Available online at www.sciencedirect.com

ScienceDirect

Theoretical Computer Science 372 (2007) 115–121

Theoretical Computer Science

www.elsevier.com/locate/tcs

Note

A simple storage scheme for strings achieving entropy bounds[☆]

Paolo Ferragina[*], Rossano Venturini

Dipartimento di Informatica, University of Pisa, Italy

Received 3 July 2006; received in revised form 13 December 2006; accepted 14 December 2006

Communicated by A. Apostolico

Abstract

We propose a storage scheme for a string $S[1, n]$, drawn from an alphabet Σ, that requires space close to the k-th order empirical entropy of S, and allows one to retrieve any substring of S of length ℓ in optimal $O(1 + \frac{\ell}{\log_{|\Sigma|} n})$ time. This matches the best known bounds [R. González, G. Navarro, Statistical encoding of succinct data structures, in: Procs CPM, in: LNCS, vol. 4009, 2006, pp. 295–306; K. Sadakane, R. Grossi, Squeezing succinct data structures into entropy bounds, in: Procs ACM-SIAM SODA, 2006, pp. 1230–1239], via the use of binary encodings and tables only. We also apply our storage scheme to the Burrows–Wheeler Transform [M. Burrows, D. Wheeler, A block sorting lossless data compression algorithm, Technical Report 124, Digital Equipment Corporation, 1994], and achieve a space bound which depends on both the k-th order entropy of S and the k-th order entropy of its BW-transformed string $\texttt{bwt}(S)$.
© 2006 Elsevier B.V. All rights reserved.

Keywords: Data structures for strings; Data compression

1. Introduction

Starting from [4], the design of compressed (self-)indexes for strings became an active field of research (see [11] and references therein). The key problem addressed in these papers consists of representing a string $S[1, n]$ drawn from an alphabet Σ within compressed space, while still being able to answer various types of query (e.g. substring, approximate, ...) in efficient time, without incurring in the whole decompression of the compressed data. In these results, compressed space usually means space close to the k-th order empirical entropy of S,[1] and efficient time means something depending on the length of the searched string and as independent as possible of S's length. Various trade-offs are known, and for them we refer the reader to [11].

Recently, Sadakane and Grossi [14] addressed the foundational problem of designing a compressed storage scheme for a string S which is *provably better* than storing S as a plain array of symbols. Here, the query operation to

[☆] Part of this work appeared in the Proceedings of the 18th ACM-SIAM Symposium on Discrete Algorithms (SODA), 2007. The work was partially supported by Italian MIUR grants Italy–Israel FIRB "Pattern Discovery Algorithms in Discrete Structures, with Applications to Bioinformatics", and by a Yahoo! Research grant on "Data compression and indexing in hierarchical memories".

[*] Corresponding address: Department of Computer Science, University of Pisa, Largo Bruno Pontecorvo 3, 56127 Pisa, Italy. Tel.: +39 050 22127 64; fax: +39 050 22127 26.

E-mail addresses: ferragina@di.unipi.it (P. Ferragina), rossano@di.unipi.it (R. Venturini).

[1] This is a lower bound to the space achieved by any k-th order compressor.

be supported is the retrieval of any ℓ-long substring of S in optimal $O(1 + \frac{\ell}{\log_{|\Sigma|} n})$ time. Previous solutions [11], based on compressed indexes, incurred in an additional sub-logarithmic time overhead. The Sadakane–Grossi storage scheme achieves the optimal time bound and occupies a number of bits upper bounded by the following function[2]: $nH_k(S) + O(\frac{n}{\log_{|\Sigma|} n}((k+1)\log|\Sigma| + \log\log n))$, where $H_k(S)$ is the k-th order entropy of string S (see Definition (3)). This storage scheme is based on a sophisticated combination of various techniques: the Ziv–Lempel string encoding [16], succinct dictionaries [12], and some novel succinct data structures for supporting navigation and path-decoding in LZ-tries. Since storing S by means of a *plain* array of symbols takes $\Theta(n \log|\Sigma|)$ bits, the scheme in [14] is effective when $k = o(\log_{|\Sigma|} n)$.

More recently, González and Navarro [6] proposed a simpler storage scheme achieving the same query time and an improved space bound:

$$nH_k(S) + O\left(\frac{n}{\log_{|\Sigma|} n}(k \log|\Sigma| + \log\log n)\right). \tag{1}$$

This storage scheme exploits a statistical encoder (namely, Arithmetic) on most of S's substrings but, unlike [14], requires fixing the order k of the entropy bound in advance.

In what follows we propose a very simple storage scheme that: (1) drops the use of any compressor (either statistical or LZ-like), and deploys only binary encodings and tables; (2) matches the space bound of Eq. (1) simultaneously over all $k = o(\log_{|\Sigma|} n)$. We then exploit this storage scheme to achieve two corollary results. The first one provides a novel bound in terms of $H_k(S)$ on the compression ratio achievable by any 0-th order compressor applied on blocks of l contiguous symbols of S, with $k \le l$ (see Theorem 3). The second result shows that our storage scheme can be used upon the string Burrows–Wheeler Transformed string $\mathtt{bwt}(S)$ in order to achieve an interesting compressed-space bound which depends on the k-th order entropy of both the strings S and $\mathtt{bwt}(S)$ (see Theorem 4).

2. Some background

Let $S[1, n]$ be a string drawn from the alphabet $\Sigma = \{a_1, \ldots, a_h\}$. For each $a_i \in \Sigma$, we let n_i be the number of occurrences of a_i in S. Let $\{P_i = n_i/n\}_{i=1}^{h}$ be the empirical probability distribution for the string S. The 0-*th order empirical entropy* of S is defined as[3]

$$H_0(S) = -\sum_{i=1}^{h} P_i \log P_i. \tag{2}$$

For any string w of length k, we denote by w_S the string of single symbols following the occurrences of w in S, taken from left to right. For example, if $S = \mathtt{mississippi}$ and $w = \mathtt{si}$, we have $w_S = \mathtt{sp}$ since the two occurrences of $\mathtt{si}$ in S are followed by the symbols $\mathtt{s}$ and $\mathtt{p}$, respectively. The k-th order empirical entropy of S is defined as

$$H_k(S) = \frac{1}{|S|} \sum_{w \in \Sigma^k} |w_S| \, H_0(w_S). \tag{3}$$

Not surprisingly, for any $k \ge 0$ we have $H_k(S) \ge H_{k+1}(S)$. The value $|S|H_k(S)$ is a lower bound to the output size of any compressor that encodes each symbol of S with a code that only depends on the symbol itself and on the k immediately preceding symbols [9].

We will use $\mathcal{B} = [\epsilon, 0, 1, 00, 01, 10, 11, 000, \ldots]$ to denote the infinite sequence of binary strings ordered first by length and then lexicographically by their content, with ϵ denoting the empty string.

2.1. The Burrows–Wheeler transform

In [2] Burrows and Wheeler introduced a new compression algorithm based on a reversible transformation now called the *Burrows–Wheeler Transform* (BWT). The BWT transforms the input string S into a new string, that is usually easier to compress, via three basic steps (see Fig. 1):

[2] As stated in [6], the term $(k+1)\log|\Sigma|$ appears erroneously as k in [14]. We therefore use the correct bound in this note.

[3] Throughout this paper we assume that all logarithms are taken to the base 2, whenever not explicitly indicated, and we assume $0 \log 0 = 0$.

```
                          F              L
mississippi#              # mississipp i
ississippi#m              i #mississip p
ssissippi#mi              i ppi#missis s
sissippi#mis              i ssippi#mis s
issippi#miss              i ssissippi# m
ssippi#missi     ⟹        m ississippi #
sippi#missis              p i#mississi p
ippi#mississ              p pi#mississ i
ppi#mississi              s ippi#missi s
pi#mississip              s issippi#mi s
i#mississipp              s sippi#miss i
#mississippi              s sissippi#m i
```

Fig. 1. Example of a Burrows–Wheeler transform for the string $S =$ `mississippi`. The matrix on the right has the rows sorted in lexicographic order. The output of the BWT is the column L; in this example the string `ipssm#pissii`.

(1) append at the end of S a special symbol # smaller than any other symbol of Σ;
(2) form a *conceptual* matrix $\mathcal{M}_S$ whose rows are the cyclic shifts of string $S\#$ in lexicographic order;
(3) construct the transformed string L by taking the last column of the sorted matrix $\mathcal{M}_S$.

Notice that every column of $\mathcal{M}_S$, and hence also the transformed string L, is a permutation of $S\#$. In particular the first column of $\mathcal{M}_S$, call it F, is obtained by lexicographically sorting the symbols of $S\#$ (or, equally, the symbols of L). Note also that when we sort the rows of $\mathcal{M}_S$, we are essentially sorting the suffixes of S because of the presence of the special symbol #. This shows that: (1) there is a strong relation between $\mathcal{M}_S$ and the *suffix array* data structure built on S; (2) symbols following the same substring (*context*) in S are grouped together in L, thus giving rise to clusters of nearly identical symbols. Property 1 is crucial for designing compressed indexes (see e.g. [11]), whereas Property 2 is the key for the design of modern data compressors (see e.g. [9,3]).

The Burrows–Wheeler Transform has changed the way in which fundamental tasks for string processing and data retrieval, such as compression and indexing, are designed and engineered (see e.g. [4,7,11]). In Section 5 we will address the problem of succinct storing of the BWT, because this impacts on most modern indexing and compression tools.

3. Our storage scheme for strings

Let $S[1, n]$ be a string drawn from an alphabet Σ, and assume that n is a multiple of $b = \lfloor \frac{1}{2} \log_{|\Sigma|} n \rfloor$. If this is not the case, we append to S the missing symbols taking them as the special *null symbol*.[4] We partition S into blocks S_i of size b each. Let $\mathcal{S}$ be the set of distinct blocks of S. The number of all blocks is $\frac{n}{b}$; the number of distinct blocks is $|\mathcal{S}| = O(|\Sigma|^b) = O(n^{1/2})$.

The encoding scheme. We sort the elements of $\mathcal{S}$ per decreasing frequency of occurrence in S's partition. Let $r(S_i)$ be the rank of the block S_i in this ordering, and let $r^{-1}(j)$ be its inverse function (namely, the one that returns the block having the given rank j). The storage scheme for S consists of the following information.

- Each block S_i is assigned a codeword $\texttt{enc}(i)$ consisting of the binary string that has rank $r(S_i)$ in $\mathcal{B}$. It is simple to see that $|enc(i)| \leq \log i \leq \frac{1}{2} \log n$. Of course, $\texttt{enc}(i)$ is not a *uniquely decodable code*, but the additional tables built below will allow us to decode it in constant time and within a space bounded by Eq. (1).
- We build a bit sequence V obtained by juxtaposing the binary encodings of all S's blocks in the order of their appearance in S. Namely $V = \texttt{enc}(1) \cdots \texttt{enc}(\frac{n}{b})$.
- We store r^{-1} as a table of $O(|\Sigma|^b)$ entries, taking $O(|\Sigma|^b \log n) = o(n)$ bits.

[4] This will add to the entropy estimation a negligible additive term equal to $O(\log |\Sigma| \ \log_{|\Sigma|} n) = O(\log n)$ bits.

- To guarantee constant-time access to the encodings of S's blocks and to ensure their decodings, we use a *two-level* storage scheme for the starting positions of $\mathtt{encs}$ (see [10]). Specifically, we *logically* group every $c = \Theta(\log n)$ contiguous blocks into one *super-block*, having thus size $bc \log |\Sigma| = \Theta(\log^2 n)$ bits. Table $T_{Sblk}[1, \frac{n}{bc}]$ stores the starting position of the encoding of every super-block in V, and table $T_{blk}[1, \frac{n}{b}]$ stores the starting position in V of the encoding of every block *relative to* the beginning of its enclosing super-block. Note that the starting position of each super-block is no more than $|V| = O(\frac{n}{b} \log n) = O(n \log |\Sigma|)$, whereas the relative position of each block within its super-block is $O(\log^2 n)$. Consequently, tables T_{Sblk} and T_{blk} occupy $O(\frac{n}{bc} \log |V| + \frac{n}{b} \log \log n) = O(\frac{n \log \log n}{\log_{|\Sigma|} n})$ bits overall, and guarantee a constant-time access to every codeword $\mathtt{enc}(i)$ and its length.[5]

Theorem 1. *Our storage scheme encodes $S[1, n]$ in $|V| + O(\frac{n \log \log n}{\log_{|\Sigma|} n})$ bits, which is upper bounded by Eq. (1), simultaneously over all $k = o(\log_{|\Sigma|} n)$.*

Proof. For every position i, $k < i \le n$, let us denote by f_i the empirical probability of seeing $S[i]$ after the k-th order context $S[i-k, i-1]$. According to the notation in Section 2, this can be rephrased by saying that f_i is the frequency of occurrence of symbol $S[i]$ within w_S, where $w = S[i-k, i-1]$. It is easy to see that a (semi-static) k-order modeler can compute all the frequencies f_i via two passes over S, and hence in $O(n)$ time.

Arithmetic encoding is one of the most effective statistical encoders [15]. Given the f_is, it represents the string S with a range of size $F = f_1 \times f_2 \times \cdots \times f_n$. It is well known [15] that $2 + \log(1/F) = 2 + \sum_{i=1}^{n} \log(1/f_i)$ bits are enough for distinguishing a number within that range. The binary representation of this number is the Arithmetic compression of S. If we compute $\sum_{i=k+1}^{n} \log(1/f_i)$, and then group all the terms referring to the same k-th order context, we obtain a summation upper bounded by $nH_k(S)$ (see Eq. (3)). Additionally, since $f_i \ge 1/n$, we have that $\sum_{i=1}^{k} \log(1/f_i) = O(k \log n)$. As a result, a (semi-static) k-th order Arithmetic encoder compresses the whole S within $nH_k(S) + 2 + O(k \log n)$ bits.

Let us introduce a compressor E that encodes each block S_i of S individually: the first k symbols of S_i are represented explicitly; the remaining $b-k$ symbols of S_i are compressed via the above k-th order Arithmetic encoder (hence using their k-th order frequencies fs). It is easy to observe that the codeword so assigned to S_i uniquely identifies it. This blocking approach increases the above Arithmetic encoding of the whole S by $O((n/b)k \log |\Sigma|)$ bits, which accounts for the cost of explicitly storing the first k symbols of each S_i.

To show that the string V produced by our storage scheme $\mathtt{enc}$ is shorter than the compressed string produced by E, it suffices to note that the codewords assigned by E are a subset of $\mathcal{B}$, whereas the codewords assigned by $\mathtt{enc}$ are the first $|\mathcal{S}|$ binary strings of $\mathcal{B}$. Given that $\mathcal{B}$ is the set of the shortest codewords assignable to $\mathcal{S}$'s strings, our encoding $\mathtt{enc}$ is better than E because it follows the *golden rule* of data compression: it assigns shorter codewords to more frequent symbols. Summing up the cost of the block's encodings and the space occupancy of the decoding table, we get the space bound of Eq. (1), whenever $k = o(\log_{|\Sigma|} n)$ and independently of it. □

We now show how to decode in constant time a generic block S_k. This will be enough to prove the result for any ℓ-long substring of S. We first derive the starting position $p(k)$ of the string $\mathtt{enc}(k)$ that encodes S_k in V. Namely, we compute the super-block number $h = \lceil k/c \rceil$ containing $\mathtt{enc}(k)$, and its starting bit-position $y = T_{Sblk}[h]$ within V. Then, we compute $x = T_{blk}[k]$ as the relative bit-position of $\mathtt{enc}(k)$ within its enclosing super-block. Thus $p(k) = x + y$. Similarly, we derive the starting position $p(k+1)$ of $\mathtt{enc}(k+1)$ in V (if any, otherwise we set $p(k+1) = |V| + 1$). We can thus fetch $\mathtt{enc}(k) = V[p(k), p(k+1) - 1]$ in constant time since $|\mathtt{enc}(k)| = p(k+1) - p(k) = O(\log n)$ bits.

We finally decode $\mathtt{enc}(k)$ as follows. Let v be the integer value represented by the binary string $\mathtt{enc}(k)$, where $v = 0$ if $\mathtt{enc}(k) = \epsilon$. Because of the canonical ordering of $\mathcal{S}$, S_k is computed as the block having rank $z = 2^{|\mathtt{enc}(k)|} + v$. That is, $S_k = r^{-1}(z)$.

Theorem 2. *Our storage scheme retrieves any substring of S of length ℓ in optimal $O(1 + \frac{\ell}{\log_{|\Sigma|} n})$ time.*

Proof. The algorithm described above allows us to retrieve any block S_k in constant time. The theorem follows by observing that any ℓ-long substring $S[j, j+l-1]$ spans $O(1 + \frac{l}{\log_{|\Sigma|} n})$ blocks of S. □

[5] It suffices to compute the starting position of $\mathtt{enc}(i)$ and $\mathtt{enc}(i+1)$, if any.

4. Huffman on blocks of symbols

It is well known that the Huffman compressor cannot represent a symbol with less than one bit. To circumvent this, the string S is usually partitioned into $\frac{n}{l}$ blocks of length l each, and then Huffman is applied onto the alphabet Σ_l of these new symbols, i.e. l-long blocks. This blocking strategy spreads the per-symbol inefficiency over the entire block, thus reducing it to $\frac{1}{l}$ bits. It is natural to ask which is the compression ratio of this block-Huffman algorithm. The following theorem bounds the 0-th order entropy of S_l by the k-th order empirical entropy of original string S.

Theorem 3. *$H_0(S_l) \leq l H_k(S) + O(k \log |\Sigma|)$, simultaneously over all $k \leq l$.*

Proof. Consider the compressor E in the proof of Theorem 1. E does not depend on the size of the blocks into which S has been decomposed. Hence, we can set $b = l$, apply E onto the blocked S and thus assign a distinct prefix-free codeword to each distinct block in $\mathcal{S}$ (i.e. symbol of Σ_l). As seen in that proof, the space necessary for representing the whole S_l is bounded by $nH_k(S) + O((n/l)k \log |\Sigma|)$ bits. The stated theorem follows by the classical information-theory lower bound, since every prefix-free encoder needs on S_l at least $|S_l| H_0(S_l) = \frac{n}{l} H_0(S_l)$ bits. □

We note that the case $k = 0$ of Theorem 3 has been proved in [13]. Given Theorem 3, the output produced by Huffman over S_l is bounded by $\frac{n}{l} H_0(S_l) + \frac{n}{l} \leq nH_k(S) + O(\frac{n}{l}(k \log |\Sigma| + 1))$.

5. BWT compression and access

In Section 2 we introduced the Burrows–Wheeler transform of a string $S[1, n]$, denoted by $\mathtt{bwt}(S)$. In this section we show that our storage scheme can be used upon the string $\mathtt{bwt}(S)$ in order to achieve an interesting compressed-space bound which depends on both $H_k(S)$ and $H_k(\mathtt{bwt}(S))$. The relation between these two entropies will be commented on below.

Theorem 4. *Our storage scheme applied on the string $L = \mathtt{bwt}(S)$ takes no more than $\min\{nH_k(L), nH_k(S)\} + o(n \log |\Sigma|)$ bits, simultaneously over all $k = o(\log_{|\Sigma|} n)$. Any ℓ-long substring of L can be retrieved in optimal $O(1 + \frac{\ell}{\log_{|\Sigma|} n})$ time.*

Proof. Let $C_k(j)$ be the k-long prefix of the j-th row of $\mathcal{M}_S$. By the properties of $\mathtt{bwt}(S)$ (see Section 2), $C_k(j)$ follows $L[j]$ in S and thus $C_k(j)$ is the *following* k-long context of $L[j]$. If we change in Definition (3) of $H_k(S)$ the notion of *preceding* k-long contexts with the one of *following* k-long contexts, the difference between the two quantities turns out negligible [4]. Therefore, to ease our discussion we consider the *following* k-long contexts and work with $H_k(S)$ as it was defined over them.

We partition L into substrings of length $b = \lfloor \frac{1}{2} \log_{|\Sigma|} n \rfloor$, say $L_1 L_2 \ldots L_{n/b}$ (called hereafter *blocks*). Note that each block L_i corresponds to a range of b rows in the BWT-matrix $\mathcal{M}_S$. We say that the block L_i is *k-prefix-equal* if all rows in $\mathcal{M}_S[(i-1)b + 1, ib]$ have the same prefix of length k. Otherwise, L_i is said to be *k-prefix-different*. The total number of k-prefix-different blocks is $O(|\Sigma|^k)$, because that is the number of distinct strings of length k over Σ. Moreover, we note that the first k symbols of S belong to k-prefix-different blocks because of the special symbol #.

To prove the theorem we consider a preliminary encoding scheme for L's blocks. A k-prefix-different block L_i is written without any compression as k occurrences of a special *null* symbol plus L_i itself. This takes $O((b + k) \log |\Sigma|) = O(\log n + k \log |\Sigma|)$ bits. Since there are $O(|\Sigma|^k)$ k-prefix-different blocks and we assumed $k = o(\log_{|\Sigma|} n)$, the (plain) encoding of the k-prefix-different blocks takes $O(|\Sigma|^k \log n) = o(n)$ bits.

As far as k-prefix-equal blocks are concerned, we encode a block L_i as follows: we write explicitly the k-long following context shared by all L_i's symbols, using $k \log |\Sigma|$ bits; and then use a k-th order Arithmetic encoder on the individual symbols of L_i. This encoder computes for any symbol $L[j]$ the empirical probability f_j of seeing this symbol *followed* by the context $C_k(j)$ in S. In the proof of Theorem 1, we considered f as the preceding contexts and showed that a (semi-static) k-th order Arithmetic encoder that uses the fs compresses S within $H_k(S) + (n/b)k \log |\Sigma| + O(k \log n) + 2$ bits. Here we are compressing L, which is a permutation of string S, and we are considering the following contexts of S's symbols. Given our comment above on the definition of $H_k(S)$, we can conclude that this bound still holds for the Arithmetic encoder applied on L. Summing up, the space required by this encoding scheme over all blocks of L is $nH_k(S) + O((n/b)k \log |\Sigma|)$ bits.

Let us now take our storage scheme enc of Section 3, apply it to string L, and compare the length of the resulting compressed string against the previous encoding. By Theorem 1, we know that enc encodes L within $nH_k(L) + O((n/b)k \log |\Sigma|)$ bits, and this proves one part of the theorem. As far as the other term $H_k(S)$ is concerned, we observe that any block L_i may occur many times in the partition of L and each occurrence may have associated a different k-long following context. As a result, the above scheme encodes all occurrences of L_i with at most $O(|\Sigma|^k)$ different codewords, because it has at most $|\Sigma|^k$ distinct k-contexts (as a k-prefix-equal block) and at most $|\Sigma|^k$ plain encodings (as a k-prefix-different block). If we re-assign to all these codewords the shortest one, we have that each distinct block of L gets one codeword in $\mathcal{B}$. Following an argument similar to the one used in the proof of Theorem 1, this encoding of L's blocks is worse than enc because this latter assigns the shortest codewords of $\mathcal{B}$ to the distinct blocks of L. Therefore enc takes no more than $nH_k(S) + O((n/b)k \log |\Sigma|)$ bits. $\square$

The relation between $H_k(S)$ and $H_k(L)$ is not fully known. In [6] it was proved that $H_1(L) \leq 1 + H_k(S) \log |\Sigma| + o(1)$ for any $k < (1 - \epsilon) \log_{|\Sigma|} n$ and any constant $0 < \epsilon < 1$. Actually the gap may be quite large. For example, let us consider the string $S = (bba)^m$ and set $k = 1$. By Eq. (3) we have

$$nH_1(S) = (m-1)H_0(b^{m-1}) + 2mH_0((ba)^m) = 2m = \frac{2}{3}n.$$

On the other hand, since $L = \mathtt{bwt}(S) = b^{2m}a^m$, we have

$$\begin{aligned} nH_1(L) &= (m-1)H_0(a^{m-1}) + 2mH_0(b^{2m-1}a) \\ &= -(2m-1)\log\frac{2m-1}{2m} - \log\frac{1}{2m} \\ &= 2m\log 2m - (2m-1)\log(2m-1) \\ &= O(\log n) \end{aligned}$$

which is exponentially smaller than $nH_1(S)$, for any $m > 1$. On the other hand, we show an example in which $nH_1(L) > nH_1(S)$. Let $S = (a_1 a_2 \ldots a_m)^m$ and $k = 1$. We have $nH_1(S) = 0$. Since $L = \mathtt{bwt}(S) = a_m^m a_1^m \ldots a_{m-1}^m$, we have

$$nH_1(L) = -(m-1)\left((m-1)\log\frac{m-1}{m} + \log\frac{1}{m}\right) = \Theta(\sqrt{n}\ \log\sqrt{n}).$$

6. Conclusions

The simplification we have proposed in this paper to the results of [6,14] drives us to two possible considerations. One is that we now have a *class-note* solution for the string storage problem that, as deeply illustrated in [14], may find successful applications into many other interesting contexts: e.g. it may turn succinct or 0-th order entropy data structures into k-th order entropy data structures (see [1,8,14] and references therein). The second consideration refers to future research. All known solutions are far from being usable in practice because of the additive term which usually *dominates* the k-th order entropy term. More research is still needed to either achieve a space bound close to the one attainable with the k-th order compressors of the BZIP-family [9,7,3], for which the additive term is $O(|\Sigma|^k \log n)$ bits rather than $o(n \log |\Sigma|)$ bits, or to show a lower bound related to k-th order entropy, in the vein of [5]. Since our storage scheme, unlike that of [14,6], does not use any sophisticated data compression machinery, we are led to think that there is room for improvement!

References

[1] J. Barbay, J.I. Munro, M. He, S.S. Rao, Succinct indexes for strings, binary relations and multi-labeled trees, in: Procs ACM-SIAM SODA, 2007.
[2] M. Burrows, D. Wheeler, A block sorting lossless data compression algorithm, Technical Report 124, Digital Equipment Corporation, 1994.
[3] P. Ferragina, R. Giancarlo, G. Manzini, M. Sciortino, Boosting textual compression in optimal linear time, Journal of the ACM 52 (4) (2005) 688–713.
[4] P. Ferragina, G. Manzini, Indexing compressed text, Journal of the ACM 52 (4) (2005) 552–581.
[5] A. Golynski, Optimal lower bounds for rank and select indexes, in: Procs ICALP, in: LNCS, vol. 4051, 2006, pp. 370–381.
[6] R. González, G. Navarro, Statistical encoding of succinct data structures, in: Procs CPM, in: LNCS, vol. 4009, 2006, pp. 295–306.

[7] R. Grossi, A. Gupta, J. Vitter, High-order entropy-compressed text indexes, in: Procs ACM-SIAM SODA, 2003, pp. 841–850.
[8] J. Jansson, K. Sadakane, K.K. Sung, Ultra-succinct representation of ordered trees, in: Procs ACM-SIAM SODA, 2007.
[9] G. Manzini, An analysis of the Burrows–Wheeler transform, Journal of the ACM 48 (3) (2001) 407–430.
[10] I. Munro, Tables, in: Procs FST-TCS, in: LNCS, vol. 1180, 1996, pp. 37–42.
[11] G. Navarro, V. Mäkinen, Compressed full-text indexes, ACM Computing Surveys, 2007 (in press).
[12] R. Raman, V. Raman, S. Srinivasa Rao, Succinct indexable dictionaries with applications to encoding k-ary trees and multisets, in: Procs ACM-SIAM SODA, 2002, pp. 233–242.
[13] K. Sadakane, New text indexing functionalities of the compressed suffix arrays, Journal of Algorithms 48 (2) (2003) 294–313.
[14] K. Sadakane, R. Grossi, Squeezing succinct data structures into entropy bounds, in: Procs ACM-SIAM SODA, 2006, pp. 1230–1239.
[15] I.H. Witten, A. Moffat, T.C. Bell, Managing Gigabytes: Compressing and Indexing Documents and Images, second edn, Morgan Kaufmann Publishers, Los Altos, CA 94022, USA, 1999.
[16] J. Ziv, A. Lempel, Compression of individual sequences via variable length coding, IEEE Transactions on Information Theory 24 (1978) 530–536.