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

Susanne Albers* Michael Mitzenmacher†

TR-95-039

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 offline and static adversaries in this situation. Previous work on such sources compared online algorithms only to 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.

*International Computer Science Institute, 1947 Center Street, Berkeley, CA 94708; and Max Planck Institute for Computer Science, Germany. Supported in part by an Otto Hahn Medal Award of the Max Planck Society. albers@icsi.berkeley.edu

†Computer Science Division, University of California at Berkeley, Berkeley, CA 94720. mitzen@cs.berkeley.edu

1 Introduction

We study deterministic online 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 i-th 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. 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 *online*, i.e., when serving a 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.
- **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 suitable in many applications, and many of our techniques generalize to other models as well. Our work is motivated by the goal to present a universal algorithm that is competitive on any sequence (in the Sleator Tarjan model) but also performs especially well on distributions. Previous results have shown that MTF is such an algorithm, whereas algorithms T and FC are not. 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 online 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.

In [1] it was shown that the algorithm achieves a competitive ratio of 2 on any request sequence, as does Move-to-front [14]. 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 offline algorithms. It should be noted that TS(0) needs memory to store, for each item x, the times of the two most recent requests to x. If these two times are not stored in separate array but with the item, the algorithm needs a second pass through the list in order to find the position at which the requested item should be inserted.

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 [3]. 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 that a request sequence is generated by a probability distribution. The performances of MTF, T, and FC have generally been compared to 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 position 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 [12] showed that for all $\vec{p}$, $E_{FC}(\vec{p})/E_{STAT}(\vec{p}) = 1$. However, the algorithm FC has the drawback that it adapts very slowly to changing probability distributions. Chung *et al.* [5] analyzed the MTF rule and proved $E_{MTF}(\vec{p})/E_{STAT}(\vec{p}) \leq \frac{\pi}{2} \approx 1.5708$ for all $\vec{p}$. This bound is tight because Gonnet *et al.* [7] showed that one can find $\vec{p}_0$ with $E_{MTF}(\vec{p}_0)/E_{STAT}(\vec{p}_0) \geq \alpha$ for any α arbitrarily close to $\frac{\pi}{2}$.

More recent research on the list update problem was inspired by Sleator and Tarjan [14] who suggested to compare the performance of an online algorithm to that of an *optimal offline* algorithm. An optimal offline algorithm knows the entire request sequence in advance and can serve it with minimum cost. An online 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 offline algorithm. Sleator and Tarjan proved that the MTF algorithm is 2-competitive. They also showed that the algorithms T and FC are not c-competitive for any constant c. The competitive ratio of 2 is the best ratio that a deterministic online algorithm for the list update problem can achieve [11].

In classical data compression, 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.* [3] showed how a 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 positions 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.

Recently, Grinberg *et al.* [8] 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[4]. 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 online algorithm to that of the optimal offline algorithm, which we shall 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 $p_i = \frac{1}{n}$, 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) to the optimal offline algorithm OPT and show $E_{TS}(\vec{p})/E_{OPT}(\vec{p}) \leq 1.5$ for all distributions $\vec{p}$. This is a performance MTF cannot match because $E_{TS}(\vec{p}_0)/E_{STAT}(\vec{p}_0) > 1.57$ for some $\vec{p}_0$, and when MTF is compared to OPT the ratio might even be worse. It is worthwhile to notice that 1.5 is the best lower bound currently known on the competitiveness that can be achieved by randomized list update algorithms against the oblivious adversary [15]. 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 $\vec{p}$, $E_{TS}(\vec{p})/E_{STAT}(\vec{p}) \leq 1.34$.

Given these results, we examine the potential for 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 provide evidence that TS(0) could be useful in practice by implementing TS(0) compression algorithms and testing them on the standard Calgary Compression Corpus files. In almost all of our tests, TS(0) encoding achieves a better compression ratio than MTF encoding.

2 Analyses for the list update problem

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.

2.1 The expected cost of TS(0)

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_i, $1 \leq i \leq n$, is
$$e_{TS}(x_i) = \frac{1}{2} + \sum_{j=1}^{n} \frac{p_j^3 + 3p_j^2 p_i}{(p_i + p_j)^3}.$$

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

Proof: Part a): The cost $e_{TS}(x_i)$ is 1 plus the expected number of items x_j, $x_j \neq x_i$, that precede x_i in the list. Let A_{ji} be the event that x_j precedes x_i in the list when TS(0) serves a request to x_i. We compute the asymptotic probability $Prob(A_{ji})$ using the following 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_j precedes x_i in the list if and only if a majority of the last three requests for x_i and x_j have been for x_j.*

Proof of Lemma 1: 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 $x_ix_ix_j$. After the second request to x_i, item x_i must precede x_j in TS(0)'s list. The 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 twice 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 not requested 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 $x_ix_ix_j$, then x_j does not move in front of x_i. ■

Lemma 1 implies that the event A_{ji} occurs if and only if the last three requests for x_i and x_j are (B_1) $x_jx_jx_j$; (B_2) $x_jx_jx_i$; (B_3) $x_jx_ix_j$; or (B_4) $x_ix_jx_j$. It is not hard to verify that $Prob(B_1) = p_j^3/(p_i+p_j)^3$ and $Prob(B_k) = p_ip_j^2/(p_i+p_j)^3$, for $k = 2,3,4$. Therefore, $Prob(A_{ji}) = (p_j^3 + 3p_j^2p_i)/(p_i+p_j)^3$ and

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

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

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

■

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

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

Proof: Rivest [12] showed $E_{MTF}(\vec{p}) = \sum_{1\le i\le j\le n} \frac{2p_ip_j}{p_i+p_j}$. Using part b) of Theorem 1, the result follows immediately. ■

2.2 Performance against dynamic offline algorithms

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

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

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) on σ can be divided into costs that are caused by each unordered pair $\{x, y\}$ of items x and y. For each pair $\{x, y\}$, we can partition σ into phases. The cost caused by the pair $\{x, y\}$ is equal to the total cost that $\{x, y\}$ incurs on all phases. In the same way we can estimate OPT's cost on a request sequence σ. This technique of evaluating cost by considering pairs of items was also used in [2, 10, 1]. In the second part of the analysis we show that, for each pair $\{x, y\}$, the asymptotic expected cost paid by TS(0) on a phase is at most $\frac{3}{2}$ times the asymptotic expected cost incurred by OPT.

In the following we will always assume that serving a request to the i-th item in the list incurs a cost of $i-1$ rather than i. If $E_{TS}(\vec{p}) \leq \frac{3}{2} E_{OPT}(\vec{p})$ holds in this $(i-1)$-*cost model*, then the inequality obviously also holds in the i-cost model. Now consider a fixed request sequence $\sigma = \sigma(1)\sigma(2), \ldots, \sigma(m)$ of length m. Let L be the set of items in the list. For an algorithm $A \in \{TS(0), OPT\}$, 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 $C_A(t, x) = 0$. The cost incurred by A on σ can be written as

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

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 $C_{TS}(\sigma_{xy})$ 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,

$$C_{TS}(\sigma_{xy}) = \sum_{\substack{t \in [1,m] \\ \sigma(t) = x}} C_{TS}(t, y) + \sum_{\substack{t \in [1,m] \\ \sigma(t) = y}} C_{TS}(t, x) \qquad \text{and} \qquad C_{TS}(\sigma) = \sum_{\substack{\{x,y\} \\ x \neq y}} C_{TS}(\sigma_{xy}). \tag{1}$$

The optimal cost $C_{OPT}(\sigma)$ can be written in a similar way:

$$C_{OPT}(\sigma_{xy}) \leq \sum_{\substack{t \in [1,m] \\ \sigma(t) = x}} C_{OPT}(t, y) + \sum_{\substack{t \in [1,m] \\ \sigma(t) = y}} C_{OPT}(t, x) \qquad \text{and} \qquad C_{OPT}(\sigma) \geq \sum_{\substack{\{x,y\} \\ x \neq y}} C_{OPT}(\sigma_{xy}). \tag{2}$$

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. Line (1) and line (2) allow us to compare $C_{TS}(\sigma)$ and $C_{OPT}(\sigma)$ by simply comparing $C_{TS}(\sigma_{xy})$ and $C_{OPT}(\sigma_{xy})$ for each pair $\{x, y\}$ of items. In the following we concentrate on one particular pair $\{x, y\}$ of items $x \neq y$.

We partition the request sequence σ_{xy} into phases, as in [1]. The first phase starts with the first request in σ_{xy} and ends when, for the first time, there are two requests to the same item and the next request is different. The second phase starts with that next request and ends in the same way as the first phase, and so on. The phases we obtain can be classified into the following types.

$T1_x(h)$:	x^h	$T1_y(h)$:	y^h	for some $h \geq 2$
$T2_x(h_1, h_2)$:	$(xy)^{h_1} x^{h_2}$	$T2_y(h_1, h_2)$:	$(yx)^{h_1} y^{h_2}$	for some $h_1 \geq 1, h_2 \geq 2$
$T3_x(h_1, h_2)$:	$(xy)^{h_1} y^{h_2}$	$T3_y(h_1, h_2)$:	$(yx)^{h_1} x^{h_2}$	for some $h_1 \geq 1, h_2 \geq 1$

Since a phase ends with (at least) two requests to the same item, the item requested last in the phase precedes the other item in the two item list maintained by TS(0). Thus the item requested first in a phase is always second in the list. Without loss of generality we can assume the same holds for OPT, because when OPT serves two consecutive requests to the same item, it cannot cost more to move that item to the front of the two item list after the first request. The costs spent by TS(0) and OPT on the various phases are given in the table below.

	$TS(0)$	OPT		$TS(0)$	OPT
$T1_x(h)$	2	1	$T1_y(h)$	2	1
$T2_x(h_1,h_2)$	$2h_1$	h_1+1	$T2_y(h_1,h_2)$	$2h_1$	h_1+1
$T3_x(h_1,h_2)$	$2h_1-1$	h_1	$T3_y(h_1,h_2)$	$2h_1-1$	h_1

We explain the entries for TS(0) in the table; the entries for OPT are obvious. Consider a phase of type $T1_x(h)$ for some integer $h \geq 2$. TS(0) pays a cost of 1 at the first request because x is stored behind y in the list. TS(0) leaves the position of x unchanged because y was requested at least twice since the last request to x. Hence TS(0) also pays a cost of 1 at the second request to x. At this second request, x is moved to the front of the list, and all remaining requests to x in the phase cost 0. Therefore the total cost of TS(0) in a phase of type $T1_x(h)$ is 2. If TS(0) serves a phase of type $T2_x(h_1,h_2)$ or $T3_x(h_1,h_2)$, then the first request to x costs 1 and, since the position of x does not change, the second request to y costs 0. On all remaining requests, the requested item is always moved to the front of the list. In a phase of type $T2_x(h_1,h_2)$ this gives a total cost of $1+0+2(h_1-1)+1 = 2h_1$; in a phase of type $T3_x(h_1,h_2)$ the total cost is $1+0+2(h_1-1) = 2h_1-1$. The costs incurred by TS(0) on types $T1_y(h), T2_y(h_1,h_2), T3_y(h_1,h_2)$ are the same.

Suppose that the request sequence σ_{xy} consists of k_{xy} phases $P_{xy}(1), P_{xy}(2), \ldots, P_{xy}(k_{xy})$. For any algorithm $A \in \{TS, OPT\}$, let $C_A(P_{xy}(i))$ denote the cost that A incurs on phase $P_{xy}(i)$. Then

$$C_A(\sigma) = \sum_{\substack{\{x,y\}\\ x\neq y}} C_A(\sigma_{xy}) = \sum_{\substack{\{x,y\}\\ x\neq y}} \sum_{i=1}^{k_{xy}} C_{TS}(P_{xy}(i)).$$

Here and in the following we assume $C_{OPT}(\sigma) = \sum_{\substack{\{x,y\}\\ x\neq y}} C_{OPT}(\sigma_{xy})$. This can only strengthen the optimal offline algorithm.

So far we have considered a fixed request sequence σ of length m. Now, let σ be a request sequence of length m that is generated according to the distribution $\vec{p} = (p_1, p_2, \ldots, p_n)$. We develop formulae for the expected cost that TS(0) and OPT incur on one request in σ. For any pair $\{x,y\}$ of items $x \neq y$, let $\mathcal{T}_{xy}$ be the set of all possible phase types mentioned above, i.e., $\mathcal{T}_{xy} = \{T1_z(h) | z = x,y;\ h = 2,3\ldots\} \cup \{T2_z(h_1,h_2) | z = x,y;\ h_1 = 1,2,\ldots;\ h_2 = 2,3,\ldots\} \cup \{T3_z(h_1,h_2) | z = x,y;\ h_1,h_2 = 1,2,\ldots\}$. For a type $T \in \mathcal{T}_{xy}$, let $C_A(T)$ be the cost that algorithm $A \in \{TS(0), OPT\}$ incur on a phase of type T. For any positive integer i, Let B^i_{xy} denote the event that σ_{xy} consists of at least i phases. The expected cost incurred by $A \in \{TS(0), OPT\}$ on a single request in σ is given by

$$\frac{1}{m} \sum_{\substack{\{x,y\}\\ x\neq y}} \sum_{i\geq 1} \sum_{T\in\mathcal{T}_{xy}} Prob(B^i_{xy} \cap (\text{phase } P_{xy}(i) \text{ has type } T)) C_A(T).$$

We are interested in the asymptotic value of the above expression. In the following we concentrate on one particular pair $\{x,y\}$ of items $x \neq y$ and show that the asymptotic expected cost incurred by TS(0) on a phase $P_{xy}(i)$ is at most $\frac{3}{2}$ times the asymptotic expected cost incurred by OPT on that phase. Given a long (potentially infinite) request sequence σ that is generated by the probability distribution $\vec{p} = (p_1, p_2, \ldots, p_n)$, the asymptotic expected cost that an algorithm $A \in \{TS(0), OPT\}$ incurs on a phase in σ_{xy} is given by

$$E_A(P_{xy}) = \sum_{T\in\mathcal{T}_{xy}} Prob(\text{phase } P_{xy} \text{ has type } T) C_A(T).$$

If the inequality

$$E_{TS}(P_{xy}) \leq \frac{3}{2} E_{OPT}(P_{xy}) \tag{3}$$

holds for any pair $\{x, y\}$ of items, then, asymptotically, the expected cost incurred by TS(0) on a request sequence σ generated by $p = (p_1, p_2, \ldots, p_n)$ can be at most $\frac{3}{2}$ times the expected cost incurred by OPT on σ. This establishes the theorem.

In remains to prove inequality (3) for any fixed pair $\{x, y\}$. We omit the subscript xy where the meaning is understood. For the following let $p = \frac{p_x}{p_x + p_y}$ and $q = \frac{p_y}{p_x + p_y}$. We have to compute, for each $T \in \mathcal{T}$, $Prob$(phase P has type T). Let $\overline{P}$ be the phase preceding P. Let B_x be the event that $\overline{P}$ ends with a request to x, and let B_y be the event that $\overline{P}$ ends with a request to y. For any $T \in \mathcal{T}$,

$$Prob(P \text{ has type } T) \;=\; Prob(B_x) \cdot Prob(P \text{ has type } T|B_x) + Prob(B_y) \cdot Prob(P \text{ has type } T|B_y).$$

We first examine the phase types $T1_x(h), T2_x(h_1, h_2), T3_x(h_1, h_2)$. Note that if $\overline{P}$ ends with a request to x, then P cannot start with a request to x because a phase is always extended as long as possible. Therefore, $Prob(P \text{ has type } T|B_x) = 0$ for all the types $T1_x(h), T2_x(h_1, h_2), T3_x(h_1, h_2)$. Consider type $T1_x(h)$ for some $h \geq 2$. We have $Prob(P \text{ has type } T1_x(h)) = Prob(B_y) \cdot Prob(P \text{ has type } T1_x(h)|B_y)$. Given that $\overline{P}$ ends with a request to y, we know that the first request in P is to x. $T1_x(h)$ occurs if exactly $h - 1$ requests to x follow, which happens with probability $p^{h-1}q$. Therefore

$$Prob(P \text{ has type } T1_x(h)) = Prob(B_y) \cdot p^{h-1}q.$$

For type $T2_x(h_1, h_2)$, $h_1 \geq 1, h_2 \geq 2$, given that $\overline{P}$ ends with y, P starts with x. Type $T2_x(h_1, h_2)$ occurs if the following requests consist of a request to y, then $2(h_1 - 1)$ alternating requests to x and y and finally h_2 requests to x. Thus

$$Prob(P \text{ has type } T2_x(h_1, h_2)) = Prob(B_y) \cdot q(pq)^{h_1-1}p^{h_2}q.$$

Using the same arguments we can show for $T3_x(h_1, h_2)$, $h_1, h_2 \geq 1$,

$$Prob(P \text{ has type } T3_x(h_1, h_2)) = Prob(B_y) \cdot q(pq)^{h_1-1}q^{h_2}p.$$

By symmetry we have similar formulas for phases that begin with y.

In order to compute $Prob(B_x)$ and $Prob(B_y)$, we imagine that the phases in σ_{xy} are generated by a Markov chain, where the types $T \in \mathcal{T}$ are states. Given a phase that ends with an x, one can calculate that the probability that the next phase ends with a y is $q/(1 - pq)$. Similarly, given a phase that ends with a y, the next phase ends with an x with probability $p/(1 - pq)$. Consider the points where phases change from ending with x to ending with y and vice versa. In the stationary distribution, the number of changes in each direction must be equal, and hence

$$Prob(B_x)\frac{q}{1 - pq} = Prob(B_y)\frac{p}{1 - pq}.$$

We obtain $Prob(B_x) = p$ and $Prob(B_y) = q$.

In total the probabilities for the types $T \in \mathcal{T}$ are as follows:

	$Prob$		$Prob$
$T1_x(h)$	q^2p^{h-1}	$T1_y(h)$	p^2q^{h-1}
$T2_x(h_1, h_2)$	$q^3(pq)^{h_1-1}p^{h_2}$	$T2_y(h_1, h_2)$	$p^3(pq)^{h_1-1}q^{h_2}$
$T3_x(h_1, h_2)$	$q^2p(pq)^{h_1-1}q^{h_2}$	$T3_y(h_1, h_2)$	$p^2q(pq)^{h_1-1}p^{h_2}$

Multiplying, for each phase type, its probability with the respective cost, we obtain

$$\begin{aligned}
E_{TS}(P) &= \sum_{h=2}^{\infty} q^2p^{h-1}2 + \sum_{h_1=1}^{\infty}\sum_{h_2=2}^{\infty} q^3(pq)^{h_1-1}p^{h_2}2h_1 + \sum_{h_1=1}^{\infty}\sum_{h_2=1}^{\infty} q^2p(pq)^{h_1-1}q^{h_2}(2h_1-1) \\
&\quad + \sum_{h=2}^{\infty} p^2q^{h-1}2 + \sum_{h_1=1}^{\infty}\sum_{h_2=2}^{\infty} p^3(pq)^{h_1-1}q^{h_2}2h_1 + \sum_{h_1=1}^{\infty}\sum_{h_2=1}^{\infty} p^2q(pq)^{h_1-1}p^{h_2}(2h_1-1) \\
&= 2pq + 2p^2q^2\sum_{h_1=1}^{\infty}(pq)^{h_1-1}h_1 + q^3\sum_{h_1=1}^{\infty}(pq)^{h_1-1}(2h_1-1) \\
&\quad +2pq + 2p^2q^2\sum_{h_1=1}^{\infty}(pq)^{h_1-1}h_1 + p^3\sum_{h_1=1}^{\infty}(pq)^{h_1-1}(2h_1-1) \\
&= 4pq + 4p^2q^2\frac{1}{(1-pq)^2} + 2(p^3+q^3)\frac{1}{(1-pq)^2} - (p^3+q^3)\frac{1}{1-pq} \\
&= \frac{1}{(1-pq)^2}(4pq + 4p^2q^2 + 4p^3q^3 + (p^3+q^3)(1+pq)) \\
&= \frac{1}{(1-pq)^2}(4pq - 4p^2q^2(1-pq) - (p^3+q^3)(1-pq) + 2(p^3+q^3)) \\
&= \frac{1}{(1-pq)^2}(2(p^3+q^3+2pq) - (1-pq)(p^3+q^3+4p^2q^2)).
\end{aligned}$$

It is easy to verify that $p^3+q^3+2pq = 1-pq$. Therefore,

$$E_{TS}(P) = (2-p^3-q^3-4p^2q^2)/(1-pq).$$

The asymptotic expected cost incurred by OPT on a phase can be computed similarly and we obtain

$$\begin{aligned}
E_{OPT}(P) &= \sum_{h=2}^{\infty} q^2p^{h-1} + \sum_{h_1=1}^{\infty}\sum_{h_2=2}^{\infty} q^3(pq)^{h_1-1}p^{h_2}\cdot(h_1+1) + \sum_{h_1=1}^{\infty}\sum_{h_2=2}^{\infty} q^2p(pq)^{h_1-1}q^{h_2}h_1 \\
&\quad + \sum_{h=2}^{\infty} p^2q^{h-1} + \sum_{h_1=1}^{\infty}\sum_{h_2=2}^{\infty} p^3(pq)^{h_1-1}q^{h_2}\cdot(h_1+1) + \sum_{h_1=1}^{\infty}\sum_{h_2=2}^{\infty} p^2q(pq)^{h_1-1}p^{h_2}h_1 \\
&= pq + p^2q^2\sum_{h_1=1}^{\infty}(pq)^{h_1-1}(h_1+1) + q^3\sum_{h_1=1}^{\infty}(pq)^{h_1-1}h_1 \\
&\quad +pq + p^2q^2\sum_{h_1=1}^{\infty}(pq)^{h_1-1}(h_1+1) + p^3\sum_{h_1=1}^{\infty}(pq)^{h_1-1}h_1 \\
&= 2pq + 2p^2q^2(\frac{1}{(1-pq)^2} + \frac{1}{1-pq}) + (p^3+q^3)\frac{1}{(1-pq)^2} \\
&= \frac{1}{(1-pq)^2}(p^3+q^3+2pq) \\
&= \frac{1}{1-pq}.
\end{aligned}$$

We conclude $E_{TS}(P) = (2-p^3-q^3-4p^2q^2)E_{OPT}(P)$. The expression $2-p^3-q^3-4p^2q^2$ is maximal for $p=q=\frac{1}{2}$, and hence $E_{TS}(P) \le \frac{3}{2}E_{OPT}(P)$. ■

2.3 Performance against static offline 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 [5] to bound the ratio between $E_{TS}(\vec{p})$ and $E_{STAT}(\vec{p})$. We assume $p_1 \geq p_2 \geq \ldots \geq p_n$. As $E_{STAT}(\vec{p}) = \sum_i ip_i = \frac{1}{2}\sum_{i,j}\min(p_i, p_j) + \frac{1}{2}$, we have

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

The result is immediate from the following theorem:

Theorem 3 *If $x_i > 0\,(1 \leq i \leq n)$, then*

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

Proof: We shall 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, $H(x, y) = \frac{\partial^2 G}{\partial x \partial y}$, 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 $G(x, y) = \frac{xy(x^2 + 6xy + y^2)}{(x+y)^3}$. Without loss of generality, let $0 < x_1 < x_2 < \ldots < x_n$. 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:

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

We now apply Lemma 2; calculating the required integral numerically is a simple exercise, and we find

$$\int_0^\infty H^+(x, 1) x^{-1/2} dx \approx 1.338765....$$

∎

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

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

and the following version of Hilbert's inequality:

Theorem 4 (Hilbert's inequality) *For $p, q > 1$ satisfying $\frac{1}{p} + \frac{1}{q} = 1$, suppose that $K(x, y)$ is non-negative 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

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

For convenience, we restate the lemma:

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, $H(x,y) = \frac{\partial^2 G}{\partial x \partial y}$, and $H^+(x,y) = max\{H(x,y), 0\}$. Then*

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

Proof: Set $F(x) = \int_\infty^x f(x)dx$. Then $\int_0^\infty \int_0^\infty \min(x,y)f(x)f(y)dxdy = \int_0^\infty F^2(x)dx$. [Lemma 2 of [5]] Similarly,

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

The second to last 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 non-negative. In fact in Theorem 3 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 [3]. Motivated by this result, we consider a similar algorithm based on TS(0). We assume the reader is somewhat familiar with the system of [3], which was briefly described in the introduction.

3.1 Theoretical results

Let $B_{TS}(\vec{p})$ 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 $B_{TS}(\vec{p})$, we have to specify how an integer j should be encoded. We use a variable length prefix code by Elias [6] which encodes the integer j using $1 + \lfloor \log j \rfloor + 2\lfloor \log(1 + \log j) \rfloor$ bits. Bentley *et al.* [3] showed that, using this prefix code, the expected number of bits needed by the MTF algorithm is $B_{MTF}(\vec{p}) \le 1 + H(\vec{p}) + 2\log(1 + H(\vec{p}))$, for all $\vec{p}$. Here $H(\vec{p}) = \sum_{i=1}^n p_i \log(\frac{1}{p_i})$ is the entropy of the source. We prove similar bounds for TS(0).

Theorem 5 *For any $\vec{p} = (p_1, p_2, \ldots, p_n)$,*

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

where $\overline{H}(\vec{p}) = \sum_{i=1}^n p_i \log(\frac{1}{p_i}) + \log(1 - \sum_{1 \le i \le j \le n} \frac{p_i p_j (p_i - p_j)^2}{(p_i + p_j)^3})$.

Note that $0 \le \sum_{1 \le i \le j \le n} \frac{p_i p_j (p_i - p_j)^2}{(p_i + p_j)^3} < 1$. Therefore we have the following corollary.

Corollary 2 *For any $\vec{p} = (p_1, p_2, \ldots, p_n)$,*

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

where $H(\vec{p}) = \sum_{i=1}^n p_i \log(\frac{1}{p_i})$ is the entropy of the source.

Proof of Theorem 5: 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 [9] inequality, is at most $f(\sum_{j=1}^n q_{ij} j)$. Jensen's inequality states 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 $e_{TS}(x_i)$ of symbol x_i in TS(0)'s list. Therefore, $B_{TS}(p) \le \sum_{i=1}^n p_i f(e_{TS}(x_i))$. In the following we show that

$$\sum_{i=1}^n p_i \log(e_{TS}(x_i)) \le \overline{H}(p). \tag{4}$$

Using this inequality, we can easily derive Theorem 5 because

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

We now show inequality (4). By Theorem 1a), we have $e_{TS}(x_i) = \frac{1}{2} + \sum_{j=1}^n \frac{p_j^3 + 3p_j^2 p_i}{(p_i + p_j)^3}$ and

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

We have $\frac{1}{2} + \sum_{j=1}^n \frac{p_j}{p_i + p_j} = \frac{1}{p_i}(\frac{1}{2}p_i + \sum_{j=1}^n \frac{p_i p_j}{p_i + p_j}) \le \frac{1}{p_i}(p_i + \sum_{\substack{j=1 \\ j \ne i}}^n p_j) = \frac{1}{p_i}$. Therefore

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

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

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

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

■

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 $A_{TS}(S)$ be the average number of bits needed to encode one symbol in the string S using the TS(0) algorithm. Similarly, let $A_{MTF}(S)$ 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.* [3] show that for any input sequence S, $A_{MTF}(S) \le 1 + H(S) + 2\log(1 + H(S))$ where $H(S) = \sum_{i=1}^{n} \frac{m_i}{m} \log(\frac{m}{m_i})$ 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 6 *For any input sequence S,*

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

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

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 k-th 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 k-th 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 $[t_{k-1}+1, t_k]$. 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 $[t_{k-1}+1, t_k]$. Note that $d_k + 1 = pos_k$. Therefore, the k-th 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

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

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

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

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

The theorem follows immediately. ■

3.2 Simulation results

Our theoretical work suggests that a compression scheme similar to move-to-front using the TS(0) scheme may provide better performance. In effect, TS(0) is a conservative version of MTF encoding; like MTF-encoding, it responds well to locality of references by moving recently requested items to the front, but it responds more slowly. Understanding this intuition is important to understand where TS-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 a simple versions of TS-encoders and decoders for text compression. Our tests use standard documents from the Calgary Compression Corpus. The current goal of these tests is not to develop an all-purpose functional compression system, but merely to demonstrate the potential gains from using TS in place of MTF. The compression is performed by turning the document into a token stream. The tokens are then encoded by their position on the list using standard variable-length prefix encodings given by Elias [6]; each integer i requires $1 + 2\lfloor \log i \rfloor$ bits. We can compare the compression of MTF and TS 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-encoding outperforms MTF-encoding significantly on the sample documents; the improvement is typically 6 – 8 %. Moreover, in all cases TS-encoding beats MTF-encoding. However, this character-based compression scheme performs far worse than standard UNIX utilities, such as pack and compress.

In order to make TS(0) and MTF encoding comparable to 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 non-white 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. 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 [3].

The results of Table 2 reflect the compression achieved[1], including only the token stream and not the dictionary. As one might expect, the gains from TS in this situation are less dramatic; however, they do reach 1% on large documents. Surprisingly TS performs worse than MTF in one instance, underscoring an important point: TS is not guaranteed to perform better than MTF.

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 [4]. Unfortunately, the results here show less

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

File	TS(0)		MTF		Original	
	Bytes	% Orig.	Bytes	% Orig.	Bytes	% Orig.
bib	99121	**89.09**	106478	**95.70**	111261	**100.00**
book1	581758	**75.67**	644423	**83.83**	768771	**100.00**
book2	473734	**77.55**	515257	**84.35**	610856	**100.00**
geo	92770	**90.60**	107437	**104.92**	102400	**100.00**
news	310003	**82.21**	333737	**88.50**	377109	**100.00**
obj1	18210	**84.68**	19366	**90.06**	21504	**100.00**
obj2	229284	**92.90**	250994	**101.69**	246814	**100.00**
paper1	42719	**80.36**	46143	**86.80**	53161	**100.00**
paper2	63654	**77.44**	69441	**84.48**	82199	**100.00**
pic	113001	**22.02**	119168	**23.22**	513216	**100.00**
progc	33123	**83.62**	35156	**88.75**	39611	**100.00**
progl	52490	**73.26**	55183	**77.02**	71646	**100.00**
progp	37266	**75.47**	40044	**81.10**	49379	**100.00**
trans	79258	**84.59**	82058	**87.58**	93695	**100.00**

Table 1: MTF vs. TS : Byte-based compression

File	TS(0)		MTF		Original	
	Bytes	% Orig.	Bytes	% Orig.	Bytes	% Orig.
bib	34117	**30.66**	35407	**31.82**	111261	**100.00**
book1	286691	**37.29**	296172	**38.53**	768771	**100.00**
book2	260602	**42.66**	267257	**43.75**	610856	**100.00**
news	116782	**30.97**	117876	**31.26**	377109	**100.00**
paper1	15195	**28.58**	15429	**29.02**	53161	**100.00**
paper2	24862	**30.25**	25577	**31.12**	82199	**100.00**
progc	10160	**25.65**	10338	**26.10**	39611	**100.00**
progl	14931	**20.84**	14754	**20.59**	71646	**100.00**
progp	7395	**14.98**	7409	**15.00**	49379	**100.00**

Table 2: MTF vs. TS : Word-based compression

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 extremely high locality of reference. Given this, it is 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 Move-to-front 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) offline 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 be extended to the case that a request sequence is generated by a Markov chain, but for general Markov chains, we cannot do better than a competitive ratio of 2.

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 Move-to-front 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 stardard corpus of test files.

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 *Proc. of the 6th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 412–419, 1995.

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

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

[4] M. Burows and D.J. Wheeler. A block-sorting lossless data compression algorithm. DEC SRC Research Report 124, 1994.

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

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

[7] G.H. Gonnet, J.I. Munro and H. Suwanda. Towards self-organizing linear search. In *Proc. 19th Annual IEEE Symposium on Foundations of Computer Science*, pages 169–174, 1979.

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

[9] G.H. Hardy, J.E. Littlewood and G. Polya. *Inequalities*. Cambridge University Press, Cambridge, England, 1967.

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

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

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

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

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

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

[16] 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.