Information and Computation 207 (2009) 849–866

Contents lists available at ScienceDirect

Information and Computation

journal homepage: www.elsevier.com/locate/ic

The myriad virtues of Wavelet Trees☆

Paolo Ferragina [a,1], Raffaele Giancarlo [b,2], Giovanni Manzini [c,*,3]

[a] Dipartimento di Informatica, Università di Pisa, Italy
[b] Dipartimento di Matematica ed Applicazioni, Università di Palermo, Italy
[c] Dipartimento di Informatica, Università del Piemonte Orientale, Italy

ARTICLE INFO

Article history:
Received 20 March 2008
Revised 17 November 2008
Available online 29 January 2009

ABSTRACT

Wavelet Trees have been introduced by Grossi et al. in SODA 2003 and have been rapidly recognized as a very flexible tool for the design of compressed full-text indexes and data compression algorithms. Although several papers have investigated the properties and usefulness of this data structure in the full-text indexing scenario, its impact on data compression has not been fully explored. In this paper we provide a throughout theoretical analysis of a wide class of compression algorithms based on Wavelet Trees. Also, we propose a novel framework, called *Pruned Wavelet Trees*, that aims for the best combination of Wavelet Trees of properly-designed shapes and compressors either binary (like, Run-Length Encoders) or non-binary (like, Huffman and Arithmetic encoders).

© 2009 Elsevier Inc. All rights reserved.

1. Introduction

The Burrows–Wheeler Transform [5] (bwt for short) 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. [9,18,19]). The transform reduces the problem of high-order entropy compression to the apparently simpler task of designing and engineering good zero-order (or memoryless) compressors. This point has lead to the paradigm of compression boosting presented in [9]. However, despite nearly 60 years of investigation in the design of good memoryless compressors, no general theory for the design of zero-order compressors suited for the bwt is available, since it poses special challenges. Indeed, bwt is a string in which symbols following the same context (substring) are grouped together, giving raise to clusters of nearly identical symbols. A good zero-order compressor must both adapt fast to those rapidly changing contexts and compress efficiently the runs of identical symbols. By now, it is understood that one needs a clever combination of classic zero-order compressors and Run-Length Encoding techniques. However, such a design problem is mostly open. Recently, Grossi et al. [10,11,13] proposed an elegant and effective solution to the posed design problem: the Wavelet Tree. It is a binary tree data structure that reduces the compression of a string over a finite alphabet to the compression of a set of binary strings. The latter problem is then solved via Run-Length Encoding or Gap Encoding techniques. (A formal definition is given in Section 4.1.)

☆ This paper extends results appeared on the Proceedings of the 33th International Colloquium on Automata and Languages (ICALP '06).

* Corresponding author. Fax: +39 0131 360198.
E-mail addresses: ferragina@di.unipi.it (P. Ferragina), raffaele@math.unipa.it (R. Giancarlo), manzini@mfn.unipmn.it (G. Manzini).

[1] Partially supported by Italian PRIN project "MainStream", Italy–Israel FIRB Project "Pattern Discovery Algorithms in Discrete Structures, with Applications to Bioinformatics", and by a Yahoo! Research Grant.

[2] Partially supported by PRIN project "Metodi Combinatori ed Algoritmici per la Scoperta di Patterns in Biosequenze", FIRB project "Bioinformatica per la Genomica e La Proteomica", and Italy–Israel FIRB Project "Pattern Discovery Algorithms in Discrete Structures, with Applications to Bioinformatics".

[3] Partially supported by Italy–Israel FIRB Project "Pattern Discovery Algorithms in Discrete Structures, with Applications to Bioinformatics".

0890-5401/$ - see front matter © 2009 Elsevier Inc. All rights reserved.
doi:10.1016/j.ic.2008.12.010

Wavelet Trees are remarkably natural since they use a well known decomposition of entropy in terms of binary entropy and, in this respect, it is surprising that it took so long to define and put them to good use. The mentioned ground-breaking work by Grossi et al. highlights the beauty and usefulness of this data structure mainly in the context of full-text indexing, and investigates a few of its virtues both theoretically and experimentally in the data compression setting. In this paper we make a step forward by providing throughout theoretical analysis of a wide class of compression algorithms based on Wavelet Trees. A part of our theoretical results either strengthen the ones by Grossi et al. or fully support the experimental evidence presented by those researchers and cleverly used in their engineering choices. The remaining part of our results highlight new virtues of Wavelet Trees. More specifically, in this paper:

(**A)** We provide a theoretic analysis of the two most common techniques used to compress a binary string β, namely Run-Length Encoding (Rle) and Gap Encoding (Ge), in terms of the zero-order entropy $H_0(\beta)$ of the string β (see Lemmas 3.3 and 3.4). These results are the key for our subsequent analysis, and are quite general that may turn to be useful in other settings.

(**B**) We provide a throughout analysis of Wavelet Trees as *stand-alone* general-purpose compressor by considering two specific cases in which either the binary strings associated to the tree nodes are compressed via Rle, and refer to it as Rle Wavelet Tree (Theorem 4.2), or via Ge, and refer to it as Ge Wavelet Tree (Theorem 4.3). Our results generalize the ones given in [14,11] to the case of a *generic prefix-free encoding* of the integers output by Rle or Ge, and hence provide compression bounds which depend on the features of these prefix-free encoders and on the zero-order entropy $H_0(s)$ of the input string s. A notable consequence of these results is Corollary 4.7 that provides the first theoretical analysis of Inversion Frequencies coding [3,4]. Then, we move to study the use of Wavelet Trees for compressing the output of the bwt, and show that Rle Wavelet Trees can achieve a compression bound in terms of $H_k(s)$ (Theorem 5.3), whereas Ge Wavelet Trees *cannot*. This result generalizes [14,11] to the case in which a generic prefix-free encoder is used in combination with Rle or Ge, and also it theoretically supports the algorithmic engineering choices made in [7,10,11], based only on an experimental analysis of the data. Note that our result has been recently strengthened in [17] where the authors show that Rle can be replaced by the succinct dictionaries of [20] (see discussion at the end of Section 5).

(**C**) We combine Wavelet Trees with the compression booster [9] to build a new compression algorithm that compresses any string s in at most $2.2618|s|H_k^*(s) + \log|s| + \Theta\left(|\Sigma|^{k+1}\right)$ bits for any $k \geqslant 0$ (Theorem 6.2 and Corollary 6.3), where $H_k^*(s)$ is the modified k-th order empirical entropy defined in Section 2. This improves the best known bound [9] that achieved a constant 2.5 in front of $|s|H_k^*(s)$. Moreover, we show that we cannot lower that constant down to 2, so that our result is actually close to being optimal. We remark that these kinds of *entropy-only bounds*, namely having the form $\lambda|s|H_k^*(s) + \log|s| + g_k$ with g_k depending only on k and the alphabet size, are much interesting because they guarantee that the compression ratio is proportional to the entropy of the input string s, even if s is highly compressible (see Section 6 for further details).

(**D**) We define *Pruned Wavelet Trees* (see Section 7) that generalize Wavelet Trees and add to this class of data structures in several ways. In order to present our results here, we need to mention some facts about Wavelet Trees, when they are used as stand-alone zero-order compressors. The same considerations apply when they are used upon the BWT. Wavelet Trees reduce the problem of compressing a string to that of compressing a set of binary strings. That set is uniquely identified by: (D.1) the shape (or topology) of the binary tree underlying the Wavelet Tree; (D.2) an assignment of alphabet symbols to the leaves of the tree. How to choose the best Wavelet Tree, in terms of number of bits produced for compression, is open. Grossi et al. establish worst-case bounds that hold for the entire family of Wavelet Trees and therefore they do not depend on (D.1) and (D.2). They also bring some experimental evidence that choosing the "best" Wavelet Tree may be difficult [11, Section 3]. It is possible to exhibit an infinite family of strings over an alphabet Σ for which changing the Wavelet Tree shape (D.1) influences the coding cost by a factor $\Theta(\log|\Sigma|)$, and changing the assignment of symbols to leaves (D.2) influences the coding cost by a factor $\Theta(|\Sigma|)$. So, the choice of the best tree cannot be neglected and remains open. Moreover, (D.3) Wavelet Trees commit to binary compressors, losing the potential advantage that might come from a mixed strategy in which only some strings are binary and the others are defined on an arbitrary alphabet (and compressed via general purpose zero-order compressors, such as Arithmetic and Huffman coding). Again, it is possible to exhibit an infinite family of strings for which a mixed strategy yields a *constant* multiplicative factor improvement over standard Wavelet Trees. So, (D.3) is relevant and open.

In Section 7 we introduce the new paradigm of Pruned Wavelet Trees that allows us to reduce the compression of a string s to the identification of a set of strings, where only a subset may be binary, which are compressed via the mixed strategy sketched above. We develop a combinatorial optimization framework so that one can address points (D.1)–(D.3) simultaneously. Moreover, we provide a *polynomial-time* algorithm for finding the *optimal* mixed strategy for a Pruned Wavelet Tree of *fixed* shape and assignment of alphabet symbols to the leaves of the tree. In addition, we provide a *polynomial-time* algorithm for selecting the *optimal tree-shape* for Pruned Wavelet Trees, when only the assignment of symbols to the leaves of the tree is fixed. Apart from their intrinsic interest, being Wavelet Trees a special case, those two results shed some light on a problem implicitly posed in [14], where it is reported that a closer inspection of the data did not yield any insights as to how to generate a space-optimizing tree, even with the use of heuristics.

2. Empirical entropies

Let s be a string over the alphabet $\Sigma = \{a_1, \ldots, a_h\}$, and for each $a_i \in \Sigma$, let n_i be the number of occurrences of a_i in s.[4] The *zero-order empirical entropy* of s is defined as $H_0(s) = -\sum_{i=1}^{h}(n_i/|s|)\log(n_i/|s|)$. It is well known that H_0 is the maximum compression we can achieve using a fixed codeword for each alphabet symbol. It is also well known that we can often achieve a compression ratio better than $H_0(s)$, if the codeword used for each symbol depends on the symbols preceding it. In this case, the maximum compression is lower bounded by the k-th order entropy $H_k(s)$ defined as follows. Let $\mathcal{S}_k$ denote the set of all length-k substrings of s. For any $w \in \mathcal{S}_k$, let w_s denote the string consisting of the concatenation of the single characters following each occurrence of w inside s. Note that the length of w_s is equal to the number of occurrences of w in s, or to that number minus one if w is a suffix of s. The k-th order empirical entropy of s is defined as

$$H_k(s) = \frac{1}{|s|}\sum_{w\in\mathcal{S}_k}|w_s|H_0(w_s)\,. \tag{1}$$

The value $|s|\,H_k(s)$ represents a lower bound to the compression we can achieve using codewords which depend on the k most recently seen symbols. For any string s and $k \geqslant 0$, it is $H_k(s) \geqslant H_{k+1}(s)$.

As pointed out in [18], for highly compressible strings $H_0(s)$ fails to provide a reasonable lower bound to the performance of compression algorithms. For that reason, [18] introduced the notion of 0-*th order modified empirical* entropy:

$$H_0^*(s) = \begin{cases} 0 & \text{if } |s| = 0 \\ (1 + \lfloor \log|s| \rfloor)/|s| & \text{if } |s| \neq 0 \text{ and } H_0(s) = 0 \\ H_0(s) & \text{otherwise.} \end{cases} \tag{2}$$

For a non-empty string s, $|s|H_0^*(s)$ is at least equal to the number of bits needed to write down the length of s in binary. Starting from H_0^* we define the k-th order modified empirical entropy H_k^* using a formula similar to (1). However, to ensure that $H_{k+1}^*(s) \leqslant H_k^*(s)$ for every string s, H_k^* is defined as the maximum compression ratio we can achieve using for each symbol a codeword which depends on a context of size *at most* k (instead of always using a context of size k as for H_k, see [18] for details). We use the following notation. Let $\mathcal{S}_k$ denote the set of all length-k substrings of s as before. Let $\mathcal{Q}$ be a subset of $\mathcal{S}_1 \cup \cdots \cup \mathcal{S}_k$. We write $\mathcal{Q} \preceq \mathcal{S}_k$ if every string $w \in \mathcal{S}_k$ has a unique suffix in $\mathcal{Q}$. The k-th order modified empirical entropy of s is defined as

$$H_k^*(s) = \min_{\mathcal{Q}\preceq\mathcal{S}_k}\left\{\frac{1}{|s|}\sum_{w\in\mathcal{Q}}|w_s|H_0^*(w_s)\right\}. \tag{3}$$

It is straightforward to verify that for any $k \geqslant 0$ and for any string s it is $H_k^*(s) \geqslant H_{k+1}^*(s)$ and $H_k^*(s) \geqslant H_k(s)$. In Section 6 we will establish bounds in terms of H_k^* and we will show that the same bounds cannot be established in terms of H_k.

We now provide some useful lemmas related to the empirical entropies. Recall that a run is a substring of identical symbols in a string, and a maximal run is a run which cannot be extended, i.e., it is not a proper substring of a longer run.

Lemma 2.1. *[16, Section 3]. The number of maximal runs in a string s is bounded by $1 + |s|H_0(s)$.*

Lemma 2.2. *For any $c \geqslant 1/2$ and for $2 \leqslant r \leqslant n/2$, we have*

$$\log(n/r) \leqslant cr\log(n/r) - cr + 2c.$$

Proof. Let $G(r) = cr\log\frac{n}{r} - cr - \log\frac{n}{r} + 2c$. For $c \geqslant 1/2$, $G(2)$ and $G(n/2)$ are non-negative and $G(r)$ is a concave function in $[2, n/2]$. Hence, $G(r) \geqslant 0$, for $2 \leqslant r \leqslant n/2$, and the thesis follows. □

Lemma 2.3. *Let s be a binary string such that $H_0(s) \neq 0$. Let $n = |s|$ and let r, $1 \leqslant r \leqslant n/2$, denote the number of occurrences of the least frequent symbol in s. We have*

$$|s|H_0(s) \geqslant r\log(n/r) + r.$$

Proof. It is

$$\begin{aligned} |s|H_0(s) &= r\log(n/r) + (n-r)\log(n/(n-r)) \\ &= r\log(n/r) + r\log\left(1 + \frac{r}{n-r}\right)^{\frac{n-r}{r}} \\ &\geqslant r\log(n/r) + r \end{aligned}$$

where the last inequality holds since $(1 + 1/t)^t \geqslant 2$ for $t \geqslant 1$. □

[4] In this paper we write log to denote $\log_2$ and we write ln to denote the natural logarithm. We also assume $0\log 0 = 0$.

Lemma 2.4. *For any string s and $k \geqslant 0$ it is*

$$|s|H_k^*(s) \geqslant \log(|s| - k).$$

Proof. Let $\mathcal{Q} \preceq \mathcal{S}_k$ denote the subset for which the minimum (3) is achieved. It is

$$|s|H_k^*(s) = \sum_{w \in \mathcal{Q}} |w_s| H_0^*(w_s) \geqslant \sum_{w \in \mathcal{Q}} \max(1, \log(|w_s|)) = \sum_{w \in \mathcal{Q}} \log \max(2, |w_s|).$$

Since $\sum_i (\log x_i) \geqslant \log(\sum_i x_i)$ whenever $\min_i x_i \geqslant 2$, we have

$$|s|H_k^*(s) \geqslant \log\left(\sum_{w \in \mathcal{Q}} |w_s|\right) \geqslant \log(|s| - k). \quad \square$$

3. Compression of binary strings

This section provides the technical presentation of the results outlined in point **(A)** of Section 1. Two widely used binary compressors are Rle and Ge. They both represent a binary string via a sequence of positive integers. Each integer in the sequence is then compressed via a prefix-free encoder of the integers. The choice of which one to pick, among the many [6], is the most critical part of the compression process. Therefore, it is very useful to have an analysis of both Rle and Ge that accounts both for H_0 and for the parameters characterizing the compression ability of the chosen encoder of the integers. We provide such an analysis here, together with a rather useful variant of Ge.

Let Pfx denote a uniquely decodable encoder of the positive integers (not necessarily prefix-free). Assume that there exist two positive constants a and b such that, for any positive integer i, we have $|\mathsf{Pfx}(i)| \leqslant a \log i + b$. For instance, γ-coding [6] satisfies the inequality with $a = 2$ and $b = 1$. Notice that the assumption $|\mathsf{Pfx}(i)| \leqslant a \log i + b$ is not restrictive, since the codeword set generated by Pfx is universal [15]. Those universal codeword sets have a nice subadditivity property, when used to encode sequences of integers, as it is the case of interest here:

Lemma 3.1 (*Subadditivity*)**.** *Let a, b be two constants such that $|\mathsf{Pfx}(i)| \leqslant a \log i + b$, for $i > 0$. Then, there exists a constant d_{ab} such that, for any sequence of positive integers $i_1, i_2, \ldots, i_k$, we have*

$$\left|\mathsf{Pfx}\left(\sum_{j=1}^{k} i_j\right)\right| \leqslant \left(\sum_{j=1}^{k} |\mathsf{Pfx}(i_j)|\right) + d_{ab}.$$

Proof. From elementary calculus, we have that $\mathsf{Pfx}(i_1 + i_2) \leqslant \mathsf{Pfx}(i_1) + \mathsf{Pfx}(i_2)$, whenever $\min(i_1, i_2) \geqslant 2$. Hence, we only need to take care of the case in which some of the i_j's are 1. For $i \geqslant 1$, we have:

$$\begin{aligned} |\mathsf{Pfx}(i+1)| - |\mathsf{Pfx}(i)| - |\mathsf{Pfx}(1)| &= a \log(1 + (1/i)) - b && (4)\\ &= (a \log e) \ln(1 + (1/i)) - b \\ &\leqslant (a \log e)/i - b, && (5) \end{aligned}$$

where the last inequality holds since $t \geqslant 0$ implies $\ln(1 + t) \leqslant t$. Let $c_{ab} = (a \log e)/b$. From (4), we get that $i \geqslant 1$ implies $\mathsf{Pfx}(i+1) \leqslant \mathsf{Pfx}(i) + \mathsf{Pfx}(1) + (a - b)$. Moreover, from (5), we get that $i \geqslant c_{ab}$ implies $\mathsf{Pfx}(i+1) \leqslant \mathsf{Pfx}(i) + \mathsf{Pfx}(1)$. Combining these inequalities, we get

$$\left|\mathsf{Pfx}\left(\sum_{j=1}^{k} i_j\right)\right| \leqslant \left(\sum_{j=1}^{k} |\mathsf{Pfx}(i_j)|\right) + c_{ab}(a - b),$$

and the lemma follows with $d_{ab} = c_{ab}(a - b)$. $\square$

In the remainder of this paper, we will use integer encoders as the basic compression procedure in different settings. The only assumption we make is that the integer encoders are of logarithmic cost, i.e., $|\mathsf{Pfx}(i)| \leqslant a \log i + b$. Since the codeword $\mathsf{Pfx}(1)$ must be at least one bit long, we have $b \geqslant 1$ for every code. Note that, since γ-codes have $a = 2$ and $b = 1$, a code with $a > 2$ (and necessarily $b \geqslant 1$) would be worse than γ-codes for any integer and therefore not interesting. Hence, from now on, we assume $a \leqslant 2$ and $b \geqslant 1$.

It is also useful to recall a prefix-free encoding based on the base 3 representation of the integers. For any positive integer n, let $(n)_3\#$ denote the string over the alphabet $\{0, 1, 2, \#\}$ consisting of the base 3 representation of n, followed by

the special symbol #. Note that the first symbol of $(n)_3$# can only be either 1 or 2. Given $(n)_3$#, we build the binary string $\mathsf{Trn}(n)$ by encoding the first symbol with one bit ($1 \to 0$, $2 \to 1$), and the other symbols with two bits ($0 \to 00$, $1 \to 01$, $2 \to 10$, $\# \to 11$). For example, we have $15 = (120)_3$ so $\mathsf{Trn}(15) = 0\,10\,00\,11$. It is straightforward to verify that Trn provides a prefix-free encoding of the integers. Moreover:

Lemma 3.2. *For any positive integer n, we have* $|\mathsf{Trn}(n)| \leqslant (\log_3 4)\log n + 3 = (1.2618\ldots)\log n + 3$.

Proof. Assume $(n)_3$ has length k. Then $n \geqslant 3^{k-1}$ and $k \leqslant \log_3 n + 1$. We have

$$|\mathsf{Trn}(n)| = 2k + 1 \leqslant 2(\log_3(n) + 1) + 1 = (\log_3 4)\log n + 3$$

as claimed. □

3.1. Analysis of Run-Length Encoding

Definition 1. For any binary string $\beta = b_1^{\ell_1} b_2^{\ell_2} \cdots b_k^{\ell_k}$, with $b_i \neq b_{i+1}$, $\mathsf{Rle}(\beta)$ is defined as the concatenation of the binary strings

$$\mathsf{Rle}(\beta) = b_1 \mathsf{Pfx}(\ell_1)\mathsf{Pfx}(\ell_2)\cdots\mathsf{Pfx}(\ell_k).$$

Note that the bit b_1 is needed to establish which symbol appears in each run.

Next lemma bounds the length of $\mathsf{Rle}(\beta)$ in terms of $H_0(\beta)$.

Lemma 3.3. *Let* Pfx *be an integer coder such that* $|\mathsf{Pfx}(i)| \leqslant a\log(i) + b$. *For any binary string* β *such that* $H_0(\beta) \neq 0$, *we have*

$$|\mathsf{Rle}(\beta)| \leqslant \max(2a, b + 2a/3)|\beta|\,H_0(\beta) + 3b + 1.$$

Proof. Let $\beta = b_1^{\ell_1} b_2^{\ell_2} \cdots b_k^{\ell_k}$, with $b_i \neq b_{i+1}$, so that $|\mathsf{Rle}(\beta)| = 1 + \sum_{i=1}^{k} |\mathsf{Pfx}(\ell_i)|$. Moreover, let $n = |\beta|$ and let r denote the number of occurrences of 1 in β. Assume 1 is the least frequent symbol in β, so that $1 \leqslant r \leqslant n/2$. We distinguish three cases, depending on the value of r.

Case $r = 1$**.** We have $|\mathsf{Rle}(\beta)| \leqslant 2a\log n + 3b + 1$. The thesis follows since $|\beta|H_0(\beta) \geqslant \log n$.

Case $2 \leqslant r < n/4$**.** This is the most complex case. We have

$$|\mathsf{Rle}(\beta)| = 1 + \sum_{i=1}^{k}(a\log\ell_i + b) = a\left(\sum_{i=1}^{k}\log\ell_i\right) + bk + 1 \tag{6}$$

Since the runs of 0's and 1's alternate, we have $k \leqslant 2r + 1$. We now show that the number of non-zero logarithms in (6) (that is, the number of logarithms for which $\ell_i > 1$) is at most $r + 1$. To see this, let g denote the number of singletons (runs of length 1) in the r occurrences of the least frequent symbol. Therefore, there are at most $g + \lfloor (r-g)/2 \rfloor$ runs of 1's, and at most $1 + g + \lfloor (r-g)/2 \rfloor$ runs of 0's. Hence the number of runs of length $\ell_i > 1$ are at most

$$\lfloor (r-g)/2 \rfloor + \left(1 + g + \lfloor (r-g)/2 \rfloor\right) \leqslant r + 1$$

as claimed. Using Jensen's inequality and the fact that the function $x\log(n/x)$ is increasing, for $x \leqslant (r+1) \leqslant (n/e)$, we get

$$\begin{aligned}|\mathsf{Rle}(\beta)| &\leqslant a\left(\sum_{\ell_i > 1}\log\ell_i\right) + b(2r+1) + 1\\ &\leqslant a(r+1)\log(n/(r+1)) + b(2r+1) + 1\\ &\leqslant a(r+1)\log(n/r) + 2br + b + 1.\end{aligned} \tag{7}$$

Assume that $b \leqslant a$. Since $(r+1) \leqslant 2r$, we get from (7) that $|\mathsf{Rle}(\beta)| \leqslant 2a(r\log(n/r) + r) + b + 1$, and the thesis follows by Lemma 2.3. Assume now that $b > a$. Applying Lemma 2.2 to (7) with $c = (2b - a)/2a \geqslant 1/2$, we get

$$\begin{aligned}|\mathsf{Rle}(\beta)| &\leqslant a(1+c)r\log(n/r) + (2b - ac)r + 2ac + b + 1\\ &= (b + a/2)r\log(n/r) + (b + a/2)r + 3b - a + 1\\ &\leqslant (b + a/2)(r\log(n/r) + r) + 3b + 1\end{aligned}$$

and the thesis follows again by Lemma 2.3.

Case $n/4 \leqslant r \leqslant n/2$**.** By Jensen's inequality, we have

$$|\mathsf{Rle}(\beta)| \;=\; 1 + a\sum_{i=1}^{k}\log \ell_i \;+\; kb \;\leqslant\; ak\log(n/k) + kb + 1.$$

Since the function $x\log(n/x)$ has its maximum for $x = n/e$ and, by Lemma 2.1, $k \leqslant 1 + |\beta|H_0(\beta)$, we get

$$|\mathsf{Rle}(\beta)| \leqslant a(n/e)\log e + b(1 + |\beta|H_0(\beta)) + 1. \tag{8}$$

Since $r \geqslant n/4$, we have $H_0(\beta) \geqslant -(1/4)\log(1/4) - (3/4)\log(3/4) = (2 - (3/4)\log 3)$. Since

$$\frac{[(\log e)/e]}{(2 - (3/4)\log 3)} = 0.654\ldots < 2/3$$

by (8), we get

$$|\mathsf{Rle}(\beta)| \leqslant (2/3)a|\beta|H_0(\beta) + b|\beta|H_0(\beta) + b + 1.$$

This completes the proof. □

Note that, although Rle encodes unambiguously a single binary string, it does not provide a uniquely decodable code for the set of all binary strings. Indeed, if we are given the concatenation $\mathsf{Rle}(\beta_1)\mathsf{Rle}(\beta_2)$, we are not able to retrieve β_1 and β_2 because the decoder does not know where the encoding of β_1 ends. To retrieve β_1 and β_2, we need some additional information, for example the length of $\mathsf{Rle}(\beta_1)$.

3.2. Analysis of Gap Encoding and of a novel variant

Definition 2. For any binary string β such that $H_0(\beta) \neq 0$, let c_0 denote its least frequent symbol. Define $c_1 = 1$, if c_0 is also the last symbol of β. Otherwise, let $c_1 = 0$. Moreover, let $p_1, p_2, \ldots, p_r$ denote the positions of the occurrences of c_0 in β, and let $g_1, \ldots, g_r$ be defined by $g_1 = p_1, g_i = p_i - p_{i-1}$, for $i = 2, \ldots, r$. If $c_1 = 1$, $\mathsf{Ge}(\beta)$, the Gap Encoding of β, is defined as:

$$\mathsf{Ge}(\beta) = c_0c_1\mathsf{Pfx}(g_1)\mathsf{Pfx}(g_2)\cdots\mathsf{Pfx}(g_r).$$

If $c_1 = 0$, $\mathsf{Ge}(\beta)$ is defined as:

$$\mathsf{Ge}(\beta) = c_0c_1\mathsf{Pfx}(g_1)\mathsf{Pfx}(g_2)\cdots\mathsf{Pfx}(g_r)\mathsf{Pfx}(|\beta| - p_r).$$

Note that the additional term $\mathsf{Pfx}(|\beta| - p_r)$ is needed since, when $c_1 = 0$, we have no information on the length of the last run in β. We now bound $|\mathsf{Ge}(\beta)|$ in terms of $H_0(\beta)$.

Lemma 3.4. *Let* Pfx *be an integer coder such that* $|\mathsf{Pfx}(i)| \leqslant a\log(i) + b$. *For every binary string* β *it is*

$$|\mathsf{Ge}(\beta)| \leqslant \max(a, b)|\beta|\,H_0(\beta) + a\log|\beta| + b + 2.$$

Proof. Let r denote the number of occurrences of the least frequent symbol in β. Using the notation of Definition 2, we have

$$|\mathsf{Ge}(\beta)| \;\leqslant\; \sum_{i=1}^{r}|\mathsf{Pfx}(g_i)| \;+\; a\log|\beta| \;+\; b \;+\; 2.$$

Since $\sum_{i=1}^{r} g_i \leqslant |\beta|$, by the concavity of the logarithm, it is

$$\sum_{i=1}^{r}|\mathsf{Pfx}(g_i)| = a\left[\sum_{i=1}^{r}\log(g_i)\right] + rb \leqslant ar\log(|\beta|/r) + rb \leqslant \max(a, b)(r\log(n/r) + r).$$

The thesis then follows by Lemma 2.3. □

Notice that, similarly to Rle, Ge does not provide a uniquely decodable code for the set of binary strings. However, as it will be self-evident later, it turns out to be convenient to devise and analyze a modified Gap Encoder Ge*, not substantially different from Ge, that provides a prefix-free, and therefore uniquely decodable, code for the set of binary strings. Since we need to perform a rather tight analysis of Ge*, we give its complete pseudo-code in Fig. 3.2. It makes use of an auxiliary procedure SeqCompr, which is described and analyzed in the following lemma.

Lemma 3.5. *Let* $d_1, d_2, \ldots, d_t$ *denote a sequence of positive integers such that* $d_{i-1} < d_i$ *for* $i = 2, \ldots, t$. *Consider the integer gaps* $g_1 = d_1, g_2 = d_2 - d_1, \ldots, g_t = d_t - d_{t-1}$ *and the string*

$$\mathsf{SeqCompr}(d_1, \ldots, d_t) = \mathsf{Pfx}(t)\mathsf{Pfx}(g_1)\mathsf{Pfx}(g_2)\cdots\mathsf{Pfx}(g_t).$$

Procedure Ge*(β)
1. Output the least frequent bit in β.
2. Let $b_1, b_2, \cdots, b_r$ be the positions in β of its least frequent bit.
3. If $r = 1$, output the bit 0, followed by $\mathsf{Pfx}(|\beta|)$, and finally followed by $\lfloor \log |\beta| \rfloor + 1$ bits encoding b_1.
4. If $r > 1$ and $b_r < |\beta|$, output the bits 10, followed by $\mathsf{SeqCompr}(b_1, b_2, \ldots, b_r, |\beta|)$.
5. If $r > 1$ and $b_r = |\beta|$, output the bits 11, followed by $\mathsf{SeqCompr}(b_1, b_2, \ldots, b_r)$.

Fig. 1. Procedure Ge* for the encoding of a binary string β with $H_0(\beta) \neq 0$.

$\mathsf{SeqCompr}(d_1, d_2, \ldots, d_t)$ *is a prefix-free encoding of* $d_1, d_2, \ldots, d_t$, *in the sense that we can detect the end of the encoding without any additional information. Moreover, we also have*:

$$|\mathsf{SeqCompr}(d_1, \ldots, d_t)| \leqslant at \log\left(\frac{d_t}{t}\right) + a \log t + b(t+1).$$

Proof. Notice that

$$|\mathsf{SeqCompr}(d_1, \ldots, d_t)| = |\mathsf{Pfx}(t)| + \sum_{i=1}^{t} |\mathsf{Pfx}(g_i)| \leqslant \sum_{i=1}^{t} (a \log(g_i) + b) + a \log t + b.$$

Since $\sum_{i=1}^{t} g_i = d_t$, using Jensen's inequality, we get

$$|\mathsf{SeqCompr}(d_1, \ldots, d_t)| \leqslant at \log\left(\frac{d_t}{t}\right) + a \log t + b(t+1)$$

and the lemma follows. □

Lemma 3.6. Ge* *is a prefix-free encoder such that, for any binary string* β *with* $H_0(\beta) \neq 0$, *we have*

$$|\mathsf{Ge}^*(\beta)| \leqslant C_{ab} |\beta| H_0(\beta) + \Theta(1)$$

with

$$C_{ab} = \begin{cases} a+1 & \text{if } b < a+2, \\ (a+b)/2 + \epsilon & \text{if } b \geqslant a+2. \end{cases} \tag{9}$$

where $\epsilon > 0$ *is an arbitrarily small positive constant.*

Proof. Clearly Ge* is prefix-free. To prove the bound we use the notation of Fig. 3.2. Assume first $r = 1$. Since $|\beta| H_0(\beta) \geqslant \log |\beta|$ we have

$$|\mathsf{Ge}^*(\beta)| = (a+1) \log |\beta| + \Theta(1) \leqslant (a+1)|\beta| H_0(\beta) + \Theta(1).$$

The thesis follows since $b \geqslant a+2$ implies $a+1 \leqslant (a+b)/2$.

Assume now $r > 1$. We consider only the subcase in which $b_r < |\beta|$, since the other is simpler and left to the reader. By Lemma 3.5, we get

$$|\mathsf{Ge}^*(\beta)| \leqslant a(r+1) \log\left(\frac{|\beta|}{r+1}\right) + a \log(r+1) + br + \Theta(1).$$

From elementary calculus, we know that, for any $\epsilon > 0$, there exists k_ϵ such that $a \log(r+1) \leqslant \epsilon r + k_\epsilon$. Hence

$$|\mathsf{Ge}^*(\beta)| \leqslant ar \log\left(\frac{|\beta|}{r}\right) + a \log\left(\frac{|\beta|}{r}\right) + (b+\epsilon) r + \Theta(1).$$

Using Lemma 2.2 to bound $\log(|\beta|/r)$, we get that, for any $c \geqslant 1/2$, it is

$$|\mathsf{Ge}^*(\beta)| \leqslant (ac+a) r \log\left(\frac{|\beta|}{r}\right) + (b + \epsilon - ac) r + \Theta(1). \tag{10}$$

If $b < 2a$, we take $c = 1/2$ and $\epsilon < 2a - b$. Plugging these values in (10) and using Lemma 2.3, we get

$$|\mathsf{Ge}^*(\beta)| \leqslant (3/2) ar \log\left(\frac{|\beta|}{r}\right) + (3/2) ar + \Theta(1) \leqslant (3/2) a |\beta| H_0(\beta) + \Theta(1). \tag{11}$$

If $b \geqslant 2a$, we choose $c = ((b-a)/2a) \geqslant 1/2$, which, plugged in (10) and using Lemma 2.3 yields

Procedure TreeLabel(u, s)

1. Assign string s to node u. If u has no children return.
2. Let u_L (resp. u_R) denote the left (resp. right) child of u. Let $\Sigma^{(u_L)}$ (resp. $\Sigma^{(u_R)}$) be the set of symbols associated to the leaves of the subtree rooted at u_L (resp. u_R).
3. Assign to node u the binary string obtained from s replacing the symbols in $\Sigma^{(u_L)}$ with 0, and the symbols in $\Sigma^{(u_R)}$ with 1.
4. Let s_L denote the string obtained from s removing the symbols in $\Sigma^{(u_R)}$. If $|s_L| > 0$, TreeLabel(u_L, s_L).
5. Let s_R denote the string obtained from s removing the symbols in $\Sigma^{(u_L)}$. If $|s_R| > 0$, TreeLabel(u_R, s_R).

Fig. 2. Procedure TreeLabel for building the full Wavelet Tree $W_f(s)$ given the alphabetic tree T_Σ and the string s. The procedure is called with $u = root(T)$.

$$|\mathsf{Ge}^\star(\beta)| \leqslant \left(\tfrac{a+b}{2}\right) r \log\left(\frac{|\beta|}{r}\right) + \left(\tfrac{a+b}{2} + \epsilon\right) r + \Theta(1) \leqslant \left(\tfrac{a+b}{2} + \epsilon\right) |\beta| H_0(\beta) + \Theta(1)\,. \tag{12}$$

To complete the proof, it suffices to verify that the bounds (11) and (12) imply that $|\mathsf{Ge}^\star(\beta)| \leqslant C_{ab}|\beta|H_0(\beta) + \Theta(1)$ (when $2a \leqslant b < a+2$ we must take $\epsilon \leqslant a+2-b$; recall also that we can assume $a \leqslant 2$). □

4. Compression of general strings: achieving H_0

This section gives the technical presentation of the results claimed in point (**B**) of Section 1, where Wavelet Trees are used as stand-alone, general purpose, zero-order compressors. In particular, we analyze the performance of Rle compacted Wavelet Trees (Section 4.2) and Ge compacted Wavelet Trees (Section 4.3), showing that Ge is superior to Rle as a zero-order compressor over Wavelet Trees. Nevertheless, we will show in Section 5 that Ge Wavelet Trees, unlike Rle Wavelet Trees, are unable to achieve the k-th order entropy, when used to compress the output of the Burrows–Wheeler Transform. This provides a theoretical ground to the practical choices and experimentation made in [10,11]. Moreover, a remarkable corollary of this section is the first theoretical analysis of Inversion Frequencies coding [3,4].

4.1. Wavelet Trees

Given a string s over the alphabet Σ, we use $\Sigma^{(s)}$ to denote the set of symbols that actually appear in s. For Wavelet Trees we will use a slightly more verbose notation than the one adopted in [13], because we need to distinguish between the base alphabet Σ and the set $\Sigma^{(s)}$.

Let T_Σ be a complete binary tree with $|\Sigma|$ leaves. We associate one-to-one the symbols in Σ to the leaves of T_Σ and refer to it as an *alphabetic tree*. Given a string s over Σ, the *full Wavelet Tree* $W_f(s)$ is the labeled tree returned by the procedure TreeLabel of Fig. 2 (see also Fig. 3). Note that we associate two strings of equal length to each internal node $u \in W_f(s)$. The first one, assigned in Step 1, is a string over Σ that we denote by $s(u)$. The second one, assigned in Step 3, is a binary string that we denote by $s^{\mathbf{01}}(u)$. Note that the length of these strings is equal to the number of occurrences in s of the symbols associated to the leaves of the subtree rooted at u.

If $\Sigma^{(s)} = \Sigma$, the Wavelet Tree $W_f(s)$ has the same shape as T_Σ and it is therefore a complete binary tree. If $\Sigma^{(s)} \subset \Sigma$, $W_f(s)$ is not necessarily a complete binary tree since it may contain unary paths. By contracting all unary paths, we obtain a *compacted Wavelet Tree* $W_c(s)$, which is a complete binary tree with $|\Sigma^{(s)}|$ leaves and $|\Sigma^{(s)}| - 1$ internal nodes (see Fig. 3).

As observed in [13], we can always retrieve s given the binary strings $s^{\mathbf{01}}(u)$ associated to the internal nodes of a Wavelet Tree and the mapping between leaves and alphabetic symbols. Hence, Wavelet Trees are a tool for *encoding arbitrary strings using an encoder for binary strings*. The following fundamental property of compacted Wavelet Trees was established in [13] and shows that, in order to achieve the zero-order entropy on s, it suffices to achieve the zero-order entropy on the binary strings associated to the internal nodes of any Wavelet Tree associated to s.

Theorem 4.1 *[13]. For any string s drawn from the alphabet $\Sigma^{(s)}$, and for any compacted Wavelet Tree $W_c(s)$, we have*

$$|s|H_0(s) = \sum_{u \in W_c(s)} |s^{\mathbf{01}}(u)| H_0(s^{\mathbf{01}}(u)),$$

where the summation is done over all internal nodes of $W_c(s)$.

4.2. Analysis of Rle Wavelet Trees

Let s be a string over the alphabet $\Sigma^{(s)}$. We define the algorithm Rle_wt as follows. First we encode $|s|$ with the codeword Pfx$(|s|)$. Then, we encode the binary strings associated to the internal nodes of the Wavelet Tree $W_c(s)$ using Rle. The internal

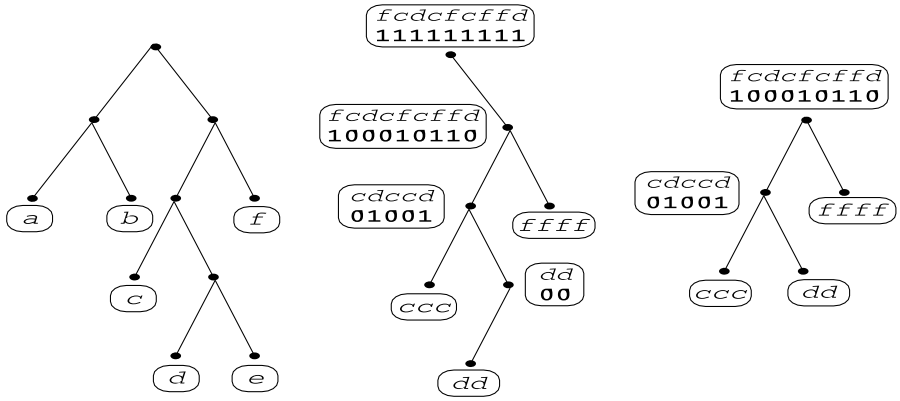

Fig. 3. An alphabetic tree (left) for the alphabet $\Sigma = \{\mathtt{a},\mathtt{b},\mathtt{c},\mathtt{d},\mathtt{e},\mathtt{f}\}$. The corresponding full Wavelet Tree (center) for the string $s = \mathtt{fcdcfcffd}$. The compacted Wavelet Tree (right) for the same string.

nodes are encoded in a predetermined order—for example heap order—such that the encoding of a node u always precedes the encoding of its children (if any).[5] We can always retrieve s from the output of Rle_wt: when we start the decoding of the string $s^{\mathbf{01}}(u)$, we already know its length $|s^{\mathbf{01}}(u)|$ and therefore no additional information is needed to mark the end of the encoding.

Theorem 4.2. *For any string s over the alphabet $\Sigma^{(s)}$, we have*

$$|\mathsf{Rle_wt}(s)| \leqslant \max(2a, b + 2a/3)\ |s|H_0(s) + a\log|s| + \Theta\left(|\Sigma^{(s)}|\right).$$

The bound holds regardless of the shape of the Wavelet Tree.

Proof. The output of Rle_wt consists of $|\mathsf{Pfx}(|s|)| \leqslant a\log|s| + b$ bits, in addition to the cost of encoding the binary string associated to each internal node of the Wavelet Tree, using Rle. That is:

$$|\mathsf{Rle_wt}(s)| = |\mathsf{Pfx}(|s|)| + \sum_{u\in W_c(s)} |\mathsf{Rle}(s^{\mathbf{01}}(u))|, \tag{13}$$

where the summation is done over internal nodes only. If $H_0(s) = 0$, there are no internal nodes and there is nothing to prove. If $H_0(s) \neq 0$, we have that, by Lemma 3.3,

$$|\mathsf{Rle}(s^{\mathbf{01}}(u))| \leqslant \max(2a, b + 2a/3)\ |s^{\mathbf{01}}(u)|H_0(s^{\mathbf{01}}(u)) + 3b + 1,$$

for each internal node u. The thesis then follows by Theorem 4.1. □

4.3. *Analysis of* Ge *Wavelet Trees*

The algorithm Ge_wt is identical to Rle_wt except that the binary strings $s^{\mathbf{01}}(u)$ associated to the internal nodes of $W_c(s)$ are encoded using Ge instead of Rle. The analysis is also very similar and provided in the next theorem.

Theorem 4.3. *For any string s over the alphabet $\Sigma^{(s)}$, we have*

$$|\mathsf{Ge_wt}(s)| \leqslant \max(a, b)\ |s|\ H_0(s)\ +\ |\Sigma^{(s)}|(a\log|s| + b + 2).$$

The bound holds regardless of the shape of the Wavelet Tree.

Proof. The output of Ge_wt(s) consists of $\mathsf{Pfx}(|s|)$, followed by the encoding of the $|\Sigma^{(s)}| - 1$ internal nodes. By Lemma 3.4, we have

$$\begin{aligned}|\mathsf{Ge_wt}(s)| &= \sum_{u\in W_c(s)} |\mathsf{Ge}(s^{\mathbf{01}}(u))|\ +\ a\log|s| + b\\ &\leqslant \max(a, b) \sum_{u\in W_c(s)} |s^{\mathbf{01}}(u)|H_0(s^{\mathbf{01}}(u))\ +\ |\Sigma^{(s)}|(a\log|s| + b + 2).\end{aligned}$$

[5] We are assuming that the Wavelet Tree shape is hard-coded in the (de)compressor.

Procedure Ge*_wt(s)
1. Encode the alphabet $\Sigma^{(s)}$ using $|\Sigma|$ bits.
2. If $|\Sigma^{(s)}| = 1$ output Pfx($|s|$) and exit.
3. Build the compacted wavelet tree $W_c(s)$.
4. Compress the binary strings associated to the internal nodes of $W_c(s)$ using the algorithm Ge*.

Fig. 4. Procedure Ge*_wt for compressing a string s over the alphabet $\Sigma^{(s)} \subseteq \Sigma$. We assume that the shape of the alphabetic tree T_Σ used for building $W_c(s)$ and the order in which internal nodes are compressed by Ge* are hard-coded in the (de)compressor.

and the thesis follows by Theorem 4.1. □

Comparing the bounds in Theorems 4.2 and 4.3, we see that the latter has a smaller constant in front of $H_0(s)$ since $\max(a, b) \leqslant \max(2a, b + 2a/3)$. Indeed, if $a \geqslant b$, as in γ-codes, we have $\max(a, b) \leqslant (1/2)\max(2a, b + 2a/3)$.

So far we have bounded the compression of Rle and Ge Wavelet Trees in terms of: (1) the entropy of the input string, and (2) the logarithm of the input string length. For most strings s, we have $\log|s| = o(|s|H_0(s))$, so $\Theta(\log|s|)$ can be regarded as a lower order term. However, if s is highly compressible, for example $s = a_1 a_2^n$, we have $|s|H_0(s) = \Theta(\log|s|)$ and a $\Theta(\log|s|)$ term cannot be considered lower order. It turns out that because of this additional $\Theta(\log|s|)$ term Rle and Ge Wavelet Trees are not the best algorithms for compressing low entropy (i.e., highly compressible) strings. Here we provide a variant of Ge Wavelet Trees that is well suited for highly compressible strings and that will be fundamental to obtain the higher order entropy-only bounds reported in Section 6.

Consider the binary compressor Ge* introduced in Section 3 and recall that it is a prefix-free encoder: the decompressor knows when the decoding of a string is complete without the need of additional information. Hence, we can compress an arbitrary string s by first encoding the alphabet $\Sigma^{(s)}$ and then building a compacted Wavelet Tree for s compressing internal nodes with Ge* (see Fig. 4). The output consists of $|\Sigma|$ bits followed by the concatenation of the compressed internal nodes.[6] The resulting algorithm, called Ge*_wt, still has the property of being prefix-free. The next lemma bounds Ge*_wt's compression in terms of the modified empirical entropy. The use of the modified empirical entropy makes it possible to prove a bound which is significant also for the highly compressible strings for which $|s|H_0^*(s) = O(\log|s|)$.

Theorem 4.4. *For any string s, we have*

$$|\mathsf{Ge^*_wt}(s)| \leqslant C_{ab}\, |s|H_0^*(s) + \Theta(|\Sigma|)\,, \tag{14}$$

where C_{ab} is the constant defined by (9).

Proof. If $|\Sigma^{(s)}| = 1$ we have $|\mathsf{Ge^*_wt}(s)| = a\log|s| + \Theta(|\Sigma|)$. The thesis follows since, by (2), we have $|s|H_0^*(s) = 1 + \lfloor \log|s| \rfloor$. If $|\Sigma^{(s)}| > 1$, by Lemma 3.6 and Theorem 4.1, we have

$$\begin{aligned}
|\mathsf{Ge^*_wt}(s)| &\leqslant \sum_{u \in W_c(s)} |\mathsf{Ge^*}(s^{\mathbf{01}}(u))| \;+\; |\Sigma| \\
&\leqslant \sum_{u \in W_c(s)} C_{ab}\, |s^{\mathbf{01}}(u)| H_0(s^{\mathbf{01}}(u)) \;+\; \Theta(|\Sigma|) \\
&\leqslant C_{ab}\, |s|H_0(s) \;+\; \Theta(|\Sigma|)\,,
\end{aligned}$$

and the thesis follows since $H_0(s) \leqslant H_0^*(s)$. □

From the above theorem and Lemma 3.2 it follows that combining Ge*_wt with the prefix encoder Trn (defined at the beginning of Section 3) the bound (14) holds with $C_{ab} = 1 + \log_3 4 \approx 2.2618$. The following theorem shows that no prefix-free algorithm can achieve a substantially better bound, in the sense that the constant in front of $H_0^*(s)$ must be greater than 2.

Theorem 4.5 *[12, Section 7]. If a compression algorithm* A *is prefix-free, then the bound*

$$|\mathsf{A}(s)| \leqslant \lambda |s| H_0^*(s) + f(|\Sigma|) \qquad \textit{for every string } s \tag{15}$$

with $f(|\Sigma|)$ independent of $|s|$, can only hold with a constant $\lambda > 2$.

[6] The algorithms Rle and Ge are not prefix-free and for this reason both Rle_wt and Ge_wt need to encode also the length of the input string.

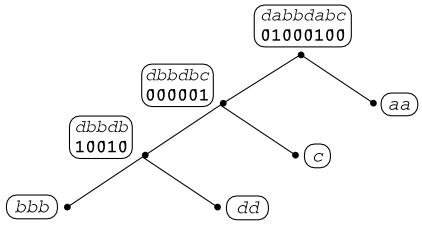

Fig. 5. The skewed Wavelet Tree for the string $s = \texttt{dabbdabc}$. Symbol b is the most frequent one and is therefore associated to the leftmost leaf.

4.4. Skewed Wavelet Trees and Inversion Frequencies coding

All results in the previous sections hold regardless of the shape of the Wavelet Tree. In this section we consider a variant of Ge_wt, in which we choose the Wavelet Tree shape and the correspondence between leaves and symbols. Although this approach leads to (minor) improvements in the compression bounds given in Theorem 4.3, it establishes an important and unexpected connection between Wavelet Tree and Inversion Frequencies coding [3,4]. To our knowledge, the results presented here provide the first bound on the output size of IF coding in terms of the entropy of the input string. They also provide a theoretical justification for the strategy, suggested in [1], of processing the symbols in order of increasing frequency.

Given a string s over the alphabet $\Sigma^{(s)}$, consider the algorithm Ge_skwt defined as follows. First, encode the length of s and the number of occurrences of each symbol using a total of $|\mathsf{Pfx}(|s|)| + |\Sigma^{(s)}|\left(\lfloor \log |s| \rfloor + 1\right)$ bits. Then, build a Wavelet Tree completely skewed to the left such that the most frequent symbol is associated to the leftmost leaf. The other symbols are associated to the leaves in reverse alphabetic order (see Fig. 5). Letting u_i denote the internal node of depth $i-1$, we finally use Ge to encode the strings $s^{\mathbf{01}}(u_1), \ldots, s^{\mathbf{01}}(u_{|\Sigma^{(s)}|-1})$, in that order. The crucial observation is that when we encode (and later decode) the string $s^{\mathbf{01}}(u_i)$, we already know:

1. its length $|s^{\mathbf{01}}(u_i)|$, which is equal to the number of occurrences of the symbols associated to the subtree rooted at u_i;
2. the number of 1's in $s^{\mathbf{01}}(u_i)$, which is equal to the number of occurrences of the symbol associated to the right child of u_i;
3. that 1 is the least frequent symbol in $s^{\mathbf{01}}(u_i)$, since the most frequent symbol in s is associated to the leftmost leaf.

We can take advantage of this extra information to encode $s^{\mathbf{01}}(u_i)$ with a "simplified" version of Ge. Indeed, with reference to Definition 2, we omit the two initial bits c_0 and c_1, and the last codeword $\mathsf{Pfx}(|s^{\mathbf{01}}(u_i)| - p_r|)$. Thus, the "simplified" Ge encoding of $s^{\mathbf{01}}(u_i)$ consists of the concatenation $\mathsf{Pfx}(g_1), \cdots, \mathsf{Pfx}(g_r)$. An analysis analogous to the one in the proof of Lemma 3.4 shows that the length of this encoding is bounded by $\max(a,b)|s^{\mathbf{01}}(u_i)|H_0(s^{\mathbf{01}}(u_i))$. Summing up, we have the following result.

Theorem 4.6. *For any string s over the alphabet $\Sigma^{(s)}$, using the skewed Wavelet Tree and the "simplified" Gap Encoding detailed above, we have*

$$|\mathsf{Ge_skwt}(s)| \leqslant \max(a,b)\,|s|\,H_0(s) \;+\; (a + |\Sigma^{(s)}|)\log|s| + |\Sigma^{(s)}| + b.$$

Proof. From the above discussion, we get

$$|\mathsf{Ge_skwt}(s)| \leqslant \max(a,b) \sum_{u \in W_c(s)} |s^{\mathbf{01}}(u)| H_0(s^{\mathbf{01}}(u)) \;+\; (a\log|s| + b) + |\Sigma^{(s)}|(\lfloor \log|s| \rfloor + 1).$$

The thesis follows by Theorem 4.1. □

Comparing the bounds in Theorems 4.3 and 4.6, we see that the latter has a smaller non-entropy term due to the use of the "simplified" Gap Encoder. Further study is needed to understand whether this advantage is outweighted (or improved) by the use of the skewed tree shape.

Assume now that $\Sigma^{(s)} = \Sigma = \{a_1, a_2, \ldots, a_h\}$ and that a_h is the most frequent symbol in s. Consider now the Ge_skwt algorithm applied to s (that is, consider a skewed Wavelet Tree whose leaves are labeled $a_h, a_{h-1}, \ldots, a_1$ from left to right). In this setting the Gap Encoding of $s^{\mathbf{01}}(u_i)$ coincides with the encoding of the positions of the symbol a_i in the string s, with the symbols $a_1, \ldots, a_{i-1}$ removed. In other words, we are encoding the number of occurrences of $a_{i+1}, \ldots, a_h$ between two consecutive occurrences of a_i. This strategy is known as *Inversion Frequencies* (IF) coding and was first suggested in [4] as an alternative to Move-to-Front encoding. Since also in IF coding we initially encode the number of occurrences of each symbol, from our analysis we get the following result.

Corollary 4.7. *Assuming that the most frequent symbol is processed last, IF coding produces a sequence of integers that we can compress within the same bound given in Theorem 4.6.*

Note that in its the original formulation IF processes symbols in alphabetic order. However, [1] showed that processing symbols in order of increasing frequency usually yields better compression. Corollary 4.7 provides theoretical support to this heuristic strategy.

5. Compression of general strings: achieving H_k

This section provides the technical details about the results claimed in point (**B**) of Section 1 concerning the combination of Rle Wavelet Trees and bwt. We show that this approach can achieve higher order entropy compression, whereas Ge Wavelet Trees cannot.

We need to recall a key property of the Burrows–Wheeler Transform of a string z [18]: If $s = \mathsf{bwt}(z)$ then, for any $k \geqslant 0$, there exists a partition $s = s_1 s_2, \cdots, s_t$ such that $t \leqslant |\Sigma|^k + k$ and $|z|\, H_k(z) = \sum_{i=1}^{t} |s_i| H_0(s_i)$. In other words, the bwt is a tool for achieving the k-th order entropy H_k, provided that we can achieve the entropy H_0 on each s_i. An analogous result holds for H_k^* as well.[7] In view of the above property, we establish our main result by showing that compressing the whole s via one Rle Wavelet Tree is not much worse than compressing each string s_i separately. In order to prove such a result, some care is needed. We can assume without loss of generality that $\Sigma^{(s)} = \Sigma$. However, $\Sigma^{(s_i)}$ will not, in general, be equal to $\Sigma^{(s)}$ and this creates some technical difficulties and forces us to consider both full and compacted Wavelet Trees. Indeed, if we "slice" the compacted Wavelet Tree $W_c(s)$ according to the partition $s = s_1, \cdots, s_t$, we get *full* Wavelet Trees for the strings s_i's.

Recall from Section 4.2 that the total cost of encoding a Wavelet Tree with Rle is given by (13). As a shorthand, we define

$$\|W_c(s)\|_{rle} = \sum_{u \in W_c(s)} |\mathsf{Rle}(s^{\mathbf{01}}(u))| \quad \text{and} \quad \|W_f(s)\|_{rle} = \sum_{u \in W_f(s)} |\mathsf{Rle}(s^{\mathbf{01}}(u))| \tag{16}$$

where both summations are done over internal nodes only. Our first lemma essentially states that, for full Rle Wavelet Trees, partitioning a string does not improve compression.

Lemma 5.1. *Let $s = s_1 s_2$ be a string over the alphabet Σ. We have*

$$\|W_f(s)\|_{rle} \leqslant \|W_f(s_1)\|_{rle} + \|W_f(s_2)\|_{rle}.$$

Proof. Let u be an internal node of $W_f(s)$ and let $\Sigma^{(u)}$ denote the set of symbols associated to u. That is, the set of symbols associated to leaves in the subtree rooted at u. If the string s_i, for $i = 1, 2$, contains at least one of the symbols in $\Sigma^{(u)}$, then $W_f(s_i)$ contains an internal node u_i that has $\Sigma^{(u)}$ as the associated set of symbols. Assume first that the symbols of $\Sigma^{(u)}$ appear only in s_1. In this case, we have that $s^{\mathbf{01}}(u)$ coincides with $s_1^{\mathbf{01}}(u_1)$, which is the binary string associated to u_1 in $W_f(s_1)$. Hence, $|\mathsf{Rle}(s^{\mathbf{01}}(u))| = |\mathsf{Rle}(s_1^{\mathbf{01}}(u_1))|$. Similarly, if the symbols of $\Sigma^{(u)}$ appear only in s_2, we have $|\mathsf{Rle}(s^{\mathbf{01}}(u))| = |\mathsf{Rle}(s_2^{\mathbf{01}}(u_2))|$.

Assume now that u_1 and u_2 both exist. In this case $s^{\mathbf{01}}(u)$ is the concatenation of $s_1^{\mathbf{01}}(u_1)$ and $s_2^{\mathbf{01}}(u_2)$. Hence, if $s^{\mathbf{01}}(u) = b_1^{\ell_1} b_2^{\ell_2}, \cdots, b_k^{\ell_k}$, there exist an index j, $1 \leqslant j \leqslant k$, and a value δ, $0 \leqslant \delta < \ell_j$, such that

$$s_1^{\mathbf{01}}(u_1) = b_1^{\ell_1} \cdots b_{j-1}^{\ell_{j-1}} b_j^{\delta}, \qquad \text{and} \qquad s_2^{\mathbf{01}}(u_2) = b_j^{\ell_j - \delta} b_{j+1}^{\ell_{j+1}} \cdots b_k^{\ell_k}.$$

Summing up, we have

$$|\mathsf{Rle}(s^{\mathbf{01}}(u))| = 1 + \sum_{i=1}^{k} |\mathsf{Pfx}(\ell_i)|$$

$$|\mathsf{Rle}(s_1^{\mathbf{01}}(u_1))| = 1 + \sum_{i=1}^{j-1} |\mathsf{Pfx}(\ell_i)| + |\mathsf{Pfx}(\delta)|$$

$$|\mathsf{Rle}(s_2^{\mathbf{01}}(u_2))| = 1 + |\mathsf{Pfx}(\ell_j - \delta)| + \sum_{i=j+1}^{k} |\mathsf{Pfx}(\ell_i)|.$$

Hence

$$|\mathsf{Rle}(s^{\mathbf{01}}(u))| - |\mathsf{Rle}(s_1^{\mathbf{01}}(u_1))| - |\mathsf{Rle}(s_2^{\mathbf{01}}(u_2))| = |\mathsf{Pfx}(\ell_j)| - |\mathsf{Pfx}(\delta)| - |\mathsf{Pfx}(\ell_j - \delta)| - 1.$$

Refining the proof of Lemma 3.1 it is easy to see that $|\mathsf{Pfx}(i_1 + i_2)| \leqslant |\mathsf{Pfx}(i_1)| + |\mathsf{Pfx}(i_2)| + (a - b)$ for any pair i_1, i_2. Hence, $|\mathsf{Pfx}(\ell_j)| \leqslant |\mathsf{Pfx}(\delta)| + |\mathsf{Pfx}(\ell_j - \delta)| + (a - b)$. Since $a \leqslant 2$ and $b \geqslant 1$, we get

$$|\mathsf{Rle}(s^{\mathbf{01}}(u))| \leqslant |\mathsf{Rle}(s_1^{\mathbf{01}}(u_1))| + |\mathsf{Rle}(s_2^{\mathbf{01}}(u_2))|.$$

The thesis follows summing over all internal nodes of $W_f(s)$. □

[7] When the bwt is used, $\log |z|$ additional bits must be included in the output file: these are required to retrieve z given s. In this section we ignore this additional cost since it would be a lower order term in the bounds of this section. The additional $\log |z|$ bits will be instead accounted for in Section 6.

Theorem 4.1 bounds the cost of *compacted* Wavelet Trees in terms of the entropy of the input string. In order to use Lemma 5.1, we need a similar result for the *full* Wavelet Tree $W_f(s)$.

Lemma 5.2. *For any non-empty string s over the alphabet* Σ, *it holds*

$$\|W_f(s)\|_{rle} \leqslant \max(2a, b + 2a/3)\ |s|H_0(s) + (|\Sigma| - 1)(a \log |s|) + \Theta(|\Sigma|)\,.$$

Proof. Assume first $H_0(s) \neq 0$. Since $W_c(s)$ is obtained from $W_f(s)$ contracting all unary paths, we have

$$\|W_f(s)\|_{rle} = \|W_c(s)\|_{rle} + \text{cost of deleted nodes.}$$

Comparing (13) and (16) we see that $\|W_c(s)\|_{rle} = |\mathsf{Rle_wt}(s)| - \mathsf{Pfx}(|s|)$. Hence, by Theorem 4.2

$$\|W_c(s)\|_{rle} \leqslant \max(2a, b + 2a/3)\ |s|H_0(s) + \Theta\Big(|\Sigma^{(s)}|\Big)\,.$$

Recall now that a node $u \in W_f(s)$ is deleted if and only if $s^{\mathbf{01}}(u)$ contains only 0's or only 1's. Hence, each deleted node u contributes to the difference $\|W_f(s)\|_{rle} - \|W_c(s)\|_{rle}$ by an amount $1 + \mathsf{Pfx}(|s^{\mathbf{01}}(u)|) \leqslant a \log |s| + b + 1$. The thesis then follows since the number of deleted nodes is at most $|\Sigma| - 1$.

Assume now $H_0(s) = 0$, i.e., $s = \sigma^n$, for some $\sigma \in \Sigma$. In this case, $W_f(s)$ contains at most $|\Sigma| - 1$ internal nodes. Their Rle cost is bounded by $a \log |s| + b + 1$, and the thesis follows. □

We are now able to bound the size of a Rle Wavelet Tree over the string $s = \mathsf{bwt}(z)$ in terms of the k-th order entropy of z.

Theorem 5.3. *Let z denote a string over the alphabet* $\Sigma = \Sigma^{(z)}$, *and let* $s = \mathsf{bwt}(z)$. *For any* $k \geqslant 0$, *we have*

$$|\mathsf{Rle_wt}(s)| \leqslant \max(2a, b + 2a/3)\ |z|\,H_k(z) + |\Sigma|^{k+1}(a \log |z|) + O\Big(|\Sigma|^{k+1}\Big)\,. \tag{17}$$

In addition, if $|\Sigma| = O(\text{polylog}(|z|))$, *for all* $k \leqslant \alpha \log_{|\Sigma|} |z|$, *constant* $0 < \alpha < 1$, *we have*

$$|\mathsf{Rle_wt}(s)| \leqslant \max(2a, b + 2a/3)\ |z|\,H_k(z) + o(|z|). \tag{18}$$

Proof. Let $C_{ab} = \max(2a, b + 2a/3)$, and let $s = s_1 \cdots s_t$ denote the partition of s such that $|z|\,H_k(z) = \sum_{i=1}^{t} |s_i|H_0(s_i)$. Since $\Sigma = \Sigma^{(z)} = \Sigma^{(s)}$, $W_f(s)$ coincides with $W_c(s)$. By (13), we have

$$|\mathsf{Rle_wt}(s)| = \|W_c(s)\|_{rle} + |\mathsf{Pfx}(|s|)| = \|W_f(s)\|_{rle} + |\mathsf{Pfx}(|s|)|. \tag{19}$$

By Lemmas 5.1 and 5.2, we get

$$\begin{aligned}
\|W_f(s)\|_{rle} &\leqslant \sum_{i=1}^{t} \|W_f(s_i)\|_{rle} \\
&\leqslant \sum_{i=1}^{t} \Big(C_{ab}\ |s_i|H_0(s_i) + (|\Sigma| - 1)(a \log |s_i|) + \Theta(|\Sigma|) \Big) \\
&\leqslant C_{ab}\ |z|\,H_k(z)\ +\ (|\Sigma| - 1) \sum_{i=1}^{t} a \log |s_i|\ +\ \Theta(t|\Sigma|)\,.
\end{aligned} \tag{20}$$

Since $\sum_{i=1}^{t} \log |s_i| \leqslant t \log(|s|/t)$, from (19) and (20) we get

$$|\mathsf{Rle_wt}(s)| \leqslant C_{ab}\ |z|\,H_k(z) + t(|\Sigma| - 1)a \log(|s|/t) + a \log(|s|) + \Theta(t|\Sigma|)\,. \tag{21}$$

Since $t \leqslant |\Sigma|^k + k$, we have

$$|\mathsf{Rle_wt}(s)| \leqslant C_{ab}\ |z|\,H_k(z) + |\Sigma|^{k+1} a \log(|s|) + \Theta\Big(|\Sigma|^{k+1}\Big)$$

which implies (17), since $|s| = |z|$. To prove (18), we start from (21) and note that Σ's size and the inequality $t \leqslant |\Sigma|^k + k$ imply $t|\Sigma| \log(|s|/t) = o(|s|) = o(|z|)$. □

Theorem 5.3 shows that Rle Wavelet Trees achieve the k-th order entropy with the same multiplicative constant $\max(2a, b + 2a/3)$ that Rle achieves with respect to H_0 (Lemma 3.3). Thus, Wavelet Trees are a sort of *booster* for Rle (cf. [9]). After the appearance of the conference version of the present paper [8], Mäkinen and Navarro [17] proved that Rle is not the only compressor that allows to achieve H_k using Wavelet Trees. More precisely, [17] considers the Wavelet Tree over the entire string $\mathsf{bwt}(z)$ in which the binary strings associated to the internal nodes of the Wavelet Tree are represented using the succinct dictionaries of [20]. Such Wavelet Tree takes at most

$$|z|\,H_k(z) + O\Big(|\Sigma|^{k+1}(\log |z|)\Big) + O\big((|z| \log |\Sigma| \log \log |z|)/\log |z|\big) \tag{22}$$

bits. Succinct dictionaries support $O(\log|\Sigma|)$ time rank and select queries [20], so the data structure of [17] is not simply a compressed file but a more powerful compressed full-text index [19]. Comparing (22) with (17), and recalling that $a \geqslant 1$, we see that (22) has a smaller multiplicative constant in front of $|z|\,H_k(z)$ but also an additional $O\big((|z|\log|\Sigma|\log\log|z|)/\log|z|\big)$ term which can become dominant for strings with small entropy.

In proving Theorem 5.3 we used some rather coarse upper bounds. So we believe it is possible to improve the bounds (17) and (18). However, there are some limits to the possible improvements. As the following example shows, the $o(|z|)$ term in (18) cannot be reduced to $\Theta(1)$ even for constant size alphabets. This is true not only for H_k, but also for the modified empirical entropy H_k^*.

Example 1. Let $\Sigma = \{1, 2, \ldots, m\}$, and let $z = (123\cdots m)^n$. We have $H_1(z) = 0$, $|z|H_1^*(z) \approx m\log n$, and $s = \mathsf{bwt}(z) = m^n 1^n 2^n \cdots (m-1)^n$. Consider a balanced Wavelet Tree of height $\lceil\log m\rceil$. It is easy to see that there exists an alphabet ordering such that the internal nodes of the Wavelet Tree all consist of alternate sequences of 0^n and 1^n. Even encoding these sequences with $\log n$ bits each would yield a total cost of about $(m\log m)\log n \approx (\log m)|z|H_1^*(z)$ bits.

It is natural to ask whether we can repeat the above analysis and prove a bound for Ge Wavelet Trees in terms of the k-th order entropy. Unfortunately, the answer is no! The problem is that, when we encode s with Ge, we have to make some global choices—e.g., the shape of the tree in Ge_wt, the role of zeros or ones in each internal node in the algorithm of [14]—and these are not necessarily good choices for every substring s_i. Hence, we can still split $W_f(s)$ into $W_f(s_1), \ldots, W_f(s_t)$, but it is not always true that $W_f(s_i) \leqslant \lambda|s_i|\,H_0(s_i) + o(|s_i|)$. As a more concrete example, consider the string $z = (01)^n$. We have $H_1(z) = 0$, $|z|H_1^*(z) = \Theta(\log n)$, and $s = \mathsf{bwt}(z) = 1^n 0^n$. $W_c(s)$ has only one internal node—the root—with associated string $1^n 0^n$. We can either encode the gaps between 1's or the gaps between 0's. In both cases, the output will be of $\Theta(n)$ bits, thus exponentially larger than $|z|H_1^*(z)$. Of course this example does not rule out the possibility that a modified version of Ge could achieve the k-th order entropy. Indeed, in view of the remark following the proof of Theorem 4.3, a modified Ge is a natural candidate for improving the bounds of Theorem 5.3.

6. Compression of general strings: achieving entropy-only bounds

This section contains the technical details of the results claimed in (**C**) of Section 1. In particular, we show how to use Wavelet Trees to achieve the best entropy-only bound known in the literature. Entropy-only bounds have the form $\lambda|s|H_k^*(s) + \log|s| + g_k$, where H_k^* is the modified k-th order empirical entropy (defined in Section 2), and g_k depends only on k and on the alphabet size. Achieving an entropy-only bound guarantees that, even for highly compressible strings, the compression ratio will be proportional to the entropy of the input string. Too see this, recall that $|s|H_k^*(s) \geqslant \log(|s| - k)$ (Lemma 2.4). Hence, if a compressor A achieves an entropy-only bound, it is $|\mathsf{A}(s)| \leqslant (1+\lambda)|s|H_k^*(s) + g_k'$ for any string s. The reason for which the term $\log|s|$ appears explicitly in entropy-only bounds is that bwt-based algorithms need to include in the output file $\log|s|$ additional bits in order to retrieve s given $\mathsf{bwt}(s)$. Keeping the term $\log|s|$ explicit provides a better picture of the performance of bwt-based compressors when $|s|H_k^*(s) = \omega(\log|s|)$. Entropy-only bounds cannot be established with the entropy H_k. To see this, consider the family of strings $s = a_1^n$. We have $|s|\,H_k(s) = 0$ for all of them and clearly we cannot hope to compress all strings in this family in $\Theta(1)$ space.

For convenience of the reader, we recall the main technical result from [9, Section 5].

Property 1. *Let* A *be a prefix-free compressor that encodes any input string $x \in \Sigma^*$ within the following space and time bounds*[8]*:*

1. $|\mathsf{A}(x)| \leqslant \lambda|x|H_0^*(x) + \mu$ bits, where λ and μ are constants,
2. the running time of A is $T(|x|)$ and its working space is $S(|x|)$, where $T(\cdot)$ is a convex function and $S(\cdot)$ is non-decreasing.

Theorem 6.1 *[9, Th. 5.2]. Given a compression algorithm* A *that satisfies Property* 1*, one can apply the compression booster in [9] so that it compresses s within $\lambda|s|H_k^*(s) + \log|s| + g_k$ bits, for any $k \geqslant 0$. The above compression takes $O(T(|s|))$ time and $O(S(|s|) + |s|\log|s|)$ bits of space.*

We can combine Theorem 6.1 with Theorem 4.4 to obtain:

Theorem 6.2. *Combining the algorithm* Ge*_wt *with the compression booster, we can compress any string s over the alphabet Σ in at most $C_{ab}|s|H_k^*(s) + \log|s| + \Theta\Big(|\Sigma|^{k+1}\Big)$ bits for any $k \geqslant 0$. The constant C_{ab} multiplying H_k^* depends, according to* (9)*, on*

[8] In [9] instead of assuming A to be prefix-free, the authors made the equivalent assumption that A appends a unique end-of-string symbol at the end of the input string x.

the parameters a and b of the integer coder Pfx *used by* Ge*_wt. *The compression process takes linear time and* $O(|s| \log |s|)$ *bits of space*.

From Theorem 6.2 and Lemma 3.2 we immediately get the best known entropy-only bound:

Corollary 6.3. *If we use the integer coder* Trn *within the algorithm* Ge*_wt, *combined with the compression booster, we can compress any string s over the alphabet* Σ *in at most*

$$(1 + \log_3 4)|s|H_k^*(s) + \log |s| + \Theta\left(|\Sigma|^{k+1}\right) = (2.2618\ldots)|s|H_k^*(s) + \log |s| + \Theta\left(|\Sigma|^{k+1}\right)$$

bits. The bound holds simultaneously for any $k \geqslant 0$.

Note that Theorem 4.5 implies that Property 1 can only hold with $\lambda > 2$. Hence, using the compression boosting theorem (Theorem 6.1) we can only prove bounds of the form $\lambda|s|H_k^*(s) + \log |s| + g_k$ with $\lambda > 2$. This implies that to substantially improve over the bound in Corollary 6.3 one needs to either refine the compression boosting theorem or use a completely different approach (possibly not based on the bwt). However, by Theorem 4.5 and the fact that $H_k^*(s) \leqslant H_0^*(s)$, no prefix-free compressor can achieve an entropy only bound of the form $\lambda|s|H_k^*(s) + g_k$ with $\lambda \leqslant 2$.

7. Pruned Wavelet Trees

In point (**D**) of Section 1, we discussed the impact on the cost of a Wavelet Tree of: (D.1) its (binary) shape, (D.2) the assignment of alphabet symbols to its leaves, (D.3) the possible use of non-binary compressors to encode the strings associated to internal nodes. Here we provide some concrete examples showing that these issues cannot be neglected. In the following, we consider Rle Wavelet Trees but similar examples can be given for Ge Wavelet Trees as well.

For (D.1), let us consider the infinite family of strings $s_n = a_1^n a_2 a_3 \cdots a_{|\Sigma|}$. For large n, the encoding cost is dominated by the $\Theta(\log n)$ cost of encoding a_1^n. If the Rle Wavelet Tree is balanced, we pay this cost $\Theta(\log |\Sigma|)$ times. If the leaf corresponding to a_1 is at depth 1, we pay this cost only once. This means that the Rle Wavelet Tree shape may impact the output size by a multiplicative factor $\Theta(\log |\Sigma|)$.

For (D.2), consider again $s_n = a_1^n a_2 a_3, \cdots, a_{|\Sigma|}$ and a skewed Wavelet Tree, in which leaves have depth $1, 2, \ldots, |\Sigma| - 1$. It is easy to see that the overall cost increases by a factor $\Theta(|\Sigma|)$, if a_1 is assigned to a leaf of depth $|\Sigma| - 1$ rather than to the leaf of depth 1. Note that the symbol-leaf mapping is critical even for balanced Wavelet Trees: Consider the string $a_1^n a_3^n a_2^n a_4^n$ and a balanced tree of height 2. If leaves are labeled a_1, a_2, a_3, a_4 (left to right) the encoding cost is $\approx 8a \log n$, whereas for the ordering a_1, a_3, a_2, a_4, the encoding cost is $\approx 6a \log n$. (Recall that we are assuming $\mathsf{Pfx}(x) \leqslant a \log x + b$.)

As for (D.3), let us consider the infinite family of strings $s_n = a_1^n (a_2 a_2 a_3 a_3)^n$ and the balanced Wavelet Tree for s_n, where the leaf corresponding to a_1 is the left child of the root r, while a_2 and a_3 are assigned to the leaves descending from the right child, say u, of the root. We have $s^{\mathbf{01}}(r) = 0^n 1^{4n}$, $s(u) = (a_2 a_2 a_3 a_3)^n$, and $s^{\mathbf{01}}(u) = (0011)^n$. We compress $s^{\mathbf{01}}(r)$ via Rle in $\Theta(\log n)$ bits, and we compress $s^{\mathbf{01}}(u)$ either via Huffman or via Rle. The former takes $4n$ bits, while the latter takes $2n|\mathsf{Pfx}(2)|$ bits, which is at least $6n$ bits, for all representations of the integers for which $|\mathsf{Pfx}(2)| > 2$ (this includes γ and δ codes [6] and all Fibonacci representations of order > 1 [2]). This shows that a mixed encoding strategy may save a constant multiplicative factor on the output size.

Those examples motivate us to introduce and discuss Pruned Wavelet Trees, a new paradigm for the design of effective zero-order compressors. Let A_{01} and A_Σ be two compression algorithms such that A_{01} is a compressor specialized onto binary strings while A_Σ is a generic compressor working on strings drawn over arbitrary alphabets Σ. We assume that A_{01} and A_Σ satisfy the following property, which holds—for example—when A_{01} is Ge* (see Section 6) and A_Σ is Arithmetic or Huffman coding.

Property 2. *Let* A_{01} and A_Σ be two compression algorithms such that:

(a) For any binary string z with $H_0(z) \neq 0$, $|\mathsf{A}_{01}(z)| \leqslant \alpha|z|H_0(z) + \beta$ bits, where α and β are constants.
(b) For any string s, $|\mathsf{A}_\Sigma(s)| \leqslant |s|H_0(s) + \eta|s| + \mu$ bits, where η and μ are constants.
(c) The running times of both A_{01} and A_Σ are convex functions (say T_{01} and T_Σ) and their working space are non-decreasing functions (say S_{01} and S_Σ).

Given the Wavelet Tree $W_c(s)$, a subset $\mathscr{L}$ of its nodes is a *leaf cover* if every leaf of $W_c(s)$ has a *unique* ancestor in $\mathscr{L}$ (see [9, Section 4]). Let $\mathscr{L}$ be a leaf cover of $W_p(s)$ and let $W_p^{\mathscr{L}}(s)$ be the tree obtained by removing all nodes in $W_p(s)$ descending from nodes in $\mathscr{L}$. We assign colors to nodes of $W_p^{\mathscr{L}}(s)$ as follows: all leaves are *black* and the remaining nodes *red*. We use A_{01} to compress all binary strings $s^{\mathbf{01}}(u)$, where $u \in W_p^{\mathscr{L}}(s)$ and it is colored *red*, while we use A_Σ to compress all strings $s(u)$, where $u \in W_p^{\mathscr{L}}(s)$ and it is colored *black*. Nodes that are leaves of $W_p(s)$ are ignored (as usual). It is a simple exercise to work

out the details on how to make this encoding decodable.[9] The *cost* $\left\|W_p^{\mathscr{L}}(s)\right\|_P$ of the Pruned Wavelet Tree is defined as the total number of bits produced by the encoding process just described:

$$\left\|W_p^{\mathscr{L}}(s)\right\|_P = \sum_{u\ red} |\mathsf{A}_{01}(s^{\mathbf{01}}(u))| + \sum_{u\ black} |\mathsf{A}_{\Sigma}(s(u))|.$$

Example 2. If $\mathscr{L} = root(W_p(s))$, then $W_p^{\mathscr{L}}(s)$ consists of one node only, namely the root, and thus we compress the entire s using A_Σ only. By Property 2(b), we have $\left\|W_p^{\mathscr{L}}(s)\right\|_P \leqslant |s|H_0(s) + \eta|s| + \mu$. The other extreme case is when $\mathscr{L}$ consists of all the leaves of $W_p(s)$, and thus $W_p(s) = W_p^{\mathscr{L}}(s)$: we never use A_Σ and we have $\left\|W_p^{\mathscr{L}}(s)\right\|_P \leqslant \alpha|s|H_0(s) + \beta(|\Sigma| - 1)$, by Property 2(a) and Theorem 4.1.

We note that when the algorithms A_{01} and A_Σ are fixed, the cost $\left\|W_p^{\mathscr{L}}(s)\right\|_P$ depends on two factors: the shape of the alphabetic tree T_Σ, and the leaf cover $\mathscr{L}$. The former determines the shape of the Wavelet Tree and the assignment of alphabet symbols to the leaves of the tree, the latter determines the assignment of A_{01} and A_Σ to the nodes of $W_p(s)$. It is therefore natural to consider the following two optimization problems.

Problem 1. Given a string s and a Wavelet Tree $W_p(s)$, find the optimal leaf cover $\mathscr{L}_{\min}$ that minimizes the cost function $\left\|W_p^{\mathscr{L}}(s)\right\|_P$. Let $C_{opt}(W_p(s))$ be the corresponding optimal cost.

Problem 2. Given a string s, find an alphabetic tree T_Σ and a leaf cover $\mathscr{L}_{\min}$ for that tree, giving the minimum of the function $C_{opt}(W_p(s))$. That is, we are interested in finding a shape of the Wavelet Tree, an assignment of alphabet symbols to the leaves of the tree, and an assignment of A_{01} and A_Σ to the Wavelet Tree nodes, so that the resulting compressed string is the shortest possible.

Problem 2 is a global optimization problem, while Problem 1 is a much more constrained local optimization problem. Note that, by Example 2, we have $C_{opt}(W_p(s)) \leqslant \min(|s|H_0(s) + \eta|s| + \mu, \alpha|s|H_0(s) + \beta(|\Sigma| - 1))$.

7.1. Optimization algorithms

The first algorithm we describe solves efficiently Problem 1. Its pseudo-code is given in Fig. 6. The key observation is a *decomposability property* of the cost functions associated to $\mathscr{L}_{\min}$, with respect to the subtrees of $W_p(s)$. We point out that such a property is essentially the same identified by Ferragina et al. for their linear time Compression Boosting algorithm [9], here exploited to devise an optimal Pruned Wavelet Tree in efficient time. In the following, with a little abuse of notation, we denote by $\mathscr{L}_{\min}(u)$ an optimal leaf cover of the subtree of $W_p(s)$ rooted at the node u and by $C_{opt}(u)$ the corresponding cost.

Lemma 7.1. *$\mathscr{L}_{\min}(u)$ consists of either the single node u, or of the union of optimal leaf covers of the subtrees rooted at its left and right children u_L and u_R, respectively.*

Proof. We can assume that s consists of at least two distinct symbols. When u is a leaf of $W_p(s)$, the result obviously holds, since the optimal leaf cover is the node itself and its cost is zero. Assume that u is an internal node of depth at least two. Note that both node sets $\{u\}$ and $\mathscr{L}_{\min}(u_L) \bigcup \mathscr{L}_{\min}(u_R)$ are leaf covers of the subtree of $W_p(s)$ rooted at u. We now show that one of them is optimal for that subtree. Let us assume that $\mathscr{L}_{\min}(u) \neq \{u\}$. Then, $\mathscr{L}_{\min}(u)$ consists of nodes which descend from u. We can then partition $\mathscr{L}_{\min}(u)$ as $\mathscr{L}(u_L) \bigcup \mathscr{L}(u_R)$, where those sets are leaf covers for the subtree rooted at u_L and u_R respectively. By the optimality of $\mathscr{L}_{\min}(u_L)$ and $\mathscr{L}_{\min}(u_R)$ we have that their cost cannot be higher than the one of $\mathscr{L}(u_L)$ and $\mathscr{L}(u_R)$ and therefore $\mathscr{L}_{\min}(u_L) \bigcup \mathscr{L}_{\min}(u_R)$ is an optimal leaf cover as well. □

Theorem 7.2. *Given two compressors satisfying Property 2 and a Wavelet Tree $W_p(s)$, the algorithm in Fig. 6 solves Problem 1 in $O(|\Sigma|(T_{01}(|s|) + T_\Sigma(|s|)))$ time and $O(|s| \log |s| + \max(S_{01}(|s|), S_\Sigma(|s|)))$ bits of space.*

Proof. The correctness of the algorithm comes from Lemma 7.1. As for the time analysis, it is based on the convexity of the functions $T_{01}(\cdot)$ and $T_\Sigma(\cdot)$, which implies that on any Wavelet Tree level we spend $O(T_{01}(|s|) + T_\Sigma(|s|))$ time, and the fact that

[9] Note that we need to encode which compressor is used at each node and (possibly) the tree shape. For simplicity, in the following, we ignore this $\Theta(|\Sigma|)$ bits overhead.

(1) If r is the only node, let $C_{opt}(r) \leftarrow |\mathsf{A}_{01}(s)|$ and $\mathcal{L}(r) \leftarrow \{r\}$.
(2) Else, visit $W_p(s)$ in post-order. Let u be the currently visited node.
  (2.1) If u is a leaf, let $Z(u) \leftarrow 0$ and $\mathcal{L}(u) \leftarrow \{u\}$. Return.
  (2.2) Compute $Z(u) \leftarrow \min\{|\mathsf{A}_\Sigma(s(u))|, |\mathsf{A}_{01}(s^{\mathbf{01}}(u))| + Z(u_L) + Z(u_R)\}$.
  (2.3) If $Z(u) = |\mathsf{A}_\Sigma(s(u))|$ then $\mathcal{L}(u) \leftarrow \{u\}$, else $\mathcal{L}(u) \leftarrow \mathcal{L}(u_L) \cup \mathcal{L}(u_R)$.
(3) Set $\mathcal{L}_{\min} \leftarrow \mathcal{L}(root(T))$.

Fig. 6. The pseudocode for the linear-time computation of an optimal leaf cover $\mathscr{L}_{\min}$ for a given decomposition tree T_s.

the tree has height at most $|\Sigma|$. Finally, the proof of the space bound can be derived from the monotonicity of the working space functions of the two compression algorithms. □

Note that the algorithm of Fig. 6 can be turned into an exhaustive search procedure for the solution of Problem 2. The time complexity would be polynomial in $|s|$ but at least exponential in $|\Sigma|$. Although we are not able to provide algorithms for the global optima with time complexity polynomial both in $|\Sigma|$ and $|s|$, we are able to settle the important special case in which the ordering of the alphabet is assigned, and so is assigned the mapping of the symbols to the leaves of the Wavelet Tree.

Fix a total order relation $\prec$ of the alphabet symbols. To simplify notation, let it be $\Sigma = \{a_1, a_2, \cdots, a_{|\Sigma|}\}$. Consider the class of Wavelet Trees such that a visit in symmetric order gives the leaves ordered according to the $\prec$ relation defined on the alphabet. We restrict Problem 2 to such a class of trees and let $C_{opt,\prec}$ be the corresponding optimal cost.

Given a string s, let $s_{i,j}$ be the string obtained by keeping only the symbols $a_i, \cdots, a_j$ in s. Moreover, given an integer k, $i < k \leqslant j$, let $s^{\mathbf{01}}_{i,j,k}$ be the binary string obtained by replacing with 0 each character a_ℓ, $i \leqslant \ell < k$, in $s_{i,j}$ and with 1 each remaining symbol. Let $C_{i,j}$ be the optimal cost of compressing string $s_{i,j}$ with Pruned Wavelet Trees, subject to the constraint that the symbols of the alphabet must appear "from left to right" in the order $a_i, \cdots, a_j$. Hence, $C_{opt,\prec} = C_{1,|\Sigma|}$. We now show that such an optimal cost, together with an optimal Pruned Wavelet Tree, can be computed via the following dynamic programming algorithm:

$$C_{i,j} = \min(|\mathsf{A}_\Sigma(s_{i,j})|, \min_{i<k\leqslant j}(C_{i,k-1} + C_{k,j} + |\mathsf{A}_{01}(s^{\mathbf{01}}_{i,j,k})|)) \tag{23}$$

where $i < j$ and the initial conditions are given by $C_{i,i} = 0$. Indeed, the correctness of the above recurrence relation comes from the observation that, for each string $s_{i,j}$, we can compress it in two possible ways: (a) use A_Σ on the entire string or (b) choose optimally a character a_k in the list $a_i, \cdots, a_j$; compress both $s_{i,k-1}$ and $s_{k,j}$ optimally and finally compress the binary sequence $s^{\mathbf{01}}_{i,j,k}$. Moreover, $C_{i,j}$ is labeled *black* if A_Σ wins. Else, it is labeled *red*. We can then recover the optimal alphabetic tree via standard traceback techniques and the color assigned to any chosen $C_{i,j}$ is assigned to the node corresponding to it. Finally, the optimal leaf cover is given by the set of *black* nodes that have no *black* node as an ancestor. As for the time analysis, it is a standard textbook exercise. We have:

Theorem 7.3. *Consider a string s and fix an ordering $\prec$ of the alphabet symbols appearing in the string. Then, one can solve Problem 2 constrained to that ordering of Σ, in $O(|\Sigma|^4(T_{01}(|s|) + T_\Sigma(|s|))$ time.*

8. Conclusions and open problems

We have provided a throughout theoretical analysis of a wide class of compression algorithms based on Wavelet Trees, and also shown how to improve their asymptotic performance by introducing a novel framework, called Pruned Wavelet Trees, that aims for the best combination of binary compressors and generic compressors in their nodes.

Our results raise several open questions. First, it would be interesting to improve the bounds established for Rle and Ge, possibly introducing small modifications to the basic algorithms. Our results for the modified Gap Encoder Ge* suggest that this should be possible. Second, it would be interesting to extend our analysis to prefix-free codes such that $|\mathsf{Pfx}(i)| \leqslant \log(i) + o(\log(i))$ (one of such codes is δ-coding). Third, further study should be devoted to understand the influence of the shape of the Wavelet Tree on compression (see also Problem 2 in Section 7). Finally, our results ask for a deeper investigation, both at the algorithmic and at the experimental level, on Pruned Wavelet Trees.

Acknowledgments

The authors would like to thank the anonymous reviewers for their careful reading and helpful suggestions.

References

[1] J. Abel, Improvements to the Burrows–Wheeler compression algorithm: after BWT stages. Available from: <http://citeseer.ist.psu.edu/abel03improvements.html>.
[2] A. Apostolico, A.S. Fraenkel, Robust transmission of unbounded strings using Fibonacci representations, IEEE Transactions on Information Theory, 33 (1987) 238–245.
[3] Z. Arnavut, Inversion coding, The Computer Journal 47 (2004) 46–57.
[4] Z. Arnavut, S. Magliveras, Block sorting and compression, in: Proceedings of IEEE Data Compression Conference (DCC), 1997, pp. 181–190.
[5] M. Burrows, D. Wheeler, A block-sorting lossless data compression algorithm. Technical Report 124, Digital Equipment Corporation, 1994.
[6] P. Elias, Universal codeword sets and representations of the integers, IEEE Transactions on Information Theory 21 (2) (1975) 194–203.
[7] P. Ferragina, R. Giancarlo, G. Manzini, The engineering of a compression boosting library: theory vs practice in BWT compression, in: Proceedings of the 14th European Symposium on Algorithms (ESA), Lecture Notes in Computer Science, vol. 4168, Springer, 2006, pp. 756–767.
[8] P. Ferragina, R. Giancarlo, G. Manzini, The myriad virtues of wavelet trees, in: Proceedings of the 33rd International Colloquium on Automata and Languages (ICALP), Lecture Notes in Computer Science, vol. 4051, Springer, 2006, pp. 561–572.
[9] P. Ferragina, R. Giancarlo, G. Manzini, M. Sciortino, Boosting textual compression in optimal linear time, Journal of the ACM 52 (2005) 688–713.
[10] L. Foschini, R. Grossi, A. Gupta, J. Vitter, Fast compression with a static model in high-order entropy, in: Proceedings of IEEE Data Compression Conference (DCC), 2004, pp. 62–71.
[11] L. Foschini, R. Grossi, A. Gupta, J. Vitter, When indexing equals compression: experiments on compressing suffix arrays and applications, ACM Transactions on Algorithms 2 (2006) 611–639.
[12] T. Gagie, G. Manzini, Move-to-front, distance coding, and inversion frequencies revisited, Technical Report TR-INF-2008-03-02-UNIPMN, Dipartimento di Informatica, Università Piemonte Orientale, 2008. Available from: <http://www.di.unipmn.it>.
[13] R. Grossi, A. Gupta, J. Vitter, High-order entropy-compressed text indexes, in: Proceedings of the 14th ACM-SIAM Symposium on Discrete Algorithms (SODA), 2003, pp. 841–850.
[14] L. Foschini, R. Grossi, A. Gupta, J. Vitter, Fast compression with a static model in high-order entropy, in: Proceedings of IEEE Data Compression Conference (DCC), 2004, pp. 62–71.
[15] K.B. Lakshmanan, On universal codeword sets, IEEE Transactions in Information Theory 27 (1981) 659–662.
[16] V. Mäkinen, G. Navarro, Succinct suffix arrays based on run-length encoding, Nordic Journal of Computing 12 (1) (2005) 40–66.
[17] V. Mäkinen, G. Navarro, Implicit compression boosting with applications to self-indexing, in: Proceedings of the 14th Symposium on String Processing and Information Retrieval (SPIRE'07), LNCS No. 4726, Springer-Verlag, 2007, pp. 214–226.
[18] G. Manzini, An analysis of the Burrows–Wheeler transform, Journal of the ACM 48 (3) (2001) 407–430.
[19] G. Navarro, V. Mäkinen, Compressed full-text indexes, ACM Computing Surveys 39 (1) (2007).
[20] R. Raman, V. Raman, S.S. Rao, Succinct indexable dictionaries with applications to encoding k-ary trees and multisets, in: Proceedings of the 13th ACM-SIAM Symposium on Discrete Algorithms (SODA), 2002, pp. 233–242.