Statistical Encoding of Succinct Data Structures

Rodrigo González[2] ⋆ and Gonzalo Navarro[1,2] ⋆⋆

[1] Center for Web Research
[2] Department of Computer Science, University of Chile.
{rgonzale,gnavarro}@dcc.uchile.cl

Abstract. In recent work, Sadakane and Grossi [SODA 2006] introduced a scheme to represent any sequence $S = s_1 s_2 \ldots s_n$, over an alphabet of size σ, using $nH_k(S) + O(\frac{n}{\log_\sigma n}(k \log \sigma + \log \log n))$ bits of space, where $H_k(S)$ is the k-th order empirical entropy of S. The representation permits extracting any substring of size $\Theta(\log_\sigma n)$ in constant time, and thus it completely replaces S under the RAM model. This is extremely important because it permits converting any succinct data structure requiring $o(|S|) = o(n \log \sigma)$ bits in addition to S, into another requiring $nH_k(S) + o(n \log \sigma)$ (overall) for any $k = o(\log_\sigma n)$. They achieve this result by using Ziv-Lempel compression, and conjecture that the result can in particular be useful to implement compressed full-text indexes.
In this paper we extend their result, by obtaining the same space and time complexities using a simpler scheme based on statistical encoding. We show that the scheme supports appending symbols in constant amortized time. In addition, we prove some results on the applicability of the scheme for full-text self-indexing.

1 Introduction

Recent years have witnessed an increasing interest on succinct data structures, motivated mainly by the growth over time on the size of textual information. This has triggered a search for less space-demanding data structures bounded by the entropy of the original text. Their aim is to represent the data using as little space as possible, yet efficiently answering queries on the represented data. Several results exist on the representation of sequences [9, 3, 18, 19], trees [15, 6], graphs [15], permutations and functions [14, 16], and texts [8, 20, 4, 7, 17, 5], to name a few.

Several of those *succinct* data structures are built over a sequence of symbols $S[1, n] = s_1 s_2 \ldots s_n$, from an alphabet A of size σ, and require only $o(|S|) = o(n \log \sigma)$ additional bits in addition to S itself (S requires $n \log \sigma$ bits[3]). A more ambitious goal is a *compressed* data structure, which takes overall space proportional to the compressed size of S and still is able to recover any substring of S and manipulate the data structure.

A very recent result by Sadakane and Grossi [22] gives a tool to convert *any* succinct data structure on sequences into a compressed data structure. More precisely, they show that S can be encoded using $nH_k(S) + O(\frac{n}{\log_\sigma n}(k \log \sigma + \log \log n))$ bits of space[4], where $H_k(S)$ is the k-th order empirical entropy of S [12]. ($H_k(S)$ is a lower bound to the space achieved by any k-th order compressor applied to S.) Their structure permits retrieving any substring of S of $\Theta(\log_\sigma n)$

⋆ Work supported by Mecesup Grant UCH 0109, Chile.
⋆⋆ Supported in part by Millennium Nucleus Center for Web Research, Grant P04-067-F, Mideplan, Chile.
[3] In this paper log stands for $\log_2$.
[4] The term $k \log \sigma$ appears as k in [22], but this is a mistake [21]. The reason is that they take from [10] an extra space of the form $\Theta(kt + t)$ as stated in Lema 2.3, whereas the proof in Theorem A.4 gives a term of the form $kt \log \sigma + \Theta(t)$.

www.manaraa.com

symbols in constant time. Under the RAM model of computation this is equivalent to having S in explicit form.

In particular, for sufficiently small $k = o(\log_\sigma n)$, the space is $H_k(S) + o(n \log \sigma)$. Any succinct data structure that requires $o(n \log \sigma)$ bits in addition to S can thus be replaced by a compressed data structure requiring $nH_k(S) + o(n \log \sigma)$ bits overall, whereas any access to S is replaced by an access to the novel structure. Their scheme is based on Ziv-Lempel encoding and is rather involved.

In this paper we show how the same result can be achieved by much simpler means. We present an alternative scheme based on k-th order modeling plus statistical encoding, just as a normal statistical compressor would process S. By adding some extra structures, we are able of retrieving any substring of S of $\Theta(\log_\sigma n)$ symbols in constant time. Although any statistical encoder works, we obtain the best results (matching exactly those of Sadakane and Grossi) using Arithmetic encoding [1]. Furthermore, we show that we can append symbols to S without changing the space complexity, in constant amortized time per symbol.

In addition, we study the applicability of this technique to full-text self-indexes. Compressed self-indexes replace a text $T[1, n]$ by a structure requiring $O(nH_0(T))$ or $O(nH_k(T))$ bits of space. In order to provide efficient pattern matching over T, many of those structures [4, 17, 5] achieve space proportional to $nH_k(T)$ by first applying the Burrows-Wheeler Transform [2] over T, $S[1, n] = bwt(T)$, and then struggling to represent S in efficient form. An additional structure of $o(|S|)$ bits gives the necessary functionality to implement the search. One could thus apply the new structure over S, so that the overall structure requires $nH_k(S) + o(|S|)$ bits. Yet, the relation between $H_k(S)$ and $H_k(T)$ remains unknown. In this paper we move a step forward by proving a positive result: $H_1(S) \leq H_k(T) \log \sigma + o(1)$ for small $k = o(\log_\sigma n)$. We note that, for example, the Run-Length FM-Index [11] achieves precisely $nH_k(T) \log \sigma + O(n)$ bits of space by rather involved means (albeit for larger k). This result shows that (essentially) the same can be achieved by applying the new structure over S.

Several indexes, however, compress $S = bwt(T)$ by means of a *wavelet tree* [7] on S, $wt(S)$. This is a balanced tree storing several binary sequences. Each such sequence B can be represented using $|B|H_0(B)$ bits of space. If we call $nH_0(wt(S))$ the overall resulting space, it turns out that $nH_0(wt(S)) = nH_0(S)$. A natural idea advocated in [22] is to use a k-th order representation for the binary sequences B, yielding space $nH_k(wt(S))$. Thus the question about the relationship between $H_k(wt(S))$ and $H_k(S)$ is raised. In this paper we exhibit examples where either is larger than the other. In particular, we show that when moving from $wt(S)$ to S, the k-th order entropy grows at least by a factor of $\Theta(\log k)$.

2 Background and notation

Hereafter we assume that $S[1, n] = S_{1,n} = s_1 s_2 \dots s_n$ is the sequence we wish to encode and query. The symbols of S are drawn from an alphabet $A = \{a_1, \dots, a_\sigma\}$ of size σ. We write $|w|$ to denote the length of sequence w. We define n_S^a de number of times that symbol $a \in A$ appears in S.

Let $B[1, n]$ be a binary sequence. Function $rank_b(B, i)$ returns the number of times b appears in the prefix $B[1, i]$. Function $select_b(B, i)$ returns the position of the i-th appearance of b within sequence B. Both *rank* and *select* can be computed in constant time using $o(n)$ bits of space in addition to B [9, 13].

المنارة للاستشارات
www.manaraa.com

2.1 The k-th order empirical entropy

The empirical entropy resembles the entropy defined in the probabilistic setting (for example, when the input comes from a Markov source). However, the empirical entropy is defined for any string and can be used to measure the performance of compression algorithms without any assumption on the input [12].

The empirical entropy of k-th order is defined using that of zero-order. This is defined as

$$H_0(S) = -\sum_{a \in A} \frac{n_S^a}{n} \log_2(\frac{n_S^a}{n}) \tag{1}$$

with n_S^a the number of occurrences of symbol a in sequence S. This definition extends to $k > 0$ as follows. Let A^k be the set of all sequences of length k over A. For any string $w \in A^k$, called a context of size k, let w_S be the string consisting of the concatenation of characters following w in S. Then, the k-th order empirical entropy of S is

$$H_k(S) = \frac{1}{n} \sum_{w \in A^k} |w_S| H_0\left(w_S\right). \tag{2}$$

The k-th order empirical entropy captures the dependence of symbols upon their context. For $k \geq 0$, $nH_k(S)$ provides a lower bound to the output of any compressor that considers a context of size k to encode every symbol of S. Note that the uncompressed representation of S takes $n \log \sigma$ bits, and that $0 \leq H_k(S) \leq H_{k-1}(S) \leq \ldots \leq H_1(S) \leq H_0(S) \leq \log \sigma$.

2.2 Statistical encoding

We are interested in the use of semi-static statistical encoders in this paper. So our problem consists in: given a sequence $S[1, n]$, and a k-th order *modeler* that yields probabilities $p_1, p_2, \ldots, p_n$ for each of those symbols, encode the successive symbols of S trying to use $-p_i \log p_i$ bits for s_i. If we reach exactly $-p_i \log p_i$ bits, the overall number of bits produced is $nH_k(S)$.

Different encoders provide different approximations to the ideal $-p_i \log p_i$ bits. The simplest encoder is probably Huffman coding [1], while the best one, from the point of view of the number of bits generated, is Arithmetic coding [1]. There are other statistical encoders such as Shannon-Fano Coding [1].

Given a statistical encoder E and a sequence S, we call $E(S)$ the bitwise output of E over S and $|E(S)|$ its bit length . We call $f_k(E, S) = |E(S)| - nH_k(S)$ the extra space in bits needed to encode S using E, on top of the k-th order empirical entropy of S. For example, the wasted space of Huffman encoding is bounded by 1 bit per symbol, and thus $f_k(\text{Huffman}, S) < |S|$ (tighter bounds exist but are not useful for this paper [1]). On the other hand, Arithmetic encoding approaches $-p_i \log p_i$ as closely as desired, requiring only at most two extra bits to terminate the whole sequence [1, Section 5.26]. Thus $f_k(\text{Arithmetic}, S) \leq 2$.

2.3 Implementing succinct full-text self-indexes

A *succinct full-text index* provides fast search functionality using a space proportional to that of the text itself. A less space-demanding index, in particular, using space proportional to that of the

المنارة للاستشارات
www.manaraa.com

compressed text is known as a *compressed full-text index*. Those indexes that contain sufficient the information to recreate the original text are known as *self-indexes*. An example of the latter ones is the FM-index family [4, 17, 5] based on the Burrows-Wheeler Transform [2] and the concept of *backward search* provided by the LF-mapping [2, 4].

The Burrows-Wheeler Transform is a reversible transformation from strings to strings.

Definition 1. *Given a text* $T_{1,n}$ *and its suffix array* $SA[1,n]$*, the* Burrows-Wheeler transform (BWT) *of* T*,* $bwt(T) = T^{bwt}$*, is defined as* $T_i^{bwt} = T_{SA[i]-1}$*, except when* $SA[i] = 1$*, where* $T_i^{bwt} = T_n$*.*

That is, T^{bwt} is formed by sequentially traversing the suffix array SA and concatenating the characters that *precede* each suffix. Although the transformation does not compress the text, the transformed text is easier to compress by local optimization methods [12].

A *cyclic shift* of $T_{1,n}$ is any string of the form $T_{i,n}T_{1,i-1}$. Let M be a matrix containing all the cyclic shifts of T in lexicographical order. Let F be the first and L the last column of M. Assuming T is terminated with a special character "\$", different from any other, the cyclic shifts $T_{i,n}T_{1,i-1}$ are sorted exactly like the suffixes $T_{i,n}$. Thus M is essentially the suffix array A of T, and L is the list of characters preceding each suffix, that is, $L = T^{bwt}$. On the other hand, F is a sorted list of all the characters in T.

In order to reverse the BWT, we need to be able two know, given a character in L, where it appears in F. This is called the *LF-mapping* [2, 4].

Definition 2. *Given strings* F *and* L *resulting from the BWT of text* T*, the* LF-mapping *is a function* $LF : [1,n] \longrightarrow [1,n]$*, such that* $LF(i)$ *is the position in* F *where character* $L[i]$ *occurs.*

The following lemma gives the formula for the LF-mapping [2, 4].

Lemma 1. *Let* $T_{1,n}$ *be a text and* F *and* L *be the result of its BWT. Let* $C : A \longrightarrow [1,n]$ *and* $Occ_L : A \times [1,n] \longrightarrow [1,n]$*, such that* $C(c)$ *is the number of occurrences in* T *of characters alphabetically smaller than c, and* $Occ_L(c,i)$ *is the number of occurrences of character* c *in* $L[1,i]$*. Then, it holds* $LF(i) = C(L[i]) + Occ_L(L[i],i)$*.*

The LF-mapping is also the key for the search procedures of full-text indexes [4, 17, 5]. As C needs only $O(\sigma \log n)$ bits, the main issue is how to compute $Occ_L(c,i)$ fast using little space.

There are different ways of implement $Occ_L(c,i)$. One way consists in using the wavelet tree [7] of $S = bwt(T) = L$.

Definition 3. *Given a sequence* $S[1,n]$ *the wavelet tree* $wt(S)$ *[7] built on* S *uses* $nH_0(S) + O(n \log\log n / \log_\sigma n)$ *bits of space and supports in* $O(\log \sigma)$ *time the following operations:*

- *given* q*,* $1 \leq q \leq n$*, the retrieval of the character* $S[q]$*;*
- *given* $c \in A$ *and* q*,* $1 \leq q \leq n$*, the computation of* $Occ_S(c,q)$*.*

The wavelet tree is a perfect binary tree of height $\lceil \log \sigma \rceil$, built on the alphabet symbols, such that the root represents the whole alphabet and each leaf represents a distinct alphabet symbol. If a node v represents alphabet symbols in the range $A^v = [i,j]$, then its left child v_l represents $A^{v_l} = [i, \frac{i+j}{2}]$ and its right child v_r represents $A^{v_r} = [\frac{i+j}{2} + 1, j]$.

المنارة للاستشارات
www.manaraa.com

We associate to each node v the subsequence S^v of S formed by the characters in A^v. However, sequence S^v is not really stored at the node. Instead, we store a bit sequence B^v telling whether characters in S^v go left or right, that is, $B_i^v = 1$ iff $S_i^v \in A^{v_r}$.

All queries on S are easily answered in $O(\log \sigma)$ time with the wavelet tree, providing each bit vector B^v with additional $o(|B^v|)$ bits to answer *rank* and *select* in constant time [9, 13].

3 A new entropy-bound succinct data structure

Given a sequence $S[1, n]$ over an alphabet A of size σ, we encode S into a compressed data structure S' within entropy bounds. To perform all the original operations over S under the RAM model, it is enough to allow extracting any $b = \frac{1}{2}\log_\sigma n$ consecutive symbols of S, using S', in constant time.

3.1 Data structures for substring decoding

We describe our data structure to represent S in $nH_k(S) + O(\frac{n}{\log_\sigma n}(k \log \sigma + \log\log n))$ bits, and to permit the access of any substring of size $b = \lfloor \frac{1}{2}\log_\sigma n \rfloor$ in constant time. This structure is built using any statistical encoder E as described in Section 2.2.

Structure. We divide S into blocks of length $b = \lfloor \frac{1}{2}\log_\sigma n \rfloor$ symbols. Each block will be represented using at most $b' = \lfloor \frac{1}{2}\log n \rfloor$ bits (and hopefully less). We define the following sequences indexed by block number $i = 0, \ldots, \lfloor n/b \rfloor$:

- $S_i = S[bi + 1, b(i + 1)]$ is the sequence of symbols forming the i-th block of S.
- $C_i = S[bi - k + 1, bi]$ is the sequence of symbols forming the k-th order context of the i-th block (a dummy value is used for C_0).
- $P_i = E(S_i)$ is the encoded sequence for the i-th block of S, initializing the k-th order modeler with context C_i.
- $\mu_i = |P_i|$ is the size in bits of P_i.
- $\tilde{P}_i = \begin{cases} S_i \text{ if } \mu_i > b' \\ P_i \text{ otherwise} \end{cases}$, is the shortest sequence among P_i and S_i.
- $\tilde{\mu}_i = |\tilde{P}_i| \le \min(b', \mu_i)$ is the size in bits of $\tilde{P}_i$.

The idea behind $\tilde{P}_i$ is to ensure that no encoded block is longer than b' bits (which could happen if a block contains many infrequent symbols). These special blocks are encoded explicitly.

Our compressed representation of S stores the following information:

- $W[0, \lfloor n/b \rfloor]$: A bit array such that
$$W[i] = \begin{cases} 0 \text{ if } \mu_i > b' \\ 1 \text{ otherwise} \end{cases},$$
with the additional $o(n/b)$ bits to answer rank queries over W in constant time [9, 13].
- $C[1, rank(W, \lfloor n/b \rfloor)]$: $C[rank(W, i)] = C_i$, that is, the k-th order context for the i-th block of S iff $\mu_i \le b'$, with $1 \le i \le \lfloor n/b \rfloor$.
- $U = \tilde{P}_0 \tilde{P}_1 \ldots \tilde{P}_{\lfloor n/b \rfloor}$: A bit sequence obtained by concatenating all the variable-length $\tilde{P}_i$.

المنارة للاستشارات
www.manaraa.com

– $T : A^k \times 2^{b'} \longrightarrow 2^b$: A table defined as $T[\alpha, \beta] = \gamma$, where α is any context of size k, β represents any encoded block of b' bits at most, and γ represents the decoded form of β, truncated to the first b symbols (as less than the b' bits will be usually necessary to obtain the b symbols of the block).
– Information to answer where each $\tilde{P}_i$ starts within U. We group together every $c = \lceil \log n \rceil$ consecutive blocks to form superblocks and store two tables:
  - $R_g[0, \lfloor n/(bc) \rfloor]$ contains the absolute position of each superblock.
  - $R_l[0, \lfloor n/b \rfloor]$ contains the relative position of each block with respect to the beginning of its superblock.

3.2 Substring decoding algorithm

We want to retrieve $q = S[i, i+b-1]$ in constant time. To achieve this, we take the following steps:

1. We calculate $j = i$ div b and $j' = (i + b - 1)$ div b.
2. We calculate $h = j$ div c, $h' = (j+1)$ div c and $u = U[R_g[h] + R_l[j], R_g[h'] + R_l[j+1] - 1]$, then
   – if $W[j] = 0$ then we have $S_j = u$.
   – if $W[j] = 1$ then we have $S_j = T[C[rank(W, j)], u']$, where u' is u padded with $b' - |u|$ dummy bits.

   We note that $|u| \leq b'$ and thus it can be manipulated in constant time.
3. If $j' \neq j$ then we repeat Step 2 for $j' = j+1$ and obtain $S_{j'}$. Then, $q = S_j[i - jb + 1, b]\ S_{j'}[1, i - jb]$ is the solution.

Lemma 2. *For a given sequence $S[1, n]$ over an alphabet A of size σ, we can access any substring of S of b symbols in $O(1)$ time using the data structures presented in Section 3.1.*

3.3 Space requirement

Let us now consider the storage size of our structures.

– We use the constant-time solution to answer the rank queries [9, 13] over W, totalizing $\frac{2n}{\log_\sigma n}(1 + o(1))$ bits.
– Table C requires at most $\frac{2n}{\log_\sigma n} k \log \sigma$ bits.
– Let us consider table U. $|U| = \sum_{i=0}^{\lfloor n/b \rfloor} |\tilde{P}_i| \leq \sum_{i=0}^{\lfloor n/b \rfloor} |P_i| = nH_k(S) + \sum_{i=0}^{\lfloor n/b \rfloor} f_k(E, S_i)$, which depends on the statistical encoder E used. For example, in the case of Huffman coding, we have $f_k(\text{Huffman}, S_i) < b$, and thus we achieve $nH_k(S) + n$ bits. For the case of Arithmetic coding, we have $f_k(\text{Arithmetic}, S_i) \leq 2$, and thus we have $nH_k(S) + \frac{4n}{\log_\sigma n}$ bits.
– The size of T is $\sigma^k 2^{b'} b \log \sigma = \sigma^k\ n^{1/2}\ \frac{\log n}{2}$ bits.
– Finally, let us consider tables R_g and R_l. Table R_g has $\lceil n/(bc) \rceil$ entries of size $\lceil \log n \rceil$, totalizing $\frac{2n}{\log_\sigma n}$ bits. Table R_l has $\lceil n/b \rceil$ entries of size $\lceil \log(b'c) \rceil$, totalizing $\frac{4n \log \log n}{\log_\sigma n}$ bits.

By considering that any substring of $\Theta(\log_\sigma n)$ symbols can be extracted in constant time by applying $O(1)$ times the procedure of Section 3.2, we have the final theorem.

المنارة للاستشارات
www.manaraa.com

Theorem 1. *Let $S[1,n]$ be a sequence over an alphabet A of size σ. Our data structure uses $nH_k(S) + O(\frac{n}{\log_\sigma n}(k \log \sigma + \log\log n))$ bits of space for any $k < (1-\epsilon)\log_\sigma n$ and any constant $0 < \epsilon < 1$, and it supports access to any substring of S of size $\Theta(\log_\sigma n)$ symbols in $O(1)$ time.*

Note that, in our scheme, the size of T can be neglected only if $k < (\frac{1}{2} - \epsilon)\log_\sigma n$, but this can be pushed as close to 1 as desired by choosing $b = \frac{1}{s}\log_\sigma n$ for constant $s \geq 2$.

Corollary 1. *The previous structure takes space $nH_k(S) + o(n \log \sigma)$ if $k = o(\log_\sigma n)$.*

These results match exactly those of [22], once one corrects their k to $k \log \sigma$ as explained.

Note that we are storing some redundant information that can be eliminated. The last characters of block S_i are stored both within $\tilde{P}_i$ and as C_{i+1}. Instead, we can choose to explicitly store the first k characters of *all* blocks S_i, and encode only the remaining $b-k$ symbols, $S_i[k+1,b]$, either in explicit or compressed form. This improves the space in practice, but in theory it cannot be proved to be better than the scheme we have given.

4 Supporting appends

We can extend our scheme to support appending symbols, maintaining the same space and query complexity, with each appended symbol having constant amortized cost. Assume our current static structure holds n symbols. We use a buffer of $n' = n/\log n$ symbols where we store symbols explicitly. When the buffer is full we use our entropy-bound succinct data structure (EBDS, Section 3) to represent those n' symbols and then we empty the buffer. We repeat this until we have $\log n$ EBDS. At this moment we reenconde all the structures plus our original n symbols, generating a new single EBDS, and restart the process with $2n$ symbols.

Data structures. We describe the additional structures needed to append symbols to the EBDS.

- $BF[1,n']$ is the sequence of at most $n' = n/\log n$ uncompressed symbols.
- AP_i is the i-th EBDS, with $0 \leq i \leq \log n$. N is the number of EBDS we currently have. We call AS_i the sequence AP_i represents. AP_0 is the original EBDS. So $|AS_0| = n$ and $|AS_i| = n/\log n$, $i > 0$.
- $WA[1,2n]$: A bit array marking the initial position of each AP_i and BF in $AS_0AS_1\ldots AS_NBF$, with the additional $o(n)$ bits to answer rank queries over WA in constant time [9, 13].

Substring decoding algorithm. We want to retrieve $q = S[i, i+b-1]$. To achieve this, we take the following steps:

- We calculate $t = rank(WA, i)$ and $t' = rank(WA, i+b-1)$. In this way, we obtain the EBDS where q belongs. The case when part of q belongs to BF is trivially solved.
- If $t = t'$ we obtain q as in Section 3.2. Otherwise, we calculate $t_{off} = i - select(WA, t) + 1$ and decode q_1, the last $n' - i + 1 \leq b$ symbols of AP_t. Then we, calculate $t'_{off} = i + b - select(WA, t')$ and decode q_2, the first t'_{off} symbols of $AP_{t'}$. Finally, we obtain $q = q_1q_2$.

Note that we can extract b symbols in constant time from any AP_i and BF, the first because $\Theta(\log_\sigma(n/\log n)) = \Theta(\log_\sigma n)$ and the second because they are explicit.

المنارة للاستشارات

www.manaraa.com

Construction time Just after we reencode everything we have that $n/2$ symbols have been reencoded one time, $n/4$ symbols 2 times, $n/8$ symbols 3 times and so on. The total number of reencodings is $\sum_{i\geq 1} n\frac{i}{2^i} = 2n$. On the other hand, we are using a semi-static statistical encoder, which takes $O(1)$ time to encode one symbol. Then we conclude each symbols have an amortized cost of appending of $O(1)$.

Space requirement. Let us now consider the storage of the appended structures.

- Table BF requires $n/\log_\sigma n$ bits
- We use the constant-time solution to answer the rank and select queries [13] over WA, totalizing $2n(1+o(1))$ bits.
- Each AP_i is an EBDS, each one using $|AS_i|H_k(AS_i) + O(\frac{|AS_i|}{\log_\sigma |AS_i|}(k\log\sigma + \log\log|AS_i|))$ bits of space.

Lemma 3. *The space requirement of all AP_i, where $0 \leq i \leq \log n$, is $\sum_{i=0}^{\log n} |AP_i| \leq |S\ AS_1 \ldots AS_N|$ $H_k(S\ AS_1 \ldots AS_N) + O(\frac{n}{\log_\sigma n}(k\log\sigma + \log\log n)) + O(\sigma^{k+1}\log^2 n) + O(k\log^2 n)$ bits.*

Proof. Consider summing any two entropies (recall Eqs. (1) and (2)).

$$\begin{aligned}|AS_1|H_k(AS_1) + |AS_2|H_k(AS_2) &= \textstyle\sum_{w\in A^k} |w_{AS_1}|H_0(w_{AS_1}) + \sum_{w\in A^k} |w_{AS_2}|H_0(w_{AS_2}) \\ &\leq \textstyle\sum_{w\in A^k} (\log \binom{|w_{AS_1}|}{|n^{a_1}_{AS_1}|, |n^{a_2}_{AS_1}|, \ldots, |n^{a_\sigma}_{AS_1}|} + \log \binom{|w_{AS_2}|}{|n^{a_1}_{AS_2}|, |n^{a_2}_{AS_2}|, \ldots, |n^{a_\sigma}_{AS_2}|}) + O(\sigma^{k+1}\log n) \\ &\leq \textstyle\sum_{w\in A^k} \log \binom{|w_{AS_1}| + |w_{AS_2}|}{|n^{a_1}_{AS_1}| + |n^{a_1}_{AS_2}|, |n^{a_2}_{AS_1}| + |n^{a_2}_{AS_2}|, \ldots |n^{a_\sigma}_{AS_1}| + |n^{a_\sigma}_{AS_2}|} + O(\sigma^{k+1}\log n) \\ &\leq |AS_1 AS_2|H_k(AS_1 AS_2) + O(\sigma^{k+1}\log n) + O(k\log n)\end{aligned}$$

where $O(\sigma^{k+1}\log n)$ comes from the relationship between the *zero*-th order entropy and the combinatorials, and $O(k\log n)$ comes from considering the symbols in the border between AS_1 ans AS_2. Note that $\sigma^{k+1}\log n = o(n)$ if $k < (1-\epsilon)\log_\sigma n$. Then the lemma follows by adding up the $N \leq \log n$ EBDSs.

Theorem 2. *The structure of Theorem 1 supports appending symbols in constant amortized time and retains the same space and query time complexities, being n the current size of the structure.*

5 Application to full-text indexing

In this section we give some positive and negative results about the application of the technique to full-text indexing, as explained in the Introduction. We have a text $T[1,n]$ over alphabet A and wish to compress a transformed version X of T with our technique. Then, the question is how does $H_k(X)$ relate to $H_k(T)$.

5.1 The Burrows-Wheeler Transform

The Burrows-Wheeler Transform, $S = bwt(T)$, is used by many compressed full-text self-indexes [8, 20, 4, 7, 17, 5, 11]. We have covered it in Section 2.3.

We show that there is a relationship between the k-th order entropy of a text T and the first order entropy of $S = bwt(T)$. For this sake, we will compress S with a first-order compressor, whose output size is an upper bound to $nH_1(S)$.

المنارة للاستشارات
www.manaraa.com

A *run* in S is a maximal substring formed by a single letter. Let $rl(S)$ be the number of runs in S. In [11] they prove that $rl(S) \leq nH_k(T) + \sigma^k$ for any k. Our first-order encoder exploits this property, as follows:

- If $i > 1$ and $s_i = s_{i-1}$ then we output bit 0.
- Otherwise we output bit 1 followed by s_i in plain form ($\log \sigma$ bits).

Thus we encode each symbol of S by considering only its preceding symbol. The total number of bits is $n + rl(S) \log \sigma \leq n(1 + H_k(S) \log \sigma + \frac{\sigma^k \log \sigma}{n})$. The latter term is negligible for $k < (1-\epsilon)\log_\sigma n$, for any $0 < \epsilon < 1$. On the other hand, the total space obtained by our first-order encoder cannot be less than $nH_1(S)$. Thus we get our result:

Lemma 4. *Let $S = bwt(T)$, where $T[1, n]$ is a text over an alphabet of size σ. Then $H_1(S) \leq 1 + H_k(T) \log \sigma + o(1)$ for any $k < (1 - \epsilon) \log_\sigma n$ and any constant $0 < \epsilon < 1$.*

We can improve this upper bound if we use Arithmetic encoding to encode the 0 and 1 bits that distinguish run heads. Their zero-order probability is $p = H_k(T) + \frac{\sigma^k}{n}$, thus the 1 becomes $-p \log p - (1 - p) \log(1 - p) \leq 1$. Likewise, we can encode the run heads s_i up to their zero-order entropy. These improvements, however, do not translate into clean formulas.

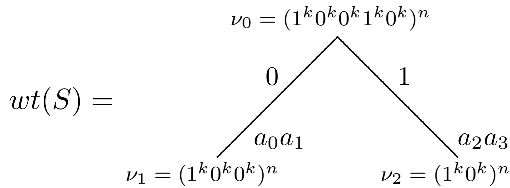
This shows, for example, that we can get (at least) about the same results of the Run-Length FM-Index [11] by compressing $bwt(T)$ using our structure.

5.2 The wavelet tree

Several FM-Index variants [11, 5] use wavelet trees to represent $S = bwt(T)$, while others [7] use them for other purposes. As explained in Section 2.3, $wt(S)$ is composed of several binary sequences. By compressing each such sequence B to $|B|H_0(B)$ bits, one achieves $nH_0(S)$ bits overall. The natural question is, thus, whether we can prove any bound on the overall space if we encode sequences B to $|B|H_k(B)$ bits. We present next two negative examples.

- First we show a case where $H_k(S) < H_k(wt(S))$. We choose $S = (a_3^k a_1^k a_0^k a_2^k a_0^k)^n$, then

$wt(S) =$

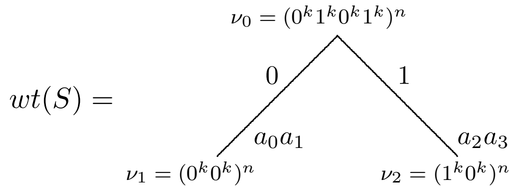


Let us compute $H_k(S)$ according to Section 2.1. Note that $H_0(w_S) = 0$ for all contexts except $w = a_0^k$, where $w_S = a_2(a_3 a_2)^{n-1}\$$, being "\$" a sequence terminator. Thus $|w_S| = \frac{2}{5k}n$ and $H_0(w_S) = -\frac{n}{2n}\log\frac{n}{2n} - \frac{n-1}{2n}\log\frac{n-1}{2n} - \frac{1}{2n}\log\frac{1}{2n} = 1 + O(\frac{\log n}{n})$. Therefore $H_k(S) \simeq \frac{2}{5k}$.
On the other hand, $H_k(wt(S)) = \sum_{i=0}^{2} H_k(\nu_i) \simeq \underbrace{\frac{2}{5k}\log k}_{\nu_0} + \underbrace{\frac{1}{3k} + \frac{\log k}{3k}}_{\nu_1}$, as $H_k(\nu_2) \simeq 0$.

Therefore, in this case, $H_k(S) < H_k(wt(S))$, by an $O(\log k)$ factor.

www.manaraa.com

– Second, we show a case where $H_k(S) > H_k(wt(S))$. Now we choose $S = (a_0^k a_3^k a_0^k a_2^k)^n$, then



In this case, $H_k(S) \simeq \frac{2}{4k}$ and $H_k(wt(S)) = \sum_{i=0}^{2} H_k(\nu_i) = O(\frac{\log n}{n})$. Thus $H_k(S) > H_k(wt(S))$ by a factor of $O(n/(k \log n))$.

Lemma 5. *The ratio between the k-th order entropy of the wavelet tree representation of a sequence S, $H_k(wt(S))$, and that of S itself, $H_k(S)$, can be at least $\Omega(\log k)$. More precisely, $H_k(wt(S))/H_k(S)$ can be $\Omega(\log k)$ and $H_k(S)/H_k(wt(S))$ can be $\Omega(n/(k \log n))$.*

What is most interesting is that $H_k(wt(S))$ can be $\Theta(\log k)$ times larger than $H_k(S)$. We have not been able to produce a larger gap. Whether $H_k(wt(S)) = O(H_k(S) \log k)$ remains open.

6 Conclusions

We have presented a scheme based on k-th order modeling plus statistical encoding to convert any succinct data structure on sequences into a compressed data structure. This structure permits retrieving any string of S of $\Theta(\log_\sigma n)$ symbols in constant time. This is an alternative to the first work achieving the same result [22], which is based on Ziv-Lempel compression. We also show how to append symbols to the original sequence within the same space complexity and with constant amortized cost per appended symbol. This method also works on the structure presented in [22].

We also analyze the behavior of this technique when applied to full-text self-indexes, as advocated in [22]. Many compressed self-indexes achieve space proportional to $nH_k(T)$ by first applying the Burrows-Wheeler Transform [2] over T, $S[1, n] = bwt(T)$. In this paper, we show a relationship between the entropies of $H_1(S)$ and $H_k(T)$. More precisely, $H_1(S) \leq H_k(T) \log \sigma + o(1)$ for small $k = o(\log_\sigma n)$. On the other hand, several indexes represent $S = bwt(T)$ as a wavelet tree [7] on S, $wt(S)$. We show in this paper that $H_k(wt(S))$ can be at least $\Theta(\log k)$ times larger than $H_k(S)$. This means that, by applying the new technique to compress wavelet trees, we have no guarantee of compressing the original sequence more than $n \min(H_0(S), O(H_k(T) \log k))$. Yet, we do have guarantees if we compress S directly.

There are several future challenges on k-th order entropy-bound data structures: (i) making them fully dynamic (we have shown how to append symbols); (ii) better understanding how the entropies evolve upon transformations such bwt or wt; (iii) testing them in practice.

Acknowledgment. We thank K. Sadakane and R. Grossi for providing us article [22].

References

1. T. Bell, J. Cleary, and I. Witten. *Text compression.* Prentice Hall, 1990.

المنارة للاستشارات
www.manaraa.com

2. M. Burrows and D. Wheeler. A block sorting lossless data compression algorithm. Technical Report 124, Digital Equipment Corporation, 1994.
3. D. Clark. *Compact Pat Trees.* PhD thesis, University of Waterloo, Canada, 1996.
4. P. Ferragina and G. Manzini. Opportunistic data structures with applications. In *Proc. 41st IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 390–398, 2000.
5. P. Ferragina, G. Manzini, V. Mäkinen, and G. Navarro. An alphabet-friendly FM-index. In *Proceedings of the 11th International Symposium on String Processing and Information Retrieval (SPIRE 2004)*, LNCS 3246, pages 150–160. Springer, 2004. Extended version to appear in *ACM TALG*.
6. R. Geary, N. Rahman, V. Raman, and R. Raman. A simple optimal representation for balanced parentheses. In *CPM: 15th Symposium on Combinatorial Pattern Matching*, pages 159–172, 2004.
7. R. Grossi, A. Gupta, and J. Vitter. High-order entropy-compressed text indexes. In *Proc. 14th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 841–850, 2003.
8. R. Grossi and J. Vitter. Compressed suffix arrays and suffix trees with applications to text indexing and string matching. In *Proc. 32nd ACM Symposium on Theory of Computing (STOC)*, pages 397–406, 2000.
9. G. Jacobson. Succinct static data structures. Technical Report CMU-CS-89-112, Carnegie Mellon University, 1989.
10. R. Kosaraju and G. Manzini. Compression of low entropy strings with Lempel-Ziv algorithms. *SIAM Journal on Computing*, 29(3):893–911, 1999.
11. V. Mäkinen and G. Navarro. Succinct suffix arrays based on run-length encoding. *Nordic Journal of Computing*, 12(1):40–66, 2005.
12. G. Manzini. An analysis of the Burrows-Wheeler transform. *Journal of the ACM*, 48(3):407–430, 2001.
13. I. Munro. Tables. In *Proc. 16th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, LNCS v. 1180, pages 37–42, 1996.
14. I. Munro, R. Raman, V. Raman, and S. Rao. Succinct representations of permutations. In *ICALP: Annual International Colloquium on Automata, Languages and Programming*, pages 345–356, 2003.
15. I. Munro and V. Raman. Succinct representation of balanced parentheses, static trees and planar graphs. In *Proc. 38th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 118–126, 1997.
16. I. Munro and S. Srinivasa Rao. Succinct representations of functions. In *ICALP: Annual International Colloquium on Automata, Languages and Programming*, pages 1006–1015, 2004.
17. G. Navarro. Indexing text using the ziv-lempel trie. *Journal of Discrete Algorithms (JDA)*, 2(1):87–114, 2004.
18. R. Pagh. Low redundancy in dictionaries with $O(1)$ worst case lookup time. In *Proc. 26th International Colloquium on Automata, Languages and Programming (ICALP)*, pages 595–604, 1999.
19. R. Raman, V. Raman, and S. Rao. Succinct indexable dictionaries with applications to encoding k-ary trees and multisets. In *Proc. 13th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 233–242, 2002.
20. K. Sadakane. Compressed text databases with efficient query algorithms based on the compressed suffix array. In *Proc. 11th International Symposium on Algorithms and Computation (ISAAC)*, LNCS v. 1969, pages 410–421, 2000.
21. K. Sadakane and R. Grossi. Personal communication, 2005.
22. K. Sadakane and R. Grossi. Squeezing succinct data structures into entropy bounds. To appear in Proc. ACM-SIAM Symposium on Discrete Algorithms (SODA), 2006.

المنارة للاستشارات
www.manaraa.com