SOFTWARE—PRACTICE AND EXPERIENCE
Softw. Pract. Exper. 0000; **00**:1–20
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/spe

Revisiting Bounded Context Block-Sorting Transformations

J. Shane Culpepper Matthias Petri Simon J. Puglisi

School of Computer Science & Information Technology, RMIT University, Melbourne VIC 3001, Australia

SUMMARY

The Burrows-Wheeler Transform (BWT) produces a permutation of a string X, denoted X^*, by sorting the n cyclic rotations of X into full lexicographical order, and taking the last column of the resulting $n \times n$ matrix to be X^*. The transformation is reversible in $\mathcal{O}(n)$ time. In this paper, we consider an alteration to the process, called k-BWT, where rotations are only sorted to a depth k. We propose new approaches to the forward and reverse transform, and show the methods are efficient in practice.
More than a decade ago, two algorithms were independently discovered for reversing k-BWT, both of which run in $\mathcal{O}(nk)$ time. Two recent algorithms have lowered the bounds for the reverse transformation to $\mathcal{O}(n \log k)$ and $\mathcal{O}(n)$ respectively. We examine the practical performance for these reversal algorithms. We find the original $\mathcal{O}(nk)$ approach is most efficient in practice, and investigate new approaches, aimed at further speeding reversal, which store precomputed context boundaries in the compressed file. By explicitly encoding the context boundaries, we present an $\mathcal{O}(n)$ reversal technique that is both efficient and effective. Finally, our study elucidates an inherently cache-friendly – and hitherto unobserved – behaviour in the reverse k-BWT, which could lead to new applications of the k-BWT transform. In contrast to previous empirical studies, we show the partial transform can be reversed significantly faster than the full transform, without significantly affecting compression effectiveness.
Copyright © 0000 John Wiley & Sons, Ltd.

Received . . .

KEY WORDS: data compression, Burrows-Wheeler transform, block-sorting, suffix array

1. INTRODUCTION

The Burrows-Wheeler Transform (BWT) – also referred to as the "block-sorting transform" – is now prevalent in lossless data compression systems [1]. The transformation process produces a permutation of a string X, denoted X^*, by sorting the n cyclic rotations of X into full lexicographical order, and taking the last column of the resulting $n \times n$ matrix to be X^*. Remarkably, the transformation is reversible in $\mathcal{O}(n)$ time. The transformation does not compress X, but X^* tends to have locally skewed first-order statistics which can be exploited to achieve better compression using relatively simple techniques. Since the transform was introduced in 1994, many empirical evaluations and theoretical investigations have followed [2, 3, 4, 5, 6, 7]. The BWT is now the basis for several off-the-shelf compression tools, such as bzip2 [8, 9]. The BWT is also a key component in fast pattern matching indexes [10, 11], and has be applied in seemingly unrelated areas such as shape analysis for computer vision, and machine translation [12]. The full potential of this versatile string permutation is only now beginning to be realized.

Contract/grant sponsor: This work was supported by Australian Research Council.; contract/grant number:

Copyright © 0000 John Wiley & Sons, Ltd.

Most BWT-based compression systems fully sort the cyclic rotations of X, and nearly all current empirical studies assume a full sorting of rotations. However, a full sorting of the rotations is resource intensive. In independent work, Schindler [13] and Yokoo [14] described an alternative approach in which the n rotations are only partially sorted to a fixed prefix depth, k. We refer to this modified transform as k-BWT. By limiting the sort depth to k, sorting can be accomplished in $\mathcal{O}(nk)$ time using radix sort, and is very fast in practice. Moreover, Schindler reports nearly identical compression effectiveness to the full transform, even for small values of k. The algorithms developed by Schindler [13] were subsequently made available in the general purpose compression tool szip. However, the simplification of the forward k-BWT transform comes at a cost: the reverse transform becomes more expensive, at least in theory.

Our contribution: First, we describe an efficient forward k-BWT algorithm based on induced sorting techniques from suffix array construction [15]. Our second contribution is a practical, $\mathcal{O}(n)$ k-BWT time reversal algorithm that implicitly stores context boundaries. Third, we provide the first thorough empirical analysis of state-of-the-art k-BWT algorithms for the forward and inverse transforms, compression effectiveness, and associated trade-offs. Lastly, we discover a previously undocumented locality of access property inherent to k-BWT algorithms, allowing fast transform reversal for small k.

2. BACKGROUND AND NOTATION

Let $X = X[0..n] = X[0]X[1]..X[n]$ be a string (or text) of $n+1$ symbols, where the first n symbols of X are drawn from an alphabet Σ and comprise the actual input; $X[n] = \$$ is a unique "end-of-string" symbol that is defined to be lexicographically smaller than all symbols in Σ. The string $X_i = X[i..n]X[0..i-1]$ represents the ith *rotation* of X or "rotation i" less formally. The substring $X[i..n]$ is the ith *suffix* of X, or "suffix i". Rotation i is always prefixed with suffix i as a result of the unique end of stream symbol \$. The k-BWT depends upon a partial sort of the rotations of X, based on an ordering of the prefixes of these rotations of length $k \geq 1$. We refer to the partial ordering as a *k-ordering* of rotations into *k-order*, and to the process itself as a *k-sort*. If two or more rotations are equal under k-order, the rotations have the same *k-rank* and therefore fall into the same *k-group*. Throughout this paper we assume a k-sort is *stable*. This assumption guarantees the ordering within each k-group coincide with their original ordering in X.

2.1. The Burrows-Wheeler Transform

String X is transformed into X^* using the following technique [1]:

1. Form a matrix $\mathcal{M}$ whose rows are the cyclic rotations of X;
2. Sort the rows of $\mathcal{M}$ into lexicographical order and let F and L be the first and last columns of $\mathcal{M}$ respectively;
3. So, $X^* = L[0]L[1]...L[n]$. To reverse the transform we also must record position I, which corresponds to the row in $\mathcal{M}$ where the original string appears.

Let $\mathcal{M}_k$ refer to the matrix $\mathcal{M}$ of rotations with the rotations stably k-sorted. Therefore, the original matrix of rotations by order of starting positions with respect to X is $\mathcal{M}_0$, and $\mathcal{M}_n$ corresponds to the fully sorted matrix of BWT. Let LF_k be the mapping of each symbol in L to its corresponding position in L for $\mathcal{M}_k$. For clarity, we use LF_n to denote the true LF-mapping, that is the LF-mapping from the fully sorted BWT. We use L_k and L_n in a similar way. Let X^*_k be the last column in $\mathcal{M}_k$. The output of BWT is X^*, and X^*_k represents the output of k-BWT. If row j of $\mathcal{M}_k$ contains rotation i then **PRED**(j) is the row containing rotation $i-1$. Figure 1 shows the the fully sorted matrix $\mathcal{M}_n$ (right) and the partially sorted matrix $\mathcal{M}_2$ (left) for the string "`knicknack$`". Observe the rows of $\mathcal{M}_n$ in the figure – up to

Copyright © 0000 John Wiley & Sons, Ltd.

DOI: 10.1002/spe

D_2	LF_2	F										L	
1	6	$	k	n	i	c	k	k	n	a	c	k	
1	10	a	c	k	$	k	n	i	c	k	k	n	
1	5	**c**	**k**	k	n	a	c	k	$	k	n	i	
0	2	**c**	**k**	$	k	n	i	c	k	k	n	a	
1	11	i	c	k	k	n	a	c	k	$	k	n	
1	3	k	$	k	n	i	c	k	k	n	a	c	
1	4	k	k	n	a	c	k	$	k	n	i	c	
1	1	**k**	**n**	i	c	k	k	n	a	c	k	$	←
0	7	**k**	**n**	a	c	k	$	k	n	i	c	k	
1	8	n	a	c	k	$	k	n	i	c	k	k	
1	9	n	i	c	k	k	n	a	c	k	$	k	

k-BWT, $k = 2$

F										L	LF_n
$	k	n	i	c	k	k	n	a	c	k	6
a	c	k	$	k	n	i	c	k	k	n	10
c	k	$	k	n	i	c	k	k	n	a	2
c	k	k	n	a	c	k	$	k	n	i	5
i	c	k	k	n	a	c	k	$	k	n	11
k	$	k	n	i	c	k	k	n	a	c	3
k	k	n	a	c	k	$	k	n	i	c	4
k	n	a	c	k	$	k	n	i	c	k	7
k	n	i	c	k	k	n	a	c	k	$	1
n	a	c	k	$	k	n	i	c	k	k	8
n	i	c	k	k	n	a	c	k	$	k	9

full BWT

Figure 1. Sorted cyclic rotations for the string "`knickknack$`" for the 2nd-order k-BWT (left) and the full BWT (right).

the $ symbol on each row – are in fact the suffixes of X in lexicographical order. Note the following important properties of $\mathcal{M}_n$ that are necessary for the reverse transform (see [1] for further details):

1. Given the ith row of $\mathcal{M}_n$, its last character $L[i]$ precedes its first character $F[i]$ in the original text. So, $X = \ldots L[i]F[i] \ldots$.
2. Let $c = L[i]$ and let r_i be the frequency of symbol c in $L[0..i-1]$. If $\mathcal{M}_n[j]$ is the r_i th row of $\mathcal{M}_n$, starting with c, then the symbol corresponding to $L[i]$ in the first column is located at $F[j]$. As such, $L[i]$ and $F[j]$ correspond to the same symbol in the input string T since F and L are both sorted by the text *following* the symbol occurrences.

The BWT algorithm is reversible using the following procedure, which produces the original string in reverse order.

1. Count the number of occurrences of each symbol in X^* and compute an array $C[0..|\Sigma|]$, such that $C[c]$ stores the number of occurrences of symbols $\{\$, 1, .., c-1\}$ in X^* (and equivalently X). The count $C[c]+1$ gives the first occurrence of the symbol c in F.
2. Next, construct the LF mapping, $LF_n[0..n]$, by making a pass over X^* and setting $LF_n[i] = C[L[i]] + \textbf{Occ}(L, L[i], i)$, where the function $\textbf{Occ}(A, b, i)$ returns the number of occurrences of symbol b in string $A[0..i-1]$. The mapping LF_n precisely defines the **Pred** function.
3. Reconstruct X backwards as follows: set $s = I$ and for each $i \in n..0$ do $X[i] \leftarrow X^*[s]$ and $s \leftarrow LF[s]$.

2.2. The Bounded Context Transform

The key difference between the full and partial transforms is the ease of implementing the **Pred** function. For $\mathcal{M}_n$, $\textbf{Pred}(i)$ is easily derived from $LF_n[i]$. But, LF_k does not necessarily represent **Pred**. The mapping $LF_n[i]$ returns the position in $\mathcal{M}_n$ of the rotation $j = \mathcal{M}_n[i] - 1$ in the fully sorted BWT, which is also the the row containing the rotation preceding shift $\mathcal{M}_n[i] - 1$. Unfortunately, $LF_k[i]$ provides no such guarantee for predecessor. It only points to a row with a rotation that shares the same k-prefix as the true predecessor, or all rows k-equal to the true predecessor. For example, $LF_2[6] = 3$ in Figure 1 is not the predecessor of rotation 3 – the actual predecessor is 4. However, 3 and 4 do share a common prefix of length $k = 2$. Using only LF_2 would recover the text "`knackknick$`" instead of "`knickknack$`". The characters `a` and `i` both have the same 2nd-order context `ck`. During reconstruction of X from

Copyright © 0000 John Wiley & Sons, Ltd.
Prepared using speauth.cls

DOI: 10.1002/spe

X_2^*, using LF_2, the character i is processed first instead of the correct choice a as LF_2 only points to the preceding context and not the correct preceding character. However, we shall see in the next section that LF_k can properly simulate **Pred** as in LF_n, using a small amount of extra information.

2.3. *Reversing k-BWT*

In this section, we describe how to reconstruct X from X_k^*, using only knowledge of the boundaries of the k-groups in $\mathcal{M}_k$. This method was originally described in [13]. For now, we assume this information is available before reconstruction w.l.o.g.. The k-group boundaries can be represented as a bitvector, D_k, or just D for short.

Definition 1
Let $D_k[1..n]$ be a bitvector, such that $D_k[1] = 1$ and for $2 \leq i \leq n$, $1 \leq k \leq n$:

$$D_k[i] = \begin{cases} 0, \text{if } \mathcal{M}_k[i][1..k] = \mathcal{M}_k[i-1][1..k] \\ 1, \text{if } \mathcal{M}_k[i][1..k] \neq \mathcal{M}_k[i-1][1..k]. \end{cases}$$

So, $D[i] = 1$ if the rotations at rows i and $i-1$ in $\mathcal{M}_k$ are in different k-groups, and $D[i] = 0$ if they belong to the same k-group. Thus, a k-group containing $v+1$ members is indicated by a substring 10^v in D. For example, the corresponding D_2 in Figure 1 is 11101111011. Recall that the rotations are defined to be in ascending order (due to the stability of the sorting process) within each k-group. During reversal, the rows in a given k-group must be visited last to first, by virtue of the fact that we are outputting the string from last character to first. We now have the pieces for a complete reversal algorithm for k-BWT:

```
j = I
for i = N downto 0 do
   X[i] ← X*_k[j]
   g ← Group(j)
   j ← Pred(j) ≡ g − GroupPos[g]
   Increment GroupPos[g]
```

As previously discussed, LF_k cannot be used to implement the **Pred** function directly as $LF_k[j]$ only guarantees to point to a symbol in the same k-group. So, the correct predecessor must be derived from the current k-group. Therefore, the preceding k-group g ($g = \textbf{Group}(j)$) is located. Then, the correct preceding character j within the k-group is determined by maintaining the count of all of the previously decoded characters within g (**GroupPos**(g)). All known methods for reversing k-BWT use a variation of this general procedure. The methods differ in how the k-group boundaries (the D_k vector) are reconstructed prior recovering X.

3. RELATED WORK

3.1. *Block-Sorting Transforms*

Construction The BWT is constructed by sorting the rows of the rotation matrix $\mathcal{M}$. This is easily accomplished by building a suffix array (SA) over X first, and then X^* is obtained from the suffix array in linear time by exploiting a duality between a suffix array and the BWT: $X^*[i] = X[\text{SA}[i] - 1]$. Suffix array construction requires $\mathcal{O}(n)$ time, but algorithms using induced sorting in $\mathcal{O}(n^2 \log n)$ worst case time tend to perform better in practice [15, 16, 17]. The BWT can therefore be computed in linear time. However, suffix array and BWT construction is still space intensive and has been the subject of several research endeavors. Suffix array construction does require $\mathcal{O}(n \log n)$ bits of space compared to $\mathcal{O}(n \log \sigma)$ space required for the BWT. Burrows and Wheeler initially proposed to split the text into smaller blocks to limit the space usage during construction, but this often lowers compression effectiveness [1, 12]. Okanohara et al. propose a BWT construction algorithm inspired by induced sorting used during

Copyright © 0000 John Wiley & Sons, Ltd.

Prepared using speauth.cls
DOI: 10.1002/spe

suffix array construction which can construct X^* in $\mathcal{O}(n)$ time and $\mathcal{O}(n \log \sigma \log \log_\sigma n)$ space [18]. Kärkkäinen [19] describes a BWT construction algorithm requiring $\mathcal{O}(n \log n)$ time and $\mathcal{O}(n \log \sigma)$ bits of space. Reducing these bounds is an active area of algorithmic research.

Reversal Reversing the BWT can also be done in $\mathcal{O}(n)$ time as described in Section 2.1. The reverse transform of Burrows and Wheeler requires $\mathcal{O}(n \log n)$ space. Kärkkäinen and Puglisi [20] categorize reversal algorithms into three categories: large-space, medium-space, and small-space. Large-space algorithms require $\mathcal{O}(n \log n)$ space, small-space algorithms require only $o(n \log \sigma)$ additional space and medium-space algorithms encompass all algorithms in between. Seward proposed the most efficient large-space BWT reversal algorithm [9]. Seward's reversal algorithm reorders the LF mapping to obtain better cache performance. Small-space algorithms require minimal extra space, but are often significantly less efficient [20]. Various medium-space algorithms also provide appealing time and space trade-offs between these two extremes [20, 21]. After reconstructing the context boundaries, a similar algorithmic approach could be used to perform space efficient k-BWT inversion as well, but this avenue remains relatively unexplored.

Applications Burrows and Wheeler proposed the BWT as part of a complete compression system. Figure 2 shows a typical sequence of modelling and coding in a BWT-based compression system. The BWT is commonly used as the first step in such compression systems to group symbols with a similar context together. The following algorithms – move-to-front (MTF) and run length encoding (RLE) – use the transformed text to create a more skewed symbol distribution which in turn can be compressed more effectively using an entropy encoder. Compression systems using the BWT are widely used in practice (bzip2) and have been analyzed in great detail [2, 3, 5]. For a more comprehensive discussion of data compression systems, see [22, 23, 24].

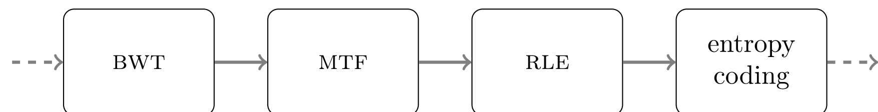

Figure 2. A typical encoding sequence for a transform-based compression system.

Ferragina et al. describe an alternative approach to MTF/RLE-based BWT compression systems, called *compression boosting* [10]. Compression boosting improves effectiveness by partitioning X^* into context aligned blocks. These sub-blocks are then individually compressed by a zero-order compressor to achieve better compression than is possible by applying a zero-order entropy coder to X^*. A greedy algorithm finds the optimal partitioning by performing bottom-up merges of the suffix tree of the input text: Suffixes are merged into a block if compression effectiveness improves, compared to compressing each suffix individually. The merge criteria is based on the compressibility of the blocks in X^*. The partitioning process adds significant extra time and space costs during the compression phase and the partition boundaries must be stored explicitly [7].

Another significant application of BWT is searching in compressed text. In seminal work, Ferragina et al. proposed the FM-Index, a self-indexing data structure that allows full-text search directly in the compressed text [10]. The FM-Index provided efficient search over a compact representation of a text by exploiting the duality between suffix arrays and the BWT, and has led to a highly productive research area in recent years [25].

Variations Several transform-based permutations similar in spirit to BWT have been proposed in recent years. Vo and Manku propose RadixZip, a block sorting transform that breaks the input text into either fixed or variable length tokens, to compress database columns

Copyright © 0000 John Wiley & Sons, Ltd.
Prepared using speauth.cls

DOI: 10.1002/spe

[26]. Inagaki et al. [27] propose a theoretical framework called the Generalized Radix Permute transform (GRP), of which BWT, k-BWT and RADIXZIP are special cases. More recently, Yokoo further showed the $\mathcal{O}(n)$ k-BWT reversal algorithm proposed by Nong et al. [28] for k-BWT can be adopted to reverse the GRP [29].

3.2. Bounded Context Block-Sorting Transforms

The first algorithm to successfully reverse k-BWT was presented by Schindler [13] (and independently by Yokoo [14]). The initial algorithm iteratively computed the k-group boundaries during reversal to construct D_k. The 1-group boundaries are recovered by sorting the symbols of L (in $\mathcal{O}(n)$ time, using counting sort) and marking each character transition in the sorted text as a context boundary. Each new context boundary is computed by traversing the text and using the character position within the current context to mark the context boundary of the next order. Overall, k passes over the text are required to reconstruct D_k, requiring $\mathcal{O}(nk)$ time in total.

Recently, Nong and Zhang [30] described an asymptotically more efficient method for computing D_k which requires $\mathcal{O}(n \log k)$ time. The key observation made by Nong and Zhang is that a k-order context is composed of two $(k/2)$-order contexts. Due to the cyclic properties inherent to the permutation matrix, the $(k/2)$-order context positions can be extracted from the previous iteration. Using a dynamic programming approach, the algorithm uses $D_{i/2}$ to produce D_i, where $i \leq k$, to recover D_k in $\mathcal{O}(n \log k)$ time. Nong et al. [28] further improve their algorithm to $\mathcal{O}(n)$ time and space for recovering the k-order contexts by computing the lengths of longest-common-prefixes (LCP) for each adjacent row in $\mathcal{M}_k$. The k-order contexts can then be retrieved by simply traversing the list of LCP values and marking a k-order context boundary each time LCP $\geq k$.

Adjeroh et al. provide an empirical analysis of the commonly available k-BWT compression system (szip [13]) and a BWT based compression system, bzip2 [12, Chapter 6.1]. In the comparison, szip achieved comparable compression was a more efficient encoder when $k < 40$. Decompression using szip was less efficient than bzip2. However, the experiments used small files ($<4\,MB$) and the two implementations use vastly differing I/O and entropy coding techniques, making it difficult to draw any conclusions about the relative performance of the systems.

4. EXPERIMENTAL SETUP

In subsequent sections, we empirically compare the effectiveness and efficiency for all k-BWT algorithms with a BWT baseline. Henceforth, *effectiveness* is a measure of compressibility of the transformed text, typically expressed in *bits per symbol* (bps) or compression ratio (%), and *efficiency* is a performance metric expressed in CPU time (sec). First, we consider the efficiency of the forward k-BWT as compared to the fully sorted BWT. Then, in Section 6, we compare the efficiency of the inverse transforms and propose new algorithmic approaches based on various time and space trade-offs. Finally, in Section 7 we evaluate the compression effectiveness for all of the algorithms and quantify the effectiveness and efficiency trade-offs between BWT and k-BWT methods.

All algorithms are implemented in ANSI C. The experimental machine was a Core2Duo 3.16 Ghz CPU, 6 MB L2 cache and 4 GB RAM running Linux-2.6.34. An efficient and freely available reference implementation for the BWT algorithm was used as a baseline*. In all experiments, the full text was processed as a single block equal to the size of the file. Standard data sets from the Pizza & Chili Corpus and the TREC collection were used [31]. Table I shows file sizes (in MB), alphabet sizes ($|\Sigma|$) and the zero-order self-information ($\mathcal{H}_0$) for each test collection. The WSJ file is the last 50 MB of the *Wall Street Journal* extracted from the Disk 2 of the TREC data

*`http://libdivsufsort.googlecode.com/` v2.01

Copyright © 0000 John Wiley & Sons, Ltd.
Prepared using speauth.cls

	$\lvert\Sigma\rvert$	Size	LCP$_{\text{mean}}$	LCP$_{\text{max}}$	$\mathcal{H}_0$	Description
WSJ	89	50	16	1,344	4.62	The last 50 MB of the *Wall Street Journal* segment (`wsj267`) of the TREC data collection.
DNA	4	50	31	14,836	1.98	Sequence of gene DNA sequences coded as A, G, C, T (all extra symbols were stripped from the text).
XML	96	50	42	1,005	5.23	Bibliographic information from `dblp`.
SOURCES	227	50	168	71,651	5.53	Concatenation of source code of the linux-2.6.11.6 and gcc-4.0.0 distributions.

Table I. Statistical properties of benchmark collection.

collection (see `http://trec.nist.gov`). In Table I, LCP$_{\text{mean}}$ is the mean of the LCP values between two adjacent rows of the transformed text. This metric is an indicator of the depth k at which symbols in the test collection will be sorted (on average) in full lexicographical order. We also show LCP$_{\text{max}}$, which is an upper bound on the maximum sort depth necessary to achieve the compression effectiveness of the full BWT. Note that all test inputs are of size 50 MB. Additional experiments were performed on variable inputs sizes ranging from 10 MB to 200 MB. However, we observed similar results in all of our experiments for differing input sizes, and therefore show results that are representative of texts of varying length.

5. FORWARD TRANSFORM EFFICIENCY

We now describe how to efficiently construct the forward bounded context transform. Schindler used a k-deep radix sort to obtain X_k^* in $\mathcal{O}(nk)$ time. However, initial experiments indicated this approach was less efficient than current suffix array construction algorithms based on induced sorting [15, 17]. Using induced sorting, as little as $\approx 30\%$ of the suffixes must be sorted to obtain the final suffix array and the resulting X^*. To compare the efficiency of the forward k-BWT to BWT, we adopt the initial induced sorting method of Itoh and Tanaka to construct X_k^* from SA$_k$ [32]. The algorithm sorts around 50% of the suffixes explicitly and induces the order of the remaining suffixes by making one additional pass over the text.

The transform can also be performed by augmenting Maniscalco and Puglisi's induced sorting suffix array construction algorithm, but would require additional passes over the text [17]. This algorithm sorts 20% fewer suffixes and allows for faster construction of the full suffix array as each suffix comparison, on average, requires LCP$_{\text{mean}}$ symbol comparisons. However, for X_k^* construction, only the first k symbols of each suffix are compared. So, the cost of more passes is greater than the decreased number of sorted suffixes, since sorting is to a fixed depth and terminates more quickly.

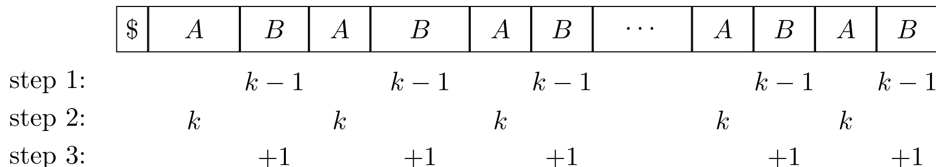

Figure 3. Induced sorting X to a depth of k in three steps: First all type B suffixes are sorted to depth $k-1$, second the k-deep sort order of all type A suffixes is induced and finally the sorting depth of all type B suffixes is increased by one.

Copyright © 0000 John Wiley & Sons, Ltd.
Prepared using `speauth.cls`

Following the approach described by Itoh and Tanaka, our algorithm partitions all suffixes in X into two types, A and B, by comparing any two prefixes of two adjacent suffixes X:

$$X[i] = \begin{cases} \textbf{Type A} & \text{if } X[i] > X[i+1] \\ \textbf{Type B} & \text{if } X[i] \leq X[i+1]. \end{cases}$$

Each individual type B suffix S_i is then assigned a bucket in SA_k according to the first character $X[i]$. All type B buckets are then sorted to a depth of $k-1$ as shown in Step 1 in Figure 3. Next, the sorted B buckets are used to induce the k sorting order for all A buckets by making one pass over SA_k as shown in Step 2. Note the special case of inducing the sorting order of an A bucket from a previous A bucket which is already in k sorted ordering, as each induction step increases the sorting order by 1 as shown in Figure 4.

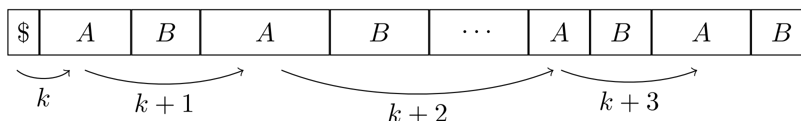

Figure 4. Inducing the sorting order of an A bucket from a previous A bucket.

To properly induce the k sorting order from an A bucket A_i, we only need to preserve the ordering if more than one position in the A_i, map to the same following bucket A_j, as the ordering between buckets is determined in Step 1. Finally, we increase the sorting depth of all type B buckets to k to obtain SA_k and X_k^* respectively.

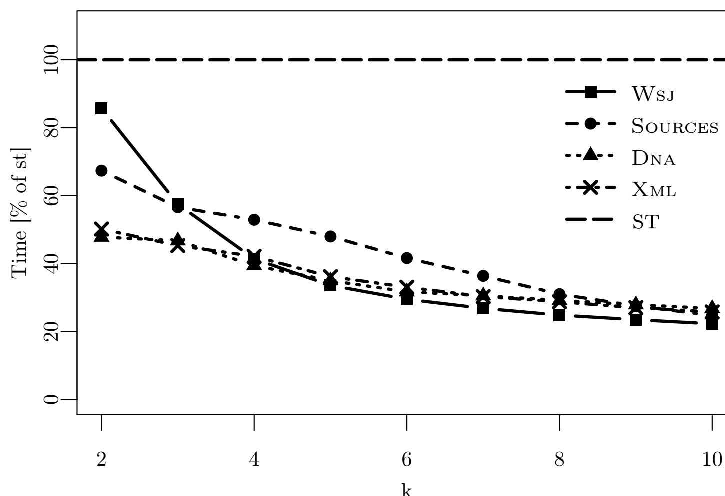

Figure 5. Efficiency of induced sorting for the forward k-BWT for the WSJ, SOURCES, DNA and XML collections proportional to Schindler's (ST) k-deep radixsort method.

Figure 5 shows the efficiency gain for an induced sorting forward k-BWT over the k-deep radixsort based transform originally proposed by Schindler. We use the radix sort method due to Kärkkäinen and Rantala [33], which is the fastest general string sorting method in practice. For all test files, induced sorting is always more efficient. As the sorting depth increases, the induced sorting transform becomes faster relative to the $\mathcal{O}(nk)$ k-deep radixsort transform as only 50% of the suffixes are being explicitly (ie. radix) sorted.

Figure 6 shows the running time of the k-BWT algorithm *proportional* to BWT for varying text inputs. Induced sorting in k-BWT outperforms the full BWT for small k. For all files with $k < 8$, the k-BWT transform is faster than BWT. For $k < 5$ the k-BWT transform is nearly

Copyright © 0000 John Wiley & Sons, Ltd.

Softw. Pract. Exper. (0000)
DOI: 10.1002/spe

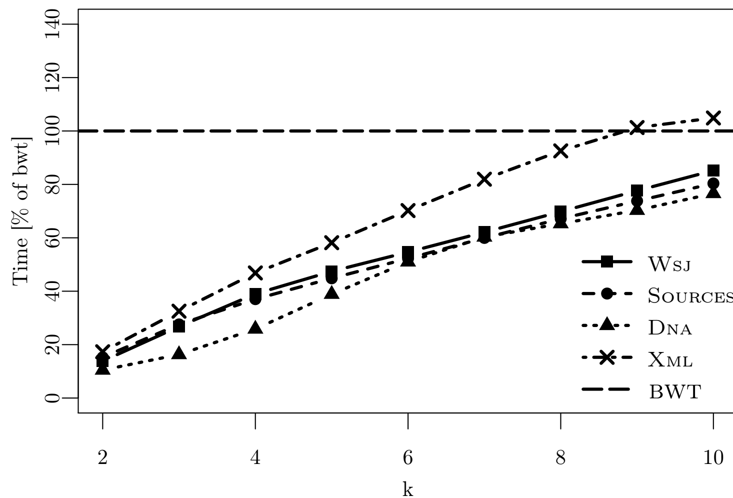

Figure 6. Efficiency of induced sorting for the forward k-BWT for the WSJ, SOURCES, DNA and XML collections relative to BWT.

twice as fast. The k-BWT sorts the initial matrix $\mathcal{M}_k$ up to a depth of k, but BWT must sort the matrix fully. Even though the construction of the transform is done using fast suffix array construction algorithms, we still expect the k-BWT transform to be more efficient as the average number of character comparisons required to compare two individual suffixes (LCP_{mean}) tends to be significantly larger than k.

The BWT and the k-BWT transform are usually part of a larger compression system. In many cases, the transform is the main contributing factor to the overall running time of the compression system. The efficiency of the complete compression system should therefore improve proportionally to the observed transform improvements. Here, we expand the investigation of efficiency of BWT and k-BWT to different transform-based compression systems that use the BWT-based transforms as a first step in a sequence of algorithms. The WSJ file is first transformed using BWT or k-BWT, then compressed with different combinations of (MTF), Run-Length Encoding (RLE) and an entropy coder. We tested the following standard entropy coders:

- Huffman coder (HUFF) [34]
- Range Coder (RANGE) [35]
- Arithmetic coder (ARITH) [36, 37]

We also replaced the MTF and RLE steps with compression boosting (BOOST) as described in [7] for k-BWT-based compression systems. The freely available boosting library framework[†] was used as the basis for these experiments. Figure 7 shows the efficiency for different compression systems using k-BWT proportional to BWT. We observe that for most systems, performance gains due to k-BWT translate into overall performance gains (that is, the BWT phase is indeed a computational bottleneck). The k-BWT compression systems can be up to 4 times faster than the full BWT. For traditional MTF-RLE compression systems with $k < 6$, k-BWT is twice as fast. Compression boosting with the k-BWT only shows marginal efficiency improvements because the post transform phase (in which the optimal partitioning is computed) is itself so computationally expensive.

[†] `http://people.unipmn.it/manzini/boosting/`

Copyright © 0000 John Wiley & Sons, Ltd.
Prepared using speauth.cls

DOI: 10.1002/spe

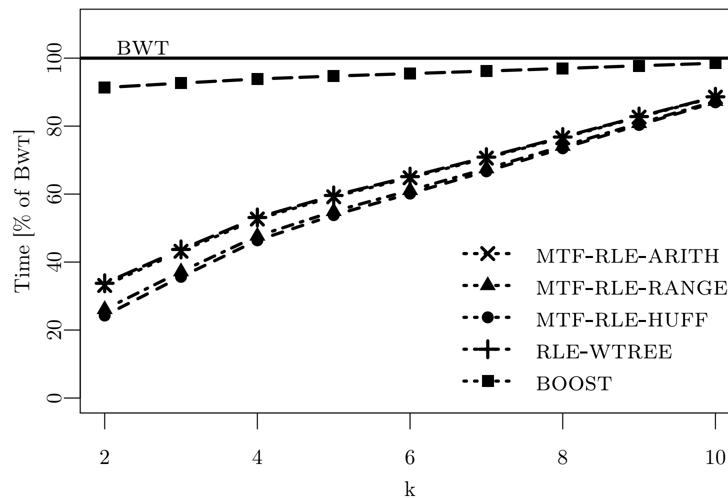

Figure 7. Compression efficiency of different compression systems using the k-BWT for WSJ *proportional* to the BWT.

6. INVERSE TRANSFORM EFFICIENCY

The recovery of the original string from all of the transforms is similar. The fundamental difference between BWT and k-BWT inversion is the need to recover context boundaries. As discussed in Section 3.2, several methods exist to recover the original context information. However, we will see shortly that the overhead for context reconstruction increases dramatically as k gets large.

Intuitively, we expect BWT inversion to be more efficient than any of the k-BWT methods, and the $\mathcal{O}(nk)$ k-BWT algorithm to be less efficient than the $\mathcal{O}(n)$ or $\mathcal{O}(n \log k)$ algorithms. However, the constant factors for the $\mathcal{O}(n)$ and $\mathcal{O}(n \log k)$ algorithms have a significant impact on their practical efficiency. This discrepancy is explored further in Section 6.2.

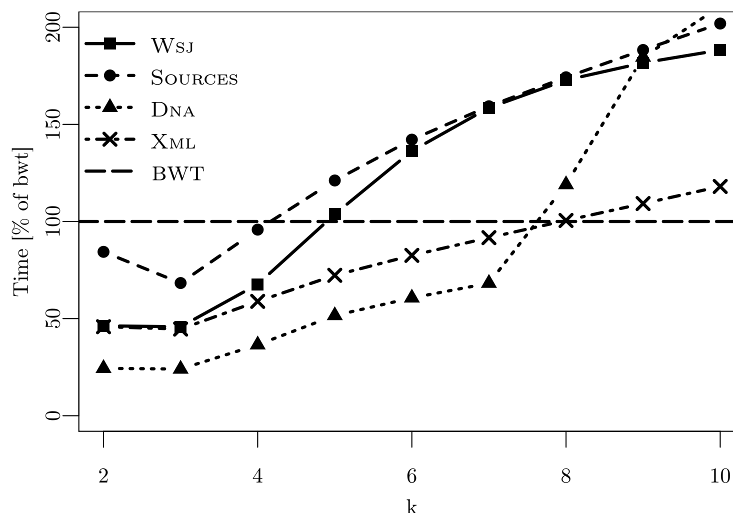

Figure 8. Reverse-transform efficiency of k-BWT using the $\mathcal{O}(nk)$ algorithm for WSJ, SOURCES, DNA and XML in time relative to BWT for each respective file.

Figure 8 shows the efficiency of Schindler's $\mathcal{O}(nk)$ algorithm for all test collections relative to the efficiency of the full BWT algorithm. For small values of k, the inverse k-BWT outperforms

Copyright © 0000 John Wiley & Sons, Ltd.
Prepared using speauth.cls

DOI: 10.1002/spe

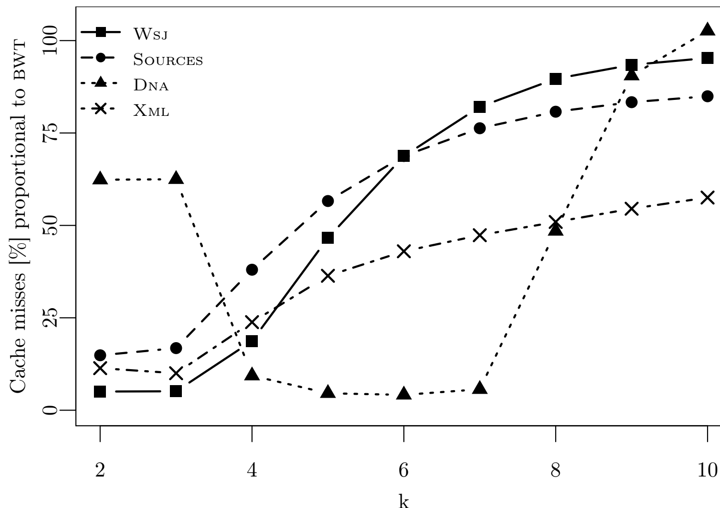

Figure 9. Cache misses for variable k in percent proportional to the cache misses of BWT for the respective file.

BWT, regardless of text input. The inverse k-BWT is up to three times faster than BWT inversion. The exact performance depends on the composition of the collection, but clearly k-BWT inversion can be more efficient than BWT inversion.

6.1. *Localization in k-*BWT

The efficiency of k-BWT is contradictory to previous results and somewhat surprising as the k-BWT algorithm performs additional work to reconstruct context information during inversion. Further experimental investigation into the discrepancy revealed a significant caching effect when k is small. In order to quantify the unexpected efficiency gain, we benchmarked the locality of access in the k-BWT inversion process. We focused on Level 2 cache performance which can be measured using the papi library version 4.1.0[‡].

Figure 9 shows the percentage of cache misses for variable k-order contexts during the reconstruction of the original text for each test collection. The cache misses are shown *proportional* to BWT inversion for identical files. Both algorithms perform the same task – reconstructing the text using the LF mapping – but, exhibit a profound difference in cache behaviour. For low order contexts, k-BWT has 90% fewer cache misses than BWT. As we will see, this remarkable difference is a natural by-product of k-BWT inversion. During inversion, the algorithm jumps between different contexts, but each partially sorted context is encoded by increasing position in the original text. The inherent locality of reference in each context allows the operating system to cycle between each context with fewer page-outs, resulting in fewer overall cache misses.

However, this observation does not hold true for Dna, where the cache misses for $k < 4$ are higher than for $k \geq 4$. This in contrast to the alternative collections where smaller values of k always results in better cache performance. Upon further investigation, our experiments show that k-BWT algorithms do not benefit from the cache effect when $|\Sigma|$ is very small. For small k and $|\Sigma|$, the number of cache misses is unusually high in Dna. Since the total number of contexts is bound by $|\Sigma|^k$, the cache can not be fully utilized: Each context has only a page size p_s of data pre-fetched in the cache. Once this data segment is processed, a cache miss occurs, and a new page is loaded. At most $p_s \cdot |\Sigma|^k$ data is cached at any given time, resulting in more cache misses as data is processed using fewer distinct memory pages. Therefore, balancing the

[‡]http://icl.cs.utk.edu/papi/

Copyright © 0000 John Wiley & Sons, Ltd.
Prepared using speauth.cls

DOI: 10.1002/spe

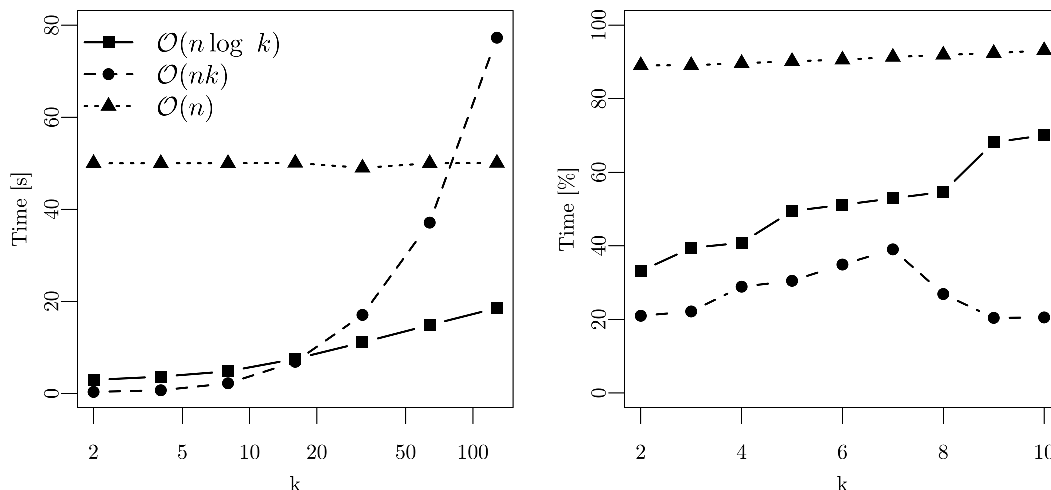

Figure 10. Context reconstruction time in seconds for variable k for all k-BWT algorithms (right) and as a percentage of the total reconstruction time for DNA (left).

number of distinct contexts and available memory pages can have a dramatic impact on overall efficiency. So, when $|\Sigma|$ is small, using larger values of k is desirable.

6.2. Context Reconstruction

The recovery of the k-order BWT and the full BWT transform are similar. Both transforms can sort the input text and use an LF mapping to recover the original string in $\mathcal{O}(n)$ time. The k-BWT method has the additional cost of reconstructing the context vector. As discussed in Section 3.2, three distinct context reconstruction algorithms have been proposed. Figure 10 compares only the cost of context reconstruction for each of these algorithms. The left subfigure shows the aggregate time for reconstruction and the right subfigure shows the percentage of total runtime consumed during context reconstruction for DNA.

The $\mathcal{O}(nk)$ algorithm is significantly more efficient for even moderate values of k. For $k = 100$, the $\mathcal{O}(n \log k)$ algorithm is still more than twice as fast as the linear variant. The constant factors of $\mathcal{O}(n)$ algorithm are not overcome until k is very large. Clearly the cost of vector reconstruction is a major contributor to efficiency loss for small k – up to 40% of the total cost for k-BWT inversion. Also, the context reconstruction time is largely independent of the text input, unlike the compression effectiveness and inverse transform efficiency.

6.3. Engineering a New Inversion Algorithm

Recovering context information in $\mathcal{O}(nk)$ time is a significant contributor to the overall time complexity of k-BWT inversion. Nevertheless, our empirical results show that k-BWT can still outperform BWT. Since vector reconstruction is a major bottleneck, up to 40%, methods which can alleviate this cost merit further consideration. Consequently, storing the contexts explicitly instead of computing the bounds during the reverse transform is a sensible space-time trade-off to consider. For small k, the number of distinct contexts is small, and, can thus be stored efficiently.

The contexts can be represented naively as a bitvector of size n bits, but the vector is sparse for small values of k, making the context information more compressible. We considered several methods to explicitly encode the distinct contexts $\mathcal{C}$. A pragmatic choice of compressed d-gap encoding is used to store the context information in our study, as the method is highly effective when $\mathcal{C} \ll n$ [38]. Furthermore, the sequential decoding limitation imposed by d-gap encoding is not problematic since random access to the context vector during reconstruction is not required. The additional cost to explicitly store the contexts relative to the cost of

Copyright © 0000 John Wiley & Sons, Ltd.

Prepared using speauth.cls
DOI: 10.1002/spe

	Theoretical		Empirical
Algorithm	Time	Space	Space(bytes)
[13]	$\mathcal{O}(nk)$	$\mathcal{O}(n)$	$5n$
[30]	$\mathcal{O}(n \log k)$	$\mathcal{O}(n)$	$22n$
[28]	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$26n$
Our Algorithm	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$5n$

Table II. Time and space bounds for inverse k-BWT context reconstruction.

reconstructing the context information using the $\mathcal{O}(nk)$ reconstruction algorithm is shown in Figure 11. For WSJ, XML and SOURCES, the cost of storing context information is negligible for $k < 6$. Context information for the DNA collection can be stored effectively for $k < 10$. Overall, there is no significant loss in compression effectiveness for moderate values of k.

6.4. Time and Space Trade-offs

Another important dimension to consider is the space required by the inversion process. Table II shows the time and space requirements with constant factors, for all context reconstruction algorithms, including our explicit storage algorithm. The empirical space usage is estimated using the program memusage available from the Pizza & Chili Corpus.

The $\mathcal{O}(n \log k)$ and $\mathcal{O}(n)$ algorithms require up to five times more space than the $\mathcal{O}(nk)$ algorithm to recover the original text. This space is further reduced by eliminating the context reconstruction requirement entirely, such as is done by our explicit context boundary encoding algorithm. Our algorithm requires only $5n$ space and can be inverted in $\mathcal{O}(n)$ time, but is only preferable when k is small.

7. COMPRESSION EFFECTIVENESS

The BWT algorithm groups symbols with similar context, allowing for more effective text compression. The transform is the first of a series of steps in a typical data compression system as described in Section 2.1. The k-BWT algorithm sorts the permutation matrix to a depth of k, and symbols with similar k-order contexts are grouped together. Consecutive contexts with an LCP $> k$ will not necessarily be exploited in k-BWT (but they may be by

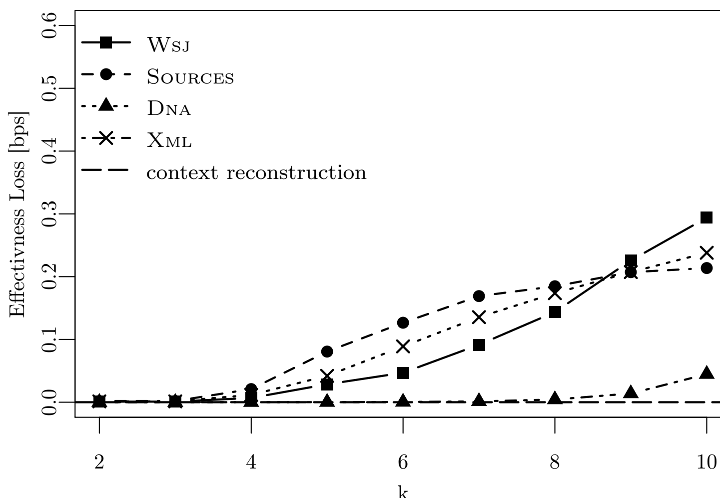

Figure 11. Entropy loss resulting from explicitly storing the contexts with d-gap encoding relative to reconstruction.

Copyright © 0000 John Wiley & Sons, Ltd.

Prepared using speauth.cls
DOI: 10.1002/spe

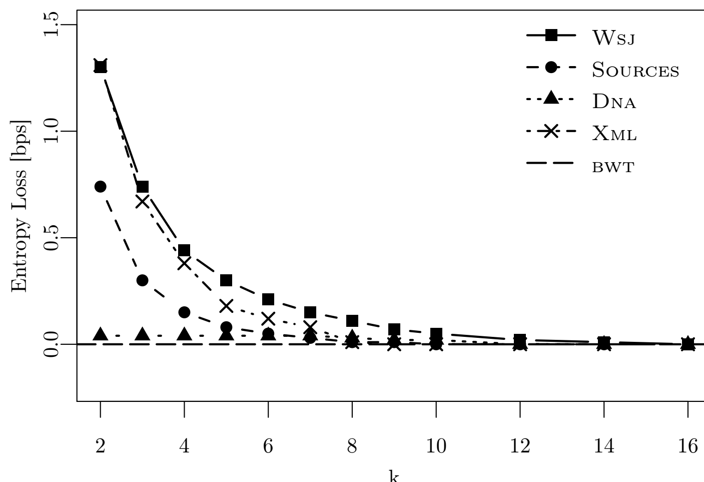

Figure 12. Effectiveness of k-BWT for WSJ, SOURCES, DNA and XML. Effectiveness is measured in entropy loss relative to BWT.

chance). Therefore, the k-BWT-based compression system can be less effective than a system using BWT.

We first consider the compression effectiveness of the algorithms relative to the sort depth k. Effectiveness is measured in bps using zero-order empirical-entropy, $\mathcal{H}_0$, after applying Move-To-Front (MTF) to the transformed text [39, 40, 41]. Figure 12 shows the effectiveness of k-BWT for increasing values of k. The effectiveness is reported as the entropy loss in bits per symbol *relative* to BWT. As k increases, the performance of k-BWT approaches the effectiveness achievable using a fully sorted BWT. For $k = 6$, the limited context transform achieves nearly identical compression effectiveness. For DNA, the performance of k-BWT is independent of k. This is unsurprising since DNA compression usually requires additional symbol model steps to improve compression effectiveness [42, 43, 44, 45]. The WSJ, SOURCES and XML collections can be compressed effectively using contexts as low as order-5. Overall, using a limited order context transform supports efficient compression, and achieves impressive compression effectiveness, even when k is small.

We now expand the empirical results for compression effectiveness in full compression systems. Each file is first transformed using BWT or k-BWT, then compressed with MTF and RLE, or BOOST. A variety of entropy coders are then applied to compress the transformed input. Table III shows the compression effectiveness in bits per symbol and efficiency in seconds of k-BWT for $k = 2, 4, 8, 10$, boosting and BWT using the different compression systems.

All compression systems show similar compression effectiveness relative to sorting depth. Huffman encoding is more efficient than Range or Arithmetic coding, but results in worse effectiveness. Compression boosting using BWT outperforms all systems in terms of effectiveness, but is an order of magnitude slower than all other combinations. When $k > 6$, the effectiveness loss is below 10 percent in k-BWT systems. When $k > 8$, the compression loss drops less to than 3 percent. The experiments show that for small k, compression effectiveness is close to that of BWT, but is much more efficient.

7.1. *Inverse Transform Effectiveness and Efficiency Trade-offs*

In the previous section, we demonstrated how a context bounded transform can achieve near identical compression results as BWT for small values of k, and that the initial $\mathcal{O}(nk)$ context reconstruction algorithm is the most efficient choice for lower order contexts. Figure 13 summarizes trade-offs between efficiency and effectiveness of the various inversion algorithms. Each sub-graph uses a distinct scaling to improve readability. We compare only BWT, Schindler's $\mathcal{O}(nk)$ k-BWT, and our $\mathcal{O}(n)$ algorithm with explicit encoding since the $\mathcal{O}(n \log k)$

Copyright © 0000 John Wiley & Sons, Ltd.

DOI: 10.1002/spe

		$\mathcal{H}_0$	MTF-RLE					
			ARITH		RANGE		HUFF	
			bps	sec	bps	sec	bps	sec
WSJ	BWT	2.12	1.70	11.14	1.67	9.93	1.78	9.80
	k-BWT-2	3.42	3.08	4.09	3.08	2.88	3.31	**2.75**
	k-BWT-4	2.56	2.12	6.15	2.12	4.94	2.34	4.81
	k-BWT-8	2.33	1.77	8.67	1.77	7.46	1.89	7.33
	k-BWT-10	2.17	1.72	9.93	1.72	8.72	1.87	8.59
	BOOST	-	**1.65**	108.07	1.66	60.53	1.68	87.06
SOURCES	BWT	1.71	1.43	11.44	1.42	9.67	1.48	9.83
	k-BWT-2	2.45	2.15	4.71	2.16	**2.94**	2.26	3.10
	k-BWT-4	1.86	1.60	6.44	1.60	4.67	1.68	4.83
	k-BWT-8	1.72	1.50	8.83	1.49	7.06	1.56	7.22
	k-BWT-10	1.71	1.47	9.88	1.48	8.11	1.53	8.27
	BOOST	-	**1.65**	108.07	1.66	60.53	1.68	87.06
XML	BWT	1.15	0.77	9.08	0.77	8.35	0.81	8.34
	k-BWT-2	2.46	1.65	3.31	1.65	2.58	1.66	**2.57**
	k-BWT-4	1.53	0.95	5.37	0.95	4.64	1.12	4.63
	k-BWT-8	1.16	0.82	8.56	0.82	7.83	0.96	7.82
	k-BWT-10	1.15	0.82	9.42	0.82	8.69	0.94	8.68
	BOOST	-	**0.74**	84.06	0.75	48.79	0.75	72.67
DNA	BWT	1.94	1.89	12.37	1.86	10.69	1.98	10.57
	k-BWT-2	1.98	1.98	4.71	1.98	3.03	2.11	**2.91**
	k-BWT-4	1.98	1.98	6.03	1.98	4.35	2.10	4.23
	k-BWT-8	1.97	1.97	9.71	1.97	8.03	2.10	7.91
	k-BWT-10	1.96	1.97	10.98	1.97	9.30	2.10	9.18
	BOOST	-	**1.86**	99.90	1.86	58.23	1.96	92.14

Table III. Effectiveness and efficiency for common transform-based compression system combinations. The $\mathcal{H}_0$ result is not shown for BOOST since this method requires each context block to be compressed separately, resulting in effectiveness approaching $\mathcal{H}_k$.

and $\mathcal{O}(n)$ k-BWT reconstruction algorithms are always less efficient when $k < 10$ in practice. As k gets large, the cost of storing an increasingly dense context vector begins to offset any gains that might be achievable by sorting to a higher k for our $\mathcal{O}(n)$ algorithm. The WSJ test collection can be decoded twice as fast using k-BWT algorithms with little compression loss. The XML can be recovered up to 50% faster and approaches BWT effectiveness at $k = 4$. The SOURCES collection can be compressed effectively and efficiently using low order contexts and the DNA collection decompresses three times as fast at identical levels of effectiveness. Implicit context storage has negligible impact on compression effectiveness for small k for all test collections.

7.2. Forward Effectiveness and Efficiency Trade-offs

Context-bound transforms achieve similar compression effectiveness to full transform based compression systems, but can compress faster. Figure 14 shows the effectiveness and efficiency trade-offs achieved using our k-bound forward transform for different compression systems. Each sub-graph uses a distinct scaling to improve readability. We compare the effectiveness and efficiency achieved for BWT and k-BWT compression systems where $2 \leq k \leq 10$. We also include the common compression systems bzip, szip, and gzip for reference. We do not show boosting as it is an order of magnitude slower than all other compression systems tested. Note the total I/O costs are included in this comparison to allow a fair comparison to the "off the shelf" compression systems.

Copyright © 0000 John Wiley & Sons, Ltd.
Prepared using speauth.cls

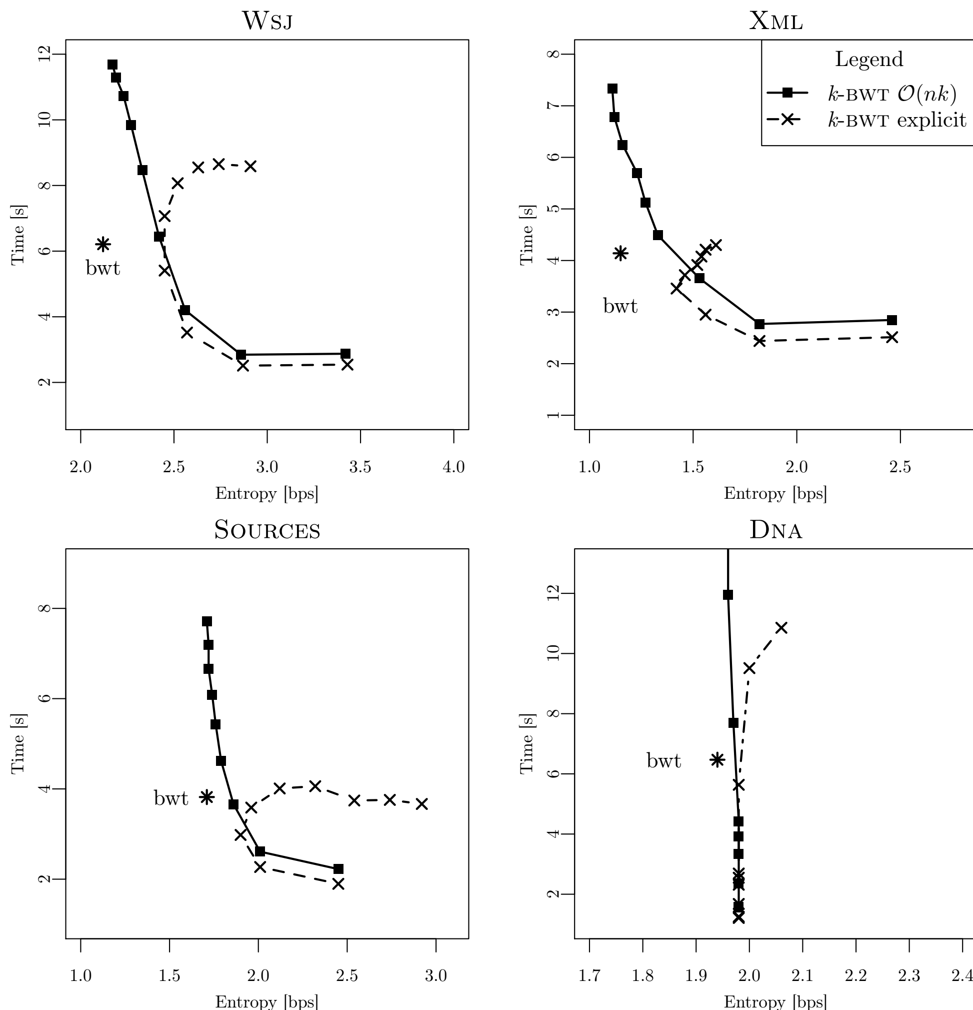

Figure 13. Efficiency vs effectiveness for block-sort inversions.

For Wsj, the k-BWT compressor is faster than gzip and at the same time achieves better compression. All compression backends show a similar effectiveness and efficiency trade-off. The BWT system is shown in the graph as an unconnected point at the top of the curve as k-BWT systems approach the full BWT as k approaches n. The BWT compression systems achieve slightly better compression but are less efficient when $k < n$. The bzip2 compressor is fast but does not achieve comparable compression effectiveness due to the fixed block size of 900 kB. The szip compressor is less efficient than our k-BWT system, for all k other than $k = 4$. For $k = 4$, szip uses a custom "super alphabet" method to only sort the text once, resulting in noticeably faster compression. This special case method can also be exploited in our compression system, but is an implementation detail we do not explore here. The Sources and Xml collections show similar behaviour. The k-BWT compressor achieves remarkable compression effectiveness on the Xml collection even for small k, but this result is more an artifact of a specific, highly compressible collection than the compression system. The Dna collection is generally not compressible without more specialized modelling steps, and thus does not show any significant trade-offs. Note that for gzip using the -9 parameter for best compression, no result is shown in the Dna graph as the running time is an order of magnitude larger than the other compressors. We also performed the same experiments for the lzma based compressor. The results are not shown in Figure 14 as the running time is significantly less efficient, and as much as 11 times

Copyright © 0000 John Wiley & Sons, Ltd.
Prepared using speauth.cls

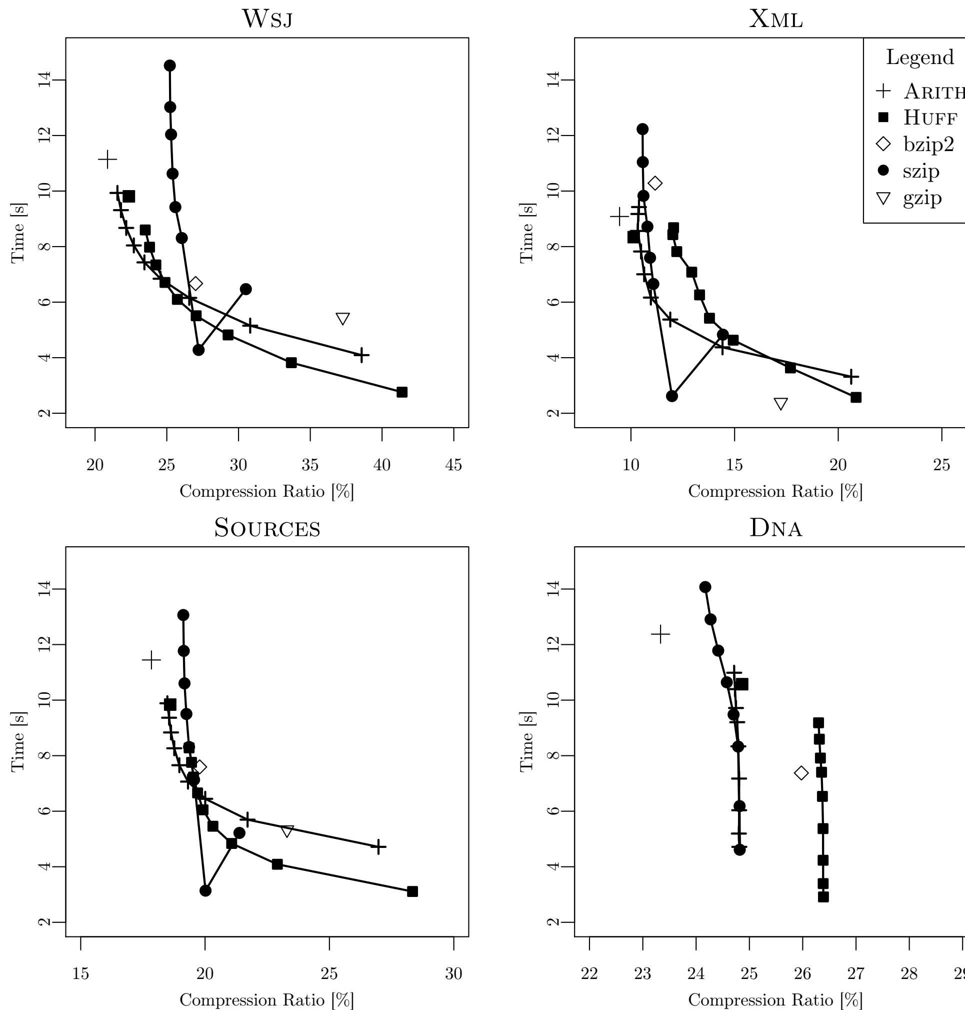

Figure 14. Efficiency vs effectiveness for bounded context-based transformation systems.

slower for DNA. However, lzma is 3% more effective on average than the best compression method shown.

8. FUTURE PROSPECTS

In this section we describe several directions for the k-BWT we wish to explore in the future.

Compression boosting is a promising technique to improve compression effectiveness when used in conjunction with block-sorted transforms [46]. However, the partitioning process has a significant time cost. The context boundaries of k-BWT could be used in combination with compression boosting to provide a more efficient partitioning process with similar effectiveness gains. Also, boosting requires the explicit storage of the partitioning information. Using the k-order boundaries to partition X_k^* would therefore make recovering the context boundaries during the reverse transform unnecessary.

Weighted Frequency Count (WFC) was proposed by Deorowicz as a second stage alternative in BWT-based compression systems [41]. The method uses context information within X^* to perform context aware frequency counting to improve effectiveness. The (WFC) tries to estimate context switches within X^* to adjust the weights for each symbol.

Copyright © 0000 John Wiley & Sons, Ltd.

Prepared using speauth.cls
DOI: 10.1002/spe

Different quantization models achieve better performance depending on properties of the input file. The context information implicitly provided by k-BWT could be used to improve these estimations, and thereby achieve better compression effectiveness.

Search in compressed text Ferragina et al. use BWT and its inherent relationship with the suffix array to support efficient search in compressed text [10]. Building a similar system based on k-BWT is an interesting open problem.

Variable depth transform Partial sorting of $\mathcal{M}$ allows faster compression. Our experiments show that for small k, compression effectiveness is comparable to BWT. However, the size of individual contexts at depth k can vary significantly. Instead of bounding the sorting depth to a fixed length in the forward transform, we could instead bound the maximum and minimum size of each context, thereby creating a variable depth transform.

9. SUMMARY

In this work, we provide an comprehensive evaluation of bounded context length block sorting. We propose a new forward transform algorithm using induced sorting that is faster than previous approaches. We also propose a new reversal algorithm which explicitly stores the context information to avoid expensive context reconstruction. We further show that existing, theoretically efficient, context recovery algorithms do not perform well in practice due to large constant factors and space usage. Finally, we highlight the efficiency and effectiveness tradeoffs for k-BWT. In future work, we intend to investigate searching in k-deep transformed, compressed text; to use the context information to improve compression effectiveness in the later stages of compression systems; and to explore variable depth transformations.

ACKNOWLEDGEMENTS

We thank Lei Chen who performed a preliminary investigation of k-BWT in his honours thesis and Ge Nong [28] who graciously provided reference implementations for context reconstruction algorithms on short notice.

REFERENCES

1. M. Burrows and D. J. Wheeler, 'A block-sorting lossless data compression algorithm.', Technical Report 124, Digital Equipment Corporation, Palo Alto, California, (May 1994).
2. P. Fenwick, 'Block sorting text compression - final report.', Technical Report 130, University of Auckland, (April 1996).
3. B. Balkenol and S. Kurtz, 'Universal data compression based on the Burrows-Wheeler transformation: Theory and practice.', *IEEE Transactions on Computers*, **49**, (10), 1043–1053, (October 2000).
4. M. Effros, 'PPM performance with BWT complexity: A new method for lossless data compression.', *Proceedings of the 10th IEEE Data Compression Conference (DCC 2000)*, eds., J. A. Storer and M. Cohn, Los Alamitos, California, March 2000, pp. 203–212. IEEE Computer Society Press.
5. G. Manzini, 'An analysis of the Burrows-Wheeler transform.', *Journal of the ACM*, **48**, (3), 407–430, (May 2001).
6. M. Effros, K. Visweswariah, S. Kulkarni, and S. Verdu, 'Universal losseless source coding with the Burrows-Wheeler transform.', *IEEE Transactions on Information Theory*, **48**, (5), 1061–1081, (May 2002).
7. P. Ferragina, R. Giancarlo, and G. Manzini, 'The engineering of a compression boosting library: Theory vs practice in BWT compression', *Proceedings of the 14th Annual European Symposium on Algorithms (ESA 2006)*, eds., Y. Azar and T. Erlebach, volume 4168 of *LNCS*. Springer, 2006, pp. 756–767.
8. J. Seward, 'On the performance of BWT sorting algorithms.', *Proceedings of the 10th IEEE Data Compression Conference (DCC 2000)*, eds., J. A. Storer and M. Cohn, Los Alamitos, California, March 2000, p. 173. IEEE Computer Society Press.
9. J. Seward, 'Space-time tradeoffs in the inverse B-W transform.', *Proceedings of the 11th IEEE Data Compression Conference (DCC 2001)*, eds., J. A. Storer and M. Cohn, Los Alamitos, California, March 2001, p. 439. IEEE Computer Society Press.
10. P. Ferragina and G. Manzini, 'Indexing compressed text.', *Journal of the ACM*, **52**, (4), 552–581, (2005). A preliminary version appeared in FOCS 2000.

Copyright © 0000 John Wiley & Sons, Ltd.

DOI: 10.1002/spe

11. P. Ferragina, R. González, G. Navarro, and R. Venturini, 'Compressed text indexes: from theory to practice', *Journal of Experimental Algorithmics*, **13**, 1.12–1.31, (2009).
12. D. Adjeroh, T. Bell, and A. Mukherjee, *The Burrows-Wheeler Transform: Data compression, suffix arrays, and pattern matching.*, Springer, New York, New York, 2008.
13. M. Schindler, 'A fast block-sorting algorithm for lossless data compression.', *Proceedings of the 7th IEEE Data Compression Conference (DCC 1997)*, eds., J. A. Storer and M. Cohn, Los Alamitos, California, March 1997, p. 469. IEEE Computer Society Press.
14. H. Yokoo, 'Notes on block-sorting data compression.', *Electronics and Communications in Japan, Part 3*, **82**, (6), 18–25, (1999).
15. S. J. Puglisi, W. F. Smyth, and A. H. Turpin, 'A taxonomy of suffix array construction algorithms.', *ACM Computing Surveys*, **39**, (2), 4–1 – 4–31, (2007).
16. J. Kärkkäinen, P. Sanders, and S. Burkhardt, 'Linear work suffix array construction.', *Journal of the ACM*, **53**, (6), 918–936, (2006).
17. M. A. Maniscalco and S. J. Puglisi, 'An efficient, versatile approach to suffix sorting', *J. Exp. Algorithmics*, **12**, 1.2:1–1.2:23, (June 2008).
18. D. Okanohara and K. Sadakane, 'A linear-time Burrows-Wheeler transform using induced sorting.', *Proceedings of the 16th International Conference on String Processing and Information Retrieval (SPIRE 2009)*, eds., J. Karlgren, J. Tarhio, and H. Hyyrö, volume 5721 of *LNCS*. Springer, August 2009, pp. 90–101.
19. J. Kärkkäinen, 'Fast BWT in small space by blockwise suffix sorting', *Theoretical Computer Science*, **387**, (3), 249–257, (2007).
20. J. Kärkkäinen and S. J. Puglisi, 'Medium-space algorithms for inverse BWT.', *Proceedings of the 18th Annual European Symposium on Algorithms (ESA 2010), Part I*, eds., M. de Berg and U. Meyer, volume 6346 of *LNCS*. Springer, 2010, pp. 451–462.
21. U. Lauther and T. Lukovszki, 'Space efficient algorithms for the Burrows-Wheeler backtransformation', *Proceedings of the 13th Annual European Symposium on Algorithms (ESA 2005)*, eds., G. S. Brodal and S. Leonardi, volume 3669 of *LNCS*. Springer, 2005, pp. 293–304.
22. T. C. Bell, J. G. Cleary, and I. H. Witten, *Text Compression*, Prentice-Hall, Englewood Cliffs, New Jersey, 1990.
23. A. Moffat and A. Turpin, *Compression and Coding Algorithms*, Kluwer Academic Publisher, Boston, first edition, 2002.
24. D. Salomon, *Data Compression: The Complete Reference*, Springer, London, fourth edition, 2007.
25. G. Navarro and V. Mäkinen, 'Compressed full text indexes', *ACM Computing Surveys*, **39**, (1), article 2, (2007).
26. B.D. Vo and G. S. Manku, 'Radixzip: linear time compression of token streams.', *Proceedings of the 33rd international conference on Very large data bases. (VLDB 2007)*, ed., C. Koch et al. ACM, September 2007, pp. 1162–1172.
27. K. Inagaki, Y. Tomizawa, and H. Yokoo, 'Novel and generalized sort-based transform for lossless data compression.', *Proceedings of the 16th International Conference on String Processing and Information Retrieval (SPIRE 2009)*, eds., J. Karlgren, J. Tarhio, and H. Hyyrö, volume 5721 of *LNCS*. Springer, August 2009, pp. 102–113.
28. G. Nong, S. Zhang, and W. H. Chan, 'Computing inverse ST in linear complexity.', *Proceedings of the 19th Annual Symposium on Combinatorial Pattern Matching (CPM 2008)*, eds., P. Ferragina and G. M. Landau, volume 5029 of *LNCS*. Springer, July 2008, pp. 178–190.
29. H. Yokoo, 'Extension and faster implementation of the GRP transform for lossless compression.', *Proceedings of the 21st Annual Symposium on Combinatorial Pattern Matching (CPM 2010)*, eds., A. Amir and L. Parida, volume 6129 of *LNCS*. Springer, June 2010, pp. 338–347.
30. G. Nong and S. Zhang, 'Efficient algorithms for the inverse sort transform.', *IEEE Transactions on Computers*, **56**, (11), 1564–1574, (2007).
31. P. Ferragina and G. Navarro. Pizza & Chili corpus – Compressed indexes and their testbeds., September 2005.
32. H. Itoh and H. Tanaka, 'An efficient method for in memory construction of suffix arrays', *Proceedings of the 6th International Symposium on String Processing and Information Retrieval (SPIRE 1999)*, eds., J. Karlgren, J. Tarhio, and H. Hyyrö. IEEE Computer Society, September 1999, pp. 81–88.
33. J. Kärkkäinen and T. Rantala, 'Engineering radix sort for strings.', *Proceedings of the 15th International Conference on String Processing and Information Retrieval (SPIRE 2008)*, eds., A. Amir, A. Turpin, and A. Moffat, volume 5280 of *LNCS*. Springer, August 2008, pp. 3–14.
34. D. Huffman, 'A method for the construction of minimum-redundancy codes', *Proc. Inst. Radio Engineers*, **40**, (9), 1098–1101, (September 1952).
35. M. Schindler, 'A fast renormalization for arithmetic coding.', *Proceedings of the 8th IEEE Data Compression Conference (DCC 1998)*, eds., J. A. Storer and M. Cohn, Los Alamitos, California, March 1998, p. 572. IEEE Computer Society Press.
36. I. H. Witten, R. M. Neal, and J. G. Cleary, 'Arithmetic coding for data compression.', *Communications of the ACM*, **30**, (6), 520–541, (June 1986).
37. A. Moffat, R. M. Neal, and I. H. Witten, 'Arithmetic coding revisited.', *ACM Transactions on Information Systems*, **16**, (3), 256–294, (1998). A preliminary version appeared in DCC 1995.
38. J. S. Culpepper and A. Moffat, 'Enhanced byte codes with restricted prefix properties', *Proceedings of the 12th International Conference on String Processing and Information Retrieval (SPIRE 2005)*, eds., M. Consens and G. Navarro, volume 3772 of *LNCS*, November 2005, pp. 1–12.
39. D. Sleator and R. E. Tarjan, 'Amortized efficiency of list update paging rules.', *Communications of the ACM*, **28**, (2), 202–208, (1985).

Copyright © 0000 John Wiley & Sons, Ltd.

Prepared using speauth.cls
DOI: 10.1002/spe

40. R. N. Horspool and G. V. Cormack, 'Constructing word-based text compression algorithms.', *Proceedings of the 2nd IEEE Data Compression Conference (DCC 1992)*, eds., J. A. Storer and M. Cohn, Los Alamitos, California, March 1992, pp. 62–71. IEEE Computer Society Press.
41. S. Deorowicz, 'Second step algorithms in Burrows-Wheeler compression algorithm', *Software Practice and Experience*, **32**, (2), 99–111, (November 2002).
42. E. Rivals, J. Delahaye, M. Dauchet, and O. Delgrange, 'A guaranteed compression scheme for repetitive DNA sequences.', *Proceedings of the 6th IEEE Data Compression Conference (DCC 1996)*, eds., J. A. Storer and M. Cohn, Los Alamitos, California, March 1996, p. 453. IEEE Computer Society Press.
43. A. Apostolico and S. Lonardi, 'Compression of biological sequences by greedy off-line textual substitution.', *Proceedings of the 10th IEEE Data Compression Conference (DCC 2000)*, eds., J. A. Storer and M. Cohn, Los Alamitos, California, March 2000, pp. 143–152. IEEE Computer Society Press.
44. X. Chen, S. Kwong, and M. Li, 'A compression algorithm for DNA sequences and its applications in genome comparison.', *Proceedings of the 4th International Conference on Research in Computational Molecular Biology (RECOMB 2000)*. ACM Press, April 2000, p. 107.
45. M. C. Brandon, D. C. Wallace, and P. Baldi, 'Data structures and compression algorithms for genomic sequence data.', *Bioinformatics*, **25**, (14), 1731–1738, (2009).
46. P. Ferragina, R. Giancarlo, G. Manzini, and M. Sciortino, 'Boosting textual compression in optimal linear time.', *Journal of the ACM*, **52**, (4), 688–713, (July 2005).

Copyright © 0000 John Wiley & Sons, Ltd.
Prepared using speauth.cls

DOI: 10.1002/spe