The Miryad Virtues of Wavelet Trees (Extended Abstract)

Paolo Ferragina[1], Raffaele Giancarlo[2], and Giovanni Manzini[3]

[1] Dipartimento di Informatica, University of Pisa, Italy.
[2] Dipartimento di Matematica ed Applicazioni, Università di Palermo, Italy
[3] Dipartimento di Informatica, University of Piemonte Orientale, Italy.

Abstract. Wavelet Trees have been introduced in [Grossi, Gupta and Vitter, ACM-SIAM 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 beauty 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 complete theoretical analysis of a wide class of compression algorithms based on Wavelet Trees. We also show how to improve their asymptotic performance by introducing a novel framework, called *Red-Black Wavelet Trees*, that aims for the best combination of binary compressors (like, Run-Length encoders) versus non-binary compressors (like, Huffman and Arithmetic encoders) and Wavelet Trees of properly-designed shapes. As a corollary, we prove high-order entropy bounds for the challenging combination of Burrows-Wheeler Transform and Wavelet Trees.

1 Introduction

The Burrows-Wheeler Transform [4] (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. [5, 6, 8, 10, 11]). The transform reduces the problem of high-order entropy compression to the apparently simpler task of designing and engineering good order-zero (or memoryless) compressors. This point has lead to the paradigm of compression boosting presented in [5]. However, despite nearly 60 years of investigation in the design of good memoryless compressors, no general theory for the design of order-zero 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 order-zero 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 order-zero compressors and run length encoding techniques. However, such a design problem is mostly open. Recently Grossi et al. [8, 9] 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 2.1.

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 groundbreaking 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. Yet, it is still open the fundamental question of whether Wavelet Trees can provide a data structural paradigm based on which one can design good order-zero compression algorithms for the Burrows-Wheeler Transform.

Our main contribution is to answer this question in the affirmative by providing the first general paradigm, and associated analytic tools, for the design of good order-zero compressors for the bwt. It is also rather fortunate that 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 the first complete theoretical analysis of Wavelet Trees as *stand-alone*, general purpose, order-zero compressors for an arbitrary string σ. We consider both the case in which binary strings associated to the tree are compressed using Run Length Encoding (RLE), and refer to it as RLE Wavelet Tree, and the case in which Gap Encoding (GE) is used, and refer to it as GE Wavelet Tree. In both cases, a generic prefix-free encoding of the integers is used as a subroutine, thus dealing with the typical scenario of use for Wavelet Trees (see [7–9]). Our analysis is done in terms of the features of these prefix-free encodings and $H_0^*(\sigma)$, the modified order-zero entropy of the string σ, defined in Section 2. As a notable Corollary, we also obtain the first analysis of Inversion Frequencies coding [2] offering a theoretical justification of the better compression observed in practice by this technique with respect to Move-to-Front coding [3].

(**B**) We study the use of Wavelet Trees to compress the output of the bwt. We show that RLE Wavelet Trees achieve a compression bound in terms of $H_k^*(\sigma)$. The technical results are in Section 4 and, although related to results in [9] (namely, cfr. Theorem 3.2), they are somewhat more general and accurate. We also show that GE Wavelet Trees *cannot achieve* analogous bounds. A striking consequence of our analytic results is to give full theoretic support to the engineering choices made in [9] (see also [7]) where, based on a punctual experimental analysis of the data, the former method is preferred to the latter to compress the output of the bwt.

(**C**) We define *Red-Black Wavelet Trees*. They 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 order-zero compressors. The same considerations apply when they are used in (B). Wavelet Trees reduce the problem of compressing a string to that of compressing a set of binary strings. That set is uniquely identified by:

(C.1) the shape (or topology) of the binary tree underlying the Wavelet Tree; (C.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 (C.1) and (C.2). They also bring some experimental evidence that choosing the "best" Wavelet Tree may be difficult [9, Sect. 3.1]. In Section 5 we exhibit an infinite family of strings over an alphabet Σ for which changing the Wavelet Tree shape (C.1) influences the coding cost by a factor $\Theta(\log|\Sigma|)$, and changing the assignment of symbols to leaves (C.2) influences the coding cost by a factor $\Theta(|\Sigma|)$. So, the choice of the best tree cannot be neglected and remains open. Moreover, (C.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 order-zero compressors, such as Arithmetic and Huffman coding). We exhibit an infinite family of strings for which a mixed strategy yields a constant factor improvement over standard Wavelet Trees. So, (C.3) is relevant and open.

We introduce the new paradigm of Red-Black Wavelet Trees that allows us to reduce the compression of a string σ to the identification of a set of strings, where only a part may be binary, which are compressed via the mixed strategy sketched above. We develop a combinatorial optimization framework so that one can address points (C.1)-(C.3) simultaneously. Moreover, we provide a *polynomial-time* algorithm for finding the *optimal* mixed strategy for a Red-Black Wavelet Tree of *fixed shape* (Theorem 5). In addition, we provide a *polynomial-time* algorithm for selecting the *optimal tree-shape* for Red-Black Wavelet Trees when the size of the alphabet is constant and the assignment of symbols to the leaves of the tree is fixed (Theorem 6). Apart from their intrinsic interest, being Wavelet Trees a special case, those two results shed some light on a problem implicitly posed in [9], 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.

Due to space limitations, all proofs of technical lemmas are in Appendix A, while proofs of Theorems are either sketched or reported in Appendix A.

2 Background and Notation

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. Throughout this paper we assume that all logarithms are taken to the base 2 and $0 \log 0 = 0$. The *0-th order empirical entropy* of the string 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. We can achieve a greater compression if the codeword we use for each symbol depends on the k symbols preceding it, since the maximum compression is now bounded by the k-th order entropy $H_k(s)$ (see [11] for the formal definition).

For highly compressible strings, $|s|\,H_k(s)$ fails to provide a reasonable bound to the performance of compression algorithms (see discussion in [5, 11]). For that

reason, [11] introduced the notion of 0-*th order modified empirical* entropy:

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

Note that 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. The *k-th order modified empirical entropy* H_k^* is then defined in terms of H_0^* as the maximum compression we can achieve by looking at *no more than* k symbols preceding the one to be compressed.

2.1 Wavelet trees

To simplify the exposition of our results we use a slightly more verbose notation that the one in [8] since we need to distinguish between the base alphabet Σ and the set of symbols that actually appear in a given string.

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. 1 (see also Fig. 2). Note that to each internal node $u \in W_f(s)$ we associate two strings of equal length. The first one, assigned in Step 1, is a string over Σ and we denote it by $s(u)$. The second one, assigned in Step 3, is a binary string and we denote it 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.

In this paper we use $\Sigma^{(s)}$ to denote the set of symbols that appear in s. If $\Sigma^{(s)} = \Sigma$, the Wavelet Tree $W_f(s)$ has the same shape as T_Σ and 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 *pruned Wavelet Tree* $W_p(s)$ which is a complete binary tree with $|\Sigma^{(s)}|$ leaves and $|\Sigma^{(s)}| - 1$ internal nodes (see Fig. 2).

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 $\Sigma^{(u_L)}$ (reps. $\Sigma^{(u_R)}$) 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. 1. 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 $u = root(T)$.

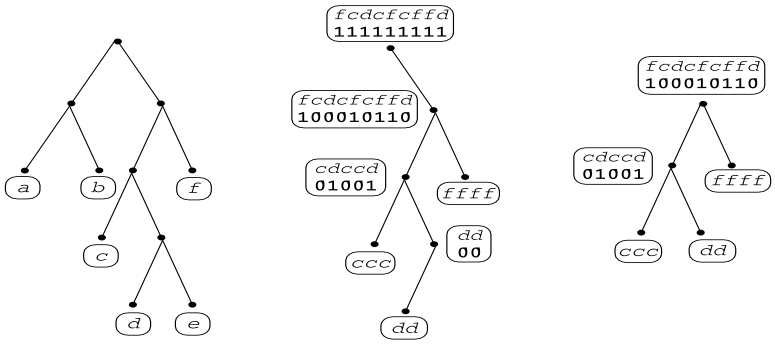

Fig. 2. An alphabetic tree (left) for the alphabet $\Sigma = \{\texttt{a,b,c,d,e,f}\}$. The corresponding full Wavelet Tree (center) for the string $s = \texttt{fcdcfcffd}$. The pruned Wavelet Tree (right) for the same string.

As observed in [8], 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 only an encoder for binary strings*. Let $\mathcal{C}$ denote any algorithm for encoding binary strings. For any internal node u we denote by $C^*(u)$ the length of the encoding of $s^{\mathbf{01}}(u)$ via $\mathcal{C}$, that is, $C^*(u) = |\mathcal{C}(s^{\mathbf{01}}(u))|$. With a little abuse of notation we write $C^*(W_p(s))$ to denote the total cost of encoding the Wavelet Tree $W_p(s)$. Namely, $C^*(W_p(s)) = \sum_{u \in W_p(s)} C^*(u)$, where the sum is done over the internal nodes only. $C^*(W_f(s))$ is defined similarly. The following fundamental property of pruned Wavelet Trees was established in [8] and shows that there is essentially no loss in compression performance when we compress an arbitrary string s using Wavelet Trees and a binary encoder.

Theorem 1 (Grossi et al., ACM Soda 2003). *Let $\mathcal{C}$ be a binary encoder such that for any binary string z the bound $|\mathcal{C}(z)| \leq \lambda |z| H_0(z) + \eta |z| + \mu$ holds with constant λ, η, μ. Then, for a string s drawn from any alphabet $\Sigma^{(s)}$, we have $C^*(W_p(s)) = \sum_{u \in W_p(s)} |\mathcal{C}(s^{\mathbf{01}}(u))| \leq \lambda |s| H_0(s) + \eta |s| + (|\Sigma^{(s)}| - 1)\mu$. The bound holds regardless of the shape of $W_p(s)$. The same result holds when the entropy H_0 is replaced by the modified entropy H_0^*.* □

3 Achieving 0-th order entropy with Wavelet Trees

This section contains a technical outline of the results claimed in (**A**) of the Introduction, where Wavelet Trees are used as stand alone, general purpose, order-zero compressors. In particular, we analyze the performance of RLE Wavelet Trees (Section 3.1) and GE Wavelet Trees (Section 3.2) showing that GE is superior to RLE as an order-zero compressor over Wavelet Trees. Nevertheless, we will show in Section 4 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 [7, 9]. Moreover, a remarkable corollary of this section is the first theoretical analysis of Inversion Frequencies coding [2].

Let $\mathcal{C}_{PF}$ denote a prefix-free encoding of the integers having logarithmic cost, namely $|\mathcal{C}_{PF}(n)| \leq a \log n + b$, for $n \geq 1$. Note that since $|\mathcal{C}_{PF}(1)| \leq b$ we must have $b \geq 1$. Also note that for γ codes we have $a = 2$ and $b = 1$. This means that it is worthwhile to investigate only prefix codes with $a \leq 2$. Indeed, a code with $a > 2$ (and necessarily $b \geq 1$) would be worse than γ codes for any n and therefore not interesting. Hence in the following we assume $a \leq 2$, $b \geq 1$ and thus $a \leq 2b$ and $a \leq b + 1$.

3.1 Analysis of RLE Wavelet Trees

For any binary string $s = a_1^{\ell_1} a_2^{\ell_2} \cdots a_k^{\ell_k}$, with $a_i \in \{0, 1\}$ and $a_i \neq a_{i+1}$, we define $\mathcal{C}_{RLE}(s) = a_1 \mathcal{C}_{PF}(\ell_1)\mathcal{C}_{PF}(\ell_2) \cdots \mathcal{C}_{PF}(\ell_k)$. Note that we need to store explicitly the bit a_1 since the values $\ell_1, \ldots, \ell_k$ alone are not sufficient to retrieve s.

Lemma 1. *For any binary string $s = a_1^{\ell_1} a_2^{\ell_2} \cdots a_k^{\ell_k}$, with $a_i \in \{0, 1\}$ and $a_i \neq a_{i+1}$, we have $|\mathcal{C}_{RLE}(s)| = 1 + \sum_{i=1,k} |\mathcal{C}_{PF}(\ell_i)| \leq 2 \max(a, b)|s|H_0^*(s) + b + 1$.* □

Combining the above Lemma with Theorem 1 we immediately get:

Corollary 1. *For any string s over the alphabet $\Sigma^{(s)}$, if the internal nodes of the Wavelet Tree $W_p(s)$ are encoded using* RLE *we have*

$$C^*(W_p(s)) \leq 2 \max(a, b)|s|H_0^*(s) + (|\Sigma^{(s)}| - 1)(b + 1). \quad \square$$

Consider now the algorithm rle_wt defined as follows. We first encode $|s|$ using $|\mathcal{C}_{PF}(|s|)| \leq a \log |s| + b$ bits. Then we encode the internal nodes of the Wavelet Tree using RLE. The internal 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).[1] This ensures that from the output of rle_wt we can always retrieve s. To see this, we observe that 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 bits are needed to mark the end of the run-length encoding. Since the output of rle_wt consists of $|\mathcal{C}_{PF}(|s|)| + C^*(W_p(s))$ bits, by Corollary 1 we get:

Theorem 2. *For any string s over the alphabet $\Sigma^{(s)}$ we have*

$$|\mathsf{rle\_wt}(s)| \leq 2 \max(a, b)|s|H_0^*(s) + (b + 1)|\Sigma^{(s)}| + a \log |s| - 1. \qquad \square$$

3.2 Analysis of GE Wavelet Trees

For any binary string s with exactly r 1's, let $p_1, p_2, \ldots, p_r$ denote their positions in s, and let $g_1, \ldots, g_r$ be defined by $g_1 = p_1$, $g_i = p_i - p_{i-1}$ for $i = 2, \ldots, r$. We denote by $\mathcal{C}_{Gap}(s)$ the concatenation $\mathcal{C}_{PF}(g_1)\mathcal{C}_{PF}(g_2) \cdots \mathcal{C}_{PF}(g_r)$.

Lemma 2. *Let s be a binary string with r 1's. If $1 \leq r \leq |s|/2$, we have*

$$|\mathcal{C}_{Gap}(s)| = |\mathcal{C}_{PF}(g_1)| + \cdots + |\mathcal{C}_{PF}(g_r)| \leq \max(a, b)|s|H_0(s). \qquad \square$$

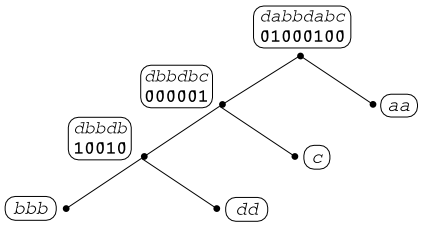

Fig. 3. The skewed Wavelet Tree for the string $s =$ `dabbdabc`. Symbol `b` is the most frequent one and is therefore associated to the leftmost leaf.

Let s be a string over the alphabet $\Sigma^{(s)}$. Consider the following algorithm called ge_wt. First we encode the length of s using $a \log |s| + b$ bits and the number of occurrences of each symbol using a total of $|\Sigma^{(s)}| \lceil \log |s| \rceil$ bits. Then, we 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. 3). Finally, we use GE to encode the strings $s^{\mathbf{01}}(u_1), \ldots, s^{\mathbf{01}}(u_{|\Sigma^{(s)}|-1})$ associated to the internal nodes of such Wavelet Tree. Note that this information is sufficient to reconstruct the input string s. The crucial point is that the decoding starts with the retrieval of the number of occurrences of each symbol. Hence, we can immediately determine the association between leaves and symbols and when we later decode a string $s^{\mathbf{01}}(u_i)$ we already know its length and the number of 1's in it.

Theorem 3. *For any string s, it is*

$$|\mathsf{ge\_wt}(s)| \leq \max(a, b)|s|\, H_0(s) + |\Sigma^{(s)}| \lceil \log |s| \rceil + a \log |s| + b.$$

Proof. We only need to show that $\sum_i |\mathcal{C}_{Gap}(s^{\mathbf{01}}(u_i))| \leq \max(a, b)|s|\, H_0(s)$. To this end we observe that assigning the most frequent symbol to the leftmost leaf ensures that each $s^{\mathbf{01}}(u_i)$ contains more 0's than 1's. The thesis follows by Lemma 2 and Theorem 1. □

Let us give a closer look to the ge_wt algorithm when $\Sigma = \{a_1, a_2, \ldots, a_h\}$ and a_h is the most frequent symbol. In this case, when we encode $s^{\mathbf{01}}(u_i)$ we are encoding 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) and was first suggested in [2] as an alternative to Move-to-Front (MTF) encoding. We have therefore the following result.

Corollary 2. *The variant of* IF*-coding in which the most frequent symbol is processed last produces a sequence of integers that we can encode with $\mathcal{C}_{PF}$ in at most* $\max(a, b)|s|\, H_0(s)$ *bits.* □

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

The standard analysis of MTF tells us that combining $\mathcal{C}_{PF}$ with MTF produces an output bounded by $a|s|\,H_0(s) + b|s|$ bits. Hence, the above corollary is the first theoretical justification of the fact, observed by practitioners, that IF-coding is superior to MTF [1, 2]. Corollary 2 also provides a theoretical justification for the strategy, suggested in [1], of processing the symbols in order of increasing frequency.

4 Achieving H_k^* with RLE Wavelet Trees and bwt

This section provides the technical details about the results claimed in (**B**) of the Introduction. In particular, we show that by using RLE Wavelet Trees as a post-processor of the bwt one can achieve higher order entropy compression. We also show that the same result cannot hold for GE Wavelet Trees.

We need to recall a key property of the Burrows-Wheeler Transform of a string σ [11]: If $s = \mathsf{bwt}(\sigma)$ then for any $k \geq 0$ there exists a partition $s = s_1 s_2 \cdots s_t$ such that[2] $t \leq |\Sigma|^k$ and $|\sigma|\,H_k^*(\sigma) = \sum_{i=1}^{t} |s_i|\,H_0^*(s_i)$. 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.

The proof idea is to show 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 pruned Wavelet Trees. Indeed, if we "slice" the Wavelet Tree $W_p(s)$ according to the partition $s = s_1 \cdots s_t$ we get *full* Wavelet Trees for the strings s_i's.

Our first lemma states that, for full RLE Wavelet Trees, partitioning a string does not improve compression.

Lemma 3. *Let $\alpha = \alpha_1\alpha_2$ be a string over the alphabet Σ. We have $C^*(W_f(\alpha)) \leq C^*(W_f(\alpha_1)) + C^*(W_f(\alpha_2))$.* □

Since Theorem 1 bounds the cost of *pruned* Wavelet Trees, in order to use Lemma 3 we need to bound $C^*(W_f(\alpha_i))$ in terms of $C^*(W_p(\alpha_i))$, for $i = 1, 2$.

Lemma 4. *Let β be a string over the alphabet Σ. We have*

$$C^*(W_f(\beta)) \leq C^*(W_p(\beta)) + (|\Sigma| - 1)(a \log |\beta| + b + 1). \qquad \square$$

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

Theorem 4. *Let σ denote a string over the alphabet $\Sigma = \Sigma^{(s)}$, and let $s = \mathsf{bwt}(\sigma)$. For any $k \geq 0$ we have*

$$C^*(W_p(s)) \leq 2\max(a, b)|\sigma|\,H_k^*(\sigma) + |\Sigma|^{k+1}(2b + 2 + a\log(|s|)). \qquad (2)$$

[2] For simplicity we ignore the end-of-file symbol and the first k symbols of σ that do not belong to any s_i. We will take care of these details in the full version.

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

$$C^*(W_p(s)) \leq 2\max(a,b)|\sigma|\, H_k^*(\sigma) + o(|\sigma|). \tag{3}$$

Proof. Let $s = s_1 \cdots s_t$ denote the partition of s such that $|\sigma|\, H_k^*(\sigma) = \sum_{i=1}^t |s_i|\, H_0^*(s_i)$. By Lemma 3, and the fact that $\Sigma = \Sigma^{(s)}$, we have that $C^*(W_p(s)) = C^*(W_f(s)) \leq \sum_{i=1}^t C^*(W_f(s_i))$. By Lemma 4, we get

$$\begin{aligned} C^*(W_p(s)) &\leq \sum_{i=1}^t C^*(W_p(s_i)) + (|\Sigma|-1)\sum_{i=1}^t (a\log|s_i| + b + 1) \\ &= \sum_{i=1}^t C^*(W_p(s_i)) + (|\Sigma|-1)\sum_{i=1}^t a\log|s_i| + t(|\Sigma|-1)(b+1) \end{aligned}$$

Since $\sum_{i=1}^t \log|s_i| \leq t\log(|s|/t)$ and $t \leq |\Sigma|^k$, using Corollary 1 we get

$$\begin{aligned} C^*(W_p(s)) &\leq 2\max(a,b)\big(\textstyle\sum_{i=1}^t |s_i|\, H_0^*(s_i)\big) \\ &\quad + t(|\Sigma|-1)a\log(|s|/t) + 2t(|\Sigma|-1)(b+1) \\ &\leq 2\max(a,b)|\sigma|\, H_k^*(\sigma) + t(|\Sigma|-1)(2b+2+a\log(|s|/t)) \end{aligned} \tag{4}$$

which implies the bound (2). To prove (3) we start from (4) and note that $|\Sigma|$'s size and the inequality $t \leq |\Sigma|^k$ imply $t|\Sigma|\log(|s|/t) = o(|s|)$. □

Theorem 4 shows that RLE Wavelet Trees achieve the k-th order entropy with the same multiplicative constant $2\max(a,b)$ that RLE achieves with respect to H_0^* (Lemma 1). Thus, Wavelet Trees are a sort of *booster* for RLE (cfr. [5]). It is possible to prove (details in the full paper) that if we apply the Compression Boosting algorithm [5] to RLE we get slightly better bounds than the ones of Theorem 4, the improvement being in the term not containing H_k^*. However, Compression Boosting makes use of a non trivial (even if linear time) partitioning algorithm. It is therefore not obvious which approach is preferable in practice.

In proving Theorem 4 we have used some rather coarse upper bounds and we believe that the result can be significantly improved. However, there are some limits to the possible improvements. The following example shows that, even for constant size alphabets, the $o(|\sigma|)$ term in (3) cannot be reduced to $\Theta(1)$.

Example 1. Let $\Sigma = \{1, 2, \ldots, m\}$, and let $\sigma = (123\cdots m)^n$. We have $|\sigma|\, H_1^*(\sigma) \approx m\log n$ and $s = \mathsf{bwt}(\sigma) = m^n 1^n 2^n \cdots (m-1)^n$. 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 $\approx (m\log m)\log n \approx (\log m)|s|H_1^*(s)$ bits. □

Finally, 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 [9]—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 $W_f(s_i) \leq \lambda |s_i| H_0(s_i) + o(|s_i|)$. As a more concrete example, consider the string $\sigma = (01)^n$. We have $|\sigma| H_1^*(\sigma) = \Theta(\log n)$ and $s = \mathsf{bwt}(\sigma) = 1^n 0^n$. $W_p(s)$ 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 $|\sigma| H_1^*(\sigma)$.

5 Red-Black Wavelet Trees

In point (**C**) of the Introduction we discussed the impact on the cost of a Wavelet Tree of: (C.1) its (binary) shape, (C.2) the assignment of alphabet symbols to its leaves, (C.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 (C.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 (C.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 $a^N c^N b^N d^N$ and a balanced tree of height 2. If leaves are labelled a, b, c, d (left to right) the encoding cost is $\approx 8a \log N$, whereas for the ordering a, c, b, d the encoding cost is $\approx 6a \log N$.

As for (C.3) let us consider the infinite family of strings $s_N = a^N (bbcc)^N$ and the balanced Wavelet Tree for s_N where the leaf corresponding to a is the left child of the root r while b, c are assigned to the leaves descending from the right child u of the root. We have $s^{\mathbf{01}}(r) = 0^N 1^{4N}$, $s(u) = (bbcc)^N$, 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 $2N|\mathcal{C}_{PF}(2)|$ bits, which is at least $6N$ bits for all representations of the integers for which $|\mathcal{C}_{PF}(2)| > 2$ (this includes γ and δ codes and all Fibonacci representations of order > 1). This shows that a mixed encoding strategy may save a constant multiplicative factor on the output size.

Motivated by these examples, we introduce and discuss Red-Black Wavelet Trees, a new paradigm for the design of effective order-zero compressors. Let $\mathcal{C}_{01}$ and $\mathcal{C}_\Sigma$ be two compression algorithms such that $\mathcal{C}_{01}$ is a compressor specialized to binary strings while $\mathcal{C}_\Sigma$ is a generic compressor. We assume that $\mathcal{C}_{01}$ and $\mathcal{C}_\Sigma$ satisfy the following property, which holds—for example—when $\mathcal{C}_{01}$ is RLE (with γ codes or order-2 Fibonacci codes used for the coding of integers, see e.g. Lemma 1) and $\mathcal{C}_\Sigma$ is Arithmetic or Huffman coding.

Property 1. (a) For any binary string x, $|\mathcal{C}_{01}(x)| \leq \alpha|x|H_0^*(x) + \beta$ bits, where α and β are constants; (b) For any string y, $|\mathcal{C}_\Sigma(y)| \leq |y|H_0(y) + \eta|y| + \mu$ bits, where η and μ are constants; (c) the running time of $\mathcal{C}_{01}$ and $\mathcal{C}_\Sigma$ is a convex function (say T_{01} and T_Σ) and their working space is a non decreasing function (say S_{01} and S_Σ). □

Given the Wavelet Tree $W_p(s)$, a subset $\mathcal{L}$ of its nodes is a *leaf cover* if every leaf of $W_p(s)$ has a *unique* ancestor in $\mathcal{L}$ (see [5, Sect. 4]). Let $\mathcal{L}$ be a leaf cover of $W_p(s)$ and let $W_p^{\mathcal{L}}(s)$ be the tree obtained by removing all nodes in $W_p(s)$ descending from nodes in $\mathcal{L}$. We assign colors to nodes of $W_p^{\mathcal{L}}(s)$ as follows: all leaves are *black* and the remaining nodes *red*. We use $\mathcal{C}_{01}$ to compress all binary strings $s^{\mathbf{01}}(u)$, $u \in W_p^{\mathcal{L}}(s)$ and *red*, while we use $\mathcal{C}_\Sigma$ to compress all strings $s(u)$, $u \in W_p^{\mathcal{L}}(s)$ and *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.[3] The *cost* $C^*(W_p^{\mathcal{L}}(s))$ is the total number of bits produced by the encoding process just described. In particular, a *red* node u contributes $|\mathcal{C}_{01}(s^{\mathbf{01}}(u))|$ bits, while a *black* node contributes $|\mathcal{C}_\Sigma(s(u))|$ bits.

Example 2. When $\mathcal{L} = root(W_p(s))$ we compress s using $\mathcal{C}_\Sigma$ only. By Property 1(b) we have $C^*(W_p^{\mathcal{L}}(s)) \leq |s|H_0(s) + \eta|s| + \mu$. The other extreme case is when $\mathcal{L}$ consists of all the leaves of $W_p(s)$. In this case we never use $\mathcal{C}_\Sigma$ and we have $C^*(W_p^{\mathcal{L}}(s)) \leq \alpha|s|\, H_0^*(s) + \beta(|\Sigma| - 1)$ by Property 1(a) and Theorem 1. □

We note that when the algorithms $\mathcal{C}_{01}$ and $\mathcal{C}_\Sigma$ are fixed, the cost $C^*(W_p^{\mathcal{L}}(s))$ depends on two factors: the shape of the alphabetic tree T_Σ, and the leaf cover $\mathcal{L}$. The former determines the shape of the Wavelet Tree, the latter determines the assignment of $\mathcal{C}_{01}$ and $\mathcal{C}_\Sigma$ to the nodes of $W_p(s)$. It is 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 $\mathcal{L}_{\min}$ that minimizes the cost function $C^*(W_p^{\mathcal{L}}(s))$. 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 $\mathcal{L}_{\min}$ for that tree giving the minimum of the function $C_{opt}^*(W_p(s))$. That is, we are interested in finding both a shape of the Wavelet Tree, and an assignment of $\mathcal{C}_{01}$ and $\mathcal{C}_\Sigma$ 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 $C_{opt}^*(W_p(s)) \leq \min(|s|\, H_0^*(s) + \eta|s| + \mu, \alpha|s|\, H_0^*(s) + \beta(|\Sigma| - 1))$.

5.1 Optimization Algorithms (Sketch)

The first algorithm we sketch is an efficient algorithm for the solution of Problem 1. The pseudo-code is given in Figure 4. We have

[3] 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 $C_{opt}(r) \leftarrow |\mathcal{C}_{01}(s)|$ 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 $Z(u) \leftarrow \min\{|\mathcal{C}_{\Sigma}(s(u))|, |\mathcal{C}_{01}(s^{\mathbf{01}}(u))| + Z(u_L) + Z(u_R)\}$.
    (2.3) If $Z(u) = |\mathcal{C}_{\Sigma}(s(u))|$ 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. 4. The pseudocode for the linear-time computation of an optimal leaf cover $\mathcal{L}_{\min}$ for a given decomposition tree T_s.

Theorem 5. *Given two compressors satisfying Property 1 and a Wavelet Tree $W_p(s)$, the algorithm in Figure 4 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. (Sketch) The correctness of the algorithm hinges on a *decomposability property* of the cost functions associated to $\mathcal{L}_{\min}$ with respect to the subtrees of $W_p(s)$. Such a property is essentially the same used in [5, Sect. 4.5], here exploited to devise an optimal Red-Black Wavelet Tree. 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. □

Since we are assuming that the alphabet is of constant size, the algorithm of Figure 4 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. Using Dynamic Programming techniques, we can show:

Theorem 6. *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 $O(|\Sigma|^4(T_{01}(|s|) + T_{gen}(|s|))$ time.* □

References

1. J. Abel. Improvements to the Burrows-Wheeler compression algorithm: After BWT stages. http://citeseer.ist.psu.edu/abel03improvements.html.
2. Z. Arnavut and S. Magliveras. Block sorting and compression. In *DCC: Data Compression Conference*, pages 181–190. IEEE Computer Society TCC, 1997.
3. J. Bentley, D. Sleator, R. Tarjan, and V. Wei. A locally adaptive compression scheme. *Communications of the ACM*, 29(4):320–330, 1986.
4. M. Burrows and D. Wheeler. A block sorting lossless data compression algorithm. Technical Report 124, Digital Equipment Corporation, 1994.
5. P. Ferragina, R. Giancarlo, G. Manzini, and M. Sciortino. Boosting textual compression in optimal linear time. *Journal of the ACM*, 52:688–713, 2005.
6. P. Ferragina and G. Manzini. Indexing compressed text. *Journal of the ACM*, 52(4):552–581, 2005.

7. L. Foschini, R. Grossi, A. Gupta, and J. Vitter. Fast compression with a static model in high order entropy. In *DCC: Data Compression Conference*, pages 62–71. IEEE Computer Society TCC, 2004.
8. R. Grossi, A. Gupta, and J. Vitter. High-order entropy-compressed text indexes. In *Proc. 14th Annual ACM-SIAM Symp. on Discrete Algorithms (SODA '03)*, pages 841–850, 2003.
9. R. Grossi, A. Gupta, and J. Vitter. When indexing equals compression: Experiments on compressing suffix arrays and applications. In *Proc. 15th Annual ACM-SIAM Symp. on Discrete Algorithms (SODA '04)*, pages 636–645, 2004.
10. R. Grossi and J. Vitter. Compressed suffix arrays and suffix trees with applications to text indexing and string matching. *SIAM Journal on Computing*, 35:378–407, 2005.
11. G. Manzini. An analysis of the Burrows-Wheeler transform. *Journal of the ACM*, 48(3):407–430, 2001.

A Appendix: Technical Lemmas

Proof of Lemma 1: Let $n = |s|$ and let r denote the number of occurrences of the less frequent symbol of s (hence, $r \leq n/2$). For simplicity in this proof we assume that $|\mathcal{C}_{RLE}(s)| = \sum_{i=1}^{k} |\mathcal{C}_{PF}(\ell_i)| = \sum_{i=1}^{k} (a \log \ell_i + b)$, that is, we ignore the cost—one bit—of encoding a_1. Note that this additional bit is accounted for in the statement of the lemma. Moreover we define $c(x) = a \log x + b$, and we recall that we can assume $a \leq 2b$.

Case $r = 0$**.** We have $|\mathcal{C}_{RLE}(s)| = a \log n + b$. The thesis follows observing that by (1) we have $|s| H_0^*(s) \geq \log n$.

Case $1 \leq r < n/4$**.** This is the most complex case. Assume that 1 is the less frequent symbol. We start by proving that for a string s with r 1's and $n-r$ 0's $|\mathcal{C}_{RLE}(s)|$ is maximized when there are no consecutive 1's. Assume for example that there is a group of $y \geq 2$ consecutive 1's. Since $r < n/4$ there must be a group of $z \geq 4$ consecutive 0's. Then, we can rearrange the symbols so that the group of y 1's is split into a group of size $y-1$ and one singleton, and the group of z 0's is split into a group of size $z-2$ and a group of size 2. Call s' this new string. Recalling that $a \leq 2b$ we have

$$\begin{aligned} |\mathcal{C}_{RLE}(s)| - |\mathcal{C}_{RLE}(s')| &= c(y) + c(z) - [c(y-1) + c(1) + c(z-2) + c(2)] \\ &= a \log(yz) + 2b - [a \log(y-1)(z-2) + a + 4b] \\ &\leq a \log(yz) - [a \log(y-1)(z-2) + 2a] \\ &= a[\log(yz) - \log(4(y-1)(z-2))]. \end{aligned}$$

To prove our claim we observe that $\log(yz) - \log(4(y-1)(z-2)) \leq 0$ since $y \geq 2$ and $z \geq 4$ imply $y \leq 2(y-1)$ and $z \leq 2(z-2)$.

We have therefore established that $|\mathcal{C}_{RLE}(s)|$ is maximized when there are no consecutive 1's. We now observe that under this assumption the $n-r$ 0's are split into k groups with $r-1 \leq k \leq r+1$ (depending on whether the string starts/ends with 1). By the concavity of the logarithm we have

$$|\mathcal{C}_{RLE}(s)| \ \leq k\, c((n-r)/k) + rc(1) \ \leq \ ak \log(n/k) + kb + rb.$$

Using the facts that $r < n/4$ and $a \leq 2b$, one can easily prove that the function $f(k) = ak\log(n/k) + kb$ is increasing for $k \leq r+1$. Hence, we get

$$\begin{aligned}|\mathcal{C}_{RLE}(s)| &\leq a(r+1)\log(n/(r+1)) + (2r+1)b \\ &\leq 2ar\log(n/r) + (2r+1)b \\ &\leq 2\max(a,b)r(\log(n/r)+1) + b.\end{aligned}$$

The thesis follows since $|s|\ H_0^*(s) = r\log(n/r) + (n-r)\log(n/(n-r)) \leq r(\log(n/r)+1)$.

Case $n/4 \leq r \leq n/2$**.** Assume for example $s = 0^{\ell_1}1^{\ell_2}\cdots 0^{\ell_k}$. We have

$$|\mathcal{C}_{RLE}(s)| = \sum_{i=1}^{k} c(\ell_i), \qquad \text{where} \qquad \sum_{i=1}^{k} \ell_i = n.$$

By the concavity of the logarithm and the fact that $k \leq n$ we get

$$|\mathcal{C}_{RLE}(s)| = a\sum_{i=1}^{k}\log(\ell_i) + kb \leq ak\log(n/k) + nb.$$

The function $k\log(n/k)$ has its maximum for $k = n/e$ hence

$$|\mathcal{C}_{RLE}(s)| \leq a(n/e)\log e + nb \leq n\max(a,b)[(\log e)/e + 1].$$

Since $H_0^*(s) \geq \mathcal{H}(1/4) = -(1/4)\log(1/4) - (3/4)\log(3/4)$, the thesis follows observing that

$$\frac{[(\log e)/e + 1]}{\mathcal{H}(1/4)} = 1.88\ldots < 2.$$

□

Proof of Lemma 2: Let $n = |s|$. We have

$$|\mathcal{C}_{Gap}(s)| = \sum_{i=1}^{r}(a\log(g_i) + b) = a + \left\lfloor \sum_{i=1}^{r}\log(g_i) \right\rfloor + rb.$$

Since $\sum_{i=1}^{r} g_i \leq n$, by the concavity of the logarithm we have

$$|\mathcal{C}_{Gap}(s)| \leq ar\log(n/r) + rb \leq \max(a,b)r(\log(n/r)+1).$$

The thesis follows since $r \leq n/2$ implies $|s|\,H_0(s) \leq r(\log(n/r)+1)$. □

Proof of Lemma 3: Let u be an internal node of $W_f(\alpha)$ and let $\Sigma^{(u)}$ denote the set of symbols associated to node u. If the string α_i, for $i = 1, 2$, contains at least one of the symbols in $\Sigma^{(u)}$ then $W_f(\alpha_i)$ 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 α_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 $W_f(\alpha_1)$. Hence, $C^*(u) = C^*(u_1)$. Similarly, if the symbols of $\Sigma^{(u)}$ appear only in α_2 we have $C^*(u) = C^*(u_2)$.

Assume now that u_1 and u_2 both exist, and define the cost function $c(x) = a \log x + b$ in which $a \leq b + 1$ (see comment at the beginning of Sect. 3). 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 $\ell_1, \ell_2, \ldots, \ell_k$ are the run lengths of $s^{\mathbf{01}}(u)$, there exist an index j, $1 \leq j \leq k$, and a value δ, $0 \leq \delta < \ell_j$, such that $C^*(u) = 1 + \sum_{i=1}^{k} c(\ell_i)$, and $C^*(u_1) = 1 + \sum_{i=1}^{j-1} c(\ell_i) + c(\delta)$ and $C^*(u_2) = 1 + c(\ell_j - \delta) + \sum_{i=j+1}^{k} c(\ell_i)$ (recall that we need one bit to determine the first symbol of each string). Using elementary calculus and the fact that $a \leq b + 1$, we can easily prove that $c(\ell_j) \leq c(\delta) + c(\ell_j - \delta)$. Hence $C^*(u) \leq C^*(u_1) + C^*(u_2)$ and the lemma follows. □

Proof of Lemma 4: We observe that an internal node $u \in W_f(\beta)$ is pruned if and only if $s^{\mathbf{01}}(u)$ contains only 0's or only 1's. Hence, u contributes to the difference $C^*(W_f(\beta)) - C^*(W_p(\beta))$ by an amount $1 + \mathcal{C}_{PF}(|s^{\mathbf{01}}(u)|)$. The lemma follows since $\mathcal{C}_{PF}(|s^{\mathbf{01}}(u)|) \leq a \log |\beta| + b$ and the number of pruned nodes is at most $|\Sigma| - 1$. □