma)$$

$$\begin{aligned} C_A^{i-1}(\sigma) + m &\leq c \cdot C_{OPT}^{i-1} + m \\ C_A^i(\sigma) &\leq c \cdot C_{OPT}^i(\sigma), \end{aligned}$$

where $C_A^{i-1}(\sigma)$ and $C_{OPT}^{i-1}(\sigma)$ are the costs of algorithm A and OPT in the $i-1$ cost model, and $C_A^i(\sigma)$ and $C_{OPT}^i(\sigma)$ are the costs of algorithm A and OPT in the i cost model.

We'll reduce the analysis to the case where

- that we have a list of only two items;
- requests are only made to these two items.

For an algorithm A, an item $x \in L$, and t, define

$$C_A(t,x) = \begin{cases} 1 & \text{if } x \text{ proceeds } y = \sigma_t \text{ in A's list} \\ 0 & \text{otherwise} \end{cases}$$

Notice that $C_A(t,x)$ is the cost incurred by x on σ_t.

Let $A \in \{BIT, TS(\sigma)\}$. We have

$$\begin{aligned} C_A(\sigma) &= \sum_{t=1}^{m} \sum_{x \in L} C_A(t,x) \\ &= \sum_{x \in L} \sum_{t=1}^{m} C_A(t,x) \\ &= \sum_{x \in L} \sum_{y \in L} \sum_{t=1 and \sigma_t = y}^{m} C_A(t,x) \\ &= \sum_{\{x,y\} and x \neq y} \left(\sum_{t=1 and \sigma_t = y}^{m} C_A(t,x) + \sum_{t=1 and \sigma_t = x}^{m} C_A(t,y) \right). \end{aligned}$$

Also we have

$$C_{OPT}(\sigma) = \sum_{\{x,y\} and x \neq y} \left(\sum_{t=1 and \sigma_t = y}^{m} C_{OPT}(t,x) + \sum_{t=1 and \sigma_t = x}^{m} C_{OPT}(t,y) + P(x,y) \right),$$

where $P(x,y)$ = total # of paid exchanges done by OPT to move x in front of y or y in front of x while processing σ.

We define another request sequence from σ. The new request sequence is denoted by σ_{xy} and is obtained from σ by deleting all requests in σ that are neither to x nor to y.

$C_A(\sigma_{xy})$ is the cost paid by A to serve σ_{xy} on the two item list.

For $A \in \{BIT, TS(\sigma)\}$, we have

$$C_A(\sigma_{xy}) = \sum_{t=1, \sigma_t = y}^{m} C_A(t,x) + \sum_{t=1, \sigma_t = x}^{m} C_A(t,y).$$

This is clearly true for. As for $TS(\sigma)$, we have the following lemma:

Lemma 24.8 *In $TS(\sigma)$, x proceeds y iff the majority of the last three requests made either to x or y were to x.*

Proof: An easy induction on the number of requests processed so far. ■

For OPT, we have

$$C_{OPT}(\sigma_{xy}) \leq \sum_{t=1,\sigma_t=y}^{m} C_{OPT}(t,x) + \sum_{t=1,\sigma_t=x}^{m} C_{OPT}(t,y).$$

Now, we want to show that for all pairs $\{x, y\}$, the following holds:

$$\frac{4}{5}E[BIT(\sigma_{xy})] + \frac{1}{5}C_{TS}(\sigma_{xy}) \leq 1.6C_{OPT}(\sigma_{xy}).$$

We split σ_{xy} into phases. A phase ends if there have been two consecutive requests to the same item and the next request is to a different item.

Next, we classify the phases:

Type	BIT	TS	OPT
x^k or y^k, for some $k \geq 2$	1.5	2	1
$(xy)^k x^l$ or $(yx)^k y^l$, for some $k \geq 1$ and $l \geq 2$	$1.5k{+}1$	$2k$	$k+1$
$(xy)^k y^l$ or $(yx)^k x^l$ for some $k \geq 1$ and $l \geq 1$	$1.5k + \frac{1}{4}$	$2k-1$	k

We can then easily check that

$$\frac{4}{5}E[BIT(\sigma_{xy})] + \frac{1}{5}C_{TS}(\sigma_{xy}) \leq 1.6C_{OPT}(\sigma_{xy}).$$

This concludes our proof of the 1.6 competitiveness of COMB. ■

References

[Albers95] S. Albers. Improved Randomized On-line Algorithms for the List Update Problem. In *Proc.of the 6th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 412–419, January 1995.

[ASW95] S. Albers and B. von Stengel and R. Werchner. A Combined BIT and TIMESTAMP Algorithm for the List Update Problem. In *Information Processing Letters*, vol. 56, pages 135–139, 1995.

[IRSW91] S. Irani, N. Reingold, D. Sleator and J. Westbrook. Randomized Competitive Algorithms for the List Update Problem. In *Proc. 2nd ACM-SIAM Symposium on Discrete Algorithms*, pages 251–260, January 1991.

[ST85] D.D. Sleator and R. E. Tarjan. Amortized Efficiency of List Update and Paging Rules. In *Communications of the ACM*, vol. 28, pages 202–208, 1985.

[Teia93] B. Teia. A Lower Bound for Randomized List Update Algorithms. In *Information Processing Letters*, vol. 47, pages 5–9, 1993.