Algorithmica (1998) 21: 312–329

Algorithmica

Average Case Analyses of List Update Algorithms, with Applications to Data Compression[1]

S. Albers[2] and M. Mitzenmacher[3]

Abstract. We study the performance of the Timestamp (0) (TS(0)) algorithm for self-organizing sequential search on discrete memoryless sources. We demonstrate that TS(0) is better than Move-to-front on such sources, and determine performance ratios for TS(0) against the optimal off-line and static adversaries in this situation. Previous work on such sources compared on-line algorithms only with static adversaries. One practical motivation for our work is the use of the Move-to-front heuristic in various compression algorithms. Our theoretical results suggest that in many cases using TS(0) in place of Move-to-front in schemes that use the latter should improve compression. Tests using implementations on a standard corpus of test documents demonstrate that TS(0) leads to improved compression.

Key Words. On-line algorithms, Competitive analysis, List update problem, Probability distribution, Data compression, Entropy.

1. Introduction. We study deterministic on-line algorithms for self-organizing sequential search. Consider a set of n items $x_1, x_2, \ldots, x_n$ that are stored in an unsorted linear linked list. At any instant of time, an algorithm for maintaining this list is presented with a *request* that specifies one of the n items. The algorithm must serve this request by *accessing* the requested item. That is, the algorithm has to start at the front of the list and search linearly through the items until the desired item is found. Serving a request to the ith item in the list incurs a cost of i. Immediately after a request, the requested item may be moved at no extra cost to any position closer to the front of the list; this can lower the cost of subsequent requests. At any time two adjacent items in the list may be exchanged at a cost of 1; these moves are called *paid exchanges*. The goal is to serve a *sequence of requests* so that the total cost incurred on that sequence is as small as possible. A list update algorithm typically works *on-line*, i.e., when serving the present request, the algorithm has no knowledge of future requests.

Early work on the list update problem assumes that a request sequence is generated by a probability distribution $\vec{p} = (p_1, p_2, \ldots, p_n)$. A request to item x_i occurs with probability p_i; the requests are generated independently. The following on-line algorithms have been investigated extensively:

- **Move-to-front (MTF):** Move the requested item to the front of the list.

[1] A portion of Albers' work was done while the author was at the International Computer Science Institute, Berkeley. A substantial portion of Mitzenmacher's research was done while at the Computer Science Department, UC Berkeley, with funding from the ONR and National Science Foundation Grant No. CCR-9505448.
[2] Max-Planck-Institut für Informatik, Im Stadtwald, 66123 Saarbrücken, Germany. albers@mpi-sb.mpg.de.
[3] Digital Equipment Corporation, Systems Research Center, 130 Lytton Ave, Palo Alto, CA 94301, USA. michaelm@pa.dec.com.

Received December 3, 1995; revised April 5, 1997. Communicated by M. Y. Kao.

- **Transpose (T):** Exchange the requested item with the immediately preceding item in the list.
- **Frequency count (FC):** Maintain a frequency count for each item in the list. Whenever an item is requested, increase its count by 1. Maintain the list so that the items always occur in nonincreasing order by frequency count.

In this paper we again investigate the list update problem under the assumption that a request sequence is generated by a probability distribution $\vec{p} = (p_1, p_2, \ldots, p_n)$; that is, it is generated by a discrete memoryless source. We note that this assumption is a suitable first approximation in many applications, especially within specific request intervals. For example, if the request sequence consists of alphanumeric characters from the concatenation of a set of files, modeling each file by a different probability distribution depending on its type provides a first-order approximation of behavior over request intervals corresponding to each file.

Although our assumption of a memoryless source is only a first approximation for real applications, our analysis provides a great deal of insight into how list update algorithms perform. Moreover, our techniques can be applied to higher-order models as well. For instance, if the sequence is generated by a Markov chain, so that the probability of a letter appearing depends only on the previous letter, generalizing our techniques is straightforward.

Our work is motivated by the goal to present a universal algorithm that achieves a good competitive ratio (in the Sleator and Tarjan model, to be presented) but also performs especially well when requests are generated by distributions. Previous results have shown that MTF is such an algorithm, whereas algorithms T and FC are not. More specifically, MTF achieves an optimal competitive ratio of 2 and has a good behavior on probability distributions. Algorithms T and FC have an even better performance on distributions but do not achieve a constant competitive ratio. Our main contribution is to show that there is an algorithm that has an even better overall performance than MTF. The algorithm we analyze belongs to the Timestamp(p) family of algorithms [1] that were introduced in the context of randomized on-line algorithms and are defined for any real number $p \in [0, 1]$. For $p = 0$, the algorithm is deterministic and can be formulated as follows:

- **Algorithm TS(0):** Insert the requested item, say x, in front of the first item in the list that has been requested at most once since the last request to x. If x has not been requested so far, leave the position of x unchanged.

As an example, consider a list of six items being in the order L: $x_3 \to x_2 \to x_4 \to x_6 \to x_1 \to x_5$. Suppose that algorithm TS(0) has to serve the second request to x_1 in the request sequence $\sigma = \ldots x_1, x_2, x_2, x_4, x_3, x_3, x_1$. Items x_4 and x_6 were requested at most once since the last request to x_1, whereas x_2 and x_3 were both requested twice. Thus, TS(0) will insert x_1 immediately in front of x_4 in the list.

In [1] it was shown that TS(0) achieves a competitive ratio of 2 on any request sequence, as does MTF [17]. Here we demonstrate that TS(0) performs better on distributions, both by developing a formula for the expected cost per request, and by comparing TS(0) with the optimal static and dynamic off-line algorithms.

Since our results show that TS(0) performs better than MTF on distributions, we consider applying the algorithm in the context of data compression, where MTF has been used to develop a locally adaptive data compression scheme [4]. Here we prove that, for all distributions $\vec{p} = (p_1, p_2, \ldots, p_n)$, the expected number of bits needed by a TS(0)-based encoding scheme to encode one symbol is linear in the entropy of the source. Our implementations also demonstrate that in practice TS(0)-based schemes can achieve better compression than MTF schemes.

1.1. *Comparison with Previous Work.* We briefly review the main results in the model where a request sequence is generated by a probability distribution. The performances of MTF, T, and FC have generally been compared with that of the *optimal static ordering*, which we call STAT. The optimal static ordering first arranges the items x_i in nonincreasing order by probabilities p_i and then serves a request sequence without changing the relative positions of items. For any algorithm A, let $E_A(\vec{p})$ denote the asymptotic expected cost incurred by algorithm A in serving one request in a request sequence generated by the distribution $\vec{p}$. Rivest [15] showed that, for all $\vec{p}$, $E_{\mathrm{FC}}(\vec{p})/E_{\mathrm{STAT}}(\vec{p}) = 1$. However, algorithm FC has the drawback that it adapts very slowly to changing probability distributions. Chung et al. [6] analyzed the MTF rule and proved $E_{\mathrm{MTF}}(\vec{p})/E_{\mathrm{STAT}}(\vec{p}) \leq \pi/2 \approx 1.5708$ for all $\vec{p}$. This bound is tight because Gonnet et al. [8] showed that one can find $\vec{p}_0$ with $E_{\mathrm{MTF}}(\vec{p}_0)/E_{\mathrm{STAT}}(\vec{p}_0) \geq \alpha$ for any α arbitrarily close to $\pi/2$.

More recent research on the list update problem was inspired by Sleator and Tarjan [17] who suggested comparing the performance of an on-line algorithm with that of an *optimal off-line* algorithm. An optimal off-line algorithm knows the entire request sequence in advance and can serve it with minimum cost. An on-line algorithm A is called c-competitive if, for all request sequences, the cost incurred by A is at most c times the cost incurred by the optimal off-line algorithm. Sleator and Tarjan proved that the MTF algorithm is 2-competitive. They also showed that algorithms T and FC are not c-competitive for any constant c that is independent of the list size n. The competitive ratio of 2 is the best ratio that a deterministic on-line algorithm for the list update problem can achieve [13].

In classical data compression theory, it is often assumed that a discrete memoryless source generates a string S to be compressed. The string S consists of *symbols*, where each symbol is an element in the alphabet $\Sigma = \{x_1, x_2, \ldots, x_n\}$. Each symbol is equal to x_i with probability p_i. Bentley et al. [4] showed how any list update algorithm can be used to develop a data compression scheme. The idea is to convert the string S of symbols into a string I of integers. Whenever the symbol x_i has to be compressed, an encoder looks up the current position of x_i in a linear list of symbols it maintains, outputs this position, and updates the list. A decoder that receives the string I can recover the original message by looking up in its own linear list, for each integer j it reads, the symbol that is currently stored at position j. The decoder also updates its list. Clearly, when the string I of integers is actually transmitted, each integer in the string should be coded again using a variable length prefix code. Bentley et al. showed that, for all $\vec{p} = (p_1, p_2, \ldots, p_n)$, the expected number of bits needed to encode one symbol in a string S using the MTF rule is linear in the entropy of the source. By Shannon's source coding theorem, this is optimal, up to a constant factor. Bentley et al. also showed that, for any string S, the average number of bits needed by MTF to encode one symbol in S is linear in the "empirical entropy" of the string.

Recently, Grinberg et al. [9] proposed a modification of the MTF encoding, which they call *MTF encoding with secondary lists*. They implemented the new compression scheme but their simulations do not show an explicit comparison between MTF and MTF with secondary lists. Also recently, a fast and efficient compression scheme that uses MTF encoding as a subroutine has been developed [5]. This algorithm appears competitive with those used in standard compression tools, and thus the examination of alternatives to MTF may lead to better practical compression algorithms.

1.2. *Our Results*. An important aspect in our work is that we compare the expected cost incurred by an on-line algorithm with that of the optimal off-line algorithm, which we denote by OPT. We recall that OPT may rearrange the list after each request and is not forced to serve a request sequence using the optimal static ordering.

First we develop a formula for TS(0)'s expected cost on a distribution $\vec{p} = (p_1, p_2, \ldots, p_n)$. This formula implies that if we have a distribution $\vec{p}$ with $p_i = 1/n$, for all i, then MTF and TS(0) have the same expected cost. On all other distributions, TS(0) has a smaller expected cost. Then we compare TS(0) with the optimal off-line algorithm OPT and show $E_{\mathrm{TS}}(\vec{p})/E_{\mathrm{OPT}}(\vec{p}) \leq 1.5$ for all distributions $\vec{p}$. This is a performance MTF cannot match because $E_{\mathrm{MTF}}(\vec{p}_0)/E_{\mathrm{STAT}}(\vec{p}_0) > 1.57$ for some $\vec{p}_0$, and when MTF is compared with OPT the ratio might even be worse. We also show that, for any $\vec{p}$ and any $\varepsilon > 0$, the cost of TS(0) is at most $1.5 + \varepsilon$ times the cost of OPT with high probability on sufficiently long sequences. It is worthwhile noticing that 1.5 is the best lower bound currently known on the competitiveness that can be achieved by randomized list update algorithms against an oblivious adversary [18]. Thus, the performance ratio of TS(0) on distributions is at least as good as the performance ratio of randomized algorithms on any input. Finally, we evaluate TS(0) against the optimal static ordering and show, for all $\vec{p}$, $E_{\mathrm{TS}}(\vec{p})/E_{\mathrm{STAT}}(\vec{p}) \leq 1.34$.

Given these results, we examine the potential of TS(0) in compression algorithms. As previously mentioned, we prove that, for all distributions $\vec{p} = (p_1, p_2, \ldots, p_n)$, the expected number of bits needed by a TS(0)-based encoding scheme to encode one symbol is linear in the entropy of the source. Our upper bounds are slightly better than similar upper bounds for MTF encoding in this case. We also prove that, for any string S, the average number of bits needed by TS(0) to encode one symbol in S is linear in the empirical entropy of S. Our bound is the same as that given for MTF by [4]. Moreover, we provide evidence that TS(0)-based compression schemes can outperform MTF-based compression schemes in practical situations by implementing these compression algorithms and testing them on the standard Calgary Compression Corpus files [19]. In almost all of our tests, TS(0) encoding achieves a better compression ratio than MTF encoding. We note that further experiments on the performance of compression schemes based on list update algorithms, including TS(0) and MTF, have recently been performed by [2].

2. Analyses for the List Update Problem.

In this section we begin by demonstrating that the asymptotic expected cost of TS(0) is always at most that of MTF on discrete memoryless sources. We then elaborate on this conclusion by determining the competitive ratio of TS(0) on such sources, against both dynamic and static adversaries.

First, we formally define the asymptotic expected cost of a list update algorithm A. Given a probability distribution $\vec{p} = (p_1, p_2, \ldots, p_n)$ and a request sequence generated according to $\vec{p}$, the configuration of A's list follows a Markov chain, with $n!$ states, that converges to a stationary distribution. For any of the $n!$ states S_i, $1 \leq i \leq n!$, let q_i be the stationary probability of S_i. Furthermore, for any item x_j, $1 \leq j \leq n$, let $pos(x_j, S_i)$ be the position of x_j in the list configuration represented by S_i. The asymptotic expected cost $e_A(x_j)$ incurred by algorithm A is serving a request to item x_j in a request sequence generated according to $\vec{p}$ is x_j's expected position in the list, i.e.,

$$e_A(x_j) = \sum_{1 \leq i \leq n!} q_i \; pos(x_j, S_i).$$

The asymptotic expected cost incurred by A in serving a single request in a request sequence generated to $\vec{p} = (p_1, p_2, \ldots, p_n)$ is

$$E_A(\vec{p}) = \sum_{j=1}^{n} p_j e_A(x_j).$$

2.1. *The Expected Cost of* TS(0). To bound the expected cost per request of TS(0), we first prove a useful lemma.

LEMMA 1. *Consider any point in the request sequence where there have been at least three requests for x_i and x_j. Then x_i precedes x_j in the list maintained by* TS(0) *if and only if a majority of the last three requests for x_i and x_j have been for x_i.*

PROOF. We show that the item of the pair $\{x_i, x_j\}$ that was requested most often during the last three requests precedes the other item of the pair $\{x_i, x_j\}$ in TS(0)'s list. Suppose that a majority of the last three requests for x_i and x_j has been to x_i. Item x_i was requested at least twice during these three last requests. First consider the case that the last request for x_i and x_j has been to x_i. Then, at that last request, TS(0) moves x_i at some position in front of x_j, provided that x_i did not precede x_j already, because x_j was requested at most once since the last request to x_i. Now assume that the last request for x_i and x_j has been to x_j, i.e., the last three requests for x_i and x_j are $x_i x_i x_j$. After the second request to x_i, item x_i must precede x_j in TS(0)'s list. Algorithm TS(0) has the important property that if it serves a request to an item x_j, then all items preceding x_j in the list that were requested at most once since the last request to x_j are stored consecutively in front of x_j. In other words, if x_j is inserted in front of the first item in the list that was requested at most once since the last request to x_j, then x_j does not pass an item that was requested at least twice since the last request to x_j. These statements were shown in [1]. Therefore, when TS(0) serves the request to x_j in the subsequence $x_i x_i x_j$, then x_j does not move in front of x_i. □

THEOREM 1. *For any probability distribution $\vec{p} = (p_1, p_2, \ldots, p_n)$,*

(a) *the asymptotic expected cost incurred by* TS(0) *in serving a request to item x_j, $1 \leq j \leq n$, is*

$$e_{\mathrm{TS}}(x_j) = \frac{1}{2} + \sum_{i=1}^{n} \frac{p_i^2 + 3p_i^2 p_j}{(p_i + p_j)^3}.$$

(b)

$$E_{\mathrm{TS}}(\vec{p}) = \sum_{1\le i\le j\le n} \frac{p_i p_j}{p_i + p_j}\left(2 - \frac{(p_i - p_j)^2}{(p_i + p_j)^2}\right).$$

PROOF. (a) The cost $e_{\mathrm{TS}}(x_j)$ is 1 plus the expected number of items x_i, $x_i \neq x_j$, that precede x_j in the list. Let A_{ij} be the event that x_i precedes x_j in the list when TS(0) serves a request to x_j. We compute the asymptotic probability $Prob(A_{ij})$ using Lemma 1.

Lemma 1 implies that the event A_{ij} occurs if and only if the last three requests for x_i and x_j are (B_1) $x_i x_i x_i$; (B_2) $x_i x_i x_j$; $(B3)$ $x_i x_j x_i$; or (B_4) $x_j x_i x_i$. It is not hard to verify that $Prob(B_1) = p_i^3/(p_i + p_j)^3$ and $Prob(B_k) = p_i^2 p_j/(p_i + p_j)^3$, for $k = 2, 3, 4$. Therefore, $Prob(A_{ij}) = (p_i^3 + 3p_i^2 p_j)/(p_i + p_j)^3$ and

$$e_{\mathrm{TS}}(x_j) = 1 + \sum_{\substack{i=1\\ i\neq j}}^{n} Prob(A_{ij}) = 1 + \sum_{\substack{i=1\\ i\neq j}}^{n} \frac{p_i^3 + 3p_i^2 p_j}{(p_i + p_j)^3} = \frac{1}{2} + \sum_{i=1}^{n} \frac{p_i^3 + 3p_i^2 p_j}{(p_i + p_j)^3}.$$

(b) The asymptotic expected cost incurred by TS(0) on one request is

$$\begin{aligned}
E_{\mathrm{TS}}(\vec{p}) &= \sum_{j=1}^{n} p_j e_{\mathrm{TS}}(x_j) = \frac{1}{2} + \sum_{j=1}^{n}\sum_{i=1}^{n} p_j \left(\frac{p_i^3 + 3p_i^2 p_j}{(p_i + p_j)^3}\right) \\
&= \frac{1}{2} + \frac{1}{2}\sum_{j=1}^{n} p_j + \sum_{1\le i<j\le n} \frac{p_i p_j (p_i^2 + 6p_i p_j + p_j^2)}{(p_i + p_j)^3} \\
&= \sum_{j=1}^{n} p_j + \sum_{1\le i<j\le n} \frac{p_i p_j (p_i^2 + 6p_i p_j + p_j^2)}{(p_i + p_j)^3} \\
&= \sum_{1\le i\le j\le n} \frac{p_i p_j}{p_i + p_j}\left(\frac{p_i^2 + 6p_i p_j + p_j^2}{(p_i + p_j)^2}\right) \\
&= \sum_{1\le i\le j\le n} \frac{p_i p_j}{p_i + p_j}\left(2 - \frac{(p_i - p_j)^2}{(p_i + p_j)^2}\right).
\end{aligned}$$

□

COROLLARY 1. *For any probability distribution* $\vec{p} = (p_1, p_2, \ldots, p_n)$,

$$E_{\mathrm{MTF}}(\vec{p}) - E_{\mathrm{TS}}(\vec{p}) = \sum_{1\le i\le j\le n} p_i p_j \frac{(p_i - p_j)^2}{(p_i + p_j)^3}.$$

PROOF. Rivest [14] showed $E_{\mathrm{MTF}}(\vec{p}) = \sum_{1\le i\le n} 2p_i p_j/(p_i + p_j)$. Using part (b) of Theorem 1, the result follows immediately. □

2.2. *Performance against Dynamic Off-Line Algorithms.*

THEOREM 2. *For any probability distribution* $\vec{p} = (p_1, p_2, \ldots, p_n)$,

$$E_{\mathrm{TS}}(\vec{p}) \le \tfrac{3}{2} E_{\mathrm{OPT}}(\vec{p}).$$

PROOF. The analysis consists of two main parts. In the first part we show that, given a fixed request sequence σ, the cost incurred by TS(0) and OPT on σ can be divided into costs that are caused by each unordered pair $\{x, y\}$ of items x and y, $x \neq y$. This technique of evaluating cost by considering pairs of items was also used in [3], [11], and [1]. In the second part of the analysis we show that, for each pair $\{x, y\}$, the asymptotic expected cost paid by TS(0) is at most 1.5 times the asymptotic expected cost incurred by OPT.

In the following we always assume that serving a request to the ith item in the list incurs a cost of $i-1$ rather than i. If $E_{\mathrm{TS}}(\vec{p}) \leq \frac{3}{2} E_{\mathrm{OPT}}(\vec{p})$ holds in this $(i-1)$-*cost model*, then the inequality also holds in the i-cost model. (We note that the reverse statement does not necessarily hold.) Now consider a fixed request sequence $\sigma = \sigma(1), \sigma(2), \ldots, \sigma(m)$ of length m. For an algorithm $A \in \{\mathrm{TS}(0), \mathrm{OPT}\}$, let $C_A(t, x)$ denote the cost caused by item x when A serves request $\sigma(t)$. That is, $C_A(t, x) = 1$ if x precedes the item requested by $\sigma(t)$ in A's list at time t; otherwise $C_A(t, x) = 0$. For any pair $\{x, y\}$ of items $x \neq y$, let $p(x, y)$ be the total number of paid exchanges that A incurs in moving x in front of y or y in front of x. Recall that in a paid exchange, an item, which is not accessed by the present request, is exchanged with the immediately preceding item in the list. The cost incurred by A on σ can be written as

$$C_A(\sigma) = \sum_{\substack{\{x,y\}\\ x\neq y}} \left(\sum_{\substack{t\in[1,m]\\ \sigma(t)=x}} C_A(t, y) + \sum_{\substack{t\in[1,m]\\ \sigma(t)=y}} C_A(t, x) + p(x, y) \right).$$

Now, for any unordered pair $\{x, y\}$ of items x and y, with $x \neq y$, let σ_{xy} be the request sequence that is obtained from σ if we delete all requests that are neither to x nor to y. Let $C_{\mathrm{TS}}(\sigma_{xy})$ be the cost incurred by TS(0) if it serves σ_{xy} on a two item list that consists of only x and y. In [1] it was shown that if TS(0) serves σ on the long list, then the relative position of x and y changes in the same way as if TS(0) serves σ_{xy} on the two item list. Therefore,

$$C_{\mathrm{TS}}(\sigma_{xy}) = \sum_{\substack{t\in[1,m]\\ \sigma(t)=x}} C_{\mathrm{TS}}(t, y) + \sum_{\substack{t\in[1,m]\\ \sigma(t)=y}} C_{\mathrm{TS}}(t, x)$$

and

$$C_{\mathrm{TS}}(\sigma) = \sum_{\substack{\{x,y\}\\ x\neq y}} C_{\mathrm{TS}}(\sigma_{xy}). \tag{1}$$

Note that TS(0) does not incur paid exchanges and hence $p(x, y) = 0$ for all pairs $\{x, y\}$. The optimal cost $C_{\mathrm{OPT}}(\sigma)$ can be written in a similar way:

$$C_{\mathrm{OPT}}(\sigma_{xy}) \leq \sum_{\substack{t\in[1,m]\\ \sigma(t)=x}} C_{\mathrm{OPT}}(t, y) + \sum_{\substack{t\in[1,m]\\ \sigma(t)=y}} C_{\mathrm{OPT}}(t, x) + p(x, y)$$

and

$$C_{\mathrm{OPT}}(\sigma) \geq \sum_{\substack{\{x,y\}\\ x\neq y}} C_{\mathrm{OPT}}(\sigma_{xy}). \tag{2}$$

Here, only inequality signs hold because if OPT serves σ_{xy} on the two items list, then it can always arrange x and y optimally in the list, which might not be possible if OPT serves σ on the entire list. In fact, an optimal off-line algorithm for serving a request sequence σ_{xy} on a two item list can be specified easily: whenever there are at least two consecutive requests to the same item, that item is moved to the front of the list after the first request if the item is not already there. On all other requests, the list remains unchanged. Clearly, such an optimal ordering of all pairs $\{x, y\}$ might not always be possible if OPT serves σ on the long list.

Equation (1) and inequality (2) allow us to compare $C_{\mathrm{TS}}(\sigma)$ and $C_{\mathrm{OPT}}(\sigma)$ by simply comparing $C_{\mathrm{TS}}(\sigma_{xy})$ and $C_{\mathrm{OPT}}(\sigma_{xy})$ for each pair $\{x, y\}$ of items. The same can be trivially shown to hold true for the asymptotic expected costs $E_{\mathrm{TS}}(\vec{p})$ and $E_{\mathrm{OPT}}(\vec{p})$ as well, by taking the expectations of both sides of (1) and (2), and using the linearity of expectations.

Hence, in the following, we concentrate on one particular pair $\{x, y\}$ of items $x \neq y$. For an algorithm, $A \in \{\mathrm{TS}(0), \mathrm{OPT}\}$, let $E_A^{xy}(\vec{p})$ be the asymptotic expected cost incurred on the two item list containing x and y if A serves a single request in σ_{xy}, given that the request sequence σ is generated by $\vec{p}$. We will show that

$$E_{\mathrm{TS}}^{xy}(\vec{p}) \leq \tfrac{3}{2} E_{\mathrm{OPT}}^{xy}(\vec{p}).$$

This proves the theorem.

We first evaluate $E_{\mathrm{TS}}^{xy}(\vec{p})$. TS(0) incurs a cost of 1 on a request in σ_{xy} if x is requested and y precedes x in TS(0)'s list or if y is requested and x precedes y in TS(0)'s list. Otherwise, TS(0) incurs a cost of 0. By Lemma 1, y precedes x in TS(0)'s list if and only if the majority of the last three requests for x and y have been for y, i.e., if the last three requests in σ_{xy} have been (B_1) yyy; (B_2) yyx; (B_3) yxy; or (B_4) xyy. In the probability distribution $\vec{p} = (p_1, p_2, \ldots, p_n)$, let p_x be the probability of a request to x and let p_y be the probability of a request to p_y. Define $p = p_x/(p_x + p_y)$ and $q = (1 - p) = p_y/(p_x + p_y)$. Clearly, p and q are the probabilities that within σ_{xy}, a request is made to x and y, respectively. Thus, the asymptotic probability that y precedes x in TS(0)'s list is $q^3 + 3q^2p$. Similarly, the asymptotic probability that x precedes y in TS(0)'s list is $p^3 + 3p^2q$. Thus

$$E_{\mathrm{TS}}^{xy}(\vec{p}) = p(q^3 + 3q^2p) + q(p^3 + 3p^2q) = pq(p^2 + 6pq + q^2).$$

Next we determine $E_{\mathrm{OPT}}^{xy}(\vec{p})$. Consider OPT's movements when it serves σ_{xy} on the two item list. As explained in the paragraph after inequality (2), we may assume without loss of generality that whenever there are two consecutive requests to the same item, OPT moves that item to the front of the list on the first request. Thus, OPT incurs a cost of 1 on a request in σ_{xy} if x is requested and the last requests in σ_{xy} were of the form $yy(xy)^i$ for some $i \geq 0$, or if y is requested and the last requests in σ_{xy} were of the form $xx(yx)^i$ for some $i \geq 0$. Therefore,

$$\begin{aligned} E_{\mathrm{OPT}}^{xy}(\vec{p}) &= p\sum_{i=0}^{\infty} q^2(pq)^i + q\sum_{i=0}^{\infty} p^2(qp)^i \\ &= p\left(\frac{q^2}{1-pq}\right) + q\left(\frac{p^2}{1-pq}\right) = \frac{pq}{1-pq}. \end{aligned}$$

We conclude that $E_{\mathrm{TS}}^{xy}(\vec{p}) \leq (1-pq)(p^2+6pq+q^2)E_{\mathrm{OPT}}^{xy}(\vec{p})$. The expression $(1-pq)(p^2+6pq+q^2)$ is maximal for $p=q=\frac{1}{2}$ and hence $E_{\mathrm{TS}}^{xy}(\vec{p}) \leq \frac{3}{2}E_{\mathrm{OPT}}^{xy}(\vec{p})$. □

We next show that, for long enough sequences on discrete memoryless sources, TS(0) will be at worst $(1.5+\varepsilon)$-competitive with high probability.

THEOREM 3. *For every distribution $\vec{p}=(p_1, p_2, \ldots, p_n)$ and $\varepsilon > 0$ there exist constants c_1, c_2, and m_0 dependent on $\vec{p}$, n and ε such that, for a request sequence σ of length $m \geq m_0$ generated according to $\vec{p}$,*

$$Prob\{C_{\mathrm{TS}}(\sigma) > (1.5+\varepsilon)C_{\mathrm{OPT}}(\sigma)\} \leq c_1 e^{-c_2 m \varepsilon^2}.$$

PROOF. Again, we begin by considering the performance of the algorithms on a pair of items $\{x, y\}$. We first show that the cost of TS(0) on a random request sequence σ is close to its expectation with high probability. Consider the eight state Markov chain that records the last three requests of σ that are from the sequence σ_{xy}. Label the states (xxx), (xxy), and so on, according to the last three requests. Then by Lemma 1 the cost to TS(0) on σ_{xy} is exactly the number of transitions from (xxx) to (xxy), plus the number of requests from (xxy) to (xyy), and so on. Consider only the number of transitions from state (xxx) to (xxy) over the course of the sequence σ. Let Z be the random number of such transitions on σ. Abusing notation somewhat, let $E[Z]$ be the *asymptotic* expected number of such transitions. We now make use of standard large deviation bounds on finite state Markov chains (see, for example, Lemma 7.6 of [16] or Corollary 4.2 of [12]), which yield Chernoff-like bounds on the deviation of the number of transitions from the asymptotic expected number of transitions. In particular, we have, for each $\varepsilon_1 > 0$ and sufficiently large m,

$$Prob\{|Z - E[Z]| > \varepsilon_1 E[Z]\} \leq c_3 e^{-c_4 m \varepsilon_1^2}$$

for some constants c_3 and c_4 dependent on p_x and p_y. That is, the number of transitions of this type is close to the expected number of transitions with high probability. We now use this argument for every transition type that corresponds to a cost of 1 for TS(0) over all possible pairs of elements. Summing and using linearity of expectations then yields

$$Prob\{C_{\mathrm{TS}}(\sigma) - E[C_{\mathrm{TS}}(\sigma)] > \varepsilon_1 E[C_{\mathrm{TS}}(\sigma)]\} \leq c_5 e^{-c_6 m \varepsilon_1^2},$$

for some constants c_5 and c_6 dependent on $\vec{p}$ and n.

Similarly, we may bound the cost of OPT on σ by bounding the deviation of OPT on the subsequence σ_{xy}. Note that this will provide only a one-sided bound for the cost of OPT on σ by (2), but this is sufficient. As shown in Theorem 2 we may assume OPT incurs a cost of 1 on a request in σ_{xy} if x is requested and the last request in σ_{xy} were of the form $yy(xy)^i$, and similarly if y is requested. Hence to count the cost to OPT on σ_{xy} we may use a six state Markov chain that records the last two items requested from σ_{xy} as well as the last item that was requested twice sequentially. Using large deviation bounds and summing over all necessary transitions over all pairs of items yields that, for each $\varepsilon_2 > 0$,

$$Prob\{C_{\mathrm{OPT}}(\sigma) - E[C_{\mathrm{OPT}}(\sigma)] > \varepsilon_2 E[C_{\mathrm{OPT}}(\sigma)]\} \leq c_7 e^{-c_8 m \varepsilon_2^2}.$$

Choosing $\varepsilon + 1 = \varepsilon_2 = \varepsilon/4$ suffices to yield the theorem. (We also note that, by this construction, the constants c_1, c_2, and m can all be made polynomial in n and $1/p_n$.) $\square$

2.3. *Performance against Static Off-Line Algorithms.* Recall that the expected cost incurred by TS(0) in serving one request in a request sequence generated by $\vec{p} = (p_1, p_2, \ldots, p_n)$ is

$$1 + \sum_i \sum_{j \neq i} \frac{p_i p_j^3 + 3p_j^2 p_i^2}{(p_i + p_j)^3} = \sum_{i,j} \frac{p_i p_j (p_i^2 + 6p_i p_j + p_j^2)}{2(p_i + p_j)^3} + \frac{1}{2}.$$

We can now adapt the techniques presented in [6] to bound the ratio between $E_{\mathrm{TS}}(\vec{p})$ and $E_{\mathrm{STAT}}(\vec{p})$. We assume $p_1 \geq p_2 \geq \cdots \geq p_n$. As $E_{\mathrm{STAT}}(\vec{p}) = \sum_i i p_i = \frac{1}{2} \sum_{i,j} \min(p_i p_j) + \frac{1}{2}$, we have

$$\frac{E_{\mathrm{TS}}(\vec{p})}{E_{\mathrm{STAT}}(\vec{p})} = \frac{\sum_{i,j} (p_i p_j (p_i^2 + 6p_i p_j + p_j^2)/2(p_i + p_j)^3) + \frac{1}{2}}{\frac{1}{2} \sum_{i,j} \min(p_i, p_j) + \frac{1}{2}}$$
$$< \frac{\sum_{i,j} (p_i p_j (p_i^2 + 6p_i p_j + p_j^2)/(p_i + p_j)^3)}{\sum_{i,j} \min(p_i, p_j)}.$$

The result is immediate from the following theorem:

THEOREM 4. *If* $x_i > 0$ $(1 \leq i \leq n)$, *then*

$$\frac{\sum_{i,j} x_i x_j (x_i^2 + 6x_i x_j + x_j^2)/(x_i + x_j)^3}{\sum_{i,j} \min(x_i, x_j)} \leq 1.34.$$

PROOF. We rely on the following lemma, to be proven later, which replaces the ratio of sums by the ratio of integrals:

LEMMA 2. *Suppose* f *is an integrable function on* $(0, \infty)$ *with* $\int_0^\infty f\,dx = 0$. *Let* $G(x, y)$ *be homogeneous of degree* 1, $H(x, y) = \partial^2 G/\partial x\,\partial y$, *and* $H^+(x, y) = \max\{H(x, y), 0\}$. *Then*

$$\frac{\int_0^\infty \int_0^\infty G(x, y) f(x) f(y)\,dx\,dy}{\int_0^\infty \int_0^\infty \min(x, y) f(x) f(y)\,dx\,dy} \leq \int_0^\infty H^+(x, 1) x^{-1/2}\,dx.$$

Let $G(x, y) = xy(x^2 + 6xy + y^2)/(x + y)^3$. Without loss of generality, let $0 < x_1 < x_2 < \cdots < x_n$. Let f_δ be a function such that $f_\delta = 1$ in neighborhoods of length δ around each x_i and 0 otherwise. Then as δ approaches 0,

$$\frac{\sum_{i,j} x_i x_j (x_i^2 + 6x_i x_j + x_j^2)/(x_i + x_j)^3}{\sum_{i,j} \min(x_i, x_j)} = \lim_{\delta \to 0} \frac{\int_0^\infty \int_0^\infty G(x, y) f_\delta(x) f_\delta(y)\,dx\,dy}{\int_0^\infty \int_0^\infty \min(x, y) f_\delta(x) f_\delta(y)\,dx\,dy}.$$

We now apply Lemma 2. Here $H^+(x, y) = \max(-6xy(x^2 - 6xy + y^2)/(x + y)^5, 0)$. We calculate the required integral (using Maple) to find

$$\int_0^\infty H^+(x, 1) x^{-1/2}\,dx \approx 1.3390\ldots. \qquad \square$$

We now move to the proof of Lemma 2. The proof depends on Hölder's inequality

$$\int f(x)g(x)\,dx \le \left(\int f^p(x)\,dx\right)^{1/p}\left(\int g^q(x)\,dx\right)^{1/q},$$

and the following version of Hilbert's inequality (see [10]):

THEOREM 5 (Hilbert's Inequality). *For $p, q > 1$ satisfying $1/p + 1/q = 1$, suppose that $K(x, y)$ is nonnegative and homogeneous of degree -1, and that*

$$\int_0^\infty K(x,1)x^{-1/p}\,dx = \int_0^\infty K(1,y)y^{-1/q}\,dx = C.$$

Then

$$\int_0^\infty dx\left(\int_0^\infty K(x,y)g(y)\,dy\right)^q \le C^q\int_0^\infty g^q(y)\,dy.$$

PROOF OF LEMMA 2. Set $F(x) = \int_\infty^x f(x)\,dx$. Then, by Lemma 2 of [6], $\int_0^\infty\int_0^\infty \min(x,y)f(x)f(y)\,dx\,dy = \int_0^\infty F^2(x)\,dx$. Similarly,

$$\begin{aligned}
&\int_0^\infty\int_0^\infty G(x,y)f(x)f(y)\,dx\,dy\\
&\quad= \int_0^\infty f(x)\,dx\left[G(x,y)F(y)|_0^\infty - \int_0^\infty \frac{\partial G}{\partial y}F(y)\,dy\right]\\
&\quad= -\int_0^\infty\int_0^\infty \frac{\partial G}{\partial y}f(x)F(y)\,dx\,dy\\
&\quad= \int_0^\infty\int_0^\infty \frac{\partial^2 G}{\partial x\partial y}F(x)F(y)\,dx\,dy\\
&\quad= \int_0^\infty\int_0^\infty H(x,y)F(x)F(y)\,dx\,dy\\
&\quad\le \int_0^\infty\int_0^\infty H^+(x,y)F(x)F(y)\,dx\,dy\\
&\quad\le \left[\int_0^\infty F^2(x)\,dx\right]^{1/2}\left[\int_0^\infty dx\left[\int_0^\infty H^+(x,y)F(y)\,dy\right]^2\right]^{1/2}\\
&\quad\le \int_0^\infty F^2(x)\,dx\int_0^\infty H^+(x,1)x^{-1/2}\,dx.
\end{aligned}$$

The second equality follows from the definition of F and the hypothesis that $\int_0^\infty f\,dx = 0$. The penultimate step follows from Hölder's inequality, and the last step utilizes Hilbert's inequality. The lemma follows immediately. □

Note the necessity of replacing $H(x, y)$ by $H^+(x, y)$ in the third to last step, as Hölder's inequality requires the functions inside the integral to be nonnegative. In fact in Theorem 4 the function $H(x, y)$ can be negative, so care must be taken in calculating the integral. It is somewhat surprising that, despite seemingly having to "cut off" part of the integral, we still realize an interesting result. It also suggests that perhaps the bound could be improved by avoiding this technical difficulty.

3. Analyses and Simulations for Data Compression.

The MTF algorithm has proved useful in the development of the locally adaptive compression scheme of [4]. Motivated by this result, we consider a similar algorithm based on TS(0). We assume the reader is somewhat familiar with the system of [4], which was briefly described in the Introduction.

3.1. *Theoretical Results*. Let $B_{\mathrm{TS}}(\vec{p})$ be the expected number of bits that TS(0) needs to encode one symbol in an input sequence that is generated by $\vec{p} = (p_1, p_2, \ldots, p_n)$. We assume $p_i > 0$ for all i. In order to analyze $B_{\mathrm{TS}}(\vec{p})$, we have to specify how an integer j should be encoded. We use a variable length prefix code by Elias [7] which encodes the integer j using $1 + \lfloor \log j \rfloor + 2\lfloor \log(1 + \log j) \rfloor$ bits. Bentley et al. [4] showed that, using this prefix code, the expected number of bits needed by the MTF algorithm is $B_{\mathrm{MTF}}(\vec{p}) \le 1 + H(\vec{p}) + 2\log(1 + H(\vec{p}))$, for all $\vec{p}$. Here $H(\vec{p}) = \sum_{i=1}^{n} p_i \log(1/p_i)$ is the entropy of the source. We prove similar bounds for TS(0).

THEOREM 6. *For any* $\vec{p} = (p_1, p_2, \ldots, p_n)$,

$$B_{\mathrm{TS}}(\vec{p}) \le 1 + \bar{H}(\vec{p}) + 2\log(1 + \bar{H}(\vec{p})),$$

where $\bar{H}(\vec{p}) = H(\vec{p}) + \log(1 - \sum_{1 \le i \le j \le n} (p_i p_j (p_i - p_j)^2/(p_i + p_j)^2))$.

Note that $0 \le \sum_{1 \le i \le j \le n} p_i p_j (p_i - p_j)^2/(p_i + p_j)^2 < 1$ and thus $\log(1 - \sum_{1 \le i \le j \le n} p_i p_j (p_i - p_j)^2/(p_i + p_j)^2 \le 0$.

PROOF. Let $f(j) = 1 + \log j + 2\log(1 + \log j)$. Consider a fixed symbol x_i, $1 \le i \le n$. For $j = 1, \ldots, n$, let q_{ij} be the asymptotic probability that x_i is at position j in TS(0)'s list. The expected number of bits to encode the symbol x_i is $\sum_{j=1}^{n} q_{ij} f(j)$, which, by Jensen's [10] inequality, is at most $f(\sum_{j=1}^{n} q_{ij} j)$. Jensen's inequality state that, for any concave function f and any set $\{w_1, w_2, \ldots, w_n\}$ of positive reals, $\sum_{i=1}^{n} w_i f(y_i) \le f(\sum_{i=1}^{n} w_i y_i)$. Note that $q_{ij} j$ is the asymptotic expected position $e_{\mathrm{TS}}(x_i)$ of symbol x_i in TS(0)'s list. Therefore, $B_{\mathrm{TS}}(\vec{p}) \le \sum_{i=1}^{n} p_i f(e_{\mathrm{TS}}(x_i))$. In the following we show that

$$\sum_{i=1}^{n} p_i \log(e_{\mathrm{TS}}(x_i)) \le \bar{H}(\vec{p}). \tag{3}$$

Using the inequality, we can easily derive Theorem 6 because

$$\begin{aligned} B_{\mathrm{TS}}(\vec{p}) &\le \sum_{i=1}^{n} p_i f(e_{\mathrm{TS}}(x_i)) \le 1 + \sum_{i=1}^{n} p_i \log(e_{\mathrm{TS}}(x_i)) \\ &\quad + 2\sum_{i=1}^{n} \log(1 + p_i \log(e_{\mathrm{TS}}(x_i))) \\ &\le 1 + \bar{H}(\vec{p}) + 2\log(1 + \bar{H}(\vec{p})). \end{aligned}$$

We now show inequality (3). By Theorem 1(a), we have $e_{\mathrm{TS}}(x_i) = \frac{1}{2} + \sum_{j=1}^n (p_j^3 + 3p_j^2 p_i)/(p_i + p_j)^3$ and

$$\begin{aligned}\sum_{i=1}^n p_i \log(e_{\mathrm{TS}}(x_i)) &= \sum_{i=1}^n p_i \log\left(\frac{1}{2} + \sum_{j=1}^n \frac{p_j^3 + 3p_j^2 p_i}{(p_i + p_j)^3}\right) \\ &= \sum_{i=1}^n p_i \log\left(\frac{1}{2} + \sum_{j=1}^n \frac{p_j}{p_i + p_j} + \sum_{j=1}^n \frac{p_i p_j^2 - p_i^2 p_j}{(p_i + p_j)^3}\right).\end{aligned}$$

We have

$$\frac{1}{2} + \sum_{j=1}^n \frac{p_j}{p_i + p_j} = \frac{1}{p_i}\left(\frac{1}{2}p_i + \sum_{j=1}^n \frac{p_i p_j}{p_i + p_j}\right) \le \frac{1}{p_i}\left(p_i + \sum_{\substack{j=1\\ j\neq i}}^n p_j\right) = \frac{1}{p_i}.$$

Therefore

$$\begin{aligned}\sum_{i=1}^n p_i \log(e_{\mathrm{TS}}(x_i)) &\le \sum_{i=1}^n p_i \log\left(\frac{1}{p_i} + \sum_{j=1}^n \frac{p_i p_j^2 - p_i^2 p_j}{(p_i + p_j)^3}\right) \\ &= \sum_{i=1}^n p_i \log\left(\frac{1}{p_i}\right) + \sum_{i=1}^n p_i \log\left(1 + \sum_{i=1}^n \frac{p_i^2 p_j^2 - p_i^3 p_j}{(p_i + p_j)^3}\right) \\ &\le \sum_{i=1}^n p_i \log\left(\frac{1}{p_i}\right) + \log\left(1 + \sum_{i=1}^n p_i \sum_{i=1}^n \frac{p_i^2 p_j^2 - p_i^3 p_j}{(p_i + p_j)^3}\right).\end{aligned}$$

The last step follows again from Jensen's inequality. We conclude

$$\sum_{i=1}^n p_i \log(e_{\mathrm{TS}}(x_i)) \le \sum_{i=1}^n p_i \log\left(\frac{1}{p_i}\right) + \log\left(1 - \sum_{1\le i\le j\le n} \frac{p_i p_j (p_i - p_j)^2}{(p_i + p_j)^2}\right). \quad \square$$

So far we have assumed that an input sequence S to be compressed is generated by a probability distribution $\vec{p} = (p_1, p_2, \ldots, p_n)$. Now consider *any* input sequence S. Let m be the length of S, and let m_i, $1 \le i \le n$, be the number of occurrences of the symbol x_i in the string S. Let $A_{\mathrm{TS}}(S)$ be the average number of bits needed to encode one symbol in the string S using the TS(0) algorithm. Similarly, let $A_{\mathrm{MTF}}(S)$ be the average number of bits needed by the MTF algorithm. Again, we assume that an integer j is encoded by means of the Elias encoding that requires $1 + \lfloor \log j \rfloor + 2\lfloor \log(1 + \log j)\rfloor$ bits. Bentley et al. [4] show that, for any input sequence S, $A_{\mathrm{MTF}}(S) \le 1 + H(S) + 2\log(1 + H(S))$ where $H(S) = \sum_{i=1}^n (m_i/m)\log(m/m_i)$ is the "empirical entropy" of S. The empirical entropy is interesting because it corresponds to the average number of bits per symbol used by the optimal static Huffman encoding of a sequence; this result implies that MTF encoding is, at worst, almost as good as static Huffman encoding. We can show a similar bound for a variation of TS(0), where after the first occurrence of a symbol it is moved to the front of the list.

THEOREM 7. *For any input sequence S,*

$$A_{\mathrm{TS}}(S) \le 1 + H(S) + 2\log(1 + H(S)),$$

where $H(S) = \sum_{i=1}^{n} (m_i/m) \log(m/m_i)$.

PROOF. Our analysis is very similar to the corresponding proof by Bentley et al. Again, let $f(j) = 1 + \log j + 2\log(1 + \log j)$. Consider a fixed symbol x_i, $1 \le i \le n$, and let $t_1, t_2, \ldots, t_{m_i}$ be the times at which the symbol x_i occurs in the string S. We assume here that after TS(0) has transmitted the first occurrence of the symbol x_i, it moves x_i to the front of the list. Furthermore, we may assume without loss of generality that the first occurrence of x_i is encoded using $f(t_1)$ bits. We show that, for $k = 2, 3, \ldots, m_i$, the kth occurrence of the symbol x_i can be encoded using $f(pos_{k-1} + t_k - t_{k-1} - pos_k)$ bits, where pos_k is the position of symbol x_i in TS(0)'s list immediately after the kth x_i is transmitted. After the $(k-1)$st occurrence of x_i is encoded, the position of x_i in TS(0)'s list is pos_{k-1}. Let d_k be the number of symbols x_j, $x_j \ne x_i$, that occur at least twice in the interval $[t_{k-1}+1, t_k-1]$. Obviously, at most $t_k - t_{k-1} - 1 - d_k$ symbols x_j can move ahead of x_i in TS(0)'s list during the time interval $[t_{k-1}+1, t_k-1]$. By the definition of TS(0), immediately after the kth occurrence of x_i is transmitted, x_i precedes all items in TS(0)'s list that were requested at most once in $[t_{k-1}+1, t_k-1]$. Also, by Lemma 1, x_i is located behind all items in the list that are requested at least twice in $[t_{k-1}+1, t_k-1]$. Thus, $d_k + 1 = pos_k$. Therefore, the kth occurrence of x_i can be encoded using at most $f(pos_{k-1} + t_k - t_{k-1} - 1 - d_k) = f(pos_{k-1} + t_k - t_{k-1} - pos_k)$ bits. The total number of bits needed to encode the m_i occurrences of the symbol x_i is at most

$$\begin{aligned}
f(t_1) &+ \sum_{k=2}^{m_i} f(pos_{k-1} + t_k - t_{k-1} - pos_k) \\
&\le m_i\, f\left(\frac{1}{m_i}\left(t_1 + \sum_{k=2}^{m_i}(pos_{k-1} + t_k - t_{k-1} - pos_k)\right)\right) \\
&= m_i\, f\left(\frac{1}{m_i}(t_{m_i} + pos_1 - pos_{m_i})\right) \\
&\le m_i\, f\left(\frac{m}{m_i}\right).
\end{aligned}$$

The first inequality follows from Jensen's inequality; in the last step we make use of the facts that $t_{m_i} \le m$ and $pos_1 = 1 \le pos_{m_i}$.

Summing up the above expression for all x_i and dividing by m, we obtain that the average number of bits needed by TS(0) to encode one symbol in the string S is

$$A_{\mathrm{TS}}(S) \le \sum_{i=1}^{n} \frac{m_i}{m} f\left(\frac{m}{m_i}\right).$$

The theorem follows immediately. □

3.2. *Simulation Results*. Our theoretical work suggests that a compression scheme similar to MTF using the TS(0) scheme may provide better performance. In effect, TS(0) is a conservative version of the MTF encoding; like MTF encoding, it responds well to locality of reference by moving recently requested items to the front, but it responds more slowly. Understanding this intuition is important to understand where TS(0) encoding can improve on MTF encoding: when the locality is very strong, then MTF encoding will perform better, since it responds more aggressively. On the other hand, TS(0) encoding is more effective when the input to be compressed resembles a string generated by a distribution, possibly with a large number of rare items each with a small probability of appearing.

We have tested our theoretical results by implementing simple versions of TS(0) encoders and decoders for text compression. Our tests use standard documents from the Calgary Compression Corpus [19]. The current goal of these these is not to develop an all-purpose functional compression system, but merely to demonstrate the potential gains from using TS(0) in place of MTF. The compression is performed by turning the document into a token stream. The tokens are then encoded by their position in the list using standard variable-length prefix encodings given by Elias [7]; each integer j requires $1 + 2\lfloor \log j \rfloor$ bits. This prefix code is different from the code we used in the analyses of the previous section; in our tests, it leads to slightly better compression. We can compare the compression of MTF and TS(0) compression by varying the adaptive discipline of the list.

In the first test ASCII characters (that is, single bytes) constitute the tokens, and the list is initialized in order of character frequency in standard text. The results of Table 1 demonstrate that TS(0) encoding outperforms MTF encoding significantly on the sample documents. The boldface figures indicate how much space the compressed files take, assuming that the original files use 100% space. The improvement of TS(0) over MTF is typically 6–8%. Moreover, in all cases TS(0) encoding beats MTF encoding. However, this character-based compression scheme performs far worse than standard

Table 1. MTF versus TS(0): Byte-based compression.

	TS(0)		MTF		Original
File	Bytes	% Orig.	Bytes	% Orig.	bytes
bib	99,121	**89.09**	106,478	**95.70**	111,261
book1	581,758	**75.67**	644,423	**83.83**	768,771
book2	473,734	**77.55**	515,257	**84.35**	610,856
geo	92,770	**90.60**	107,437	**104.92**	102,400
news	310,003	**82.21**	333,737	**88.50**	377,109
obj1	18,210	**84.68**	19,366	**90.06**	21,504
obj2	229,284	**92.90**	250,994	**101.69**	246,814
paper1	42,719	**80.36**	46,143	**86.80**	53,161
paper2	63,654	**77.44**	69,441	**84.48**	82,199
pic	113,001	**22.02**	119,168	**23.22**	513,216
progc	33,123	**83.62**	35,156	**88.75**	39,611
prog1	52,490	**73.26**	55,183	**77.02**	71,646
progp	37,266	**75.47**	40,044	**81.10**	49,379
trans	79,258	**84.59**	82,058	**87.58**	93,695

Table 2. MTF versus TS(0): Word-based compression.

File	TS(0)		MTF		Original
	Bytes	% Orig.	Bytes	% Orig.	bytes
bib	34,117	**30.66**	35,407	**31.82**	111,261
book1	286,691	**37.29**	296,172	**38.53**	768,771
book2	260,602	**42.66**	267,257	**43.75**	610,856
news	116,782	**30.97**	117,876	**31.26**	377,109
paper1	15,195	**28.58**	15,429	**29.02**	53,161
paper2	24,862	**30.25**	25,577	**31.12**	82,199
progc	10,160	**25.65**	10,338	**26.10**	39,611
progl	14,931	**20.84**	14,754	**20.59**	71,646
progp	7,395	**14.98**	7,409	**15.00**	49,379

UNIX utilities, such as `pack`, `compress`, and `gzip`, which generally compress text files by 30% to 80%. Among the Unix utilities, `compress` is superior to `pack`, and `gzip` is superior to `compress`.

In order to make TS(0) and MTF encoding comparable with the standard UNIX utilities, we have to use words as the tokens, which we do in our second test. A word is taken to be a sequence of nonwhite space characters between white space. This technique assumes that the decompressor has a dictionary consisting of a list of all words in the document; in practice, this dictionary (in compressed or uncompressed form) can be included as part of the compressed document. In the results of Table 2, we compare the size of the MTF- and TS(0)-based encodings. These figures do not include the dictionary cost, so that a direct comparison between MTF and TS(0) can be seen. Also, for convenience, we placed no memory limitation on the compressor or decompressor; that is, the length of the list was allowed to grow as large as necessary. In practice one might wish to devise a more memory-efficient scheme, using the list as a cache as in [4].

The results of Table 2 reflect the compression achieved,[4] including only the token stream and not the dictionary. As one might expect, the gains from TS(0) in this situation are less dramatic, but still noticeable.

To compare the TS(0) compression with the standard UNIX utilities, we add to the results in Table 2 the size of the dictionary after being compressed using `pack`. These results are presented in Table 3. We emphasize that our results for TS(0) could be improved by compressing the dictionary using other methods, but the results are quite suggestive of TS(0) performance: the scheme performs better than `pack` and occasionally as well as `compress`, but not as well as `gzip`.

We have also attempted to use TS(0) encoding in place of MTF encoding in the data compression algorithm recently presented by Burrows and Wheeler [5]. Unfortunately, the results here show less promise. In some cases, TS(0) led to improved compression, but in most cases MTF encoding yielded better results. Although it is not entirely clear why this is the case, we note that the Burrows–Wheeler compression scheme attempts to use MTF on a stream with an extremely high locality of reference. Given this, it is

[4] Because the current implementation handles only ASCII characters, we do not have results for all files.

Table 3. TS(0) word-based compression versus UNIX utilities.

File	TS(0) (% orig.)	`pack` (% orig.)	`compress` (% orig.)	`gzip` (% orig.)
bib	**51.51**	**63.91**	**41.82**	**31.51**
book1	**50.66**	**57.04**	**43.19**	**40.76**
book2	**56.06**	**60.31**	**41.05**	**33.84**
news	**57.07**	**65.37**	**48.29**	**38.41**
paper1	**53.74**	**62.94**	**47.17**	**34.94**
paper2	**49.88**	**58.07**	**43.99**	**36.20**
progc	**65.11**	**65.71**	**48.33**	**33.51**
progl	**37.54**	**60.15**	**37.89**	**22.71**
progp	**42.79**	**61.42**	**38.90**	**22.77**

not entirely surprising that MTF would outperform TS(0). We remain optimistic that TS(0)-based encoding will prove useful in other situations.

4. Conclusion.

We have analyzed the performance of the deterministic list update algorithm TS(0) when a request sequence is generated by a probability distribution $\vec{p} = (p_1, p_2, \ldots, p_n)$. We have demonstrated that TS(0) has a better overall performance than the MTF algorithm on such distributions. In particular, we have shown that on all distributions, the expected cost incurred by TS(0) is at most 1.5 times the expected cost incurred by the optimal (dynamic) off-line algorithm. We note that a similar analysis can also be used to study the Timestamp(p) algorithms [1], but TS(0) yields the best competitive ratio against distributions. Also, the techniques we used can easily be extended to the case that a request sequence is generated by a Markov chain, but for general Markov chains, we cannot prove that TS(0) has a better competitive ratio than 2. (MTF can easily be shown to be 2-competitive against a Markov chain that cycles among the n elements of the list.)

List update algorithms can be used to develop locally adaptive data compression schemes. Our theoretical results show that TS(0)-based encoding can be better than MTF-based encoding. We have supported our theoretical observations by building encoders and decoders with TS(0) encoding that lead to improved compression over MTF encoding on a standard corpus of test files.

We suggest some open questions based on our work. A tight bound on the competitive ratio of TS(0) against static off-line algorithms, perhaps based on a tight bound for the expression in Theorem 4, remains open. A more general question is whether there is quick, simple way to determine which list update strategy (say between MTF and TS(0)) is expected to perform better on a given higher-order Markovian source. In theory, the expected cost per request can be determined, but this seems excessively time-consuming. Our work settles the question for discrete memoryless sources in favor of TS(0); for higher-order sources, the question appears more difficult.

Acknowledgments. The authors would like to thank Michael Burrows for general information on compression and outlines for the code.

References

[1] S. Albers. Improved randomized on-line algorithms for the list update problem. In *Proceedings of the* 6*th Annual ACM–SIAM Symposium on Discrete Algorithms*, pages 412–419, 1995.

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

[3] J.L. Bentley and C.C. McGeoch. Amortized analyses of self-organizing sequential search heuristics. *Communications of the ACM*, **28**:404–411, 1985.

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

[5] M. Burrows and D.J. Wheeler. A Block-Sorting Lossless Data Compression Algorithm. DEC SRC Research Report 124, 1994.

[6] F.R.K. Chung, D.J. Hajela, and P.D. Seymour. Self-organizing sequential search and Hilbert's inequality. In *Proceedings of the* 17*th Annual Symposium on the Theory of Computing*, pages 217–223, 1985.

[7] P. Elias. Universal codeword sets and the representation of the integers. *IEEE Transactions on Information Theory*, **21**:194–203, 1975.

[8] G.H. Gonnet, J.I. Munro, and H. Suwanda. Exegesis of self-organizing linear search. *SIAM Journal on Computing*, **10**:613–637, 1981.

[9] D. Grinberg, S. Rajagopalan, R. Venkatesan, and V.K. Wei. Splay trees for data compression. In *Proceedings of the* 6*th Annual ACM–SIAM Symposium on Discrete Algorithms*, pages 522–530, 1995.

[10] G.H. Hardy, J.E. Littlewood, and G. Polya. *Inequalities*. Cambridge University Press, Cambridge, 1994.

[11] S. Irani. Two results on the list update problem. *Information Processing Letters*, **38**:301–306, 1991.

[12] N. Kahale. Large Deviation Bounds for Markov Chains. DIMACS Technical Report 94-39.

[13] R. Karp and P. Raghavan. From a personal communication cited in [14].

[14] N. Reingold, J. Westbrook, and D.D. Sleator. Randomized competitive algorithms for the list update problem. *Algorithmica*, **11**(1):15–32, 1994.

[15] R. Rivest. On self-organizing sequential search heuristics. *Communications of the ACM*, **19**:63–67, 1976.

[16] A. Shwartz and A. Weiss. *Large Deviations for Performance Analysis*: *Queues, Communications, and Computing*. Chapman and Hall, London, 1995.

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

[18] B. Teia. A lower bound for randomized list update algorithms. *Information Processing Letters*, **47**:5–9, 1993.

[19] I.H. Witten and T. Bell. The Calgary/Canterbury text compression corpus. Anonymous ftp from ftp.cpsc.ucalgary.ca : /pub/text.compression/corpus/text.compression.corpus.tar.Z.