A New Perspective on List Update: Probabilistic Locality and Working Set

Reza Dorrigiv and Alejandro López-Ortiz

Cheriton School of Computer Science, University of Waterloo,
Waterloo, Ont., N2L 3G1, Canada,
rdorrigiv,alopez-o@uwaterloo.ca

Abstract. In this paper we study the performance of list update algorithms under arbitrary distributions that exhibit strict locality of reference and prove that MTF is the best list update algorithm under any such distribution. Furthermore, we study the working set property of online list update algorithms. The working set property indicates the good performance of an online algorithm on sequences with locality of reference. We show that no list update algorithm has the working set property. Nevertheless, we can distinguish among list update algorithms by comparing their performance in terms of the working set bound. We prove bounds for several well known list update algorithms and conclude that MTF attains the best performance in this context as well.

1 Introduction

The list update problem is one of the most studied online problems. It was first studied by McCabe [26] more than 45 years ago in the context of maintaining a sequential file. Since then, various list update algorithms have been proposed (e.g., [14, 30, 20, 9, 32, 23, 34, 29, 5, 2]) and different aspects of the problem have studied (e.g., [21, 28, 25, 6, 3, 18]). Despite this, there still are various interesting aspects of the problem not yet explored. In this paper we aim to provide new insights for the list update problem by studying the performance of list update algorithms under probabilistic and deterministic inputs with locality of reference.

Consider an unsorted list L of ℓ items. An online list update algorithm $\mathcal{A}$ is a strategy for reordering the elements of L after each access. The input to the algorithm is an *access sequence* $X = \langle x_1, x_2, \ldots, x_m \rangle$ that must be served in an online manner. To serve a request to an item x_j, $\mathcal{A}$ linearly searches the list until it finds x_j. If x_j is the i-th item in the list, $\mathcal{A}$ incurs a cost i to access x_j. Immediately after this access, $\mathcal{A}$ can move x_j to any position closer to the front of the list at no extra cost. This is called a *free exchange*. Also $\mathcal{A}$ can exchange any two consecutive items at a cost of 1. These are called *paid exchanges*. An efficient algorithm can thus use free and paid exchanges to minimize the overall cost of serving a sequence.

Three well-known deterministic online algorithms for list update are *Move-To-Front* (MTF), *Transpose*, and *Frequency-Count* (FC). MTF moves the requested item to the front of the list whereas Transpose exchanges the requested

item with the item that immediately precedes it. FC maintains an access count for each item ensuring that the list always contains items in non-increasing order of frequency of access. *Timestamp* (TS) is an efficient list update algorithm introduced by Albers [2]. After accessing an item a, TS inserts a in front of the first item b that is before a in the list and was requested at most once since the last request for a. If there is no such item b, or if this is the first access to a, TS does not reorganize the list.

In the early stages, list update algorithms were analyzed using the distributional or average-case model (e.g. [26, 14, 30, 9, 20]). In this model, the request sequences are generated according to a probability distribution and the efficiency of an algorithm is related to the expected cost it incurs. According to this model, FC is the best online list update algorithm, followed by Transpose and TS, and finally MTF. In contrast, in some real-life applications of list update, e.g., data compression [10, 13, 17], MTF has the best performance among these algorithms, and Transpose and FC have much worse performance than MTF and TS. This inconsistency can be explained by the fact that sequences for list update usually exhibit *locality of reference* [21, 11] and online list update algorithms try to take advantage of this property [21, 29]. A sequence has high locality of reference if a recently accessed item is more likely to be accessed in the near future. Summarizing experimental results on list update, Albers and Lauer [3] conclude that the performance and ranking of list update algorithms depend on the amount of locality in the input. In addition, it has been commonly assumed, based on intuition and experimental evidence, that MTF is the best algorithm on sequences with high locality of reference, e.g., Hester and Hirschberg [21] claim: "move-to-front performs best when the list has a high degree of locality" (see also [4], page 327). Although this was observed more than twenty years ago [21], only recently some theoretical models for list update with locality of reference have been proposed [6, 3, 16]. These models show the superiority of MTF to other online list update algorithms on sequences with high locality of reference.

However, to the best of our knowledge, no probabilistic model for list update with locality of reference has been proposed so far. We introduce such a model by refining the distributional analysis using the diffuse adversary model of Koutsoupias and Papadimitriou [24]. More specifically, we restrict the "acceptable" probability distributions to those with high locality of reference. So far, the diffuse adversary model has only been applied to paging algorithms [24, 8]. Under this model we prove the superiority of MTF and the non-optimality of static list update algorithms. Furthermore we show that the performance of MTF improves as the amount of locality increases.

We also study the *working set* property [33] of list update algorithms. The working set property is based on the idea that an operation on a recently accessed item should take less time. The working set property of most other self-organizing data structures has been studied before [33, 12, 22]. We show that although no list update algorithm has the working set property, their performance can be expressed in terms of the working set bound. Our analysis shows that MTF is the best list update algorithm in this setting. Considering the connection

between the working set property and locality, this result confirms (yet again) that MTF is the best online list update algorithm on sequences with high locality of reference.

2 List Update with Locality of Reference

In this section we refine the distributional model for analysis of list update algorithms by incorporating locality. First we provide more details about the distributional model and review the known results in this model. Let $L = (a_1, a_2, \ldots, a_\ell)$ be the list of items and $p = (p_1, p_2, \ldots, p_\ell)$ be a vector of positive probabilities with $\sum_{i=1}^{n} p_i = 1$. At each step, item a_i is requested with probability p_i. For a list update algorithm $\mathcal{A}$, let $E_{\mathcal{A}}(p)$ be the asymptotic expected cost of $\mathcal{A}$ in serving a single request in a request sequence generated by p. Traditionally, the performance of online list update algorithms was compared to that of the optimal static ordering, SOPT. SOPT knows the probability distribution and initially rearranges the items in non-increasing order of their probabilities and does not change their order afterwards. By the strong law of large numbers we have $E_{\mathrm{FC}}(p) = E_{SOPT}(p)$ for any p [30]. For MTF, Chung et al. [15] showed that for any probability distribution p, $E_{\mathrm{MTF}}(p) \leq (\pi/2)E_{SOPT}(p)$ and Gonnet et al. [19] showed that this bound is tight. Transpose outperforms MTF in this model: Rivest [30] proved that for any distribution p, we have $E_{Transpose}(p) \leq E_{\mathrm{MTF}}(p)$. For TS, we have $E_{\mathrm{MTF}}(p) \leq (\pi/2)E_{SOPT}(p)$ for any probability distribution p [4].

Therefore, FC is the best online list update algorithm in this model, followed by Transpose and TS, and finally MTF. As stated before, this is not consistent with experimental results and one apparent reason for this is the fact that the model does not incorporate locality of reference assumptions. In this section we analyze list update algorithms under probability distributions with locality of reference and show that MTF outperforms other algorithms under this model. Our model is based on the *diffuse adversary* model, in which we restrict the set of "acceptable" probability distributions.

Definition 1. *[24] Let $\mathcal{A}$ be an online algorithm for a minimization problem and let Δ be a class of distributions over the input sequences. Then $\mathcal{A}$ is c-competitive against Δ, if there exists a constant b, such that*

$$E_{\sigma \in D}[\mathcal{A}(\sigma)] \leq c \cdot E_{\sigma \in D}[OPT(\sigma)] + b,$$

for every distribution $D \in \Delta$, where $\mathcal{A}(\sigma)$ denotes the cost of $\mathcal{A}$ on the input sequence σ and $E[\,]$ denotes the expectation under D.

We model locality of reference by considering a class Δ of sequences that exhibit locality of reference. Let $L = (a_1, a_2, \ldots, a_\ell)$ be the list of items and σ be a sequence of requests to the items. We define $p(a_i, \sigma)$, the probability of accessing item a_i after the sequence σ, in a way that reflects locality of reference. The idea is to favour recently accessed items. Let the *age* of an item a_i in a

sequence σ, denoted by $age(a_i, \sigma)$, be j if a_i is the j-th most recently accessed item in σ. To handle the case that an item is not requested in σ, we assume that all sequences are prepended by the sequence $a_\ell, a_{\ell-1}, \ldots, a_1$. For example, for the empty sequence ε we have $age(a_i, \varepsilon) = i$. Observe that the items have unique ages between 1 and ℓ, i.e., the set of ages at each time is exactly $\{1, 2, \ldots, \ell\}$. Now we define probability of accessing a_i after σ in terms of the age of a_i in σ: $p(a_i, \sigma) = f(p(a_i, \sigma))$, where the non-increasing function f is a probability distribution on $\{1, 2, \ldots, \ell\}$. Observe that in contrast to the traditional probabilistic models for list update, we consider a dynamic probability distribution on items. By requiring f to be non-increasing we ensure that more recently accessed items are more probable, thus reflecting the locality of reference assumption. Furthermore, we can measure the amount of locality of such probability distributions. Define a random variable X_f such that $X_f = x$ with probability $f(x)$. The expected value of X_f, $E[X_f] = \sum_{i=1}^{\ell} i \cdot f(i)$ can be considered as a measure for the amount of non-locality of reference. if $E[X_f]$ is small then we know that the probability of requesting most recently accessed items is much higher than accessing the rest of items. We also require $f(i) > 0$ for $1 \leq i \leq \ell$ to ensure that all items can be accessed. The following examples show different possible amounts of locality.

1. Consider the probability distribution $f_1(i) = 1/\ell$ for $1 \leq i \leq \ell$. Intuitively, f_1 does not have much locality and we have $E[X_{f_1}] = \sum_{1 \leq i \leq \ell} (i/\ell) = (\ell+1)/2$, which is a relatively large number. Actually, this is the largest $E[X_f]$ for non-increasing probability distributions on $\{1, 2, \ldots, \ell\}$.
2. Consider the probability distribution f_2 for which $f_2(i+1) = f_2(i)/2$, i.e., the probability of accessing an item is halved as its age is increased by one unit. It can be proved that $E[X_{f_2}] < 2$, so f_2 has constant expected value and high amount of locality.
3. Let f_3 be the Zipfian distribution $f_3(i) = \alpha/i$, i.e., probability of accessing an item is inversely proportional to its age. We have

$$\sum_{i=1}^{\ell} f_3(i) = 1 \Rightarrow \sum_{i=1}^{\ell} \frac{\alpha}{i} = 1 \Rightarrow \alpha H_\ell = 1 \Rightarrow \alpha = 1/H_\ell,$$

where H_ℓ is the ℓ-th Harmonic number. We have $f_3(i) = \frac{1}{iH_\ell}$ and

$$E[X_{f_3}] = \sum_{i=1}^{\ell} i \cdot f_3(i) = \sum_{i=1}^{\ell} i \cdot \frac{1}{iH_\ell} = \frac{\ell}{H_\ell}.$$

Thus the expected value of f_3 is between the expected values of f_1 and f_2.

We computed the empirical probability of accessing items in terms of their ages in the files of Calgary Corpus [35] and Canterbury Corpus [1], which are the standard benchmarks for data compression. As stated before list update algorithms are widely used in data compression. These two corpora include files of different types such as English text (technical writing, poetry, fiction and non-fiction books), source code in various programming languages, picture

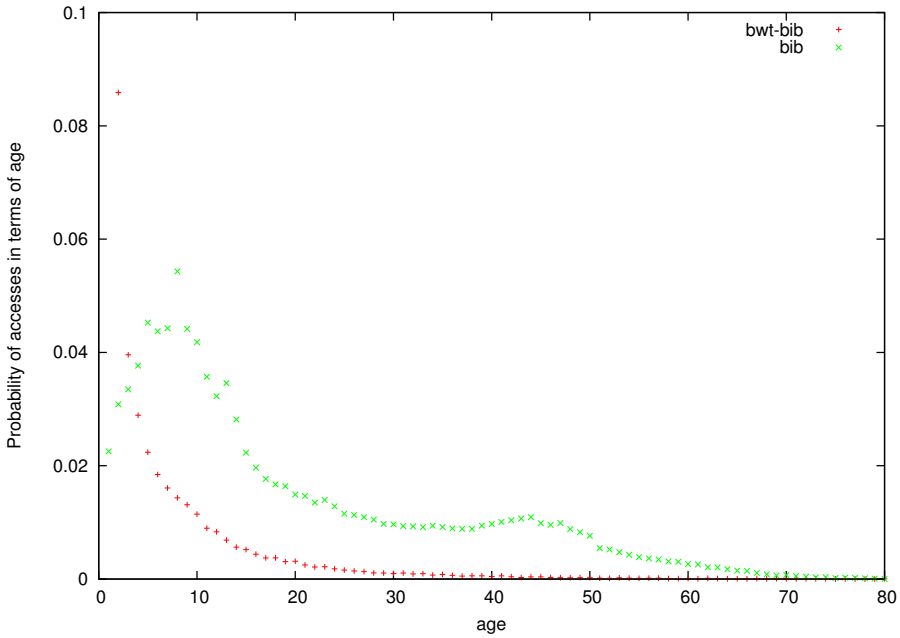

Fig. 1. Prob. of accessing items in terms of their age in file **bib** before and after BWT. The prob. of accessing the youngest age after BWT (0.67) is off-scale and thus not shown.

files, object code, and spreadsheets. We computed the empirical probabilities for files before and after *Burrows-Wheeler Transform* (BWT). The Burrows-Wheeler transform (BWT) rearranges a string of symbols to one of its permutations that is believed to have more locality of reference. Then list update algorithms are used to encode this transform. The well known compression program bzip2 [31] is based on the BWT. The results for file *bib* in Calgary Corpus are shown in Figure 1. The results for other files are very similar and can be found in the appendix. Observe from Figure 1 that after BWT the probability distribution f is non-increasing and has a very low $E[X_f]$, i.e., high locality, while before BWT the function is increasing at some intervals and has a much higher expected value. This confirms our intuition that BWT increases the amount of locality.

Observe that the probability of accessing a particular item changes over time in our model. Thus $E_{\mathcal{A}}(p)$ could be different at different times and we need to incorporate time in the definition. We define $E^t_{\mathcal{A}}(f)$ to be the expected cost of $\mathcal{A}$ in serving the t-th request in a sequence generated under f. It is not obvious whether $E^t_{\mathcal{A}}(f)$ converges as $t \to \infty$. We define $E_{\mathcal{A}}(f)$ to be $\lim_{t\to\infty} E^t_{\mathcal{A}}(f)$ if the corresponding limit exists. Observe that MTF maintains the items in decreasing order of their ages. Thus the cost of MTF on an item is exactly the age of that item and we have $E^t_{\mathrm{MTF}}(f) = E[X_f]$ for every t. Therefore $E_{\mathrm{MTF}}(f) = E[X_f]$. If we have high locality of reference, then $E[X_f]$ is small and the expected cost of

MTF will be low. Hence, MTF has good performance on sequences with locality of reference as expected. Good performance of MTF in this model is due to the fact that MTF tries to take advantage of locality by moving younger items to the front of its list. On the other hand, static strategies do not adapt to locality and so we expect them not to be optimal in the new model even if they know the distribution. This intuition is formalized in the following Lemma.

Lemma 1. *Let $\mathcal{A}$ be a static list update algorithm. Then there exists a non-increasing function f such that $E_{\mathcal{A}}(f) > E_{\mathrm{MTF}}(f)$, even if $\mathcal{A}$ knows f.*

Proof. Define the function f as follows: $f(1) = 0.9$ and $f(i) = 0.1/(l-1)$ for $2 \leq i \leq \ell$. We have

$$E_{\mathrm{MTF}}(f) = 0.9 \times 1 + \frac{0.1}{l-1}(2 + 3 + \cdots + \ell) = \ell/20 + 1.$$

$\mathcal{A}$ can rearrange the list $(a_1, a_2, \ldots, a_\ell)$ at the beginning and then cannot change the order of the items. Since a_1 is the youngest item at the beginning it seems reasonable for $\mathcal{A}$ to leave a_1 at the front of the list. So assume that this is the case. We have $E^1_{\mathcal{A}}(f) = \ell/20 + 1$ and so $\mathcal{A}$ has the same expected cost as MTF on the first request. In order to compute the asymptotic performance of $\mathcal{A}$, we define a two-state Markov chain as follows. We are in state A if we have a request to a_1 and we are in state B otherwise. If we are at state A, the probability of staying at A is 0.9 and the probability of going to B is 0.1. If we are at B, the probability of going to A is $0.1/(\ell - 1)$ and the probability of staying at B is $1 - 0.1/(\ell - 1)$. Let $[q_1 \quad q_2]$ be the stationary distribution of the Markov chain. We have

$$q_1 \times (-0.1) + q_2 \times \frac{0.1}{l-1} = 0 \Rightarrow q_1 = \frac{q_2}{l-1}.$$

Furthermore, $q_1 + q_2 = 1$ and so $q_1 = 1/\ell$ and $q_2 = 1 - 1/\ell$. Therefore asymptotically we have

$$E_{\mathcal{A}}(f) = (1/\ell) \times 1 + (1 - 1/\ell) \times \frac{2 + 3 + \cdots + \ell}{\ell - 1} = \frac{\ell + 1}{2}.$$

Thus the asymptotic expected cost of $\mathcal{A}$ is 10 times more than that of MTF.

Thus MTF outperforms any static strategy. Actually, we can show that the performance of a static list update algorithm is the same for any function f.

Theorem 1. *Let $\mathcal{A}$ be a static list update algorithm and f be an arbitrary probability distribution on $\{1, 2, \ldots, \ell\}$. We have $E_{\mathcal{A}}(f) = \frac{\ell+1}{2}$.*

Proof. We define a Markov chain based on f. We sort items by their age and consider a single state for any permutation of $(a_1, a_2, \ldots, a_\ell)$. So originally we are at state $(a_1, a_2, \ldots, a_\ell)$. From this state we move to state $(a_2, a_1, a_3, \ldots, a_\ell)$ with probability $f(2)$, to state $(a_3, a_1, a_2, a_4, \ldots, a_\ell)$ with probability $f(3)$,...,and to state $(a_\ell, a_1, \ldots, a_{\ell-1})$ with probability $f(\ell)$. The Markov chain has $l!$ states and we remain in the same state with probability $f(1)$. Let M be the transition

matrix of the Markov chain and $[q_1\ q_2\ \ldots\ q_{\ell!}]$ be its stationary distribution. Since we have $f(i) > 0$ for $1 \leq i \leq \ell$, this Markov chain is irreducible and aperiodic. Consider an arbitrary state $(a_{i_1}, a_{i_2}, \ldots, a_{i_\ell})$. We move to this state from $(a_{i_2}, a_{i_1}, \ldots, a_{i_\ell})$ with probability $f(2)$, from $(a_{i_2}, a_{i_3}, a_{i_1}, \ldots, a_{i_\ell})$ with probability $f(3)$,..., and from $(a_{i_2}, a_{i_3}, \ldots, a_{i_\ell}, a_{i_1})$ with probability $f(l)$. Thus the column corresponding to this state in M sums to $f(1) + f(2) + \cdots + f(\ell) = 1$. Therefore M is doubly stochastic. It is known (e.g., [27], page 157) that the stationary distribution of doubly stochastic matrices is the uniform distribution. Therefore we have $q_j = \frac{1}{\ell!}$ for $1 \leq j \leq \ell!$. Let $(a_{j_1}, a_{j_2}, \ldots, a_{j_\ell})$ be the static list maintained by $\mathcal{A}$. a_{j_k} appears as the i-th item in exactly $(\ell-1)!$ states of the Markov chain, for $1 \leq i \leq \ell$. If we are in such a state, the probability of accessing a_{j_k} is $f(i)$. Thus the asymptotic probability of accessing a_{j_k} is $\sum_{i=1}^{\ell}(\ell-1)! \cdot \frac{1}{\ell!} \cdot f(i) = \frac{1}{\ell}(f(1) + f(2) + \cdots + f(\ell)) = 1/\ell$. The cost of $\mathcal{A}$ on a_{j_k} is k and the asymptotic expected cost of $\mathcal{A}$ is $\sum_{k=1}^{\ell} k/\ell = (\ell+1)/2$.

This theorem shows that the performance of static strategies does not improve by increasing the amount of locality. For instance, for the probability distribution f_2 defined above we have $E_{\mathcal{A}}(f_2) = (\ell+1)/2$ while $E_{\mathrm{MTF}}(f_2)$ is a constant smaller than 2. Next we prove that MTF has the best possible performance and cannot be beaten by any other strategy.

Lemma 2. *Let $\mathcal{A}$ be a list update algorithm, $t > 0$, and f be a non-increasing probability distribution. We have $E^t_{\mathrm{MTF}}(f) \leq E^t_{\mathcal{A}}(f)$.*

Proof. Let σ be an arbitrary sequence of length $t-1$ and $(a_1, a_2, \ldots, a_\ell)$ be the list maintained by MTF after serving σ. We know that $Pr[a_i|\sigma] \geq Pr[a_{i+1}|\sigma]$, i.e., after requesting σ the probability of requesting a_i is at least the probability of requesting a_{i+1}, for $1 \leq i \leq \ell - 1$. Therefore we have

$$\sum_{i=1}^{\ell} (Pr[a_i|\sigma] \times \mathrm{MTF}^t(\sigma \cdot a_i)) \leq \sum_{i=1}^{\ell} (Pr[a_i|\sigma] \times \mathcal{A}^t(\sigma \odot a_i)),$$

where $\mathcal{A}^t(\sigma \odot a_i)$ denotes the cost incurred by $\mathcal{A}$ in serving the t-th request of $\sigma \odot a_i$, i.e., the sequence obtained by appending a_i to σ. Observe that we have $\mathrm{MTF}^t(\sigma \odot a_i) = i$. Since this holds for any sequence σ of length $t-1$, we conclude that $E^t_{\mathrm{MTF}}(f) \leq E^t_{\mathcal{A}}(f)$.

3 Working Set Property for List Update

In this section we study the performance of list update algorithms in terms of the *working set bound*. Consider the access sequence $X = \langle x_1, x_2, \cdots, x_m \rangle$. The working set number of an item z at time i, $t_i(z)$, is the number of distinct items that are requested since the last request to z (including z) or the number of distinct items that are requested so far if this is the first access to z. The working set bound of X is defined as $WS(X) = \sum_{i=1}^{m} \log(t_i(x_i) + 1)$.[1] If the

[1] In this paper all logarithms are base 2.

total cost of X in a data structure is $O(WS(X))$ (or equivalently, the amortized cost of x_i is $O(\log(t_i(x_i)+1))$ we say that data structure has the working set property. Observe that we have $t_i(x_i) = age(x_1x_2\ldots x_{i-1}, x_i)$ and so there is a close relationship between the working set bound and the probabilistic model for locality in the previous section.

As stated before, list update algorithms are used in data compression. As noted in [17], the cost model is different in this case: the cost of encoding an item in position i is $\Theta(\log i)$. It is not hard to see that under this logarithmic cost model MTF has the working set property as $t_i(x_i)$ is the position of x_i in the list maintained by MTF at time i. In contrast, the following lemma shows that no online list update algorithm in the standard cost model has the working set property.

Lemma 3. *Let $\mathcal{A}$ be an online list update algorithm. There is an access sequence X such that $\mathcal{A}(X) \geq \frac{\ell}{\log(\ell+1)} \cdot WS(X)$.*

Proof. Consider an access sequence X of length m obtained by requesting the item that is in the last position of list maintained by $\mathcal{A}$ at each time. We have $\mathcal{A}(X) = m \cdot \ell$. Also we have $t_i(x_i) \leq \ell$ for $1 \leq i \leq m$, because we do not have more than ℓ distinct items. Therefore $WS(X) \leq m \cdot \log(\ell+1)$, and $\mathcal{A}(X) \geq \frac{WS(X)}{\log(\ell+1)} \cdot \ell$.

However we can still rank list update algorithms by comparing how far their performance is from the working set bound. First we prove a general upper bound.

Lemma 4. *Let $\mathcal{A}$ be an online list update algorithm. For any access sequence X we have $\mathcal{A}(X) \leq \ell \cdot WS(X)$.*

Proof. Consider an arbitrary sequence X of length m. Since the maximum cost that $\mathcal{A}$ incurs on a request is ℓ, we have $\mathcal{A}(X) \leq m \cdot \ell$. At the same time clearly $t_i(x_i) \geq 1$ for any i. Thus we have $WS(X) \geq m \cdot \log 2 = m$ which implies $\mathcal{A}(X) \leq \ell \cdot WS(X)$.

The following lemma shows that MTF achieves the best possible performance in terms of the working set bound.

Lemma 5. *For any access sequence X we have* $\mathrm{MTF}(X) \leq \frac{\ell}{\log(\ell+1)} \cdot WS(X)$.

Proof. Consider an arbitrary access sequence X of length m. We have

$$\frac{\mathrm{MTF}(X)}{WS(X)} = \frac{\sum_{i=1}^{m} t_i(x_i)}{\sum_{i=1}^{m} \log(t_i(x_i)+1)},$$

where $1 \leq t_i(x_i) \leq \ell$. Since the terms in the numerator grow exponentially compared to terms in the denominator, this expression takes its maximum when we have $t_i(x_i) = \ell$ for $1 \leq i \leq m$. Therefore

$$\frac{\mathrm{MTF}(X)}{WS(X)} \leq \frac{\sum_{i=1}^{m} \ell}{\sum_{i=1}^{m} \log(\ell+1)} = \frac{m \cdot \ell}{m \cdot \log(\ell+1)} = \frac{\ell}{\log(\ell+1)},$$

which implies $\mathrm{MTF}(X) \leq \frac{\ell}{\log(\ell+1)} \cdot WS(X)$.

Other list update algorithms do not behave optimally in terms of the working set bound.

Theorem 2. *In the worst case we have*
a) $Transpose(X) \geq \frac{\ell}{\log 3} \cdot WS(X)$.
b) $\mathrm{FC}(X) \geq \frac{\ell+1}{2} \cdot WS(X)$.
c) $\mathrm{TS}(X) \geq \frac{2\ell}{\log(\ell+1)+1} \cdot WS(X)$

Proof. Due to space constraints, we only provide the proof of part (c) here. The proofs of the other two parts can be found in the appendix. Let $\mathcal{L}_0 = (a_1, a_2, \ldots, a_\ell)$ be the initial list and k be an arbitrary integer. Consider the access sequence X obtained by repeating k times the block $a_\ell^2 a_{\ell-1}^2 \ldots a_1^2$. Let B be such a block in X. Each item a_i is accessed twice in B. TS does not move a_i after its first access in B, because all other items have been accessed twice since the last access to a_i. After the second access, TS moves the item to the front of the list. Therefore each access is to the last item of the list and TS incurs a cost of ℓ on each access. We have $\mathrm{TS}(X) = 2k \cdot \ell^2$. Next we compute $WS(X)$. The first and second access to a_i in block B contributes $\log(\ell+1)$ and $\log 2$ to $WS(X)$, respectively. Considering the special case of the first block, we have $WS(X) = \ell + \sum_{i=2}^{\ell+1} \log i + (k-1) \cdot \ell(\log(\ell+1) + \log 2)$. Therefore

$$\frac{\mathrm{TS}(X)}{WS(X)} = \frac{2k \cdot \ell^2}{\ell + \sum_{i=2}^{\ell+1} \log i + (k-1) \cdot \ell(\log(\ell+1) + 1)},$$

which becomes arbitrarily close to $\frac{2\ell}{\log(\ell+1)+1}$ as k grows.

We can also analyze the performance of randomized list update algorithms in terms of the working set bound by considering their expected cost. Algorithm *Bit*, introduced by Reingold et al. [29], is a simple randomized algorithm that achieves a competitive ratio 1.75, thus beating any deterministic algorithm [11]. Bit allocates a bit $b(a_i)$ for each item a_i and initializes these bits uniformly and independently at random. Upon an access to a_i, it first complements $b(a_i)$, then if $b(a_i) = 0$ it moves a_i to the front, otherwise it does nothing. The following lemma shows that although randomization (for Bit) can improve the competitive ratio it cannot lead to the working set property. In fact the performance of Bit in terms of the working set bound is worse than MTF. This is inconsistent with competitive analysis but consistent with experimental results [7].

Lemma 6. *In the worst case* $E(Bit(X)) \geq \frac{3\ell+1}{2(\log(\ell+1)+1)} \cdot WS(X)$.

Proof. Let $\mathcal{L}_0 = (a_1, a_2, \ldots, a_\ell)$ be the initial list and k be an arbitrary integer. Consider the access sequence $X = \{a_\ell^2 a_{\ell-1}^2 \ldots a_1^2\}^k$. Let x_i and x_{i+1} be two consecutive accesses to a_j. After two consecutive accesses to each item, a_j will have been moved to the front of the list with probability 1. Therefore a_j is in the last position of the list maintained by Bit at the time of request x_i and Bit incurs cost ℓ on this request. After this request, Bit moves a_j to the front of the

list if and only if $b(a_j)$ is initialized to 1. Since $b(a_j)$ is initialized uniformly and independently at random, this will happen with probability 1/2. Therefore the expected cost of Bit on x_{i+1} is $\frac{1}{2}(\ell+1)$ and the expected cost of Bit on X is $k \cdot \ell(\ell + \frac{\ell+1}{2})$. We have $WS(X) = \ell + \sum_{i=2}^{\ell+1} \log i + (k-1) \cdot \ell(\log(\ell+1)+1)$. Therefore

$$\frac{E(Bit(X))}{WS(X)} = \frac{k \cdot \ell(\ell + \frac{\ell+1}{2})}{\ell + \sum_{i=2}^{\ell+1} \log i + (k-1) \cdot \ell(\log(\ell+1)+1)},$$

which becomes arbitrary close to $\frac{3\ell+1}{2(\log(\ell+1)+1)}$ as k grows.

Table 1. Working set bounds of files in Canterbury Corpus (normalized by their sizes) before and after Burrows-Wheeler Transform

File	Category	Size(bytes)	l	WS/n	$WS/$n (BWT)
alice29.txt	English text	152089	74	3.9	2.0
asyoulik.txt	Shakespeare play	125179	68	3.6	1.8
cp.html	HTML source	24603	86	3.8	1.8
fields.c	C source	11150	90	3.5	1.6
grammar.lsp	LISP source	3721	76	3.3	1.7
kennedy.xls	Excel Spreadsheet	1029744	256	2.6	1.5
lcet10.txt	Technical writing	426754	84	3.4	1.6
plrabn12.txt	Poetry	481861	81	3.5	1.8
ptt5	CCITT test set	513216	159	1.4	1.1
sum	SPARC Executable	38240	255	3.1	1.7
xargs.1	GNU manual page	4227	74	3.6	1.9

4 Experimental Results

In this section we compute the working set bound for some real life inputs for list update and study the performance of well known list update algorithms in terms of the working set bound. We computed the working set of files of Calgary and Canterbury corpora before and after BWT. Table 1 shows the results for the Canterbury Corpus. The results for the Calgary Corpus are similar and are shown in Table 3 in the appendix. From these results we conclude that the working set bound for BWT of each file is much less than the working set bound of the original file. This reflects the intuition that BWT increases the amount of locality of reference in a sequence.

We also computed the performance of list update algorithms on the BWT of these files. Table 2 shows the corresponding costs normalized by the working set bound of each file. Comparing the experimental results with the theoretical bounds we proved in Section 3 shows that the actual performance of the algorithms is much better than the theoretical worst case bounds. In particular, the worst case lower bound of $\frac{l}{\log l+1}$ seems pessimistic. Furthermore, MTF and TS have close performance and outperform FC and TR. This is consistent with our theoretical results.

Table 2. Performance of list update algorithm (normalized by the working set bound) on files of Canterbury and Calgary Corpora after Burrows-Wheeler Transform

File	MTF	TS	FC	TR	$\frac{l}{\log l+1}$
alice29.txt	1.98	2.04	5.80	2.67	11.88
asyoulik.txt	2.10	2.14	6.24	2.79	11.13
cp.html	2.96	3.31	8.20	6.09	13.23
fields.c	2.66	3.43	8.96	9.23	13.83
grammar.lsp	3.04	3.91	6.42	10.57	12.13
kennedy.xls	3.90	3.79	5.55	5.15	22.15
lcet10.txt	1.93	1.97	6.41	2.48	13.10
plrabn12.txt	1.99	1.96	5.26	2.21	12.74
ptt5	1.23	1.20	1.31	1.22	12.25
sum	3.51	4.02	10.00	8.37	18.26
xargs.1	3.04	3.64	6.38	9.01	11.88

File	MTF	TS	FC	TR	$\frac{l}{\log l+1}$
bib	2.18	2.38	9.22	3.78	12.74
book1	1.98	1.92	5.28	2.16	12.86
book2	2.03	2.12	6.90	2.83	14.55
geo	5.66	5.35	6.07	5.74	18.26
news	2.68	2.95	8.34	4.02	14.78
obj1	4.86	5.04	7.45	8.74	18.26
obj2	3.07	3.40	9.60	5.88	18.26
paper1	2.44	2.82	7.32	5.14	14.43
paper2	2.19	2.34	5.64	3.58	13.95
pic	1.79	1.71	2.18	2.04	21.38
progc	2.77	3.22	8.90	6.35	14.07
progl	2.03	2.38	7.66	4.04	13.47
progp	2.16	2.72	9.03	5.38	13.71
trans	2.13	2.75	13.12	5.76	14.90

5 Conclusions

We introduced a probabilistic model for list update with locality of reference. This model is based on the diffuse adversary model and considers a dynamic probability distribution for accessing the items. We proved that MTF outperforms other algorithms in this model and its performance improves as the locality increases. Analyzing other list update algorithms under this model remains open. Furthermore, we analyzed several online list update algorithms in terms of the working set bound. We proved that MTF achieves the optimal performance in terms of the working set bound, while several other algorithms do not. Thus, both these models confirms the well known belief that MTF is the best list update algorithm on sequences with high locality of reference. Our experiments showed that the working set bound of files decreases after applying BWT. This is consistent with our intuition that BWT increases locality of reference.

References

1. The canterbury corpus. Available at http://corpus.canterbury.ac.nz/index.html.
2. S. Albers. Improved randomized on-line algorithms for the list update problem. *SICOMP*, 27(3):682–693, 1998.
3. S. Albers and S. Lauer. On list update with locality of reference. In *ICALP*, pages 96–107, 2008.
4. S. Albers and M. Mitzenmacher. Average case analyses of list update algorithms, with applications to data compression. *Algorithmica*, 21(3):312–329, 1998.
5. S. Albers, B. von Stengel, and R. Werchner. A combined BIT and TIMESTAMP algorithm for the list update problem. *IPL*, 56:135–139, 1995.
6. S. Angelopoulos, R. Dorrigiv, and A. López-Ortiz. List update with locality of reference. In *LATIN*, pages 399–410, 2008.
7. R. Bachrach and R. El-Yaniv. Online list accessing algorithms and their applications: Recent empirical evidence. In *SODA*, pages 53–62, 1997.
8. L. Becchetti. Modeling locality: A probabilistic analysis of LRU and FWF. In *ESA*, pages 98–109, 2004.

9. J. Bentley and C. McGeoch. Amortized analyses of self-organizing sequential search heuristics. *CACM*, 28:404–411, 1985.
10. J. L. Bentley, D. D. Sleator, R. E. Tarjan, and V. K. Wei. A locally adaptive data compression scheme. *CACM*, 29:320–330, 1986.
11. A. Borodin and R. El-Yaniv. *Online Computation and Competitive Analysis*. Cambridge University Press, 1998.
12. P. Bose, K. Douïeb, and S. Langerman. Dynamic optimality for skip lists and B-trees. In *SODA*, pages 1106–1114, 2008.
13. M. Burrows and D. J. Wheeler. A block-sorting lossless data compression algorithm. Technical Report 124, DEC SRC, 1994.
14. P. Burville and J. Kingman. On a model for storage and search. *Journal of Applied Probability*, 10:697–701, 1973.
15. F. R. Chung, D. J. Hajela, and P. D. Seymour. Self-organizing sequential search and hilbert's inequalities. In *STOC*, pages 217–223, 1985.
16. R. Dorrigiv, M. R. Ehmsen, and A. López-Ortiz. Parameterized analysis of paging and list update algorithms. In *WAOA*, pages 104–115, 2009.
17. R. Dorrigiv, A. López-Ortiz, and J. I. Munro. An application of self-organizing data structures to compression. In *SEA*, pages 137–148, 2009.
18. M. Ehmsen, J. Kohrt, and K. Larsen. List factoring and relative worst order analysis. In *WAOA*, pages 118–129, 2010.
19. G. H. Gonnet, J. I. Munro, and H. Suwanda. Toward self-organizing linear search. In *FOCS*, pages 169–174, 1979.
20. G. H. Gonnet, J. I. Munro, and H. Suwanda. Towards self-organizing linear search. In *FOCS*, pages 169–174. IEEE, 1979.
21. J. H. Hester and D. S. Hirschberg. Self-organizing linear search. *ACM Computing Surveys*, 17(3):295, Sept. 1985.
22. J. Iacono. Improved upper bounds for pairing heaps. In *SWAT*, pages 32–45, 2000.
23. S. Irani. Two results on the list update problem. *IPL*, 38:301–306, 1991.
24. E. Koutsoupias and C. Papadimitriou. Beyond competitive analysis. *SICOMP*, 30(1):300–317, 2000.
25. C. Martínez and S. Roura. On the competitiveness of the move-to-front rule. *Theoretical Computer Science*, 242(1–2):313–325, 2000.
26. J. McCabe. On serial files with relocatable records. *Op. Res.*, 12:609–618, 1965.
27. R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.
28. J. I. Munro. On the competitiveness of linear search. In *ESA*, pages 338–345, 2000.
29. N. Reingold, J. Westbrook, and D. Sleator. Randomized competitive algorithms for the list update problem. *Algorithmica*, 11:15–32, 1994.
30. R. Rivest. On self-organizing sequential search heuristics. *CACM*, 19:63–67, 1976.
31. J. Seward. bzip2, a program and library for data compression. http://www.bzip.org/.
32. D. D. Sleator and R. E. Tarjan. Amortized efficiency of list update and paging rules. *CACM*, 28:202–208, 1985.
33. D. D. Sleator and R. E. Tarjan. Self-adjusting binary search trees. *Journal of the ACM*, 32(3):652–686, 1985.
34. B. Teia. A lower bound for randomized list update algorithms. *IPL*, 47:5–9, 1993.
35. 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.

A Proofs

Theorem 2

In the worst case we have
a) $Transpose(X) \geq \frac{\ell}{\log 3} \cdot WS(X)$.
b) $\mathrm{FC}(X) \geq \frac{\ell+1}{2} \cdot WS(X)$.
c) $\mathrm{TS}(X) \geq \frac{2\ell}{\log(\ell+1)+1} \cdot WS(X)$

Proof. The proof of part (c) is provided in the paper. Here we provide the proofs of the other two parts.
Proof of part (a) Let $\mathcal{L}_0 = (a_1, a_2, \ldots, a_\ell)$ be the initial list. Consider a sequence X of length m obtained by several repetitions of the pattern $a_\ell a_{\ell-1}$. Then $Transpose(X) = m \cdot \ell$. Observe that $t_1(x_1) = 1$ and $t_i(x_i) = 2$ for $2 \leq i \leq m$. Therefore $WS(X) = 1 + \sum_{i=2}^{m} \log(2+1) = 1 + (m-1) \cdot \log 3$, and

$$\frac{Transpose(X)}{WS(X)} = \frac{m \cdot \ell}{1 + (m-1) \cdot \log 3} \geq \frac{m \cdot \ell}{m \cdot \log 3} = \frac{\ell}{\log 3}.$$

Proof of part (b) Let $\mathcal{L}_0 = (a_1, a_2, \ldots, a_\ell)$ be the initial list and k be an arbitrary integer. Consider the following access sequence: $X = a_1^k a_2^{k-1} a_3^{k-2} \ldots a_\ell^{k-\ell+1}$. On serving X, FC does not change the order of items in its list and incurs cost

$$\sum_{i=1}^{\ell} (k-i+1) \times i = \frac{k \cdot \ell(\ell+1)}{2} + \frac{\ell(1-\ell^2)}{3}.$$

We have $WS(X) = (k-1) \cdot \log 2 + (k-2) \cdot \log 2 + \cdots + (k-\ell) \cdot \log 2 + \sum_{i=2}^{\ell+1} \log i = k\ell - \ell(\ell+1)/2 + \sum_{i=2}^{\ell+1} \log i$. Therefore

$$\frac{\mathrm{FC}(X)}{WS(X)} = \frac{k\ell(\ell+1)/2 + l(1-\ell^2)/3}{k\ell - \ell(\ell+1)/2 + \sum_{i=2}^{\ell+1} \log i}.$$

Since k can be selected to be arbitrary larger than ℓ, we get

$$\frac{\mathrm{FC}(X)}{WS(X)} \geq \frac{k\ell(\ell+1)/2}{k\ell} = \frac{\ell+1}{2}.$$

B Empirical Probability of Access

In this section we provide the graphs for empirical probability of accessing items in terms of their ages in the files of Calgary and Canterbury Corpora. Figures 2–10 show the corresponding graphs for **book1**, **book2**, **news**, **paper1**, **paper2**, **progc**, **progl**, **progp**, and **trans**, respectively.

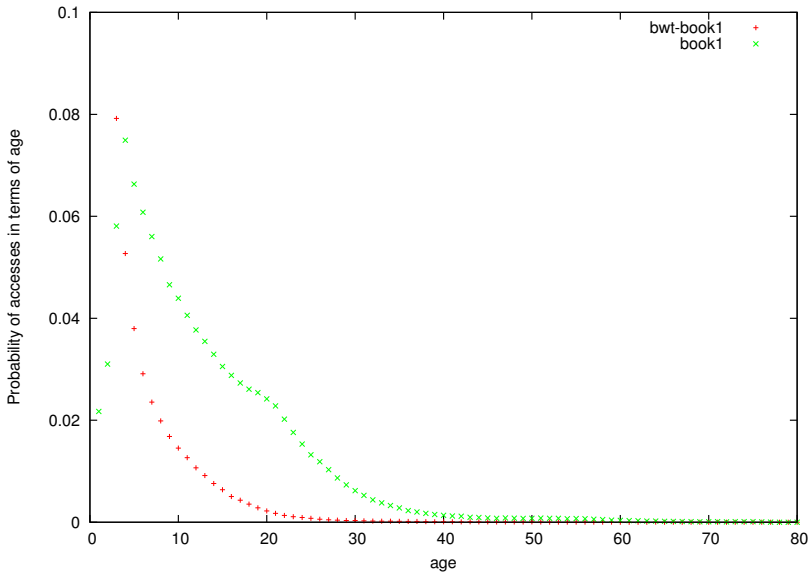

Fig. 2. Prob. of accessing items in terms of their age in file **book1** before and after BWT. For bwt-book1, $f(1) = 0.49$ and $f(2) = 0.15$ are off-scale and thus not shown.

C Experimental Results on Files of Calgary Corpus

Table 3 shows the working set bound of files in Calgary Corpus before and after BWT. These results are similar to the experimental results presented in Section 4 for files of Canterbury Corpus.

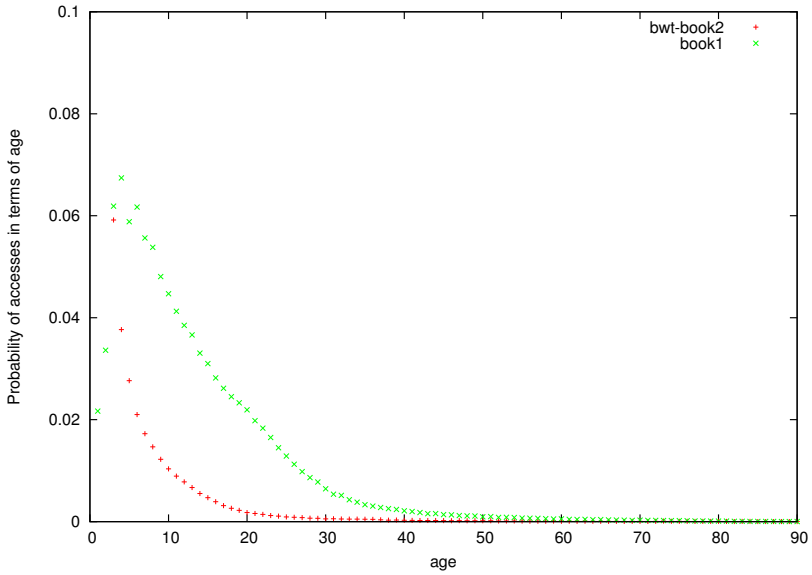

Fig. 3. Prob. of accessing items in terms of their age in file **book2** before and after BWT. For bwt-book2, $f(1) = 0.60$ and $f(2) = 0.12$ are off-scale and thus not shown.

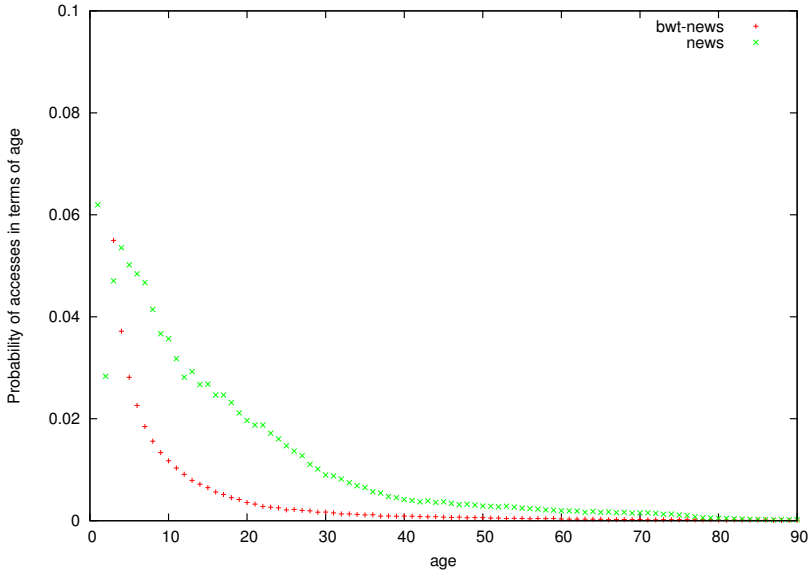

Fig. 4. Prob. of accessing items in terms of their age in file **news** before and after BWT. For bwt-news, $f(1) = 0.57$ and $f(2) = 0.11$ are off-scale and thus not shown.

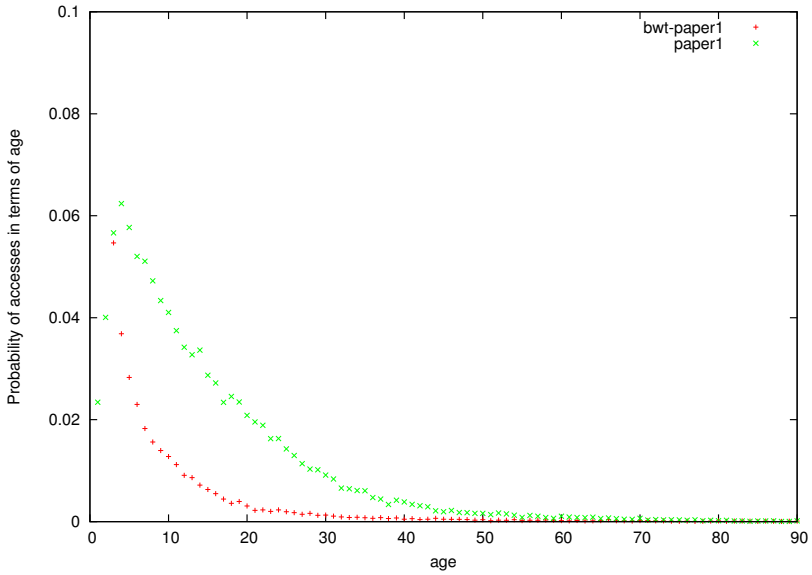

Fig. 5. Prob. of accessing items in terms of their age in file **paper1** before and after BWT. For bwt-paper1, $f(1) = 0.58$ and $f(2) = 0.11$ are off-scale and thus not shown.

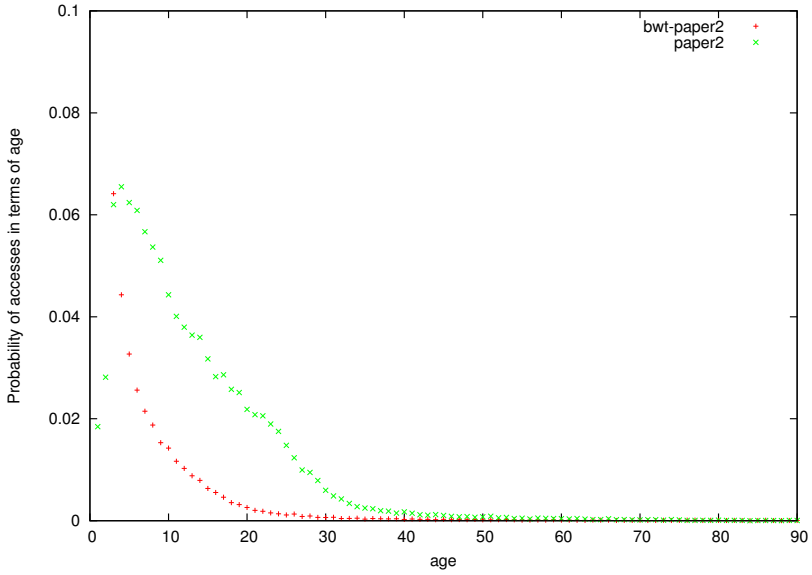

Fig. 6. Prob. of accessing items in terms of their age in file **paper2** before and after BWT. For bwt-paper2, $f(1) = 0.55$ and $f(2) = 0.12$ are off-scale and thus not shown.

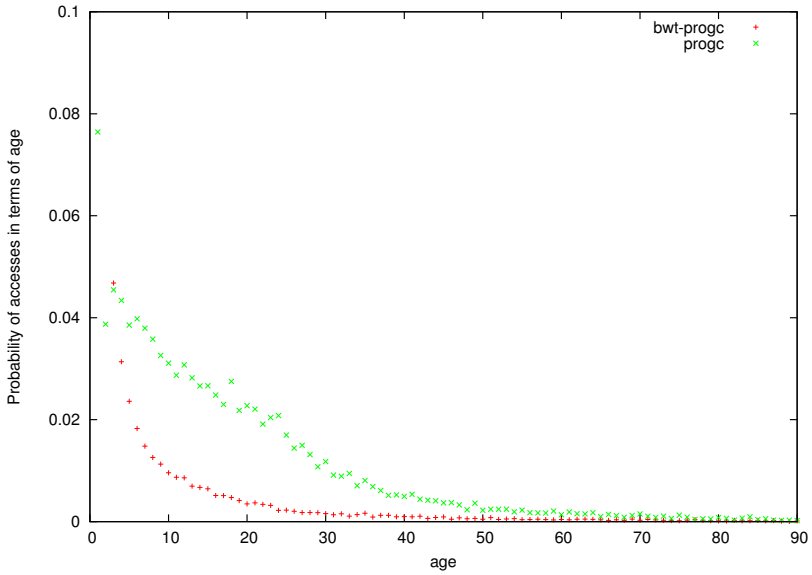

Fig. 7. Prob. of accessing items in terms of their age in file **progc** before and after BWT. For bwt-progc, $f(1) = 0.60$ and $f(2) = 0.11$ are off-scale and thus not shown.

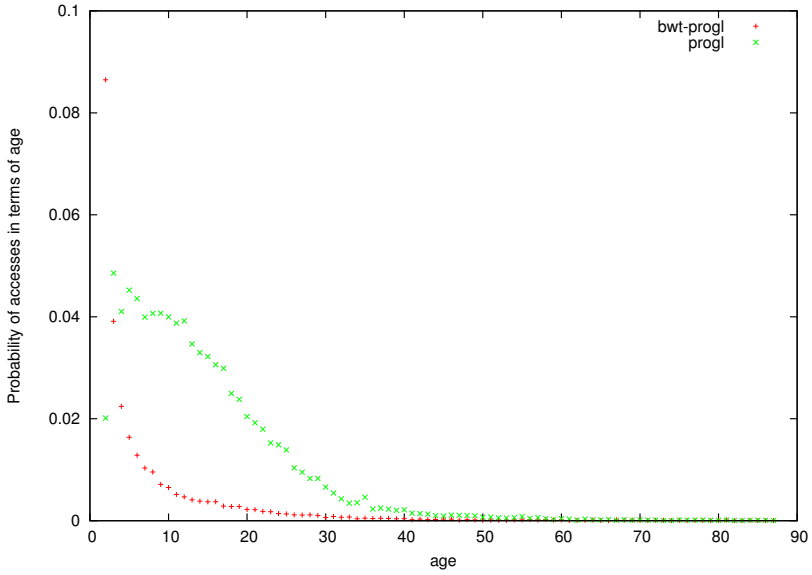

Fig. 8. Prob. of accessing items in terms of their age in file **progl** before and after BWT. For bwt-progl, $f(1) = 0.72$ is off-scale and thus not shown.

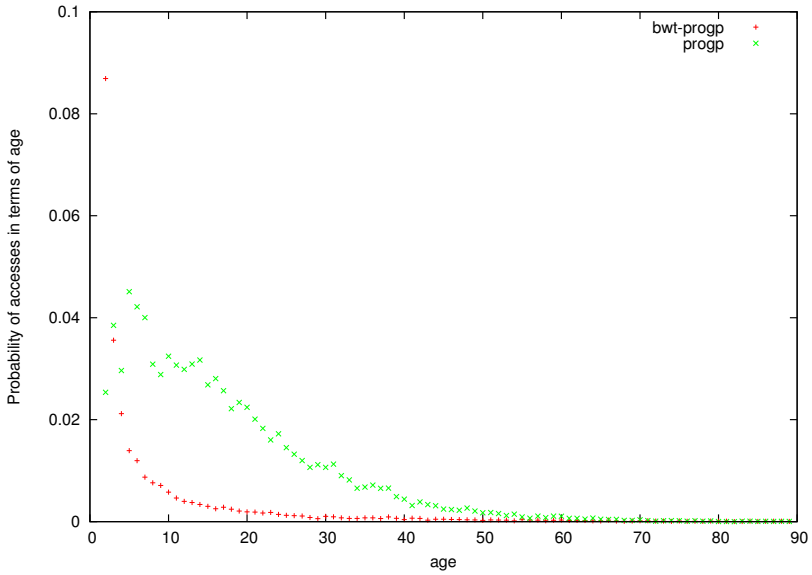

Fig. 9. Prob. of accessing items in terms of their age in file **progp** before and after BWT. For bwt-progp, $f(1) = 0.74$ is off-scale and thus not shown.

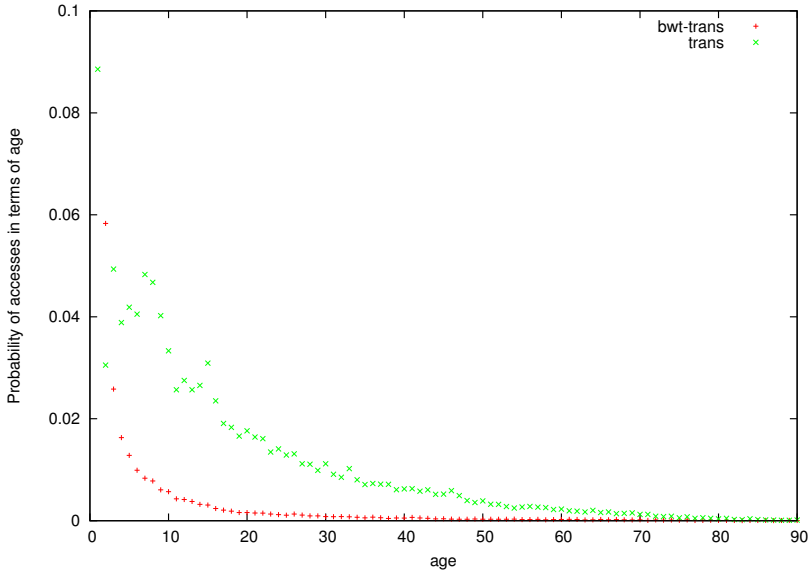

Fig. 10. Prob. of accessing items in terms of their age in file **trans** before and after BWT. For bwt-trans, $f(1) = 0.79$ is off-scale and thus not shown.

Table 3. Working set bounds of files in Calgary Corpus (normalized by their sizes) before and after Burrows-Wheeler Transform

File	Category	Size (bytes)	l	WS/n	$WS/$n (BWT)
bib	Bibliography	111261	81	3.9	1.6
book1	Fiction book	768771	82	3.4	1.7
book2	Non-fiction book	610856	96	3.5	1.6
geo	Geophysical data	102400	256	4.2	2.3
news	USENET batch file	377109	98	3.6	1.8
obj1	Object code for VAX	21504	256	3.8	2.1
obj2	Object code for Mac	246814	256	4.1	1.5
paper1	Technical paper	53161	95	3.56	1.73
paper2	Technical paper	82199	91	3.47	1.72
pic	fax picture	513216	159	1.37	1.25
progc	Source code in "C"	39611	92	3.65	1.74
progl	Source code in LISP	71646	87	3.22	1.45
progp	Source code in PASCAL	49379	89	3.38	1.44
trans	Transcript of terminal session	93695	99	3.61	1.38