Move-to-Front, Distance Coding, and Inversion Frequencies Revisited*

Travis Gagie[†] Giovanni Manzini[†]

Abstract

Move-to-Front, Distance Coding and Inversion Frequencies are three simple and effective techniques used to process the output of the Burrows-Wheeler Transform. In this paper we provide the first complete comparative analyses of these techniques, establishing upper and lower bounds on their compression ratios.

We describe simple variants of these three techniques that compress any string up to a constant factor of its kth-order empirical entropy for any $k \geq 0$. At the same time we prove lower bounds for the compression of arbitrary strings which show that these variants are nearly optimal. The bounds we establish are "entropy-only" bounds in the sense that they do not involve non-constant overheads.

Our analyses provide new insights into the inner workings of these techniques, partially explain their good behavior in practice, and suggest strategies for improving their performance.

1 Introduction

Burrows-Wheeler compression [7] is important in itself and as a key component of compressed full-text indices [24]. It is therefore not surprising that this topic has received a great deal of attention (see [15] and references therein). Despite more than ten years of investigation, however, some important questions remain open. For example, although it is now well understood why the Burrows-Wheeler Transform helps compression, it is still unclear which is the best way to process the output of this transformation. In the original Burrows-Wheeler compression algorithm [7] the output of the Burrows-Wheeler Transform is processed by Move-to-Front encoding [5, 25] followed by a 0th-order encoder. Extensive experimental work has investigated the role and usefulness of these two steps and several researchers have proposed variants of this basic scheme [1, 2, 3, 4, 6, 9, 11]. Unfortunately, these variants mostly rely on clever heuristics to improve the compression of "typical" strings and usually defy theoretical analysis. More recently, some researchers have devised new tools for Burrows-Wheeler compression, namely Wavelet Trees [13, 16, 22] and Compression Boosting [14, 18]. Although these new approaches have nice theoretical properties and guaranteed compression bounds, so far their behavior in practice does not appear to be substantially superior to the simpler strategies based on Move-to-Front and 0th-order encoding [12].

Given this state of affairs, it is natural to further investigate the simple and effective techniques like Move-to-Front with the twofold objective of gaining greater insight into their inner workings and establishing entropy bounds on their compression performance. Recently, [19] has provided a simple and elegant analysis of the original Burrows-Wheeler compression algorithm showing that, for any string s, its output size is upper bounded by $\mu|s|H_k(s) + O(|s|)$ bits for any $\mu > 1$ and $k \geq 0$, where $H_k(s)$ is the kth-order empirical entropy of s. [19] also analyzes the compressor in which Move-to-Front is replaced by Distance Coding [6, 9] and proves that it produces an output bounded by $1.7286|s|H_k(s) + O(\log |s|)$ bits. These bounds provide an important theoretical complement to the good practical behavior of these techniques. However, the presence of the terms $O(|s|)$ and $O(\log |s|)$ makes it difficult to evaluate how

*Partially supported by Italian MIUR Italy-Israel FIRB Project "Pattern Discovery Algorithms in Discrete Structures, with Applications to Bioinformatics". A preliminary version of this work has appeared in the *Proceedings of the 18th Symposium on Combinatorial Pattern Matching (CPM '07)*.

[†]Department of Computer Science, University of Eastern Piedmont, Italy. {travis,manzini}@mfn.unipmn.it.

close the compression ratio is to the entropy of the input, especially when s is highly compressible. For example, for $s = \sigma_1\sigma_2^n$, for $k \geq 0$ it is $|s|H_k(s) = O(\log|s|)$ so a bound of the form $\mu|s|H_k(s) + O(|s|)$ tells one nothing about how close the compression ratio is to the entropy of the input string.

The above observation suggests that it is worthwhile to consider *entropy-only* bounds, that is, bounds of the form $\lambda|s|H_k^*(s) + O(1)$, where $\lambda > 1$ is a constant independent of k, $|s|$, and of the alphabet size. Note that entropy-only bounds are expressed in terms of the *modified kth-order entropy* H_k^* since they cannot be established in terms of H_k (see Section 2). 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. Note that not every compression algorithm can achieve such bounds since many compressors have non-constant overheads that become non-negligible when the input string is highly compressible. Indeed, the capability of achieving entropy-only bounds is one of the features that differentiate Burrows-Wheeler compression algorithms from the family of Lempel-Ziv compressors [23].

In this paper we analyze Move-to-Front (Mtf), Distance Coding (Dc), and Inversion Frequencies Coding (If) [2, 3] and we study their effectiveness in compressing the output of the Burrows-Wheeler transform (bwt from now on). Our main results can be summarized as follows:

1. The procedures Mtf, Dc and If all output sequences of positive integers. These sequences are usually encoded using either a 0th-order encoder or a prefix-free integer encoder; in this paper we establish upper bounds for both options without making assumptions on the inner working of the final encoder. To this end we extend a technique introduced in [19] for the analysis of 0th-order encoders in terms of integer coders (Lemma 3.3); this extension may be of independent interest.
2. We provide the first theoretical analysis of If when used to compress the output of the bwt (Theorem 6.3).
3. We describe simple variants of Mtf, Dc, and If achieving entropy-only bounds (Corollaries 4.5, 5.9, and 6.6). The variant of Mtf simply uses Run-Length Encoding (Rle), while the variants of Dc and If make use of a novel "escape and re-enter" technique.
4. Our best entropy-only bound holds for a variant of Dc that compresses every string s into at most $(2.69 + C_0)|s|H_k^*(s) + \log|s| + \Theta(1)$ bits for any $k \geq 0$, where C_0 is the per symbol overhead of the 0th-order encoder. We prove (Theorem 7.1) that no compression algorithm (not necessarily based on the bwt) can achieve an entropy-only bound of the form $\lambda|s|H_0^*(s) + \Theta(1)$ for a constant $\lambda < 2$. In addition, we prove that, under the mild assumption that concatenations of encoded strings are uniquely decodable, even $\lambda = 2$ is not achievable (Theorem 7.2).

Comparison with Related results. The first entropy-only bound for Mtf as a post-processor of the bwt has been established in [23]. With a rather complex analysis [23] shows that the compression achieved by the bwt followed by Mtf, Rle, and a 0th-order encoder is bounded by $(5+3C_0)|s|H_k^*(s) + \log|s| + \Theta(1)$ bits for any string s and for any $k \geq 0$ (C_0 is again the per symbol overhead of the 0th-order encoder). In this paper we consider a slightly different version of Rle for which we establish a bound of the same form with the constant in front of $|s|H_k^*(s)$ reduced to $(4.4 + C_0)$. Our analysis is simpler than the one in [23] and provides upper bounds also for the case in which the 0th-order encoder is replaced by a prefix-free integer encoder.

In [19] the authors provide the first analysis of Dc combined with the bwt and a 0th-order encoder. They show that the output of this compressor is bounded by $1.7286|s|H_k(s) + \Theta(\log|s|)$ bits. This bound holds only if the 0th-order encoder is an "ideal" version of Arithmetic Coding for which the overhead per symbol is $(\log|s|)/|s|$. Using the techniques of this paper it is possible to refine the analysis of [19] and prove that for a 0th-order encoder with a constant per symbol overhead C_0 the output of Dc is bounded by $(1.7286 + C_0)|s|H_k(s) + \Theta(\log|s|)$ bits. However, this is not an entropy-only bound because of the $\Theta(\log|s|)$ term (no tight bounds are known on the size of the constant hidden in the asymptotic notation; from the analysis in [19] it follows that it is at most h^{k+1}, where h is the alphabet size).

No bounds of any kind were previously known for the compression achieved by lf when used to process the bwt. The only known bound for lf was the one given in [13] which applies to lf used as a stand-alone compressor. Finally, the lower bounds proven in Section 7 complement previous ones established in [18, 19, 20] but are not directly comparable to them: the bounds in [19] are not expressed in terms of the empirical entropy and the bounds in [18, 20] apply only to bwt-based compressors.

For the convenience of the reader, we have confined to Appendix B the proofs of some lemmas that are purely technical and not related to the inner workings of the algorithms being considered.

2 Notation and Background

Let s be a string drawn from the alphabet $\Sigma = \{\sigma_1, \ldots, \sigma_h\}$. For $i = 1, \ldots, |s|$ we write $s[i]$ to denote the ith character of s. For each $\sigma_i \in \Sigma$, let n_i be the number of occurrences of σ_i in s. The 0*th-order empirical entropy* of the string s is defined as[1] $H_0(s) = -\sum_{i=1}^{h}(n_i/|s|)\log(n_i/|s|)$. It is well known that H_0 is the maximum compression we can achieve using a fixed codeword for each alphabet symbol. The following definition captures the abstract notion of a compressor which is able to achieve H_0 up to a constant overhead per symbol and an additional overhead depending on the alphabet size.

Definition 1 *An algorithm* A *is a 0th-order algorithm if for any input string* s *we have*

$$|\mathsf{A}(s)| \le |s|H_0(s) + C_0|s| + O(h \log h)$$

where $h = |\Sigma|$*. The parameter* C_0 *is the per symbol overhead of* A*.* ∎

Examples of 0th-order algorithms are Huffman coding, for which $C_0 = 1$, and Arithmetic coding, for which the overhead per symbol can in principle be made arbitrarily small (for typical implementations it is $C_0 \approx .01$). It is well known that we can often achieve a compression ratio better than $H_0(s)$ if the codeword we use for each symbol depends on the k symbols preceding it. In this case, the maximum compression is bounded from below by the kth-order entropy $H_k(s)$ (see [23] for a full discussion or Appendix A for a summary).

In [19] the authors analyze the original Burrows-Wheeler compressor in which the output of the bwt is processed by Mtf followed by a 0th-order algorithm and they prove that its output is bounded by

$$\mu|s|H_k(s) + (\log(\zeta(\mu)) + C_0)|s| + \log|s| + O\Big(h^{k+1}\log h\Big) \tag{1}$$

bits, where $\zeta(\mu) = \sum_{j>0} j^{-\mu}$ is the Riemann zeta function and C_0 is the per symbol overhead of the 0th-order algorithm. The above bound holds simultaneously for any $\mu > 1$ and $k \ge 0$. This means we can get close to the kth-order entropy for any $k \ge 0$. Unfortunately, in (1) there is also a $\Theta(|s|)$ term which becomes dominant when s is highly compressible. For example, for $s = \sigma_1\sigma_2^n$ we have $|s|H_0(s) = \log|s| + O(1)$. In this case, the bound (1) does not guarantee that the compression ratio is within a constant factor of the entropy.

In order to get significant bounds for highly compressible strings as well, it would be desirable to prove entropy-only bounds of the form $\lambda|s|H_k(s) + \Theta(1)$; unfortunately, such bounds cannot be established. To see this, consider the family of strings $s = \sigma_1^n$; we have $|s|H_0(s) = 0$ for all of them and we cannot hope to compress all strings in this family in $\Theta(1)$ space. For that reason, [23] introduced the notion of 0*th-order modified empirical* entropy:

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

[1] In the following, log means $\log_2$ and ln denotes the natural logarithm. We assume $0\log 0 = 0$.

Note that if $|s| > 0$, $|s|H_0^*(s)$ is at least equal to the number of bits needed to write down the length of s in binary. The *kth-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 (again, see [23] for a full discussion or Appendix A for a summary). An entropy-only bound in terms of H_k^* is proven in [23] for the algorithm consisting of the bwt, followed by Mtf and Rle encoding, followed by 0th-order encoding. A key tool for the analysis in [23] is the notion of local optimality.

Definition 2 *A compression algorithm* A *is* locally λ-optimal *if there exists a constant* c_h *such that for any string* s *and for any partition* $s_1 s_2 \cdots s_t$ *of* s *we have*

$$\mathsf{A}(s) \leq \lambda \left[\sum_{i=1}^{t} |s_i| H_0^*(s_i) \right] + \; c_h t, \tag{3}$$

where c_h *depends only on the alphabet size* h*. If the bound* (3) *holds with a parameter* λ_h *that depends on the alphabet size* h*, we say that the algorithm* A *is* locally pseudo optimal*.* ■

The importance of local optimality stems from the following lemma which establishes that processing the output of the bwt with a locally optimal algorithm yields an algorithm achieving an entropy-only bound.

Lemma 2.1 ([23]) *If* A *is locally* λ*-optimal then the bound*

$$|\mathsf{A}(\mathsf{bwt}(s))| \leq \lambda |s| H_k^*(s) + \log|s| + c_h h^k \tag{4}$$

holds simultaneously for any $k \geq 0$*.* ■

Note that the term $\log|s|$ in (4) is due to the fact that $\mathsf{bwt}(s)$ consists of a permutation of s—which is compressed using A—and an integer in $[1, |s|]$ whose encoding takes $1 + \lfloor \log|s| \rfloor$ bits. Since $|s|H_k^*(s) \geq \log(|s| - k)$ (see Lemma A.1), we could rewrite the right-hand side of (4) as $(\lambda + 1)|s|H_k^*(s) + c_h h^k$ (this justifies the expression 'entropy-only bound'). However, since for many strings it is $\log|s| \ll |s|H_k^*(s)$, keeping the term $\log|s|$ explicit provides a better picture of the performance of bwt-based compressors.

We conclude this section with two lemmas relating the order zero entropy of a string to its length and the number of runs in it. Given a string s, a run is a substring $s[i]s[i+1]\cdots s[i+k]$ of identical symbols, and a maximal run is a run which cannot be extended; that is, it is not a proper substring of a larger run.

Lemma 2.2 ([21, Sect. 3]) *The number of maximal runs in a string* s *is bounded from above by* $1 + |s|H_0(s)$*.* ■

Lemma 2.3 *Let* s *be a string containing* $\mathsf{runs}(s)$ *maximal runs and let* α*,* β *and* ϵ *be positive constants; then*

$$\alpha \log|s| + \beta \mathsf{runs}(s) \leq \max(\alpha, \beta + \epsilon)|s|H_0^*(s) + O(1).$$

Proof: See Appendix B. ■

3 Integer and 0th-order encoders

Move-to-Front, Distance Coding, and Inversion Frequencies all output sequences of positive integers. These sequences are usually compressed using either a 0th-order encoder (see Definition 1) or a prefix-free encoding for the integers. Prefix-free encoders of the integers use a fixed codeword for each integer regardless of its frequency and are therefore faster and easier to implement. 0th-order encoders (especially arithmetic coders) are slower but usually achieve a significantly better compression. Unfortunately, they are also more difficult to analyze when used in connection with the bwt. In this section we show that the compression achieved by a generic 0th-order encoder can be bounded in terms of the best compression achieved by a *family* of integer coders. This result will make it possible to translate compression bounds for integer coders into compression bounds for 0th-order encoders. In the following we denote by Enc a uniquely decodable encoder of the positive integers (not necessarily prefix-free). For any $i \geq 1$ we denote by $\mathsf{Enc}(i)$ the fixed codeword encoding the integer i. Note that we admit also "ideal" coders in which the codewords have fractional lengths. Our only assumption is that there exist two positive constants a and b such that for any $i \geq 1$ we have $|\mathsf{Enc}(i)| \leq a \log i + b$. For example, for γ-coding [10] the above inequality holds for $a = 2$ and $b = 1$. In our analysis we will often make use of the following property.

Lemma 3.1 (Subadditivity) *Let a, b be two constants such that for $i \geq 1$ it is $|\mathsf{Enc}(i)| \leq a \log i + b$. Then, there exists a constant d_{ab} such that for any sequence of positive integers $x_1, x_2, \ldots, x_k$ we have*

$$\left|\mathsf{Enc}\Big(\sum\nolimits_{j=1}^{k} x_j\Big)\right| \;\leq\; \Big(\sum\nolimits_{j=1}^{k} |\mathsf{Enc}(x_j)|\Big) \;+\; d_{ab}.$$

Proof: See Appendix B. ∎

The next lemma, which follows from the analysis in [19], establishes a connection between integer and 0th-order coders by showing that if we feed a sequence of integers to a 0th-order encoder the output is essentially no larger than the output produced by an ideal integer encoder with parameters $a = \mu$ and $b = \log(\zeta(\mu)) + C_0$ for any $\mu > 1$.

Lemma 3.2 *Let $\mathsf{Order0}$ be a 0th-order encoder with per character overhead C_0 and let $x_1 x_2 \cdots x_n$ be a sequence of integers such that $1 \leq x_i \leq h$ for $i = 1, \ldots, n$. Then, for any $\mu > 1$ we have*

$$|\mathsf{Order0}(x_1 x_2 \cdots x_n)| \leq \sum\nolimits_{i=1}^{n} \Big(\mu \log(x_i) + \log \zeta(\mu) + C_0\Big) \;+\; O(h \log h)\,.$$

Proof: For any $\mu > 1$, consider the probability distribution over the positive integers defined by $q(j) = (\zeta(\mu)\, j^{\mu})^{-1}$. By the definition of Riemann zeta function it is $\zeta(\mu) = \sum_{j>0} j^{-\mu}$ hence $\sum_{j>0} q(j) = 1$. By Gibb's inequality it is $nH_0(x_1 \cdots x_n) \leq -\sum_{i=1}^{n} \log(q(x_i))$. By Definition 1 we get

$$\begin{aligned}
|\mathsf{Order0}(x_1 x_2 \cdots x_n)| &\leq nH_0(x_1 \cdots x_n) + nC_0 + O(h \log h) \\
&\leq -\sum\nolimits_{i=1}^{n} \log(q(x_i)) + nC_0 + O(h \log h) \\
&\leq \sum\nolimits_{i=1}^{n} (\mu \log(x_i) + \log \zeta(\mu) + C_0) + O(h \log h)\,.
\end{aligned}$$

∎

In the following we will also make use of a compression algorithm that combines the advantages of integer and 0th-order encoders. The reason for introducing a new algorithm is that many of the procedures considered in this paper produce sequences of positive integers whose magnitude can be as large as the length of the input string s. This is not a problem when we compress such sequences using integer encoders since, by definition, such encoders handle arbitrarily large integers. Unfortunately, large integers can be a problem for 0th-order encoders as typical 0th-order algorithms have an overhead of $O(h \log h)$ bits, where h is the size of the input alphabet (see Definition 1). If we need to encode values as large as $|s|$ such overhead could make the use of the encoder unprofitable. For example, the run-length encoding of the string $s = 1010^2 10^3 10^4 1 \cdots 10^k 1$ produces $\Theta(\sqrt{|s|})$ distinct integers: if we encode these integers with a 0th-order algorithm, the overhead deriving from the alphabet size would be much larger than $|s| H_0(s)$.

Researchers are well aware of this phenomenon and circumvent it by using a 0th-order algorithm for encoding "small" integers and ad-hoc techniques for handling the (usually few) occurrences of large integers. We now describe one such scheme based on δ-coding and show it is equivalent to an ideal integer coder with parameters $a = \mu$ and $b = \log(\zeta(\mu)) + C_0 + \nu$ for any constants $\mu > 1$ and $\nu > 0$. For descriptions of more sophisticated techniques, see [26] and references therein.

Recall that δ-coding [10] is a prefix-free encoding of the integers such that for any $x \geq 1$ it is $|\delta(x)| \leq 1 + \log x + 2\log(1 + \log x)$. Hence, for any $\mu > 1$ and $\nu > 0$ we can find an integer t such that for $x \geq t$ it is

$$|\delta(x)| \leq \mu\log(x) + \log\zeta(\mu) + \nu - \log(2^\nu/(2^\nu - 1)). \tag{5}$$

Given an encoder $\mathsf{Order0}$ with per symbol overhead C_0 we define a new encoder $\mathsf{Order0*}$ that uses δ-coding for encoding the integers larger than t. Given the sequence $x_1 \cdots x_n$ let $y_1 \cdots y_n$ denote the sequence with all integers greater than t replaced by copies of t and let $z_1 \cdots z_\ell$ be all the integers at least t. To encode $x_1 \cdots x_n$ the algorithm $\mathsf{Order0*}$ encodes $y_1 \cdots y_n$ with $\mathsf{Order0}$ and $z_1 \cdots z_\ell$ with δ-coding.

Lemma 3.3 *Let* $\mathsf{Order0}$ *be an order zero encoder with per symbol overhead* C_0*. For any sequence of positive integers* $x_1 x_2 \cdots x_n$ *and constants* $\mu > 1$ *and* $\nu > 0$ *we have*

$$|\mathsf{Order0*}(x_1 x_2 \cdots x_n)| \leq \sum\nolimits_{i=1}^{n} \Big(\mu\log(x_i) + \log\zeta(\mu) + \nu + C_0\Big) + O(1)\,.$$

Proof: Note that the sequence $y_1 \cdots y_n$ contains only integers between 1 and t. Assign to each integer in this range the weight $q(\cdot)$ defined by $q(j) = (2^\nu\zeta(\mu)j^\mu)^{-1}$ for $j = 1, \ldots, t-1$, and $q(t) = 1 - 2^{-\nu}$. Since $\sum_{j=1}^{t} q(j) \leq 1$, we have $nH_0(y_1 \cdots y_n) \leq -\sum_{i=1}^{n} \log(q(y_i))$. By Definition 1,

$$\begin{aligned}
|\mathsf{Order0*}(x_1 \cdots x_n)| &= \sum_{i=1}^{\ell} |\delta(z_i)| \;+\; \mathsf{Order0}(y_1 \cdots y_n)\\
&\leq \sum_{i=1}^{\ell} |\delta(z_i)| \;+\; nH_0(y_1 \cdots y_n) \;+\; nC_0 \;+\; O(t\log t)\\
&\leq \sum_{x_i \geq t} |\delta(x_i)| \;-\; \sum_{i=1}^{n} \log(q(y_i)) \;+\; nC_0 \;+\; O(t\log t)\\
&\leq \sum_{x_i \geq t} \Big(\mu\log x_i + \log\zeta(\mu) + \nu - \log(2^\nu/(2^\nu - 1))\Big) \;+\\
&\quad\; \sum_{x_i < t} \Big(\mu\log x_i + \log\zeta(\mu) + \nu\Big) \;-\; \sum_{x_i \geq t} \log(q(t)) \;+\; nC_0 \;+\; O(t\log t)\,.
\end{aligned}$$

Observing that $\log(q(t)) = -\log(2^\nu/(2^\nu - 1))$, we conclude that

$$|\mathsf{Order0*}(x_1 \cdots x_n)| \leq \sum_{i=1}^{n} \Big(\mu\log x_i + \log\zeta(\mu) + \nu + C_0\Big) \;+\; O(t\log t)\,.$$

The thesis follows since t depends only on the constants μ and ν. ∎

4 Analysis of Move-to-Front encoding

The Move-to-Front (Mtf) procedure encodes a string by replacing each symbol with the number of distinct symbols seen since its last occurrence plus one. To this end, Mtf maintains a list of the symbols ordered by recency of occurrence; when the next symbol arrives the encoder outputs its current rank and moves it to the front of the list. If the input string is defined over the alphabet Σ we assume that ranks are in the range $[1, h]$, where $h = |\Sigma|$. To completely determine the encoding procedure we must specify the initial status of the recency list. However, changing the initial status increases the output size by at most

$O(h \log h)$ bits so we will add this overhead and ignore the issue. Let Enc denote an integer coder such that $|\mathsf{Enc}(i)| \le a \log i + b$ and let $\mathsf{Mtf}+\mathsf{Enc}$ denote the algorithm in which the ranks produced by Mtf are encoded using Enc. From the analysis in [5] it follows that for any string s we have $|\mathsf{Enc}(\mathsf{Mtf}(s))| \le a|s|H_0(s)+b|s|+O(h \log h)$. In addition, if $\mathsf{Order0}$ is a 0th-order compressor with per character overhead C_0, Lemma 3.2 implies that $|\mathsf{Order0}(\mathsf{Mtf}(s))| \le \mu|s|H_0(s) + (\log \zeta(\mu) + C_0)|s| + O(h \log h)$ for any $\mu > 1$. Unfortunately, the following example shows that $\mathsf{Mtf} + \mathsf{Enc}$ is not powerful enough to achieve entropy-only bounds.

Example 1 Fix an integer coder Enc and let ℓ denote the length of the shortest codeword produced by Enc. Let $s = \sigma_1^n$. Since $|\mathsf{bwt}(s)| = |s|$, we have $|\mathsf{Enc}(\mathsf{Mtf}(\mathsf{bwt}(s)))| \ge \ell|s|$. Since $|s|H_0^*(s) = 1 + \lfloor \log |s| \rfloor$ it follows that the combined algorithm $\mathsf{bwt} + \mathsf{Mtf} + \mathsf{Enc}$ cannot achieve an entropy-only bound that holds for every possible input string. ∎

The above example shows that if we feed to the final encoder $\Theta(|s|)$ symbols it is unlikely we can achieve an entropy-only bound. This observation suggests the algorithm $\mathsf{Mtf\_rle}$ that combines Mtf with Rle. Assume $\sigma = s[i+1]$ is the next symbol to be encoded. Instead of simply encoding the Mtf rank r of σ, $\mathsf{Mtf\_rle}$ finds the maximal run $s[i+1] \cdots s[i+\ell]$ of consecutive occurrences of σ and encodes the pair[2] $\langle r, \ell \rangle$. We define the algorithm $\mathsf{Mtf\_rle} + \mathsf{Enc}$ as the algorithm which encodes each such pair with Enc. Since the Mtf rank r is always greater than one, to save space we encode each pair as follows: If $\ell = 1$, we encode $\langle r, \ell \rangle$ with the codewords $\langle \mathsf{Enc}(1), \mathsf{Enc}(r) \rangle$, while if $\ell > 1$ we encode $\langle r, \ell \rangle$ with the codewords $\langle \mathsf{Enc}(r), \mathsf{Enc}(\ell - 1) \rangle$.

Lemma 4.1 *Let* $\mathsf{A}_0 = \mathsf{Mtf\_rle} + \mathsf{Enc}$. *For any string s we have*

$$|\mathsf{A}_0(s)| \le 2a|s|H_0^*(s) + a \log \ell + (2b - a)\mathsf{runs}(s) + O(h \log h)\,.$$

where $\mathsf{runs}(s)$ *is the number of runs in s, and ℓ is the length of the last run.*

Proof: Assume $H_0(s) \neq 0$ (otherwise $s = \sigma^n$ and the proof follows by an easy computation). Let $\langle r_1, \ell_1 \rangle$, $\langle r_2, \ell_2 \rangle$, ..., $\langle r_t, \ell_t \rangle$ denote the set of pairs generated by $\mathsf{Mtf\_rle}$. Because of the way A_0 encodes the pairs $\langle r_j, \ell_j \rangle$, if we define $|\mathsf{Enc}(0)|$ to be equal to $|\mathsf{Enc}(1)|$ the encoding of each pair $\langle r_j, \ell_j \rangle$ takes precisely $|\mathsf{Enc}(r_j)| + |\mathsf{Enc}(\ell_j - 1)|$ bits. Hence, we can write

$$|\mathsf{A}_0(s)| = \sum_{j=1}^{t} (|\mathsf{Enc}(r_j)| + |\mathsf{Enc}(\ell_j - 1)|) \;+\; O(h \log h)\,.$$

To bound $|\mathsf{A}_0(s)|$ we charge each term in the above summation to a character $\sigma \in \Sigma$ as follows: we charge the term $|\mathsf{Enc}(r_j)|$ to the character forming the jth run and the term $|\mathsf{Enc}(\ell_j - 1)|$ to the character forming the $j+1$-st run. Note that this leaves out the last run length ℓ_t: its corresponding cost $|\mathsf{Enc}(\ell_t - 1)|$ is accounted for explicitly in the statement of the lemma.

For any given character σ let (α_1, β_1), (α_2, β_2), ..., (α_k, β_k) denote the starting and ending positions of the runs of σ. For $i = 1, \ldots, k$ let $\langle r'_i, \ell'_i \rangle$ denote the pair encoding the run (α_i, β_i) (so we have $\ell'_i = \beta_i - \alpha_i + 1$). Finally, let m_i denote the length of the run immediately preceding the run (α_i, β_i). The total cost charged to σ is therefore

$$\sum_{i=1}^{k} \Big(|\mathsf{Enc}(r'_i)| + |\mathsf{Enc}(m_i - 1)| \Big)\,. \tag{6}$$

Define $\beta_0 = 0$. We now show that for $i \ge 1$ we have

$$|\mathsf{Enc}(r'_i)| + |\mathsf{Enc}(m_i - 1)| \le 2 \log(\alpha_i - \beta_{i-1}) + 2b - a. \tag{7}$$

[2]Here and in the following we use angle brackets to show that certain values form a pair or a triple with a particular meaning: such brackets are not part of the output.

Assume first $m_i > 1$. Recall r'_i is the number of distinct characters in the substring from $s[\beta_{i-1}+1]$ to $s[\alpha_i]$. If, immediately before $s[\alpha_i]$, there is a run of m_i equal symbols, we have $r'_i \le \alpha_i - \beta_{i-1} - (m_i - 1)$. Hence

$$\begin{aligned} |\mathsf{Enc}(r'_i)| + |\mathsf{Enc}(m_i - 1)| &= a(\log(r'_i) + \log(m_i - 1)) + 2b \\ &\le 2a\log((r'_i + m_i - 1)/2) + 2b \\ &\le 2a\log(\alpha_i - \beta_{i-1}) + 2b - a. \end{aligned}$$

If $m_i = 1$, then $|\mathsf{Enc}(m_i - 1)| = b$. Since $2 \le r'_i \le \alpha_i - \beta_{i-1}$, we have

$$\begin{aligned} |\mathsf{Enc}(r'_i)| + |\mathsf{Enc}(m_i - 1)| &= a\log(r'_i) + 2b \\ &\le 2a\log(\alpha_i - \beta_{i-1}) + 2b - a \end{aligned}$$

thus establishing (7). Using (7), the total cost (6) charged to σ can be bounded by

$$2a\left[\log(\alpha_1 - \beta_0) + \log(\alpha_2 - \beta_1) + \cdots + \log(\alpha_k - \beta_{k-1})\right] + k(2b - a) \tag{8}$$

bits. Summing the cost $k(2b-a)$ over all characters in Σ yields a total of $(2b-a)\,\mathsf{runs}(s)$ bits. To complete the proof we bound the content of the square brackets in (8). Since $\log(1) = 0$, the content of the square brackets is equal to

$$\log(\alpha_1 - \beta_0) + \cdots + \log(\alpha_k - \beta_{k-1}) + (\beta_1 - \alpha_1 + \beta_2 - \alpha_2 + \cdots + \beta_k - \alpha_k)\log(1). \tag{9}$$

The sum of the coefficients of the logarithms in (9) is $k + \sum_{i=1}^{k}(\beta_i - \alpha_i) = \sum_{i=1}^{k}(\beta_i - \alpha_i + 1)$ which is equal to the number n_σ of occurrences of σ in s. Hence, by Jensen's inequality, (9) is bounded by

$$n_\sigma \log\left(\frac{(\alpha_1 - \beta_0) + \cdots + (\alpha_k - \beta_{k-1}) + (\beta_1 - \alpha_1 + \cdots + \beta_k - \alpha_k)}{n_\sigma}\right) = n_\sigma \log((\beta_k - \beta_0)/n_\sigma)$$

which is at most $n_\sigma \log(|s|/n_\sigma)$. Summing $n_\sigma \log(|s|/n_\sigma)$ over all σ's yields $|s|H_0(s)$ and the lemma follows. ■

Since the length of the last run is bounded by $|s|$, combining Lemmas 4.1 and 2.3 we get

Corollary 4.2 *Let* $\mathsf{A}_0 = \mathsf{Mtf\_rle} + \mathsf{Enc}$. *For any string s and $\epsilon > 0$ we have*

$$|\mathsf{A}_0(s)| \le \max(3a, a + 2b + \epsilon)|s|H_0^*(s) + O(h\log h)\,. \quad ■$$

Theorem 4.3 *The algorithm* $\mathsf{A}_0 = \mathsf{Mtf\_rle} + \mathsf{Enc}$ *is locally* $\max(3a, a + 2b + \epsilon)$*-optimal for any* $\epsilon > 0$.

Proof: By Corollary 4.2 it suffices to prove that

$$|\mathsf{A}_0(s_1 s_2)| \le |\mathsf{A}_0(s_1)| + |\mathsf{A}_0(s_2)| + O(h\log h).$$

To prove this inequality observe that compressing s_2 independently of s_1 changes the encoding of the Mtf rank of only the first occurrence of each character in s_2. This gives an $O(h\log h)$ overhead. In addition, there could be a run of equal characters crossing the boundary between s_1 and s_2. In this case the length of the first part of the run will be encoded in s_1 and the length of the second part in s_2. By Lemma 3.1 this produces an $O(1)$ overhead and the theorem follows. ■

Note that combining the above theorem with Lemma 2.1 we immediately get an entropy-only bound for bwt + Mtf_rle + Enc with parameter $\lambda = \max(3a, a + 2b + \epsilon)$. In addition, using Lemma 3.3, we can extend Theorem 4.3 to the case in which the output of Mtf_rle is compressed with the algorithm Order0* described at the end of Section 3.

Procedure Distance Coding

1. Write the first character in s;
2. For each other character $\sigma \in \Sigma$, write the distance to the first σ in s, or 1 if σ does not occur (notice no distance is 1, because we do not reconsider the first character in s);
3. For each maximal run of a character σ, write the distance from the ending position of that run to the starting position of the next run of σ's, or 1 if there are no more σ's (again, no distance is 1);
4. Encode the length ℓ of the last run in s.

Figure 1: Distance coding of a string s over the alphabet $\Sigma = \{\sigma_1, \dots, \sigma_h\}$.

Theorem 4.4 *The algorithm* $\mathsf{Mtf\_rle} + \mathsf{Order0^*}$ *is locally* $(4.40 + C_0)$*-optimal.*

Proof: By Lemma 3.3 we know that for any $\mu > 1$ and $\nu > 0$ the output of $\mathsf{Order0^*}$ on input $\mathsf{Mtf\_rle}(s)$ is bounded by the output of an integer coder with parameters $a = \mu$ and $b = \log(\zeta(\mu)) + \nu + C_0$. The thesis follows by Theorem 4.3 taking $\mu = 22/15$ and $\nu = \epsilon = 0.001$. ■

Corollary 4.5 *For any string* s *and* $k \geq 0$ *we have*

$$|\mathsf{Order0^*}(\mathsf{Mtf\_rle}(\mathsf{bwt}(s)))| \leq (4.40 + C_0)|s|H_k^*(s) + \log|s| + O\Big(h^{k+1}\log h\Big).$$

Proof: Immediate by Theorem 4.4 and Lemma 2.1. ■

5 Analysis of Distance Coding

Distance Coding (Dc) is an encoding procedure which is relatively little-known, probably because it was originally described only on a Usenet post [6]. The basic idea of Dc is to encode the starting position of each maximal run. The details of the algorithm are given in Figure 1. Note that Dc does not encode the length of the runs since the ending position of the current run is determined by the starting position of the next run. The distance between two characters is defined as the number of characters between them plus one (so the distance is one if the two characters are consecutive). The distance of a character from the beginning of s is defined as the number of characters preceding it plus one (so the distance is one for the first character of the string s). We define $\mathsf{Dc} + \mathsf{Enc}$ as the algorithm in which the integers produced by Dc are encoded using the integer coder Enc.

Lemma 5.1 *Let* $\mathsf{A}_1 = \mathsf{Dc} + \mathsf{Enc}$*. For any string* s *and for any* $\epsilon > 0$ *we have*

$$|\mathsf{A}_1(s)| \leq \max(2a, a + b + \epsilon)|s|H_0^*(s) + O(h).$$

Proof: Assume $H_0(s) \neq 0$ (otherwise $s = \sigma^n$ and the proof follows by an easy computation). Writing the first character in s takes $O(\log h)$ bits; we write h copies of 1 while encoding s (or $h+1$ if the first character is a 1), which takes $O(h)$ bits. Writing the length of the last run takes $|\mathsf{Enc}(\ell)|$ which is at most $a\log\ell + b$ bits. We are left with the task of bounding the cost of encoding: 1) the starting position of the first run of each character, 2) the distance between the ending position of each run and the starting position of the next run of the same character. We account these costs separately for each $\sigma \in \Sigma$. Let (α_1, β_1), (α_2, β_2), …, (α_k, β_k) denote the starting and ending positions of the runs of σ. Dc encodes these runs with the sequence of codewords

$$\mathsf{Enc}(\alpha_1), \mathsf{Enc}(\alpha_2 - \beta_1), \mathsf{Enc}(\alpha_3 - \beta_2), \dots, \mathsf{Enc}(\alpha_k - \beta_{k-1})$$

whose overall size is bounded by (setting $\beta_0 = 0$)

$$a\left[\log(\alpha_1 - \beta_0) + \log(\alpha_2 - \beta_1) + \cdots + \log(\alpha_k - \beta_{k-1})\right] + bk \tag{10}$$

bits. Summing the above term over all σ and reasoning as in the proof of Lemma 4.1 (compare (10) with (8)) we get

$$|\mathsf{A}_1(s)| \le a\log\ell + a|s|H_0(s) + b\,\mathsf{runs}(s) + O(h),$$

where $\mathsf{runs}(s)$ is the number of runs in s. The thesis follows by Lemma 2.3. ∎

The above lemma tells us that $\mathsf{Dc}+\mathsf{Enc}$ compresses any string up to its 0th-order entropy. Unfortunately, our next result shows that this algorithm combined with the bwt cannot achieve an entropy-only bound in terms of H_k^* for $k \ge 1$.

Theorem 5.2 *For any integer encoder* Enc*, there exists an infinite number of strings* s *such that*

$$|\mathsf{Enc}(\mathsf{Dc}(\mathsf{bwt}(s)))| \ge (h-2)|s|H_1^*(s) - \Theta(h^2),$$

where h *is the size of the input alphabet.*

Proof: Every uniquely decodable integer encoder Enc must satisfy the extended Kraft's inequality [8, Theorem 5.2.2]:

$$\sum\nolimits_{i\ge 1} 2^{-|\mathsf{Int}(i)|} \le 1.$$

Hence, there exists an infinite number of integers m such that $|\mathsf{Enc}(m)| \ge \log m$. For each such integer m let $n = m - (h-1)$. Note that

$$|\mathsf{Enc}(n + (h-1))| \;=\; |\mathsf{Enc}(m)| \;\ge\; \log m \;\ge\; \log n. \tag{11}$$

Consider the string

$$s = \sigma_1\sigma_3\ \sigma_1\sigma_4\ \sigma_1\sigma_5\ \cdots\ \sigma_1\sigma_{h-1}\ \sigma_1\sigma_h\ \ \sigma_1\sigma_2^n\sigma_3\ \ \sigma_3\sigma_4\ \sigma_3\sigma_5\ \cdots\ \sigma_3\sigma_{h-1}\ \sigma_3\sigma_h.$$

The string s consists of the concatenation of the pairs $\sigma_1\sigma_i$, for $i = 3, \ldots, h$, followed by $\sigma_1\sigma_2^n\sigma_3$, followed by the concatenation of the pairs $\sigma_3\sigma_i$ for $i = 4, \ldots, h$. $\mathsf{bwt}(s)$ is obtained by sorting the characters of s using the substring $s[0, i-1]$ as the sorting key for the character $s[i]$.[3] A tedious computation shows that

$$\mathsf{bwt}(s) = \sigma_1\ \underbrace{\sigma_3\sigma_4\sigma_5\cdots\sigma_h\sigma_2}_{w_1}\ \underbrace{\sigma_2^{n-1}\sigma_3}_{w_2}\ \underbrace{\sigma_1\sigma_3\sigma_4\sigma_5\cdots\sigma_h}_{w_3}\ \underbrace{\sigma_1\sigma_3}_{w_4}\ \underbrace{\sigma_1\sigma_3}_{w_5}\ \cdots\ \underbrace{\sigma_1\sigma_3}_{w_{h-1}}\ \underbrace{\sigma_1}_{w_h}.$$

In the above representation of $\mathsf{bwt}(s)$, for $i = 1, \ldots, h$ we have highlighted the string w_i containing the set of characters immediately following σ_i in s (the initial σ_1 in $\mathsf{bwt}(s)$ corresponds to the initial σ_1 in s and therefore does not belong to any w_i). Thus, we have

$$|s|H_1^*(s) \;\le\; \sum_{i=1}^{h} |w_i|H_0^*(w_i) \;\le\; \log n + 2h\log h + 2(h-3). \tag{12}$$

At the same time we notice that Dc applied to $\mathsf{bwt}(s)$ generates $h-2$ times the integer $n+h-1$ since there are exactly that many characters between the first two occurrences of the characters $\sigma_1, \sigma_4, \sigma_5, \ldots, \sigma_h$. By (11) and (12) we have

$$|\mathsf{Enc}(\mathsf{Dc}(\mathsf{bwt}(s)))| \;\ge\; (h-2)|\mathsf{Enc}(n+h-1)| \;\ge\; (h-2)\log(n) \;\ge\; (h-2)|s|H_1^*(s) - \Theta(h^2\log h)$$

as claimed. ∎

[3] We are assuming substrings are compared in right-to-left lexicographic order. Note that the bwt is more often defined using the substring $s[i+1, n-1]$ as the sorting key for $s[i]$: the two definitions can be made equivalent by reversing the input string. See [14] for details.

Although it is possible that the above result does not hold if we replace the integer encoder Enc with a 0th-order encoder, the above theorem suggests that the repeated encoding of large distances could be a cause of inefficiency for Dc. For this reason, we introduce a new algorithm called Distance Coding with escapes (Dc_esc). The main difference between Dc and Dc_esc is that, whenever Dc would write a distance, Dc_esc compares the cost of writing that distance to the cost of escaping and re-entering later, and does whichever is cheaper.

Whenever Dc would write 1, Dc_esc writes $\langle 1, 1\rangle$; this lets us use $\langle 1, 2\rangle$ as a special re-entry sequence. To escape after a run of σ's, we write $\langle 1, 1\rangle$; to re-enter at the next run of σ's, we write $\langle 1, 2, \ell, \sigma\rangle$, where ℓ is the length of the preceding run (necessarily of some other character). To see how Dc_esc works, suppose we are encoding the string

$$s = \cdots\ \sigma_1^j\, \sigma_2^k\, \sigma_3^\ell\, \sigma_1^m\ \cdots.$$

When Dc reaches the run σ_1^j it encodes the value $k+\ell+1$ which is the distance from the last σ_1 in σ_1^j to the first σ_1 in σ_1^m. Instead, Dc_esc compares the cost of encoding $k+\ell+1$ with the cost of encoding an escape (sequence $\langle 1, 1\rangle$) plus the cost of re-entering. In this case the re-entry sequence would be written immediately after the code associated with the run σ_3^ℓ and would consist of the sequence $\langle 1, 2, \ell, \sigma_1\rangle$. When the decoder finds such a sequence it knows that the current run (in this case of σ_3's) will only last for ℓ characters and, after that, there is a run of σ_1's. (Recall that Dc only encodes the starting position of each run: the end of the run is induced by the beginning of a new run. When we re-enter an escaped character we must explicitly provide the length of the ongoing run).

Notice we do not distinguish between instances in which $\langle 1, 1\rangle$ indicates a character does not occur, cases in which it indicates a character does not occur again, and cases in which it indicates an escape; we view the first two types of cases as escapes without matching re-entries.

Lemma 5.3 *Let* $\mathsf{A}_1 = \mathsf{Dc} + \mathsf{Enc}$ *and let* $\mathsf{A}_2 = \mathsf{Dc\_esc} + \mathsf{Enc}$*. For any string* s *and for any partition* $s = s_1 \cdots s_t$

$$|\mathsf{A}_2(s)| \le \sum_{i=1}^{t} |\mathsf{A}_1(s_i)| \ +\ O(ht \log h).$$

Proof: Fix a partition $s = s_1 \cdots s_t$ and consider the algorithm Dc_esc* that, instead of choosing at each step whether to escape or not, escapes if and only if the current distance crosses the boundary between two different partition elements. That is, Dc_esc* uses the escape sequence every time it encodes the distance between a run ending in s_i and a run starting in s_j with $j > i$. Let $\mathsf{A}_2^* = \mathsf{Dc\_esc^*} + \mathsf{Enc}$. Since Dc_esc always performs the most economical choice, we have $|\mathsf{A}_2(s)| \le |\mathsf{A}_2^*(s)|$; we prove the lemma by showing that

$$|\mathsf{A}_2^*(s)| \le \sum_{i=1}^{t} |\mathsf{A}_1(s_i)| \ +\ O(ht \log h).$$

Clearly Dc_esc* escapes at most th times. The parts of an escape/re-enter sequence that cost $\Theta(\log h)$ (that is, the codewords for $\langle 1, 1\rangle$, $\langle 1, 2\rangle$ and the encoding of the escaped character σ) are therefore included in the $O(ht \log h)$ term. Thus, for each escape sequence we have only to take care of the cost of encoding the value ℓ that provides the length of the run immediately preceding the re-entry point. We now show that the cost of encoding the run lengths ℓs is bounded by costs paid by Dc and not paid by Dc_esc*. Let σ denote the escaped character. Let s_j denote the partition element containing the re-entry point and let m denote the position in s_j where the new run of σ's starts (that is, at position m of s_j there starts a run of σ's; the previous one ended in some s_i with $i < j$ so Dc_esc* escaped σ and is now re-entering). Let σ_p denote the character immediately preceding the re-entry point: with our notation we have that the re-entry point is preceded by the run σ_p^ℓ. We consider two cases:

$\ell \le m$. In this case the run σ_p^ℓ starts within s_j. This implies that the cost $|\mathsf{Enc}(\ell)|$ paid by Dc_esc* is no greater than the cost $|\mathsf{Enc}(m)|$ paid by Dc for encoding the first position of σ in s_j.

$\ell > m$. In this case the run σ_p^ℓ starts in a partition element preceding s_j. Let $m' = \ell - m$. If $m' < |s_{j-1}|$ the run σ_p^ℓ starts within s_{j-1}. Under this assumption, by Lemma 3.1, the cost $|\mathsf{Enc}(\ell)|$ paid by $\mathsf{Dc\_esc^*}$ is at most d_{ab} plus the cost $|\mathsf{Enc}(m)|$ paid by Dc for encoding the first position of σ in s_j, plus the cost $|\mathsf{Enc}(m')|$ paid by Dc to encode the length of the last run in s_{j-1}. If $m' > |s_{j-1}|$ then the run σ_p^ℓ spans several partition elements s_{j-k}, s_{j-k+1}, ..., s_j. In this case, again by Lemma 3.1, the cost $|\mathsf{Enc}(\ell)|$ is bounded by d_{ab} plus the cost paid by Dc for encoding the following items: 1) the last run in s_{j-k}, 2) the last (and only) run in s_{j-k+1}, ..., s_{j-1}, 3) the first position of σ in s_j.

■

Combining Lemma 5.3 with Lemma 5.1 and Lemma 2.1 we immediately get

Theorem 5.4 *The algorithm* $\mathsf{A}_2 = \mathsf{Dc\_esc} + \mathsf{Enc}$ *is locally* $\max(2a, a+b+\epsilon)$*-optimal for any* $\epsilon > 0$*, hence for any string* s *and* $k \geq 0$ *it is* $|\mathsf{A}_2(\mathsf{bwt}(s))| \leq \max(2a, a+b+\epsilon)|s|H_k^*(s) + \log|s| + O(h^{k+1}\log h)$. ■

We now consider the case in which the output of $\mathsf{Dc\_esc}$ is compressed with the encoder $\mathsf{Order0^*}$. The main tool for our analysis will again be Lemma 3.3, which establishes a relationship between the output size of $\mathsf{Order0^*}$ of that of an integer coder. However, there is the technical difficulty that for a generic 0th-order we do not necessarily have the concept of a codeword assigned to each input symbol. The concept of a codeword is well-defined for Huffman coding, for example, but not for Arithmetic coding. This could be a problem for $\mathsf{Dc\_esc}$ because, in order to decide whether to escape or not, it compares the cost of encoding two different set of symbols.

Theorem 5.5 *The algorithm* $\mathsf{Dc\_esc} + \mathsf{Order0^*}$ *is locally* $(2.94 + C_0)$*-optimal.*

Proof: Fix $\mu > 1$ and $\nu > 0$. Let $\mathsf{Enc}_{\mu\nu}$ be the ideal integer coder such that $|\mathsf{Enc}_{\mu\nu}(i)| = \mu\log i + \log(\zeta(\mu)) + \nu + C_0$ (see Lemma 3.3). Let $\mathsf{Dc\_esc}_{\mu\nu}$ denote the algorithm that decides whether to escape or not on the basis of the costs given by $\mathsf{Enc}_{\mu\nu}$. By Theorem 5.4 $\mathsf{Dc\_esc}_{\mu\nu} + \mathsf{Enc}_{\mu\nu}$ is locally $\max(2\mu, \mu+\log(\zeta(\mu))+\nu+\epsilon+C_0)$-optimal for any $\epsilon > 0$. Since by Lemma 3.3 $|\mathsf{Order0^*}(\mathsf{Dc\_esc}_{\mu\nu}(s))| \leq |\mathsf{Enc}_{\mu\nu}(\mathsf{Dc\_esc}_{\mu}(s))| + O(1)$ the local optimality result stated in Theorem 5.4 holds for $\mathsf{Dc\_esc}_{\mu\nu} + \mathsf{Order0^*}$ as well. The theorem follows taking $\mu = 1.47$ and $\nu = \epsilon = 0.001$. ■

5.1 Using an explicit escape symbol

We now show how to improve the performance of $\mathsf{Dc\_esc}$ by using a special escape symbol to introduce escape/re-enter sequences. The rationale is that escape/re-enter sequences are relatively rare so it pays to use a special low-probability symbol for them. This escape symbol will be used also by our variant of the Inversion Frequencies algorithm.

Lemma 5.6 *Let* Enc *be a code for the integers such that for* $i > 0$ *it is* $|\mathsf{Enc}(i)| \leq a\log i + b$*. For any* $\delta > 0$ *there exists a code* Enc^δ *such that: 1) for* $i > 0$ *it is* $|\mathsf{Enc}^\delta(i)| \leq (1+\delta)(a\log i) + b$*, 2) in addition to the positive integers* Enc^δ *can encode a special escape symbol* esc.

Proof: Given $\delta > 0$ let i_δ denote the smallest integer such that $\log(i+1) \leq (1+\delta)\log i$. We define the code Enc^δ as follows: $\mathsf{Enc}^\delta(\mathsf{esc}) = \mathsf{Enc}(i_\delta)$ and

$$\mathsf{Enc}^\delta(i) = \begin{cases} \mathsf{Enc}(i) & \text{for } i < i_\delta, \\ \mathsf{Enc}(i+1) & \text{for } i \geq i_\delta. \end{cases}$$

The lemma follows since the concavity of $\log x$ ensures $|\mathsf{Enc}^\delta(i)| \leq (1+\delta)(a\log i) + b$ for any $i \geq 1$. ■

Let Esc_1 denote the procedure that, given a sequence of positive integers, replaces every occurrence of 1 with the symbol esc. For example: $\mathsf{Esc}_1(2113314) = 2\,\mathsf{esc}\,\mathsf{esc}\,3\,3\,\mathsf{esc}\,4$. Let $\mathsf{B}_2 = \mathsf{Dc\_esc} + \mathsf{Esc}_1 + \mathsf{Enc}^\delta$. Note that in B_2 every occurrence of the symbol 1 produced by $\mathsf{Dc\_esc}$ is eventually encoded with the codeword $\mathsf{Enc}^\delta(\mathsf{esc})$. We assume that $\mathsf{Dc\_esc}$ assigns the cost $|\mathsf{Enc}^\delta(\mathsf{esc})|$ to the symbol 1 when it has to decide whether to escape or not.

Lemma 5.7 *For any positive constants ϵ, δ, the algorithm $\mathsf{B}_2 = \mathsf{Dc\_esc} + \mathsf{Esc}_1 + \mathsf{Enc}^\delta$ is locally λ-optimal with $\lambda = \max(2a', a' + b + \epsilon)$, $a' = a(1+\delta)$.*

Proof: Let $\mathsf{B}_1 = \mathsf{Dc} + \mathsf{Esc}_1 + \mathsf{Enc}^\delta$. Since Dc outputs the symbol 1 at most $2h$ times, replacing it with esc introduces an $O(h)$ overhead. Replacing Enc with Enc^δ introduces a multiplicative overhead of $(1+\delta)$ to each log term; repeating the proof of Lemma 5.1 we get

$$|\mathsf{B}_1(s)| \le \max(2a', a' + b + \epsilon)|s|H_0^*(s) + O(h). \tag{13}$$

Consider now $\mathsf{B}_2^* = \mathsf{Dc\_esc*} + \mathsf{Esc}_1 + \mathsf{Enc}^\delta$, where $\mathsf{Dc\_esc*}$ is defined as in the proof of Lemma 5.3. Reasoning as in Lemma 5.3 we have that for any partition $s = s_1 \cdots s_t$

$$|\mathsf{B}_2(s)| \le |\mathsf{B}_2^*(s)| \le \sum_{i=1}^{t} |\mathsf{B}_1(s_i)| \ + \ O(ht \log h)$$

where the second inequality follows by the fact that $\mathsf{Dc\_esc*}$ outputs the esc symbol at most $O(ht)$ times. The lemma follows combining the above inequality with (13). ∎

Theorem 5.8 *The algorithm $\mathsf{Dc\_esc} + \mathsf{Esc}_1 + \mathsf{Order0*}$ is locally $(2.69 + C_0)$-optimal.*

Proof: Fix $\mu > 1$ and $\nu > 0$. Since

$$\textstyle\sum_{j\ge 2} ((\zeta(\mu) - 1)j^\mu)^{-1} \ = \ (\zeta(\mu) - 1)^{-1}\Big(\sum_{j \ge 2} j^{-\mu}\Big) \ = \ 1,$$

by repeating the proof of Lemma 3.3 one can show that applying $\mathsf{Order0*}$ to a sequence of integers greater than one produces an output size bounded by the output size of an ideal integer coder $\mathsf{Enc}_{\mu\,\nu}$ for the set $\{j \mid j \ge 2\}$ such that $|\mathsf{Enc}_{\mu\,\nu}(i)| = \mu \log i + \log(\zeta(\mu) - 1) + \nu + C_0$.

Let $\mathsf{Enc}^\delta_{\mu\,\nu}$ be the coder for the set $\{\mathsf{esc}\} \cup \{2, 3, 4, \ldots\}$ obtained by applying Lemma 5.6 to $\mathsf{Enc}_{\mu\,\nu}$. By Lemma 5.7, for any $\epsilon, \delta > 0$, the algorithm $\mathsf{Dc\_esc} + \mathsf{Esc}_1 + \mathsf{Enc}^\delta_{\mu\,\nu}$ is λ-optimal with $\lambda = \max(2\mu(1+\delta), \mu(1+\delta) + \log(\zeta(\mu) - 1) + \nu + C_0 + \epsilon)$. Since $\mathsf{Enc}^\delta_{\mu\,\nu}$ is defined in terms of $\mathsf{Enc}_{\mu\,\nu}$ and $\mathsf{Order0*}$ produces at most $O(1)$ more bits than $\mathsf{Enc}_{\mu\,\nu}$, the same local optimality result holds for $\mathsf{Order0*}$ as well. The theorem follows taking $\mu = 1.343$, and $\nu = \epsilon = \delta = 0.001$. ∎

Corollary 5.9 *For any string s and $k \ge 0$ we have*

$$|\mathsf{Order0*}(\mathsf{Esc}_1(\mathsf{Dc\_esc}(\mathsf{bwt}(s))))| \le (2.69 + C_0)|s|H_k^*(s) + \log|s| + O\Big(h^{k+1}\log h\Big).$$

Proof: Immediate by Theorem 5.8 and Lemma 2.1. ∎

6 Analysis of Inversion Frequencies Coding

Inversion Frequencies coding (If for short) is a coding strategy first proposed in [3] as an alternative to Mtf. Given a string s over an ordered alphabet $\Sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_h\}$, in its original formulation If works in $h-1$ phases. In the ith phase If encodes the distance between every pair of consecutive occurrences of σ_i: in the computation of such distances If ignores the characters smaller than σ_i. In other words, in

Inversion Frequencies with Run-Length Encoding (If_rle)

1. Write $h = |\Sigma|$ bits to indicate which characters are actually present in s (from now on we assume all characters are present);
2. For $i = 1, \ldots, h-1$: write the number ℓ_i of characters greater than σ_i preceding the first occurrence of σ_i in s; if $\ell_i = 0$ write esc instead.
3. Set $j = 1$ and repeat while $j \leq |s|$:
   (a) Let $\sigma_i = s[j]$. Let $s[m]$ be the first occurrence of a symbol greater than σ_i to the right of $s[j]$, and let $s[p]$ be the first occurrence of the symbol σ_i to the right of $s[m]$.
   (b) Write the pair $\langle k, \ell \rangle$ where k is the number of occurrences of σ_i in $s[j] \cdots s[m-1]$ and ℓ is the number of occurrences of symbols greater than σ_i in $s[m] \cdots s[p-1]$.
   (c) Set j to be the next position in s containing a character different from σ_i and σ_h.
4. Write the pair $\langle \mathsf{esc}, \mathsf{esc} \rangle$.

Figure 2: Inversion Frequencies with Run Length Encoding.

the ith phase If conceptually builds the string $s^{(i)}$ removing from s the characters smaller than σ_i and encodes the distances between consecutive occurrences of σ_i in $s^{(i)}$. Note that If does not encode explicitly the occurrences of σ_h. The output of If consists of the concatenation of the output of the single phases prefixed by an encoding of the number of occurrences of each symbol σ_i (this information is needed by the decoder to determine when a phase is complete). For example, if $s = \sigma_2\sigma_2\sigma_1\sigma_3\sigma_3\sigma_1\sigma_3\sigma_1\sigma_3\sigma_2$, the first phase encodes the occurrences of σ_1 in s, producing the sequence $\langle 3, 3, 2 \rangle$, and the second phase encodes the occurrences of σ_2 in $s^{(2)} = \sigma_2\sigma_2\sigma_3\sigma_3\sigma_3\sigma_3\sigma_2$, producing the sequence $\langle 1, 1, 5 \rangle$. The output of If is an encoding of the number of occurrences of σ_1, σ_2, and σ_3 (3, 3, and 4 in our example), followed by the sequence $\langle 3, 3, 2, 1, 1, 5 \rangle$.

Recently, [13, Sect. 3.2] has shown that If is equivalent to coding the string s with a skewed wavelet tree combined with Gap Encoding. The analysis in [13] shows that, if the alphabet is reordered so that σ_h is the most frequent symbol, the output of If + Enc is bounded by

$$|\mathsf{Enc}(\mathsf{If}(s))| \leq \max(a, b)|s|H_0(s) + (|\Sigma| + a)\log|s| + O(1). \tag{14}$$

Unfortunately, the following example shows that If+Enc is not powerful enough to achieve an entropy-only bound.

Example 2 Consider the string $s = (\sigma_2\sigma_1)^n$. It is $\mathsf{bwt}(s) = \sigma_2^n\sigma_1^n$. No matter how we order the alphabet, If applied to $\mathsf{bwt}(s)$ produces $n-1$ copies of the symbol 1, hence $|\mathsf{Enc}(\mathsf{If}(\mathsf{bwt}(s)))| = \Theta(n)$ which is exponentially larger than $|s|H_1^*(s) \approx 2\log n$. ■

To prove entropy-only bounds for If we develop two variants and we show they are locally optimal according to Definition 2. The first variant, called If_rle, simply combines If with Run Length Encoding. If_rle produces a sequence over the set $\{\mathsf{esc}\} \cup \{1, 2, \ldots\}$ so its output will be compressed using the Enc^δ encoder described in Lemma 5.6.

The outline of the procedure If_rle is described in Figure 2. Note that in the main body of If_rle (Step 3) we are essentially encoding the following information: "starting from the current character $\sigma_i = s[j]$ there are k occurrences of σ_i before we reach the first character greater than σ_i; after that there are ℓ characters greater than σ_i before we find another occurrence of σ_i". Note also that, similarly to If, the procedure If_rle does not encode explicitly the occurrences of σ_h. In Step 3a we are assuming that the characters $s[p]$ and $s[m]$ always exist: this is not the case for the last run of each character that is handled using the escape symbol. If $s[m]$ or $s[p]$ does not exist (there are no characters greater than σ_i to the right of $s[j]$, Decoding Procedure for If_rle

1. Read $h = |\Sigma|$ bits to determine which characters are actually present in s (from now on we assume all characters are present);
2. For $i = 1, \ldots, h-1$, read ℓ_i and set *To_be_skipped*$[i] \leftarrow \ell_i$ and *To_be_written*$[i] \leftarrow 0$ (if $\ell_i = \mathsf{esc}$ set *To_be_skipped*$[i] \leftarrow 0$ instead);
3. Repeat until the pair $\langle \mathsf{esc}, \mathsf{esc} \rangle$ has been read:
   (a) Let i be the smallest index such that *To_be_skipped*$[i] = 0$, read the next pair $\langle k, \ell \rangle$ and set *To_be_written*$[i] \leftarrow k$, *To_be_skipped*$[i] \leftarrow \ell$ (if all *To_be_skipped*$[i]$ are nonzero do nothing);
   (b) Let i be the smallest index such that *To_be_written*$[i] \neq 0$; if all *To_be_written*$[i]$ are zero, let $i = h$;
   (c) Write σ_i to the output file;
   (d) For $j = 1, 2, \ldots, i-1$ set *To_be_skipped*$[j] \leftarrow$ *To_be_skipped*$[j] - 1$;

Figure 3: Decoding procedure for Inversion Frequencies with Run Length Encoding.

or there are no occurrences of σ_i to the right of $s[m]$), then If_rle writes the pair $\langle k, \mathsf{esc} \rangle$ and the character σ_i is no longer considered.[4]

The procedure for decoding the output of If_rle is shown in Figure 3. The decoder maintains two arrays *To_be_written*$[1, \ldots, h-1]$ and *To_be_skipped*$[1, \ldots, h-1]$ such that *To_be_written*$[i]$ stores how many σ_i's have to be written before we find a character greater than σ_i and *To_be_skipped*$[i]$ stores how many characters greater than σ_i there are between the end of the current run of σ_i's and the next one (again runs and distances for σ_i are defined ignoring smaller characters). For a single character σ_i the decoding procedure works as follows. While *To_be_written*$[i] > 0$ the decoder outputs σ_i and decreases *To_be_written*$[i]$ by one. When *To_be_written*$[i]$ reaches zero the decoder decreases *To_be_skipped*$[i]$ by one each time it outputs a character greater than σ_i. When *To_be_skipped*$[i]$ also reaches zero the decoder needs new instructions for σ_i so it reads a new pair $\langle k, \ell \rangle$ from the compressed file and sets *To_be_written*$[i] \leftarrow k$ and *To_be_skipped*$[i] \leftarrow \ell$. The actual decoding procedure is more complex since it has to work on all characters $\sigma_1, \ldots, \sigma_h$ at the same time. So it is often the case that more than one *To_be_written*$[i]$ is greater than zero: in this case the smallest i wins; the reason for this is that, if $i < j$, the encoding of σ_j ignores the occurrences of σ_i so σ_i must take precedence. Note that the decoder outputs a character σ_h every time *To_be_skipped*$[j] > 0$ for every $j < h$. The last run of each character is handled as follows: if the decoder reads the pair $\langle k, \mathsf{esc} \rangle$ it sets *To_be_written*$[i] \leftarrow k$ and *To_be_skipped*$[i] \leftarrow \infty$, meaning there are k more occurrences of σ_i and no more.

As a preliminary to the analysis of If_rle, we establish the following two technical lemmas. Note that Lemma 6.1, which we restate here for completeness, is a known property of wavelet trees [17].[5]

Lemma 6.1 *For $i = 1, 2, \ldots, h-1$ let $z^{(i)}$ denote the binary string obtained from s deleting all characters smaller than σ_i, replacing the occurrences of σ_i with* 1*, and replacing the occurrences of characters greater than σ_i with* 0*. We have*

$$\sum_{i=1}^{h-1} |z^{(i)}| H_0(z^{(i)}) = |s| H_0(s). \tag{15}$$

Proof: See Appendix B. ∎

[4] There is an exception to this rule: if the input string ends with a run σ_h^ℓ, then the penultimate run σ_i^k must be encoded with the pair $\langle k, \ell \rangle$ rather than with the escape sequence $\langle k, \mathsf{esc} \rangle$. This is necessary since otherwise the output would contain no information on the last run since σ_h's occurrences are not explicitly encoded.

[5] An attentive reader might have already noticed that there is a relationship between If_rle and a skewed wavelet tree [13, Sec. 3.2] whose internal nodes are compressed with Run Length Encoding. This is true even if the two algorithms have a completely different structure.

Lemma 6.2 *Let z be a binary string of the form $z = 0^{\ell_1}1^{\ell_2}\cdots\sigma^{\ell_m}$, where $\sigma = 0$ if m is odd, and $\sigma = 1$ if m is even. Define*

$$RLE(z) = \sum_{i=1}^{m} |\mathsf{Enc}^{\delta}(\ell_i)|. \tag{16}$$

If $|\mathsf{Enc}^{\delta}(\ell)| \leq (1+\delta)(a\log\ell) + b$ as in Lemma 5.6, then setting $a' = a(1+\delta)$ we have

$$RLE(z) \leq a'|z|H_0(z) \ + \ a'\log|z| + \ b\,\mathsf{runs}(z).$$

Proof: See Appendix B. ∎

Let $\mathsf{A}_3 = \mathsf{lf\_rle} + \mathsf{Enc}^{\delta}$. For $i = 1,\ldots,h-1$, let $s^{(i)}$ denote the string obtained by removing from s the characters smaller than σ_i. If in $s^{(i)}$ we replace σ_i with 1 and $\sigma_{i+1},\ldots,\sigma_h$ with 0 we get precisely the string $z^{(i)}$ defined in Lemma 6.1. Let RLE be defined by (16). We first observe that

$$|\mathsf{A}_3(s)| \leq \sum_{i=1}^{h-1} RLE(z^{(i)}) \ + \ O(h)\,. \tag{17}$$

Indeed, apart from h bits at Step 1 and $O(h)$ esc symbols, $\mathsf{lf\_rle}$'s output consists precisely of the lengths of the runs of zeros and ones in $z^{(i)}$ for $i = 1,\ldots,h-1$. Consider, for example, the encoding of the character σ_i. At Step 2 $\mathsf{lf\_rle}$ encodes the number ℓ_i of characters greater than σ_i preceding the first occurrence of σ_i in s: this is precisely the length of the first run of 0's in $z^{(i)}$. Then, each pair $\langle k,\ell\rangle$ written at Step 3 represents a run of σ_i in $s^{(i)}$—corresponding to a run of 1's in $z^{(i)}$—followed by a run of characters greater than σ_i—corresponding to a run of 0's in $z^{(i)}$ (note that, except for the case mentioned in Footnote 4, the last run of each character σ_i is encoded with the escape sequence $\langle k,\mathsf{esc}\rangle$; in other words $\mathsf{lf\_rle}$ does not explicitly encode the length of the last run of 0's in $z^{(i)}$).

Having established (17) we are now ready to prove that A_3 is locally pseudo optimal.

Theorem 6.3 *Let Enc denote an integer encoder such that $|\mathsf{Enc}(i)| \leq a\log i + b$. For any pair of positive constants ϵ,δ the algorithm $\mathsf{A}_3 = \mathsf{lf\_rle} + \mathsf{Enc}^{\delta}$ is locally pseudo optimal with parameter $\lambda_h = \max(ha', a'+b+\epsilon)$, where $a' = a(1+\delta)$.*

Proof: We need to prove that for any partition $s = s_1s_2\cdots s_t$ it is

$$|\mathsf{A}_3(s)| \leq \max(ha', a'+b+\epsilon)\sum_{j=1}^{t}|s_j|H_0^*(s_j) \ + \ O(th)\,. \tag{18}$$

For $i = 1,\ldots,h-1$ let $s^{(i)}$ and $z^{(i)}$ be defined as above. The partition $s = s_1\cdots s_t$ naturally induces the partitions $s^{(i)} = s_1^{(i)}\cdots s_t^{(i)}$ and $z^{(i)} = z_1^{(i)}\cdots z_t^{(i)}$ (note that if s_j contains only symbols smaller than σ_i then $s_j^{(i)}$ and $z_j^{(i)}$ are both empty). If RLE is defined by (16), by Lemma 3.1 it is

$$RLE(z^{(i)}) \leq \sum_{j=1}^{t} RLE(z_j^{(i)}) \ + \ O(t) \tag{19}$$

that, combined with (17), yields

$$\begin{aligned} |\mathsf{A}_3(s)| \ &\leq \ \sum_{i=1}^{h-1}\sum_{j=1}^{t} RLE(z_j^{(i)}) \ + \ O(th) \\ &\leq \ \sum_{j=1}^{t}\left(\sum_{i=1}^{h-1} RLE(z_j^{(i)})\right) \ + \ O(th)\,. \end{aligned}$$

Hence, to prove (18) it suffices to show that for any partition element s_j it is

$$\sum_{i=1}^{h-1} RLE(z_j^{(i)}) \leq \max(ha', a'+b+\epsilon)|s_j|H_0^*(s_j) + O(h). \tag{20}$$

Fix s_j and let d_j denote the number of distinct characters appearing in s_j. To prove (20) we distinguish three cases according to the size of d_j (clearly $1 \leq d_j \leq h$).

CASE $d_j = h$. In this case for every character σ_i it is $H_0(z_j^{(i)}) \neq 0$ which implies $H_0(z_j^{(i)}) = H_0^*(z_j^{(i)})$. By Lemmas 6.2 and 2.3, for any $\epsilon > 0$ we have

$$RLE(z_j^{(i)}) \leq \max(2a', a'+b+\epsilon)|z_j^{(i)}|H_0(z_j^{(i)}) + O(1). \tag{21}$$

Combining the above inequality with Lemma 6.1, we get

$$\begin{aligned} \sum_{i=1}^{h-1} RLE(z_j^{(i)}) &\leq \max(2a', a'+b+\epsilon)\sum_{i=1}^{h-1} |z_j^{(i)}|H_0(z_j^{(i)}) + O(h) \\ &\leq \max(2a', a'+b+\epsilon)|s_j|H_0(s_j) + O(h) \end{aligned}$$

which proves (20).

CASE $d_j = 1$. Let σ_f denote the only symbol appearing in s_j. In this case we have

$$z_j^{(i)} = \begin{cases} \mathsf{0}^{|s_j|} & \text{for } i = 1, \ldots, f-1, \\ \mathsf{1}^{|s_j|} & \text{for } i = f, \\ \text{empty} & \text{for } i > f. \end{cases} \tag{22}$$

Since $|s_j|H_0^*(s_j) = 1 + \lfloor \log |s_j| \rfloor$, (20) follows observing that

$$\sum_{i=1}^{h-1} RLE(z_j^{(i)}) \;\leq\; (h-1)\mathsf{Enc}^{\delta}(|s_j|) \;\leq\; (h-1)(a' \log |s_j| + b) \;\leq\; (h-1)a'|s_j|H_0^*(s_j) + b(h-1).$$

CASE $1 < d_j < h$. This is the most complex case. Let σ_e, σ_f denote, respectively, the smallest and the largest symbols appearing in s_j. Let $\ell_j = |s_j|$ and let m_j denote the number of occurrences of σ_f in s_j. For $i = 1, \ldots, h-1$, it is

$$z_j^{(i)} = \begin{cases} \mathsf{0}^{\ell_j} & \text{for } i < e, \\ \mathsf{1}^{m_j} & \text{for } i = f, \\ \text{empty} & \text{for } i > f. \end{cases}$$

Note that for $\sigma_e < \sigma_i < \sigma_f$ we can still have $z_j^{(i)} = \mathsf{0}^{r_i}$ (this happens when s_j does not contain σ_i), however our hypothesis ensures that $H_0(z_j^{(e)}) \neq 0$ and $|z_j^{(e)}| = \ell_j$. Let W_j denote the subset of $\{\sigma_1, \sigma_2, \ldots, \sigma_{h-1}\}$ such that $z_j^{(i)}$ consists of a single non-empty run of $\mathsf{0}$'s or $\mathsf{1}$'s. By Lemma 6.2 it is

$$\sum_{i=1}^{h-1} RLE(z_j^{(i)}) \leq \sum_{i \notin W_j} \Big(a'|z_j^{(i)}|H_0(z_j^{(i)}) + a' \log |z_j^{(i)}| + b\,\mathsf{runs}(z_j^{(i)})\Big) \;+\; \sum_{i \in W_j} a' \log |z_j^{(i)}| \;+\; O(h). \tag{23}$$

Note that for $i \in W_j$ it is $|z_j^{(i)}| \leq |z_j^{(e)}|$. In addition, since $\sigma_e \notin W_j$ it is $|W_j| \leq h-2$. Combining these facts we get

$$\sum_{i \in W_j} a' \log |z_j^{(i)}| \;\leq\; a'(h-2) \log |z_j^{(e)}| \tag{24}$$

which, plugged into (23) yields

$$\sum_{i=1}^{h-1} RLE(z_j^{(i)}) \leq \sum_{i \notin W_j} \left(a'|z_j^{(i)}|H_0(z_j^{(i)}) + a'(h-1)\log|z_j^{(i)}| + b\,\mathsf{runs}(z_j^{(i)}) \right) \;+\; O(h).$$

By Lemma 2.3 and the fact that for $i \notin W_j$ it is $H_0(z_j^{(i)}) = H_0^*(z_j^{(i)})$, for any $\epsilon > 0$ we have

$$\sum_{i=1}^{h-1} RLE(z_j^{(i)}) \leq \sum_{i \notin W_j} \max(a'h, a'+b+\epsilon)|z_j^{(i)}|H_0(z_j^{(i)}) \;+\; O(h).$$

Finally, since for $i \in W_j$ it is $H_0(z_j^{(i)}) = 0$, we have

$$\sum_{i=1}^{h-1} RLE(z_j^{(i)}) \leq \sum_{i=1}^{h-1} \max(a'h, a'+b+\epsilon)|z_j^{(i)}|H_0(z_j^{(i)}) \;+\; O(h),$$

and (20) follows by Lemma 6.1. ∎

Theorem 6.3 proves that $\mathsf{If\_rle}$ is locally pseudo optimal since the factor in front of the entropy grows linearly with the alphabet size. This appears to be an intrinsic limitation of the $\mathsf{If\_rle}$ algorithm. To see this, we observe that $\mathsf{If\_rle}$ sometimes pays for the encoding of the same substring more than once. Consider for example the string: $s = \sigma_1\sigma_2\sigma_3^n\sigma_2\sigma_1$. Assuming $\sigma_1 < \sigma_2 < \sigma_3$ we see that, because of the presence of the σ_3^n substring, $\mathsf{If\_rle}$ pays an $\Theta(\log n)$ cost for the encoding of both σ_1 and σ_2. Generalizing this argument, the following example suggests that the bound in Theorem 6.3 cannot be substantially improved.

Example 3 Consider the partition $s = s_1 s_2 \cdots s_{2h}$ where

$$s_1 = \sigma_1\sigma_2\cdots\sigma_h \quad s_2 = \sigma_1^n \quad s_3 = s_1 \quad s_4 = \sigma_2^n \quad s_5 = s_1 \quad s_6 = \sigma_3^n$$

and so on up to $s_{2h} = \sigma_h^n$. We have $\sum_{i=1}^{2h} |s_i| H_0^*(s_i) = O(h \log n)$, whereas, no matter how we order the alphabet it is $|\mathsf{A}_3(s)| = |\mathsf{Enc}^\delta(\mathsf{If\_rle}(s))| = \Theta(h^2 \log n)$. ∎

To overcome the limitations of $\mathsf{If\_rle}$, we now introduce an *escape and re-enter* mechanism. The new algorithm, called Inversion Frequencies with RLE and Escapes ($\mathsf{If\_rle\_esc}$), works as follows. Assume that $s[j] = \sigma_i$ is the next character to be encoded, and let $s[m]$, $s[p]$, k, and ℓ be defined as for the algorithm $\mathsf{If\_rle}$: $s[m]$ is the first symbol greater than σ_i to the right of $s[j]$, $s[p]$ is the first occurrence of σ_i to the right of $s[m]$, k is the number of occurrences of σ_i in $s[j]\cdots s[m-1]$, and ℓ is the number of occurrences of symbols greater than σ_i in $s[m]\cdots s[p-1]$. Moreover, let o denote the largest index such that $m < o < p$ and $s[o-1] > s[o]$ (o does not necessarily exist). If o does not exist, $\mathsf{If\_rle\_esc}$ behaves as $\mathsf{If\_rle}$ and outputs the pair $\langle k, \ell\rangle$. If o exists, $\mathsf{If\_rle\_esc}$ chooses the most economical option between 1) encoding $\langle k, \ell\rangle$ and 2) escaping σ_i (which means encoding the pair $\langle k, \mathsf{esc}\rangle$) and re-entering it at the position o. It is possible to re-enter at o since the condition $s[o-1] > s[o]$ implies that when the decoder reaches the position o it will need to read new data from the compressed file. To see this, let $s[o] = \sigma_e$ and observe that when the decoder outputs $s[o-1] > \sigma_e$ we must have *To_be_written*$[e] = 0$ (see Step 3b in Fig. 3). Since the decoder can output $s[o] = \sigma_e$ only if *To_be_written*$[e] > 0$, it must be that after outputting $s[o-1]$ the variable *To_be_skipped*$[e]$ has reached zero and the decoder has read a new pair from the compressed file.

The code for re-entering is the triple $\langle \mathsf{esc}, \ell'+1, \sigma_i\rangle$, where σ_i is the re-entering character and ℓ' is the number of characters greater than σ_i in $s[o]\cdots s[p-1]$: we encode $\ell'+1$ since it is possible that $\ell' = 0$. Note however that $\ell'+1$ is never larger than the value ℓ that would have been written if we had not escaped. After reading the re-enter triple $\langle \mathsf{esc}, \ell'+1, \sigma_i\rangle$, the decoder sets *To_be_written*$[i] = 0$ and *To_be_skipped*$[i] = \ell'$ and reads the next pair from the compressed file (that would be the *To_be_written*, *To_be_skipped* pair for $s[o]$ unless there is another re-enter sequence).

$$\cdots \overbrace{\sigma_1\,\sigma_1\,\sigma_2\,\sigma_2\,\sigma_3\,\sigma_3\,\sigma_3}^{s_{j-1}}\ \overbrace{\sigma_4\,\sigma_4\,\sigma_4\,\sigma_4\,\sigma_4\,\sigma_4\,\sigma_5\,\sigma_4}^{s_j}\ \overbrace{\sigma_4\,\sigma_4\,\sigma_4\,\sigma_2\,\sigma_2\,\sigma_1\,\sigma_1\,\sigma_3}^{s_{j+1}} \cdots$$

$$\cdots \underbrace{\langle 2, \mathsf{esc}\rangle}_{\sigma_1} \underbrace{\langle 2, \mathsf{esc}\rangle}_{\sigma_2} \underbrace{\langle 3, \mathsf{esc}\rangle}_{\sigma_3} \underbrace{\langle 6, 1\rangle}_{\sigma_4} \underbrace{\langle 1, ?\rangle}_{\sigma_5} \underbrace{\langle \mathsf{esc}, 4, \sigma_2\rangle}_{\sigma_2 \text{ re-enter}} \underbrace{\langle 4, ?\rangle}_{\sigma_4} \underbrace{\langle \mathsf{esc}, 3, \sigma_1\rangle}_{\sigma_1 \text{ re-enter}} \underbrace{\langle 2, ?\rangle}_{\sigma_2} \underbrace{\langle \mathsf{esc}, 1, \sigma_3\rangle}_{\sigma_3 \text{ re-enter}} \underbrace{\langle 2, ?\rangle}_{\sigma_1} \cdots$$

Figure 4: The escape mechanism at work in $\mathsf{lf\_rle\_esc^*}$. The top row shows the portion of the string $s = s_1 s_2 \cdots s_t$ around s_j and the bottom row shows the corresponding encoding. Some of the values in the encoding are marked with ? since they depend on the forthcoming portion of the string. Note that the last runs of σ_1, σ_2, and σ_3 in s_{j-1} are escaped, but only σ_1 and σ_3 are totally escaped in s_j according to the definition given in the proof of Theorem 6.4. Since its re-entry point is inside s_j, σ_2 is not totally escaped in s_j.

Example 4 Consider the string $s = \cdots \sigma_1 \sigma_2^2 \sigma_3^3 \sigma_4^n \sigma_2^4 \sigma_3 \sigma_1 \sigma_2^5 \cdots$ over the alphabet $\Sigma = \{\sigma_1, \sigma_2, \sigma_3, \sigma_4\}$. If n is sufficiently large $\mathsf{lf\_rle\_esc}$ escapes the characters σ_1 and σ_3 and produces the output

$$\cdots \underbrace{\langle 1, \mathsf{esc}\rangle}_{\sigma_1} \underbrace{\langle 2, n+3\rangle}_{\sigma_2} \underbrace{\langle 3, \mathsf{esc}\rangle}_{\sigma_3} \underbrace{\langle \mathsf{esc}, 6, \sigma_1\rangle}_{\sigma_1 \text{ re-enter}} \underbrace{\langle \mathsf{esc}, 1, \sigma_3\rangle}_{\sigma_3 \text{ re-enter}} \underbrace{\langle 4, 1\rangle}_{\sigma_2} \cdots$$

(recall σ_4's occurrences are not explicitly encoded). $\mathsf{lf\_rle\_esc}$ cannot escape σ_2 since between the runs σ_2^2 and σ_2^4 there is no position o such that $s[o-1] > s[o]$. ■

Notice $\mathsf{lf\_rle\_esc}$ does not distinguish between cases in which $\langle k, \mathsf{esc}\rangle$ indicates a character does not occur again, as in $\mathsf{lf\_rle}$, and cases in which it indicates an escape sequence: the former is seen as an escape without a matching re-enter. Note also that the decoder can always distinguish a re-enter sequence from a normal pair $\langle k, \ell\rangle$, an escape/end-of-character pair $\langle k, \mathsf{esc}\rangle$, and an end-of-file pair $\langle \mathsf{esc}, \mathsf{esc}\rangle$. We define $\mathsf{A}_4 = \mathsf{lf\_rle\_esc} + \mathsf{Enc}^\delta$ as the algorithm that encodes the output of $\mathsf{lf\_rle\_esc}$ with Enc^δ.

Theorem 6.4 *Let* Enc *denote an integer encoder such that* $|\mathsf{Enc}(i)| \le a \log i + b$*. For any pair of positive constants* ϵ, δ *the algorithm* $\mathsf{A}_4 = \mathsf{lf\_rle\_esc} + \mathsf{Enc}^\delta$ *is locally* λ*-optimal for* $\lambda = \max(4a', a' + b + \epsilon)$*, where* $a' = a(1+\delta)$*.*

Proof: We need to prove that for any partition $s = s_1 s_2 \cdots s_t$ we have

$$|\mathsf{A}_4(s)| \le \max(4a', a' + b + \epsilon) \sum_{i=1}^{t} |s_i| H_0^*(s_i) + O(th \log h)\,. \tag{25}$$

Fix a partition $s = s_1 s_2 \cdots s_t$ and consider the algorithm $\mathsf{lf\_rle\_esc^*}$ that, instead of choosing at each step whether to escape or not, considers the possibility of escaping the symbol σ_i only if the characters $s[m]$ and $s[p]$ belong to two different partition elements (recall that whether $\mathsf{lf\_rle\_esc^*}$ actually escapes σ_i depends on the existence of a position o, such that $m < o < p$ and $s[o-1] > s[o]$). Let $\mathsf{A}_4^* = \mathsf{lf\_rle\_esc^*} + \mathsf{Enc}^\delta$. Since $\mathsf{lf\_rle\_esc}$ always performs the most economical choice, we have $|\mathsf{A}_4(s)| \le |\mathsf{A}_4^*(s)|$. We prove the theorem by showing that (25) holds with $\mathsf{A}_4(s)$ replaced by $\mathsf{A}_4^*(s)$.

As a preliminary, we establish that the escape mechanism in A_4^* yields an overhead of at most $O(th \log h)$ bits with respect to A_3. For $i = 1, \ldots, h-1$, let $s^{(i)}$ and $z^{(i)}$ be defined as in Theorem 6.3. We have already observed in the proof of Theorem 6.3 that, apart from $O(h)$ bits at Step 1, the output of $\mathsf{lf\_rle}$ consists of the lengths of the runs of zeros and ones in $z^{(i)}$ for $i = 1, \ldots, h-1$. Each pair $\langle k, \ell\rangle$ written at Step 3 of $\mathsf{lf\_rle}$ represents a run of σ_i in $s^{(i)}$—corresponding to a run of 1's in $z^{(i)}$—followed by a run of characters greater than σ_i—corresponding to a run of $\mathsf{0}$'s in $z^{(i)}$. For the algorithm $\mathsf{lf\_rle\_esc^*}$ the only difference is that, instead of the pair $\langle k, \ell \rangle$ sometimes we encode the escape sequence $\langle k, \mathsf{esc} \rangle$ later followed by the re-enter sequence $\langle \mathsf{esc}, \ell' + 1, \sigma_i \rangle$. Since $\ell' + 1 \leq \ell$ each escape sequence introduces at most a $O(\log h)$ bits overhead. Since by construction $\mathsf{lf\_rle\_esc^*}$ escapes at most th times we conclude that the escape mechanism introduces an overhead of at most $O(th \log h)$ bits with respect to the strategy of simply encoding all run lengths as in A_3.

Now we turn to analyzing the savings introduced by the escape mechanism. As in the proof of Theorem 6.3, we observe that for $i = 1, \ldots, h-1$, the partition $s = s_1 \cdots s_t$ naturally induces the partitions $s^{(i)} = s_1^{(i)} \cdots s_t^{(i)}$ and $z^{(i)} = z_1^{(i)} \cdots z_t^{(i)}$. We say that a character σ_i is *totally escaped* in s_j if the following three conditions hold simultaneously (see also Figure 4):

(e1) $z_j^{(i)} = \mathsf{0}^\ell$, that is, s_j contains only characters larger than σ_i;

(e2) the last run of σ_i's before the beginning of s_j produces an escape sequence;

(e3) the corresponding re-entry point is at a position $s[o]$ which is after the end of s_j.

The crucial observation is that, if σ_i is totally escaped in s_j, then the algorithm $\mathsf{lf\_rle\_esc^*}$ does not "pay" for the encoding of $z_j^{(i)}$. To see this, observe that $z_j^{(i)} = \mathsf{0}^{\ell_j}$ is a substring of a larger run $\mathsf{0}^{m_j}$ in $z^{(i)}$. Because of the escape mechanism, instead of the length m_j, $\mathsf{lf\_rle\_esc^*}$ only encodes a length n_j with $n_j \leq m_j - \ell_j$. So $\mathsf{lf\_rle\_esc^*}$ only pays for the encoding of a run $\mathsf{0}^{n_j}$ which starts *after* the end of s_j and pays nothing for the encoding of $z_j^{(i)} = \mathsf{0}^{\ell_j}$.

Let $U_j \subseteq \{\sigma_1, \ldots, \sigma_{h-1}\}$ denote the set of characters *not* totally escaped in s_j. Using the above observations, and reasoning as in the proof of Theorem 6.3, we have

$$|\mathsf{A}_4^*(s)| \leq \sum_{j=1}^{t} \sum_{i \in U_j} RLE(z_j^{(i)}) \; + \; O(th \log h) \;, \tag{26}$$

where RLE is defined by (16). We conclude the proof by showing that for $j = 1, \ldots, t$

$$\sum_{i \in U_j} RLE(z_j^{(i)}) \leq \max(4a', a' + b + \epsilon)|s_j| H_0^*(s_j) \; + \; O(h) \tag{27}$$

which combined with (26) proves (25). The crucial observation for proving (27) is that the set U_j contains at most one character σ_k such that $z_j^{(k)} = \mathsf{0}^{\ell_k}$. Indeed, if for both σ_i and σ_k it is $z_j^{(i)} = \mathsf{0}^{\ell_i}$ and $z_j^{(k)} = \mathsf{0}^{\ell_k}$ one of them—the one that occurs later after the end of s_j—will certainly be escaped. For example, if the first occurrence of σ_i (resp. σ_k) after the end of s_j is at position $s[p_i]$ (resp. $s[p_k]$) with $p_i > p_k$ then σ_i will be escaped at s_j. To see this, observe that condition (e1) trivially holds and conditions (e2)–(e3) hold as well since there is certainly a position o with $s[o-1] > s[o]$ between the end of s_j and $s[p_k]$ given that s_j contains only characters larger than $\sigma_k = s[p_k]$.

To prove (27) we follow closely the proof of Theorem 6.3. Let d_j denote the number of distinct characters appearing in s_j. If $d_j = h$ then (21) holds for every $i \in \{1, \ldots, h-1\}$, and (27) follows by Lemma 6.1. If $d_j = 1$ then (22) holds but, since U_j contains at most one character σ_k such that $z_j^{(k)} = \mathsf{0}^{\ell_k}$, it is $|U_j| \leq 2$ and therefore

$$\sum_{i \in U_j} RLE(z_j^{(i)}) \leq 2\, \mathsf{Enc}^\delta(|s_j|) \leq 2\,(a' \log |s_j| + b) \leq 2a'|s_j| H_0^*(s_j) + 2b.$$

Finally, if $1 < d_j < h$ we reason again as in the proof of Theorem 6.3. We define σ_e and W_j as in that proof and, instead of (23), we get

$$\sum_{i \in U_j} RLE(z_j^{(i)}) \leq \sum_{i \in U_j \,\wedge\, i \notin W_j} \Big(a'|z_j^{(i)}| H_0(z_j^{(i)}) + a' \log |z_j^{(i)}| + b\, \mathsf{runs}(z_j^{(i)}) \Big) + \sum_{i \in U_j \cap W_j} a' \log |z_j^{(i)}| + O(h).$$

Since U_j contains at most one character such that $z_j^{(i)} = \mathsf{0}^{\ell_i}$ it is $|U_j \cap W_j| \le 2$ so instead of (24) we have

$$\sum_{i \in U_j \cap W_j} a' \log |z_j^{(i)}| \le 2a' \log |z_j^{(e)}|$$

which plugged into the above inequality yields

$$\sum_{i \in U_j \wedge i \notin W_j} RLE(z_j^{(i)}) \le \sum_{i \in U_j} \left(a' |z_j^{(i)}| H_0(z_j^{(i)}) + 3a' \log |z_j^{(i)}| + b\, \mathsf{runs}(z_j^{(i)}) \right) \; + \; O(h)$$

and (27) follows by Lemmas 2.3 and 6.1. ■

Note that combining the above theorem with Lemma 2.1 we get that for any string s it is $|\mathsf{A}_4(\mathsf{bwt}(s))| \le \max(4a', a' + b + \epsilon)|s|H_k^*(s) + \log|s| + O(h^{k+1} \log h)$ for any $k \ge 0$. Finally, repeating verbatim the proof of Theorem 5.5 with $\mu = 1.105$, $\nu = \epsilon = \delta = 0.001$, we get a bound for the output size of lf_rle_esc followed by Order0*.

Theorem 6.5 *The algorithm* lf_rle_esc + Order0* *is locally* $(4.45 + C_0)$*-optimal.* ■

Corollary 6.6 *For any string* s *and* $k \ge 0$ *we have*

$$|\mathsf{Order0^*}(\mathsf{lf\_rle\_esc}(\mathsf{bwt}(s)))| \le (4.45 + C_0)|s|H_k^*(s) + \log|s| + O\left(h^{k+1} \log h\right).$$

Proof: Immediate by Theorem 6.5 and Lemma 2.1. ■

7 Lower bounds for entropy-only compression

In this section we show that no compression algorithm A can compress every string s in less than $2|s|H_0^*(s) + \Theta(1)$ bits. We prove this result assuming only that A is non-singular; that is, for any pair of strings $s_1 \ne s_2$ we have $A(s_1) \ne A(s_2)$. An immediate consequence of this result is that there cannot be an algorithm which is locally λ-optimal for $\lambda < 2$ (consider the trivial partition with $t = 1$ in Definition 2).

Theorem 7.1 *If* A *is a non-singular compressor, then the bound*

$$|A(s)| \le \lambda |s| H_0^*(s) + \eta \qquad \textit{for every string } s$$

can hold only with a constant $\lambda \ge 2$.

Proof: For $i = 1, 2, \ldots$ let T_i denote the set of binary strings such that $s \in T_i$ if and only if $2^{i-1} < |s| \le 2^i$ and s contains exactly one $\mathsf{1}$ and $(|s| - 1)$ $\mathsf{0}$'s. Elementary calculus shows that

$$|T_i| = \frac{2^i(2^i + 1)}{2} - \frac{2^{i-1}(2^{i-1} + 1)}{2} \ge \frac{3}{8} \cdot 4^i. \tag{28}$$

In addition, recalling that $t \ge 1$ implies $(1 + \frac{1}{t})^t < e$, for $s \in T_i$ it is

$$\begin{aligned} |A(s)| &\le \lambda |s| H_0^*(s) + \eta \\ &= \lambda \left(\log|s| + (|s| - 1) \log \left(\tfrac{|s|}{|s|-1} \right) \right) + \eta \\ &\le \lambda (\log 2^i + \log e) + \eta \\ &= \lambda\, i + \eta' \end{aligned} \tag{29}$$

with $\eta' = \eta + \lambda \log e$. Since there are at most $2^{z+1} - 1$ distinct binary codewords of length at most z, we have that less than

$$2^{\lambda i + \eta' + 1} = 2^{\eta' + 1} (2^\lambda)^i \tag{30}$$

are available for encoding the strings in T_i. Comparing (28) and (30) implies that, for sufficiently large i, if every $s \in T_i$ must be assigned a different codeword, then we must have $2^\lambda \ge 4$ and therefore $\lambda \ge 2$. ■

We now show that even $\lambda = 2$ is not achievable if A induces a uniquely decodable code over the set of all strings, that is, if there are no two sequences of strings $s_1, \ldots, s_t$ and $w_1, \ldots, w_k$ such that $A(s_1) \cdots A(s_t) = A(w_1) \cdots A(w_k)$.

Theorem 7.2 *If A induces a uniquely decodable code over the set of all strings, then the bound*

$$|A(s)| \leq \lambda |s| H_0^*(s) + \eta \qquad \textit{for every string } s$$

can hold only with a constant $\lambda > 2$.

Proof: Let T_i be defined as in the proof of Theorem 7.1. Since A is uniquely decodable it must satisfy the McMillan-Kraft Inequality [8, Sect. 5.5]. Applying this inequality to the set $\cup_{i \geq 1} T_i$ yields

$$\sum\nolimits_{i \geq 1} \sum\nolimits_{s \in T_i} 2^{-|A(s)|} \leq 1. \tag{31}$$

By (29) and (28) we get

$$\sum\nolimits_{i \geq 1} \sum\nolimits_{s \in T_i} 2^{-|A(s)|} \;\geq\; \sum\nolimits_{i \geq 1} 2^{-\lambda i - \eta'} |T_i| \;\geq\; \sum_{i > 0} 2^{-\eta'} \, \frac{3}{8} \left(\frac{4}{2^{\lambda}} \right)^{i}.$$

Hence, to satisfy (31) we must have $\lambda > 2$ as claimed. ∎

References

[1] J. Abel. Incremental frequency count—a post BWT-stage for the Burrows-Wheeler compression algorithm. *Software: Practice and Experience*, 37:247–265, 2007.

[2] Z. Arnavut. Inversion coding. *The Computer Journal*, 47:46–57, 2004.

[3] Z. Arnavut and S. Magliveras. Block sorting and compression. In *Proc. of IEEE Data Compression Conference (DCC)*, pages 181–190, 1997.

[4] B. Balkenhol, S. Kurtz, and Y. M. Shtarkov. Modification of the Burrows and Wheeler data compression algorithm. In *Proc. of IEEE Data Compression Conference (DCC)*, pages 188–197, 1999.

[5] J. Bentley, D. Sleator, R. Tarjan, and V. Wei. A locally adaptive data compression scheme. *Communications of the ACM*, 29(4):320–330, 1986.

[6] E. Binder. Distance coder, 2000. Usenet group: comp.compression, http://groups.google.com/group/comp.compression/msg/27d46abca0799d12.

[7] M. Burrows and D. Wheeler. A block-sorting lossless data compression algorithm. Technical Report 124, Digital Equipment Corporation, 1994.

[8] T. M. Cover and J. A. Thomas. *Elements of Information Theory.* Wiley Interscience, 1991.

[9] S. Deorowicz. Second step algorithms in the Burrows-Wheeler compression algorithm. *Software: Practice and Experience*, 32(2):99–111, 2002.

[10] P. Elias. Universal codeword sets and representations of the integers. *IEEE Transactions on Information Theory*, 21(2):194–203, 1975.

[11] P. Fenwick. Burrows-Wheeler compression with variable length integer codes. *Software: Practice and Experience*, 32:1307–1316, 2002.

[12] P. Ferragina, R. Giancarlo, and G. Manzini. The engineering of a compression boosting library: Theory vs practice in BWT compression. In *Proc. 14th European Symposium on Algorithms (ESA)*, Lecture Notes in Computer Science vol. 4168, pages 756–767. Springer, 2006.

[13] P. Ferragina, R. Giancarlo, and G. Manzini. The myriad virtues of wavelet trees. *Information and Computation*, pages 849–866, 2009.

[14] P. Ferragina, R. Giancarlo, G. Manzini, and M. Sciortino. Boosting textual compression in optimal linear time. *Journal of the ACM*, 52:688–713, 2005.

[15] P. Ferragina, G. Manzini, and S. Muthukrishnan (Eds). Special issue on the Burrows-Wheeler transform and its applications. *Theoretical Computer Science*, 387(3), 2007.

[16] L. Foschini, R. Grossi, A. Gupta, and J. Vitter. When indexing equals compression: Experiments on compressing suffix arrays and applications. *ACM Transactions on Algorithms*, 2:611–639, 2006.

[17] R. Grossi, A. Gupta, and J. Vitter. High-order entropy-compressed text indexes. In *Proc. 14th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 841–850, 2003.

[18] A. Gupta, R. Grossi, and J. Vitter. Nearly tight bounds on the encoding length of the Burrows-Wheeler transform. In *Proc. ALENEX/ANALCO '08*, pages 191–202, 2008.

[19] H. Kaplan, S. Landau, and E. Verbin. A simpler analysis of Burrows-Wheeler based compression. *Theoretical Computer Science*, 387:220–235, 2007.

[20] H. Kaplan and E. Verbin. Most Burrows-Wheeler based compressors are not optimal. In *Proc. of the 18th Symposium on Combinatorial Pattern Matching (CPM '07)*, LNCS 4580, pages 107–118. Springer-Verlag, 2007.

[21] V. Mäkinen and G. Navarro. Succinct suffix arrays based on run-length encoding. *Nordic Journal of Computing*, 12(1):40–66, 2005.

[22] V. Mäkinen and G. Navarro. Implicit compression boosting with applications to self-indexing. In *Proc. of the 14th Symposium on String Processing and Information Retrieval (SPIRE '07)*, pages 214–226. Springer-Verlag LNCS n. 4726, 2007.

[23] G. Manzini. An analysis of the Burrows-Wheeler transform. *Journal of the ACM*, 48(3):407–430, 2001.

[24] G. Navarro and V. Mäkinen. Compressed full-text indexes. *ACM Computing Surveys*, 39(1), 2007.

[25] B. Y. Ryabko. Data compression by means of a 'book stack'. *Prob.Inf.Transm*, 16(4):16–21, 1980.

[26] E. Yang and Y. Jia. Universal lossless coding of sources with large or unbounded alphabets. In *Numbers, Information and Complexity,* I. Althöfer *et al.*, editors. Kluwer, 2000.

A Empirical entropies

For any length-k word $w \in \Sigma^k$, 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

$$H_k(s) = \frac{1}{|s|} \sum_{w \in \Sigma^k} |w_s| H_0(w_s)\,.$$

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

Starting from H_0^* we define the k-th order modified empirical entropy H_k^* using a formula similar to the one above. Unfortunately, if we simply replace H_0 with H_0^*, then the resulting entropy H_k^* does not satisfy the inequality $H_{k+1}^*(s) \leq H_k^*(s)$ for every string s. In other words, when H_0 is replaced by H_0^*, the use of a longer context for the prediction of the next symbol does not always yield an increase in compression. For this reason, we define H_k^* 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). We use the following notation. Let $\mathcal{S}_k$ denote the set of all k-letter substrings of s. 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

$$H_k^*(s) = \min_{\mathcal{Q} \preceq \mathcal{S}_k} \left\{ \frac{1}{|s|} \sum_{w \in \mathcal{Q}} |w_s| H_0^*(w_s) \right\}. \tag{32}$$

It is straightforward to verify that with the above definition $H_{k+1}^*(s) \leq H_k^*(s)$ for every string s. The following lemma establishes an useful lower bound for $H_k^*(s)$.

Lemma A.1 *For any string s and $k \geq 0$ it is*

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

Proof: Let $\mathcal{Q} \preceq \mathcal{S}_k$ denote the subset for which the minimum (32) is achieved. It is

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

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

$$|s|H_k^*(s) \;\geq\; \log\Big(\sum\nolimits_{w \in \mathcal{Q}} |w_s|\Big) \;\geq\; \log(|s| - k).$$

■

B Proofs of the technical lemmas

Proof of Lemma 2.3: First suppose $\mathsf{runs}(s) \leq 2\alpha/\epsilon + 2 = O(1)$; since $\log|s| \leq |s|H_0^*(s)$ we have

$$\alpha \log|s| + \beta \mathsf{runs}(s) \leq \alpha |s| H_0^*(s) + O(1). \tag{33}$$

Now suppose $\mathsf{runs}(s) > 2\alpha/\epsilon + 2$. This assumption implies that the frequency of the most common character in s is at most $|s| - \lfloor \mathsf{runs}(s)/2 \rfloor < |s| - \alpha/\epsilon$, with equality if and only if all odd-numbered runs contain the same character and every even-numbered run has length 1. Since $H_0(s)$ is minimized when the distribution of characters is as skewed as possible, we have

$$\begin{aligned} |s|H_0(s) \;&>\; (|s| - \alpha/\epsilon) \log\left(\frac{|s|}{|s| - \alpha/\epsilon}\right) + (\alpha/\epsilon) \log\left(\frac{|s|}{\alpha/\epsilon}\right) \\ &\geq\; (\alpha/\epsilon) \log\left(\frac{|s|}{\alpha/\epsilon}\right) = (\alpha/\epsilon)\log|s| - O(1)\,, \end{aligned}$$

so $\log|s| \leq (\epsilon/\alpha)|s|H_0(s) + O(1)$. Combining this inequality with Lemma 2.2 we get

$$\alpha \log|s| + \beta \mathsf{runs}(s) \leq (\beta + \epsilon)|s| H_0^*(s) + O(1)$$

which, together with (33) proves the lemma. ■

Proof of Lemma 3.1: Using elementary calculus it is easy to show that $\mathsf{Enc}(x_1+x_2) \le \mathsf{Enc}(x_1)+\mathsf{Enc}(x_2)$ whenever $\min(x_1, x_2) \ge 2$. Hence we need only take care of the case in which some of the x_j's are 1. For $x \ge 1$ we have:

$$\begin{aligned} |\mathsf{Enc}(x+1)| - |\mathsf{Enc}(x)| - |\mathsf{Enc}(1)| &= a\log(1+(1/x)) - b & (34)\\ &= (a\log e)\ln(1+(1/x)) \;-\; b \\ &\le (a\log e)/x - b, & (35) \end{aligned}$$

where the last inequality holds since $t \ge 0$ implies $\ln(1+t) \le t$. Let $c_{ab} = (a\log e)/b$. From (34) we get that $x \ge 1$ implies $\mathsf{Enc}(x+1) \le \mathsf{Enc}(x) + \mathsf{Enc}(1) + (a-b)$ and from (35) we get that $x \ge c_{ab}$ implies $\mathsf{Enc}(x+1) \le \mathsf{Enc}(x) + \mathsf{Enc}(1)$. Combining these inequalities we get

$$\left|\mathsf{Enc}\Big(\textstyle\sum_{j=1}^k x_j\Big)\right| \;\le\; \Big(\textstyle\sum_{j=1}^k |\mathsf{Enc}(x_j)|\Big) + c_{ab}\max(a-b,0),$$

and the lemma follows with $d_{ab} = c_{ab}\max(a-b,0)$. ∎

Proof of Lemma 6.1: For $i = 1, 2, \ldots, h$ let n_i denote the number of occurrences of σ_i in s and let $w_i = n_i + \cdots + n_h$. Note that $|z^{(i)}| = n_i + w_{i+1} = w_i$. We have

$$\begin{aligned} \sum_{i=1}^{h-1} |z^{(i)}|H_0(z^{(i)}) &= \sum_{i=1}^{h-1} \left[n_i \log\left(w_i/n_i\right) + w_{i+1}\log\left(w_i/w_{i+1}\right)\right] \\ &= \sum_{i=1}^{h-1} \left[w_i\log(w_i) - n_i\log(n_i) - w_{i+1}\log(w_{i+1})\right] \\ &= w_1\log(w_1) - \sum_{i=1}^{h-1} n_i\log(n_i) \;-\; w_h\log(w_h) \\ &= |s|\log|s| - \sum_{i=1}^{h} n_i\log(n_i) = |s|H_0(s). \end{aligned}$$

∎

Proof of Lemma 6.2: Observe first that $m = \mathsf{runs}(z)$. Assume $\mathsf{1}$ is the less frequent symbol (otherwise the proof is symmetrical) and let n_1 denote the number of occurrences of $\mathsf{1}$ in z. We distinguish three cases according to the size of n_1.

Case $n_1 = 0$. We have $m = 1$, $z = \mathsf{0}^{\ell_1}$ and $RLE(z) = |\mathsf{Enc}^{\delta}(\ell_1)| \le a'\log\ell_1 + b$.

Case $1 \le n_1 \le (|z|/e)$. We prove that $RLE(z) = \sum_{i=1}^m |\mathsf{Enc}^{\delta}(\ell_i)| \le a'|z|H_0(z) + bm$ assuming that m is even. If m is odd, the cost $|\mathsf{Enc}^{\delta}(\ell_m)|$ of the last run is accounted for by the term $a'\log|z|$ in the statement of the lemma. We have

$$RLE(z) = \sum_{i=1}^{m} |\mathsf{Enc}^{\delta}(\ell_i)| \le \sum_{i=1}^{m} a'\log\ell_i \;+\; bm. \qquad (36)$$

Let t denote the number of nonzero logarithms in (36) (that is, we do not count the logarithms for which $\ell_i = 1$). We show that $t \le n_1$ by charging each nonzero logarithm to a different $\mathsf{1}$ in z as follows. For $k = 1, \ldots, m/2$, if $\ell_{2k} > 1$ we charge both $\log(\ell_{2k-1})$ and $\log(\ell_{2k})$ to the the ones in $\mathsf{1}^{\ell_{2k}}$; if $\ell_{2k} = 1$ then $\log(\ell_{2k})$ is zero and we charge $\log(\ell_{2k-1})$ to the single $\mathsf{1}$ in $\mathsf{1}^{\ell_{2k}}$. Using Jensen's inequality and the fact that the function $x\log(|z|/x)$ is increasing for $x \le n_1 \le (|z|/e)$ we get

$$\sum_{i=1}^{m}\log(\ell_i) = \sum_{\ell_i>1}\log(\ell_i) \le t\log\left(\frac{\sum_{\ell_i>1}\ell_i}{t}\right) \le t\log(|z|/t) \le n_1\log(|z|/n_1) \le |s|H_0(s).$$

Combining the above inequality with (36) yields the thesis.

Case $(|z|/e) < n_1 \leq (|z|/2)$. From Jensen's inequality we get

$$\sum_{i=1}^{m} |\mathsf{Enc}^{\delta}(\ell_i)| = \sum_{i=1}^{m} a' \log \ell_i \; + \; bm \;\leq\; a'm \log(|z|/m) \; + \; bm\,.$$

Since the function $x \log(|z|/x)$ has its maximum for $x = (|z|/e)$ the above inequality becomes

$$\sum_{i=1}^{m} |\mathsf{Enc}^{\delta}(\ell_i)| \leq a'|z|(\log e)/e + bm.$$

The lemma follows since the hypothesis $(|z|/2) \geq n_1 > (|z|/e)$ implies $|z|H_0(z) \geq |z|(\log e)/e$. ∎