Electronic Colloquium on Computational Complexity, Report No. 141 (2018)

Local Decodability of the Burrows-Wheeler Transform

Sandip Sinha
Columbia University
sandip@cs.columbia.edu

Omri Weinstein
Columbia University
omri@cs.columbia.edu

Abstract

The Burrows-Wheeler Transform (BWT) is among the most influential discoveries in text compression and DNA storage. It is a *reversible* preprocessing step that rearranges an n-letter string into runs of identical characters (by exploiting context regularities), resulting in highly compressible strings, and is the basis for the ubiquitous `bzip` program. Alas, the decoding process of BWT is inherently sequential and requires $\Omega(n)$ time even to retrieve a *single* character.

We study the succinct data structure problem of locally decoding short substrings of a given text under its *compressed* BWT, i.e., with small redundancy r over the *Move-To-Front* based (`bzip`) compression. The celebrated BWT-based FM-index (FOCS '00), and other related literature, gravitate toward a tradeoff of $r = \tilde{O}(n/\sqrt{t})$ bits, when a single character is to be decoded in $O(t)$ time. We give a near-quadratic improvement $r = \tilde{O}(n \cdot \lg t/t)$. As a by-product, we obtain an *exponential* (in t) improvement on the redundancy of the FM-index for counting pattern-matches on compressed text. In the interesting regime where the text compresses to $n^{1-o(1)}$ bits, these results provide an $\exp(t)$ *overall* space reduction. For the local decoding problem, we also prove an $\Omega(n/t^2)$ cell-probe lower bound for "symmetric" data structures.

We achieve our main result by designing a compressed Rank (partial-sums) data structure over BWT. The key component is a locally-decodable Move-to-Front (MTF) code: with only $O(1)$ extra bits per block of length $n^{\Omega(1)}$, the decoding time of a single character can be decreased from $\Omega(n)$ to $O(\lg n)$. This result is of independent interest in algorithmic information theory.

ISSN 1433-8092

1 Introduction

Exploiting text regularities for data compression is an enterprise that received a tremendous amount of attention in the past several decades, driven by large-scale digital storage. Motivated by this question, in 1994 Burrows and Wheeler [6] proposed a preprocessing step that 'tends' to rearrange strings into more 'compressible form': Given $x \in \Sigma^n$, generate the $n \times n$ matrix whose rows are all *cyclic shifts* of x, sort these rows lexicographically, and output the last *column* L of the matrix. The string L is called the Burrows-Wheeler Transform (BWT) of x. Note that L is a *permutation* of the original string x, as each row of the sorted matrix is still a unique cyclic shift. The main observation is that, in the permuted string L, characters with identical *context*[1] appear consecutively, hence if individual characters in the original text x tend to be predicted by a reasonably small context (e.g., as in English texts), then the string $L := \mathsf{BWT}(x)$ will exhibit local similarity, namely, identical symbols will tend to recur at *close vicinity*. (For more elaboration on the properties of BWT and the role it plays in bioinformatics and information retrieval, we refer the reader to [1] and references therein). This property suggests a natural way to compress L, using a relative *recency coding* method, whereby each symbol in L is replaced by the *number of distinct symbols that appeared since its last occurrence*. Indeed, since in L symbols have local similarity, i.e., tend to recur at close vicinity, we expect the output string to consist mainly of *small integers* (and in particular, 0-runs) and hence much cheaper to encode. This relative encoding, known as the *Move-to-Front* transform [3], followed by run-length coding of 0-runs and an arithmetic coding stage (henceforth denoted $\mathtt{RLX}(L)$), forms the basis for the `bzip2` program [35]. The `RLX` ("bzip") compression benchmark was justified both theoretically and empirically [9, 19, 23], where among other properties, it was shown to approximately converge to *any finite-order* empirical entropy H_k (see Section 2.2.2 for the formal definitions and comparative analysis against other compressors).

The most remarkable property of the Burrows-Wheeler transform is that it is *invertible*. The crux of the decoding process is the fact that the transform preserves the *order* of occurrences (a.k.a *rank*) of any given character in both the first column and last column (L) of the BWT matrix. This crucial fact facilitates an iterative decoding process, whereby, given the decoded position of x_{i+1} in L, one can decode the previous character x_i using $O(|\Sigma|)$ $\textsc{Rank}$[2] queries *to* L (see Section 2.1.1 below for the formal decoding algorithm). Alas, this decoding process is inherently *sequential*, and therefore requires $\Omega(n)$ time to decode even a single coordinate of x [20, 24]. In fact, no sub-linear decoding algorithm for x_i is known even if L is stored in *uncompressed* form.

This is an obvious drawback of the Burrows-Wheeler transform, as many storage applications, such as genetic sequencing and alignment, need local searching capabilities inside the compressed database [29]. For example, if $x^1, x^2, \ldots, x^m \in \Sigma^n$ is a collection of m separate files with very similar contexts, e.g., DNA sequences, then we expect $|\mathtt{RLX}(L_{x^1 \circ x^2 \circ \ldots \circ x^m})| \ll \sum_{j=1}^{m} |\mathtt{RLX}(L_{x^j})|$, but jointly compressing the files would have the drawback that, when the application needs to retrieve only a single file x^j, it will need to spend $\Omega(n \cdot m)$ I/Os (instead of $O(n)$) to invert $\mathsf{BWT}(x^1 \circ x^2 \circ \ldots \circ x^m)$. The main question we address in this paper is, whether small *additive* space redundancy (over the compressed BWT string ($|\mathtt{RLX}(\mathsf{BWT}(x))|$) can be used to speed up the decoding time of a single character (or a short substring) of the original text, in the word-RAM model :

> **Problem 1.** *What is the least amount of space redundancy $r = r(t)$ needed beyond the compressed $\mathsf{BWT}(x)$ string, so that each coordinate x_i can be decoded in time t ?*

Since $|\mathtt{RLX}(\mathsf{BWT}(x))|$ approaches *any* finite-order empirical entropy $H_k(x)$ (see Section 2.2.2),

[1] The *k-context* of a character x_i in x is the k consecutive characters that precede it.

[2] The $\textsc{Rank}$ of a character $x_i \in x$ is the number of occurrences of identical symbols in x that precede it : $\textsc{Rank}(x, i) := |\{j \leq i \ : x_j = x_i\}|$. See also Section 2.

this data structure problem can be viewed as the *succinct dictionary*[3] problem under *infinite-order* entropy space benchmark. A long line of work has been devoted to succinct dictionaries in the *context-free* regime, i.e., when the information theoretic space benchmark is the *zeroth-order* empirical entropy $H_0(x) := \sum_{c\in\Sigma} n_c \lg \frac{n}{n_c}$ of marginal frequencies (e.g., [17, 30, 31] to mention a few). However, as discussed below, much less is known about the best possible trade-off under higher order entropy benchmarks which take context (i.e., correlation between x_i's) into account.

A related but incomparable data structure problem to Problem 1, is that of compressed pattern-matching in strings, a.k.a *full-text indexing*, where the goal is to succinctly represent a text in *compressed* form as before, so that all $occ(p)$ occurrences of a pattern $p \in \Sigma^\ell$ in x can be reported, or more modestly, counted, in near-optimal time $O(\ell+occ(p))$. The celebrated BWT-based compressed text index of Ferragina and Manzini [13], commonly known as the *FM-index*, achieves the latter task using $|\texttt{RLX}(L)| + O(n/\lg n)$ bits of space[4], and despite a long line of subsequent developments in the field, remains one of the most prominent tools in pattern matching for bioinformatics and text applications (see [29] and references therein). The core of the FM-index is a compressed data structure that computes RANK queries over the (compressed) BWT string L in constant time and redundancy $r = O(n/\lg n)$ bits, and more generally, in time t' and redundancy $r = \tilde{\Theta}(n/t')$. Their data structure, combined with simple "marking index"[5] of the BWT permutation on blocks of length t, yields a solution to Problem 1 with overall redundancy $\tilde{O}(n/t + n/t')$ and time $O(t \cdot t')$ (as simulating each sequential step of the BWT decoder requires $O(1)$ rank queries, and there are t coordinates per block). In other words, if the desired decoding time of a coordinate is t, [13] gives redundancy $r = \tilde{O}(n/\sqrt{t})$ over $\texttt{RLX}$. In fact, even for *randomized* succinct dictionaries approaching $H_k(x)$ bits of space, the best known trade-off is $r = \Theta(1/\sqrt{t})$: Dutta et al. [8] gave a randomized data structure with space $(1+\varepsilon)\tilde{H}_k(x)$,[6] and expected decoding time $\Theta(1/\varepsilon^2)$ to retrieve each x_i. Our first main result is a near-quadratic improvement over this trade-off for Problem 1:

Theorem 1 (Local Decoding of BWT, Informal)**.** *For any t and any string $x \in \Sigma^n$, there is a succinct data structure that stores $|\textit{RLX}(\mathsf{BWT}(x))| + \tilde{O}\left(\frac{n\lg t}{t}\right) + n^{0.9}$ bits of space, so that each coordinate x_i can be retrieved in $O(t)$ time, in the word-RAM model with word size $w = \Theta(\lg n)$.*

Our data structure directly implies that a contiguous substring of size ℓ of x can be decoded in time $O(t+\ell)$ without increasing the space redundancy. It is noteworthy that achieving a linear trade-off as above between time and redundancy with respect to *zeroth order* entropy ($H_0(x)$) is trivial, by dividing x into n/t blocks of length t, and compressing each block using Huffman (or arithmetic) codes, as this solution would lose at most 1 bit per block. However, this solution fails miserably with respect to higher-order entropy benchmarks H_k (in fact, even against H_1), since in the presence of contexts, the loss in compressing each block *separately* can be arbitrarily large (e.g., $H_0\left((ab)^{n/2}\right) = n$ but $H_1\left((ab)^{n/2}\right) = \lg n$). This example illustrates the qualitative difference between the DICTIONARY problem in the independent vs. correlated setting. In fact, we prove a complementary cell-probe lower bound of $r \geq \Omega(n/t^2)$ on Problem 1 for "symmetric" data structures, which further decode the dispositions of any x_i in $L = \mathsf{BWT}(x)$ and vice versa, in time $O(t)$ (see Theorem 3.2 below). While removing this (natural) restriction remains an interesting open question, this result provides a significant first step in understanding the cell-probe complexity of Problem 1, more on this below.

Our second main result, which is a by-product of our central data structure, is an *exponential* improvement (in t) on the redundancy of the FM-index for compressed pattern-matching counting:

[3] For a string $x \in \Sigma^n$ and an index $i \in [n]$, DICTIONARY(x, i) returns x_i, the i^{th} character in x.

[4] For *reporting* queries, [13] requires significantly larger space $|\texttt{RLX}(L)| \cdot \lg^\varepsilon n + O(n/\lg^{1-\varepsilon} n)$ for some $\varepsilon \in (0,1)$.

[5] I.e., recording shortcut pointers to the location of x_i in L for every block $i \in [j \cdot t]$, $j \in [n/t]$, see Section 3.

[6] Here, $\tilde{H}_k$ denotes the Lempel-Ziv [41] codeword $|\mathsf{LZ78}(x)|$ (up to $o(n)$ additive terms), hence $\lim_{n\to\infty} \tilde{H}_k/n = H_k$.

Theorem 2. *There is a small constant $\delta > 0$ such that for any $x \in \Sigma^n$ and any $t \leq \delta \lg n$, there is a compressed index using $|\mathtt{RLX}(\mathsf{BWT}(x))| + n \lg n/2^t + n^{1-\Omega_\delta(1)}$ bits of space, counting the number of occurrences of any pattern $p \in \Sigma^\ell$ in time $O(t\ell + occ(p))$ time.*

To the best of our knowledge, Theorem 2 provides the first compressed text index for pattern-matching counting queries, that can provably go below the $\tilde{\Omega}(n/\lg n)$ space barrier while maintaining near-optimal query time. In particular, it implies that at the modest increase of query time by a $O(\lg \lg n)$ factor, *any* $n/\mathrm{poly} \lg n$ redundancy is achievable. In the interesting setting of compressed pattern-matching, where the text compresses to $n^{1-o(1)}$ bits (e.g., $H_k(x) = n/\lg^{O(1)} n$), this result provides an exponential (in t) *overall* (i.e., multiplicative) space reduction over the FM-index. Compressed string-matching in the the $o(1)$ per-bit entropy regime was advocated in the seminal work of Farach and Thorup, see [10] and references therein. For *reporting* queries, we obtain a quadratic improvement over the FM-index, similar to Theorem 1 (see Section 6).

The main ingredient of both Theorem 1 and 2 is a new succinct data structure for computing Rank queries over the compressed BWT string $L = \mathsf{BWT}(x)$, with *exponentially* small redundancy $r \approx n/2^t$ with respect to $|\mathtt{RLX}(L)|$ (see Theorem 3.1 below). Our data structure builds upon and is inspired by the work of Pătraşcu [31], who showed a similar exponential trade-off for the Rank problem, with respect to the *zeroth order* entropy $H_0(L)$, i.e., in the *context-free* setting. In that sense, our work can be viewed as a certain higher-order entropy analogue of [31].

The most challenging part of our data structure is dealing with the *Move-to-Front* (MTF) encoding of L. This adaptive coding method comes at a substantial price: decoding the ith character from its encoding $\mathsf{MTF}(x)_i$ requires the decoder to know the current "state" $S_i \in \mathcal{S}_{|\Sigma|}$ of the encoder, namely, the precise *order* of recently occurring symbols, which itself depends on the *entire history* $x_{<i}$. This feature of the MTF transform, that the codebook itself is *dynamic*, is a qualitative difference from other compression schemes such as Huffman coding or (non-adaptive) arithmetic codes, in which the codebook is *fixed*. In fact, in that sense the MTF transform is conceptually closer to the BWT transform itself than to Huffman or arithmetic codes, since in both transforms, decoding the ith character is a sequential function of the decoded values of previous characters. Fortunately, it turns out that MTF has a certain local property that can be leveraged during preprocessing time, leading to the following key result:

Theorem 3 (Locally-decodable MTF, Informal)**.** *For any string $x \in \Sigma^n$, there is a succinct data structure that encodes x using at most $H_0(\mathsf{MTF}(x)) + \frac{n}{(\lg(n)/t)^t} + n^{0.9}$ bits of space, such that x_i can be decoded in time $O(t)$. Moreover, it supports* Rank *queries with the same parameters.*

The additive $n^{0.9}$ term stems from the storage space of certain look-up tables, which are *shared* across n/r blocks of size $r = (\lg(n)/t)^t$. Hence these tables occupy $o(1)$ bits per block in an amortized sense. Theorem 3 therefore has the surprising corollary that, with only $O(1)$ bits of redundancy per block (even with respect to the MTF code *followed* by arithmetic coding), decoding time can be reduced from $\Omega(n)$ to $O(\lg n)$.

Techniques. Our techniques are substantially different from those used in previous compressed text indexing literature, and in particular from the work of [13]. At a high-level, our main result uses a combination of succinct Predecessor search (under some appropriate encoding) along with ad-hoc design of two "labeled" B-tree data structures, known as *augmented aB-trees* (see Section 2.3 below). We then use a theorem of Pătraşcu [31] to *compress* these aB-trees, resulting in succinct representations while preserving the query time. Our main tree structure relies on a new local preprocessing step of the MTF codeword, which in turn facilitates a *binary-search* algorithm for

the dynamic MTF codebook (stack state). We show that this algorithm can be 'embedded' as a (compressed) augmented aB-tree. Our cell-probe lower bound (Theorem 3.2) relies on a new "entropy polarization" lemma for BWT permutations, combined with a 'nonuniform' adaptation of Golynski's cell-elimination technique for the succinct permutations problem [16].

1.1 Related Work

A large body of work has been devoted to succinct data structures efficiently supporting RANK/SELECT and (easier) DICTIONARY queries under *zeroth-order* empirical entropy, i.e., using $H_0(x)+o(n)$ bits of space. In this "*context-free*" regime, early results showed a near-linear $r = \tilde{O}(n/t)$ trade-off between time and redundancy (e.g., [5,17,30]), and this was shown by Miltersen [26] to be optimal for *systematic*[7] data structures. This line of work culminated with a surprising result of Pătraşcu [31], who showed that an *exponential* trade-off between time and redundancy can be achieved using *non-systematic* data structures in the word-RAM model, supporting all the aforementioned operations in query time $O(t)$ and $s \approx H_0(x) + O\left(n/(\lg(n)/t)^t\right)$ bits of space. For RANK/SELECT, this trade-off was shown to be optimal in the cell-probe model with word-size $w = \Theta(\lg n)$ [33], while for DICTIONARY (and MEMBERSHIP) queries, the problem is still open [7, 37, 38]. There are known string-matching data structures, based on context-free grammar compression (e.g., LZ or SLPs [4]), that achieve logarithmic query time for DICTIONARY queries, at the price of linear (but *not* succinct) space in the compressed codeword [4, 8]. However, these data structures have an $O(n/\lg n)$ additive space term, regardless of the compressed codeword, which becomes dominant in the $o(1)$ per-bit entropy regime, the interesting setting for this paper (see [10] for elaboration).

The problem of compressed pattern matching has been an active field of research for over four decades, since the works of McCreight [25] and Manber and Myers [22], who introduced suffix trees and suffix arrays. Ferragina and Manzini [13] were the first to achieve a *compressed* text index (with respect to higher-order entropy H_k), supporting pattern-matching counting and reporting queries in *sublinear* ($o(n)$) space and essentially optimal query time. Their BWT-based data structure, known as the *FM-index*, is still widely used in both theory and practice, and its applications in genomics go well beyond the scope of this paper [36]. Subsequent works, e.g. [10,18,34], designed compressed text indices under other entropy-coding space benchmarks, such as Lempel-Ziv compression, but to the best of our knowledge, all of them again require $\Omega(n/\lg n)$ bits of space, even when the text itself compresses to $o(n/\lg n)$ bits. We remark that for *systematic* data structures, linear trade-off ($r = \tilde{\Theta}(n/t)$) is the best possible for counting pattern-matches [15, 16], hence Theorem 2 provides an exponential separation between systematic and non-systematic data structures for this problem. For a more complete state of affairs on compressed text indexing, we refer the reader to [13, 29].

Another related problem to our work is the *succinct permutations* problem [16,28,39,40], where the goal is to succinctly represent a permutation $\pi \in S_n$ using $\lg n! + r$ bits of space, supporting evaluation ($\pi(i)$) and possibly inverse ($\pi^{-1}(i)$) queries in time t and q respectively. For the latter problem, an essentially tight trade-off $r = \Theta(n \lg n/tq)$ is known in the regime $t, q \in \tilde{\Theta}(\lg n)$ [16,28].

Organization. We start with some necessary background and preliminaries in Section 2. Section 3 provides a high-level technical overview of our main results. Sections 4,5 contain our main data structure and proofs of Theorems 1,2 and 3. Section 6 describes application to improved pattern-matching (reporting) queries. In Section 7 we prove the cell-probe lower bound (Theorem 3.2).

[7] Systematic data structures are forced to store the *raw* input database x, followed by an r-bit additional *index*.

2 Background and Preliminaries

For an n-letter string $x \in \Sigma^n$, let n_c be the number of occurrences, i.e., the *frequency*, of the symbol $c \in \Sigma$ in x. For $1 \leq i < j \leq n$, let $x[i:j]$ denote the substring $(x_i, x_{i+1}, \cdots, x_j)$. For convenience, we use the shorthand $x_{<i}$ to denote the prefix $(x_1, x_2, \cdots, x_{i-1})$. The k^{th} *context* of a character x_i in x is the substring of length k that precedes it. A *run* in a string x is a maximal substring of repetitions of the same symbol. For a compression algorithm $\mathcal{A}$, we denote by $|\mathcal{A}(x)|$ the output size *in bits*. The *zeroth order empirical entropy* of the string x is $H_0(x) := \sum_{c \in \Sigma} n_c \lg \frac{n}{n_c}$ (all logarithms throughout the paper are base-2, where by standard convention, $0 \lg 0 = 0$). It holds that $0 \leq H_0(x) \leq n \lg |\Sigma|$. For a substring $y \in \Sigma^k$, let y_x denote the concatenated string consisting of the single characters following all occurrences of y in x. The k^{th} *order empirical entropy* of x is defined as $H_k(x) := \sum_{y \in \Sigma^k} H_0(y_x)$. This prior-free measure intuitively captures "conditional" entropies of characters in correlated strings with bounded context, and is a lower bound on the compression size $|\mathcal{A}(x)|$ of any k-local compressor $\mathcal{A}$, i.e., any compressor that encodes each symbol with a code that only depends on the symbol itself and on the k immediately preceding symbols; for elaboration see [23]. For all $k \geq 0$, we have $H_{k+1}(x) \leq H_k(x)$. Note that the space benchmark H_k can be significantly smaller than H_0. For example, for $x = (ab)^{n/2}$, $H_0(x) = n$ but $H_k(x) = 0$ for any $k \geq 1$ (assuming the length n is known in advance; $H_k(x) \leq \lg n$ otherwise). For a random variable $X \sim \mu$, $H(X)$ denotes the Shannon entropy of X. Throughout the paper, we assume the original alphabet size is $|\Sigma| = O(1)$.

Succinct data structures. We work in the word-RAM model of word-length $w = \Theta(\lg n)$, in which arithmetic and shift operations on memory words require $O(1)$ time. A *succinct data structure* for an input $x \in \Sigma^n$ is a data structure that stores a small *additive* space overhead $r = o(n)$ beyond the "information-theoretic minimum" space $h(x)$ required to represent x, while supporting queries efficiently. In the "prior-free" setting, $h(x)$ is usually defined in terms of empirical entropy $H_k(x)$. The space overhead r is called the *redundancy*, and is measured in *bits*.

2.1 The Burrows-Wheeler Transform

Given a string $x \in \Sigma^n$, the Burrows-Wheeler Transform of x, denoted $\mathsf{BWT}(x)$, is defined by the following process. We append a unique end-of-string symbol '\$', which is lexicographically smaller than any character in Σ, to x to get $x\$$ (without this technicality, invertibility is only up to cyclic shifts). We place all $n+1$ cyclic shifts of the string $x\$$ as the rows of an $(n+1) \times (n+1)$ matrix, denoted by $\widehat{\mathcal{M}}$. Then we sort the rows of $\widehat{\mathcal{M}}$ in lexicographic order. The sorted matrix, denoted $\mathcal{M}$, is henceforth called the "BWT matrix" of x. Finally, we output $L \in (\Sigma \cup \{\$\})^{n+1}$, the last column of the BWT matrix $\mathcal{M}$. We henceforth use the shorthand $L := \mathsf{BWT}(x)$.

We observe that every column in $\mathcal{M}$ is a permutation of $x\$$. Let F and L be the *first* and *last column* of $\mathcal{M}$ respectively. See an example in Figure 2.1 below. For ease of notation, we shall refer to x\$ as x, denote its length by n, and include \$ in Σ.

2.1.1 Decoding BWT and the "LF Mapping"

While not obvious at first glance, BWT is an *invertible* transformation. An important first observation for this fact is that the *first* column F of the BWT matrix $\mathcal{M}$ is actually known "for free" (as long as the frequencies of each symbol are stored, using negligible $O(|\Sigma| \lg n)$ additive space), since $\mathcal{M}$ is sorted lexicographically (See Figure 2.1). To see why this is useful, we first introduce the following central definition:

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

Figure 1: Burrows-Wheeler Transform for the string $x =$"mississippi", with the unsorted matrix $\widehat{\mathcal{M}}$ on the left and the sorted matrix $\mathcal{M}$ on the right. The output is $L = \mathsf{BWT}(x) =$"ipssm$pissii".

Definition 1 (Rank of a character)**.** *Let $y \in \Sigma^n$. For any $c \in \Sigma, i \in [n]$, $rk_y(c, i)$ denotes the number of occurrences of the symbol c in the i^{th} prefix $y_{\leq i} = y[1 : i]$.*

Note that $rk_L(c, n) = n_c$, recalling that n_c is the frequency of c in x. We define the *Last-to-First (LF) column mapping* $\pi_{LF} : [n] \mapsto [n]$ by setting $\pi_{LF}(i) = j$ if the character L_i is located at F_j, i.e., L_i is the first character in the j^{th} row of the BWT matrix $\mathcal{M}$. We note that π_{LF} is a permutation.

An indispensable feature of BWT, the *LF Mapping Property*, states that for any character $c \in \Sigma$, the occurrences of c in the first column F and last column L follow *the same order*. In other words, the permutation π_{LF} preserves the ordering among all occurrences of c.

Fact 1 (LF Property)**.** *For $i \in [n], c \in \Sigma$, we have $rk_L(c, i) = rk_F(c, \pi_{LF}(i)) = \pi_{LF}(i) - \sum_{c'<c} n_c$.*

The second equality follows directly from the fact that the first column F is sorted lexicographically by construction, while the first equality also requires the fact that L is sorted by its right context. The formal argument can be found in Section A of the Appendix. The LF Mapping Property leads to the following lemma, which is the heart of the BWT decoding algorithm:

Lemma 1. *Fix a data structure D that returns $rk_L(c, i)$ for given $i \in [n], c \in \Sigma$. Let $j \in [n]$. If we know the position i of x_j in L, then we can compute (even without knowing j) the character $x_j = L_i$, and (if $j \geq 2$) the position i' of x_{j-1} in L, with $O(|\Sigma|)$ calls to D.*

Proof. Given the position $i \in L$ of x_j, the character $L_i = x_j$ can be decoded via $2|\Sigma|$ rank queries on L, by computing $rk_L(c, i) - rk_L(c, i-1)\ \forall\ c \in \Sigma$, which is nonzero only for $c^* := x_j$. Now, given the rank $rk_L(c^*, i)$ of x_j in L, the LF-property (Fact 1) allows us to translate it to the index $i' := \pi_{LF}(i)$ of x_j *in* F. As such, $F_{i'} = x_j$. But this means that $L_{i'} = x_{j-1}$, as every row of $\mathcal{M}$ is a cyclic shift of x (i.e., in each row, L and F contain consecutive characters of x). □

The decoding argument asserts that a RANK data structure over L allows us to "move back" *one character* in x. Note thatthe decoding algorithm implied by Lemma 1 is inherently sequential: decoding a *single* character x_{n-i} of x requires $O(|\Sigma| \cdot i)$ calls to D, hence $\Omega(n)$ worst-case time.

2.2 Compressing BWT

2.2.1 Move-to-Front encoding (MTF)

As mentioned in the introduction, when reasonably short contexts tend to predict a character in the input text x, the BWT string $L = \mathsf{BWT}(x)$ will exhibit local similarity, i.e, identical symbols will tend to recur at close vicinity. As such, we expect the integer string $\mathsf{MTF}(L)$ to contain many *small integers*. This motivates the following *relative* encoding of L:

The *Move-to-Front* transform (Bentley et al. 1986 [3]) replaces each character of L with the *number of distinct characters seen since its previous occurrence*. Formally, the encoder maintains a list, called the *MTF-stack*, initialized with all characters $c \in \Sigma$ ordered alphabetically. To encode the i^{th} character, the encoder outputs its RANK in the *current stack* $S_{i-1} \in \mathcal{S}_{|\Sigma|}$ (with the character at the top of S_{i-1} having RANK 0), and moves $c = L_i$ to the top of the stack, generating S_i. At any instant, the MTF-stack contains the characters ordered by recency of occurrence. Denote the output of this sequential algorithm by $m(L) := \mathsf{MTF}(L) = (m_1, m_2, \ldots, m_n) \in \{0, 1, \cdots, |\Sigma| - 1\}^n$.

A few remarks are in order: First, note that *runs of identical characters* in L are transformed into *runs of* 0*s* (except the first character in the run) in the resulting string $m(L)$. Second, at each position $i \in [n]$, the corresponding MTF-stack S_i defines a unique *permutation* $\pi_i \in \mathcal{S}_{|\Sigma|}$ on $[|\Sigma|]$.

2.2.2 The RLX compression benchmark

Based on the MTF transform and following the original paper of Burrows and Wheeler [6], [13, 23] analyzed the following compression algorithm[8] to encode $L = \mathsf{BWT}(x)$, henceforth denoted $\mathtt{RLX}(L)$:

1. Encode L using the Move-to-Front transform to produce $\mathsf{MTF}(L)$.
2. Denote by L^{runs} the concatenation of substrings of $\mathsf{MTF}(L)$ corresponding to *runs of* 0*s*, and by $\mathsf{MTF}(L^{-runs}) := [n] \setminus L^{runs}$ the remaining coordinates. Encode all 0-runs in L^{runs} using *Run-Length encoding*, where each run is replaced by its length (encoded using a prefix-free code), and denote the output by $\mathsf{RLE}\,(L^{runs})$.
3. Encode the remaining (non-runs) symbols in $\mathsf{MTF}(L^{-runs})$ using a 0-order entropy code[9] (e.g., Huffman or Arithmetic coding), to obtain the final bit stream $\mathtt{RLX}(L)$ (suitably modified to be prefix-free over the alphabet comprising non-zero MTF symbols and run-length symbols).

See illustration in Figure 2. For justification of the RLX space benchmark and comparison to other compressors, we refer the reader to Section B of the Appendix.

$$\begin{aligned} \mathrm{x} &= \text{“b a n n a n a a a”}. \\ \mathsf{MTF}(x) &= (1, 1, 13, 0, 1, 1, 1, 0, 0). \\ \mathtt{RLX}(x) &= (1, 1, 13, 1', 1, 1, 1, 2'). \end{aligned}$$

Figure 2: Move-to-Front (MTF) and RLX Encoding for the string x = “bannanaaa”. Run-Length Encoding (RLE) symbols in $\mathtt{RLX}(x)$ are shown in blue and with $'$ attached. The final prefix-free code is not shown. Note that the three occurrences of ‘n’ get assigned 3 distinct MTF symbols.

By a slight abuse of notation, the output length of the algorithm[10] is

$$|\mathtt{RLX}(L)| = |\mathsf{RLE}(L^{runs})| + \lceil H_0(\mathsf{MTF}(L^{-runs}))\rceil. \tag{1}$$

[8] Excluding the final arithmetic coding step.

[9] A 0-order encoder assigns a unique bit string to each symbol independent of its context, such that we can decode the concatenation of these bit strings.

[10] up to prefix-free coding overheads.

2.3 Augmented B-Trees [31]

Central to our data structure is the notion of "augmented B-trees", or *aB-trees* for short. Let $B \geq 2$, $t \in \mathbb{N}$, and let $A \in \Sigma^r$ be an array of length $r := B^t$. An aB-tree $\mathcal{T}$ over A is a B-ary tree of depth t, with leaves corresponding to elements of A. Each node $v \in \mathcal{T}$ is augmented with a value φ_v from an alphabet Φ. This value φ_v must be a function of the subarray of A corresponding to the leaves of the subtree $\mathcal{T}_v$ rooted at v. In particular, the value of a leaf must be a function of its array element, and the value of an internal node must be a function of the values of its B children.

The query algorithm starts at the root and traverses down the tree along a path which can be *adaptive*. Whenever it visits a node, it reads all the values of its B children and recurses to one of them, until it reaches a leaf node and returns the answer. We ensure the query algorithm spends $O(1)$ time per node, by packing all the augmented values of the children in a single word.

For a given aB-tree $\mathcal{T}$ and value $\varphi \in \Phi$, let $\mathcal{N}(r, \phi)$ be the number of possible arrays $A \in \Sigma^r$ such that the root is labeled with φ. A reasonable information-theoretic space benchmark for this data structure, conditioned on the root value φ, is therefore $\lg \mathcal{N}(n, \varphi)$. Pătraşcu proved the following remarkable result, which allows to *compress* any aB-tree, while preserving its query time :

Theorem 4 (Compressing aB-trees, [31])**.** *Let $B = O\left(\frac{w}{\lg(r+|\Phi|)}\right)$. We can store an aB-tree of size r with root value φ using $\lg \mathcal{N}(n, \varphi) + 2$ bits. The query time is $O(\lg_B r) = O(t)$, assuming precomputed look-up tables of $O\left(|\Sigma| + |\Phi|^{B+1} + B \cdot |\Phi|^B\right)$ words, which only depend on r, B and the aB-tree query algorithm.*

The proof idea is to use recursion in order to encode the root value φ_r, followed by an encoding of the augmented values φ_v of every child of the root, *"conditioned" on* φ_r, and so on, without losing (almost) any entropy (recursive encoding is needed to achieve this, since $\mathcal{N}(r, \phi_r)$ may not be a power of 2). Theorem 4 allows us to represent any aB-tree with a redundancy of merely 2 bits (over the *zeroth-order* empirical entropy of the leaves). Since the extra look-up tables do not depend on the array A, in our application, we use a similar trick as in [31] and divide the original array of length n into blocks of length $r = B^t$, building an aB-tree over each block. We then invoke Theorem 4 separately on each tree, adding a certain auxiliary data structure that aggregates query answers across blocks so as to answer the query on the original array (for further details, see [31]). Beyond facilitating the desired query time, this application renders the extra space occupied by the look-up tables in Theorem 4 inconsequential, as they can be *shared* across blocks. We remark that this "splitting" trick of [31] only applies when the augmented values φ are *composable*, in the sense that $\varphi(A \circ B) = f(\varphi(A), \varphi(B))$, where $A \circ B$ is the concatenation of the arrays A, B. The aB-trees we design shall use augmented *vector* values which are (component-wise) composable.

3 Technical Overview

Both Theorem 1 and Theorem 2 follow from the next result, which is the centerpiece of this work:

Theorem 3.1. *There exists a small constant $\delta > 0$ such that for any $x \in \Sigma^n$ and $t \leq \delta \lg n$, there is a succinct data structure $\mathcal{D}_{rk}$ that supports* Rank *queries on $L = \mathsf{BWT}(x)$ in time $O(t')$, using at most $|\mathtt{RLX}(L)| + n \lg n / 2^{t'} + n^{1-\Omega(1)}$ bits of space, in the $w = \Theta(\lg n)$ word-RAM model.*

Theorem 2 is a direct corollary of Theorem 3.1, as it turns out that counting the number of occurrences of a given pattern $p \in \Sigma^\ell$ in x, amounts to $O(\ell)$ successive Rank queries on L (see [13]).

To see how Theorem 1 follows from Theorem 3.1, consider the following data structure for locally-decoding a coordinate x_i of x in time t: Let $t' < t$ be a parameter to be determined shortly.

Let $\mathcal{D}_{rk}$ be the data structure supporting rank queries on L in time $O(t')$. We divide x into $\lceil n/T \rceil$ blocks of size $T := O(t/t')$, and store, for each ending index j of a block, the position *in* L corresponding to x_j. In other words, we simply record "shortcuts" of the BWT transform after every block of size T. Given an index $i \in [n]$, the data structure first computes the endpoint $j := \left\lceil \frac{i}{T} \right\rceil T$ of the block to which i belongs, reads from memory the position of x_j in L, and then simulates $(j-i) \leq T = O(t/t')$ sequential steps of the LF-mapping decoding algorithm from Section 2.1.1, to decode x_i. By Lemma 1, each step requires $O(|\Sigma|)$ RANK queries on L, each of which can be done using $\mathcal{D}_{rk}$ in $O(t')$ time, hence the overall running time is $O(T \cdot t') = O(t)$. To balance the redundancy terms, observe that the overall space of our data structure (up to $O(n^{\varepsilon})$ terms) is

$$s = |\texttt{RLX}(L)| + \frac{n \lg n}{2^{t'}} + \frac{n \lg n}{T}. \tag{2}$$

Thus, setting $t' = \Theta(\lg t)$, leads to overall redundancy $r = O\left(\frac{n \lg n \lg t}{t}\right) = \tilde{O}\left(\frac{n \lg t}{t}\right)$, as claimed in Theorem 1. Next, we provide a high-level overview of the proof of Theorem 3.1.

3.1 Proof Overview of Theorem 3.1

Recall (Section 2.2), that the RLX compression of $L = \mathsf{BWT}(x)$ can be summarized as :

$$|\texttt{RLX}(L)| = |\mathsf{RLE}(L^{runs})| + \lceil H_0(\mathsf{MTF}(L^{-runs})) \rceil.$$

Since RLX compresses the two parts L^{runs} and L^{-runs} using two conceptually different encodings (RLE and MTF, respectively), it makes sense to design a RANK data structure for each part separately (along with an efficient mapping for combining the two answers to compute the overall rank of a character in L). This modular approach simplifies the presentation and, more importantly, enables us to achieve a significantly better redundancy for Theorem 3 (i.e., $n/(\lg n/t)^t$ instead of $n/2^t$), but is slightly suboptimal in terms of space (by an $\Omega(|\texttt{RLX}(L)|)$ additive term). In the actual proof, we show how the two data structures below can be "merged" to avoid this overhead.

A Rank data structure over $\mathsf{RLE}(L^{runs})$. Our first goal is to design a compressed data structure $\mathcal{D}_{\mathsf{RLE}}$ that reports, for each symbol $c \in \Sigma$ and index $i \in [n]$, the number of occurrences of c in $L[1:i]$ that are contained *in* L^{runs}, i.e., the number of consecutive 0's in $MTF(L)[1:i]$ that correspond to runs of c. Since RLX represents this substring by "contracting" each run into a singleton (denoting its length), solving this problem succinctly essentially entails a Predecessor search[11] on the universe $[n]$ with $\kappa = \kappa(L)$ "keys", where κ denotes the number of runs in L. Alas, under the standard representation of this input, as a κ-sparse string in $\{0,1\}^n$, Predecessor search clearly requires at least $\lg \binom{n}{\kappa}$ bits of space [31,32], which could be $\gg |\texttt{RLX}(L)|$ (for example, when all but a single 0-run are of constant length and separation, which is an oblivious feature to the previous representation). To adhere to the RLE space benchmark, we use a more suitable alternative representation of L^{runs}.

To this end, suppose for simplicity of exposition, that L consists entirely of runs (i.e., $L = L^{runs}$), and that the character $c \in \Sigma$ corresponding to each 0-run is known at query time (this will be handled in the integrated data structure in Section 5). For $i \in [\kappa]$, let $\ell_i \in [n]$ denote the length of the i^{th} run, and let $L' = (\ell_1, \ell_2, \ldots, \ell_\kappa) \in [n]^\kappa$ be the string that encodes the run lengths. Note that RLX spends precisely $\sum_i \lg \ell_i$ bits to encode this part (ignoring prefix-coding issues).

To compute $rk_{L^{runs}}(c, i)$, we design an adaptive augmented aB-tree, that essentially implements a predecessor search over the new representation L' of L^{runs}: We first construct a B-tree $\mathcal{T}$ over

[11] For a set of keys $S \subset \mathcal{U}$ with $|S| = \kappa$, PREDECESSOR(i, S) returns $\max\{x \in S \mid x \leq i\}$.

the array $L' \in [n]^\kappa$, and augment each intermediate node v of the tree with the (vector-valued) function $\varphi_{RLE}(v) := (\varphi_\ell^c(v))_{c\in\Sigma} \in [n]^{|\Sigma|}$, where $\varphi_\ell^c(v)$ counts the total sum $\sum_{j\in\mathcal{T}_v} \ell_j$ of run-lengths in the subtree of v, corresponding to runs of c. Given an index $i \in [n]$ and character $c \in \Sigma$, the query algorithm iteratively examines the labels of all B children of a node $v \in \mathcal{T}$ starting from the root, and recurses to the rightmost child u of v for which $\sum_c \varphi_\ell^c(u) \le i$ (i.e., to the subtree that contains the interval ℓ_j to which i belongs), collecting the sum of $\varphi_\ell^c(u)$'s along the query path.

To ensure query time $O(t')$, we break up the array as in [31] into sub-arrays each of size $B^{t'}$ (for $B = \Theta(1)$), and build the aforementioned tree over each sub-array (this is possible since the augmented vector φ_{RLE} is a (component-wise) composable function). To ensure the desired space bound for representing $\mathcal{T}$, we further augment each node v with a "zeroth-order entropy" constraint $\varphi_0(v)$, counting the sum of marginal empirical entropies $n_c^v \lg(n_c/n)$[12] of the elements in the subtree $\mathcal{T}_v$ (which can be done recursively due to additivity of φ_0 w.r.t v's). A standard packing argument then ensures $\mathcal{N}(2\kappa, \varphi) \le 2^{\varphi_0(v)} \lesssim 2^{H_0(v)}$, as desired. We then invoke Theorem 4 to compress $\mathcal{T}$ to $H_0(L') + O\left(\frac{n\lg n}{B^{t'}}\right)$ bits, yielding exponentially small redundancy (up to $n^{1-\varepsilon}$ additive terms). This ensures that the total space (in bits) occupied by $\mathcal{D}_{\mathsf{RLE}}$ is essentially

$$H_0(L') + O\left(\frac{n\lg n}{B^{t'}}\right) \le |\mathsf{RLE}(L^{runs})| + O\left(\frac{n\lg n}{B^{t'}}\right).$$

The actual proof is slightly more involved, since the merged data structure needs to handle characters from both L^{runs} and $MTF(L^{-runs})$ simultaneously, hence it must efficiently distinguish between 0-runs corresponding to different symbols. Another issue is that Theorem 4 of [31] is only useful for truly sub-linear alphabet sizes, whereas $(L')_i \in [n]$, hence in the actual proof we must also split long runs into chunks of length $\le n^\varepsilon$. A simple application of the log-sum inequality ensures this truncation does not increase space by more than an $\tilde{O}(n^{1-\varepsilon})$ additive term.

A Rank data structure over $\mathsf{MTF}(L^{-runs})$. The more challenging task is computing $rk_{L^{-runs}}(c, i)$, i.e., the frequency of c in $L[1 : i]$ contained in the substring $\mathsf{MTF}(L^{-runs})$, which is obtained by applying the MTF transform to L and deleting all 0-runs (see Figure 2). Note that the mapping from $i \in L$ to its corresponding index $i' \in \mathsf{MTF}(L^{-runs})$ amounts to subtracting all runs before i. This operation can be performed using a single partial-sum query to our integrated data structure (in Section 5), which collects the sum of $\varphi_\ell^c(u)$'s *over all* $c \in \Sigma$ along the query path.

As discussed in the introduction, the adaptive nature of the MTF encoding has the major drawback that decoding the j^{th} symbol $\mathsf{MTF}(L^{-runs})_j$, let alone computing its rank, requires knowing the corresponding MTF stack state $S_{j-1} \in \mathcal{S}_{|\Sigma|}$ (i.e., the precise order of recently occurring symbols), which itself depends on the *entire* history $L^{-runs}_{<j}$. A straightforward solution is to store the MTF stack state after every block of length t' (where t' is the desired query time), much like the "marking" solution for decoding BWT, yielding a linear search for the stack-state S_j from the nearest block, and thus a linear time-space tradeoff.

To speed up the search for the local stack-state, we observe the following key property of the MTF transform: Let $\mathsf{MTF}(x) := (m_1, m_2, \ldots, m_n)$ be the MTF transform of $x \in \Sigma^n$ (see Figure 2 for illustration). Let $\mathcal{I} = [i, j]$ be any sub-interval of $[n]$, and denote by $S_{i-1}, S_j \in \mathcal{S}_{|\Sigma|}$ the corresponding stack-states at the start and endpoints of $\mathcal{I}$. Now, consider the *permutation* $\pi_{\mathcal{I}} := \mathbf{Id_\Sigma} \mapsto \widehat{S_j}$, obtained by simulating the MTF decoder on $(m_i, \ldots, m_j)$ *starting from the identity* state $\mathbf{Id_\Sigma}$, i.e., "restarting" the MTF decoding algorithm but running it on the *encoded* substring $(\mathsf{MTF}(x)_i, \ldots, \mathsf{MTF}(x)_j)$, arriving at some final state $(\widehat{S_j})$ at the end of $\mathcal{I}$ (note that this

[12] For $c \in \Sigma$ and node v, n_c^v denotes the frequency of c in the sub-array rooted at v.

process is well-defined). Then the true stack-state S_j satisfies: $S_j = \pi_{\mathcal{I}} \circ S_{i-1}$. The crucial point is that $\pi_{\mathcal{I}}$ is *independent* of the (true) stack state S_{i-1}, i.e., it is a *local* function of $\mathsf{MTF}(x)_{\mathcal{I}}$ only.

We show that this "decomposition" property of the MTF transform (Proposition 1), facilitates a *binary search* for the local stack-state S_{j-1} (rather than linear-searching) with very little space overhead, as follows: At preprocessing time, we build an augmented B-tree over the array $\mathsf{MTF}(x_1, \ldots, x_n)$, where each intermediate node v is augmented with the *permutation* $\pi_v \in \mathcal{S}_{|\Sigma|}$ corresponding to its subtree $\mathsf{MTF}(\mathcal{I}_v)$, obtained by "*restarting*" the MTF decoder to the identity state $\mathbf{Id_\Sigma}$, and simulating the MTF decoder from start to end of $\mathcal{I}_v$, as described above. Note that this procedure is well defined, and that the aforementioned observation is crucially used here, as the definition of aB-trees requires each augmented value of an intermediate node to be a *local function* of its own subtree. At query time, the query algorithm traverses the root-to-leaf(j) path, *composing* the corresponding (possibly inverse) permutations between the stack-states along the path, depending on whether it recurses to a right or left subtree. We show this process ensures that when the query algorithm reaches the leaf $\mathsf{MTF}(x)_j$, it possesses the correct stack-state S_{j-1}, and hence can correctly decode x_j. While this algorithm supports only "local decoding" (DICTIONARY) queries, with an extra simple trick, the above property in fact facilitates a similar aB-tree supporting RANK queries under the MTF encoding (see Section 4).

Once again, in order to impose the desired space bound ($\approx H_0(\mathsf{MTF}(L^{runs}))$) and to enable arbitrary query time t', we augment the nodes of the tree with an additional zeroth-order entropy constraint, and break up the array into sub-arrays of size $\Theta(B^{t'})$, this time for $B \approx \frac{\lg n}{t'}$. Compressing each tree using Theorem 4, and adding an auxiliary data structure to aggregate query answers across sub-arrays, completes this part and establishes Theorem 3.

3.2 Lower Bound Overview

We prove the following cell-probe lower bound for a somewhat stronger version of Problem 1, which requires the data structure to efficiently decode *both* forward and *inverse* dispositions of the induced BWT permutation between X and $L := \mathsf{BWT}(X)$ (we note that both the FM-Index and our data structure from Theorem 1 satisfy this natural requirement[13], and elaborate on it in Section 7):

Theorem 3.2. *Let $X \in_R \{0,1\}^n$ and let $\Pi_X \in \mathcal{S}_n$ be the induced BWT permutation from indices in $L := \mathsf{BWT}(X)$ to indices in X. Then, any data structure that computes $\Pi_X(i)$ and $\Pi_X^{-1}(j)$ for every $i, j \in [n]$ in time t, q respectively, such that $t \cdot q \leq \delta \lg n / \lg\lg n$ (for some constant $\delta > 0$), in the cell-probe model with word size $w = \Theta(\lg n)$, must use $n + \Omega\left(n/tq\right)$ bits of space in expectation.*

We stress that Theorem 3.2 is more general, as our proof can yield nontrivial lower bounds against general (non-product) distributions μ on Σ^n with "sufficient block-wise independence", though a lower bound against uniform strings is in some sense stronger, as it states that the above redundancy cannot be avoided even if Π_X is stored in *uncompressed* form (see also Section 7).

Our proof of Theorem 3.2 is based on a "nonuniform" variation of the "cell-elimination" technique of [16], who used it to prove a lower bound of $r \geq \Omega(n \lg n/tq)$ on the space redundancy of any data structure for the *succinct permutations* problem $\textsc{Perms}_n$. In this problem, the goal is to represent a random permutation $\Pi \in_R \mathcal{S}_n$ succinctly using $\lg n! + o(n \lg n)$ bits of space, supporting forward and inverse evaluation queries in query times t, q respectively, as above. Alas, this compression argument crucially requires that

$$t, q \leq O\left(\frac{H(\Pi)}{n \cdot \lg\lg n}\right). \tag{3}$$

[13] I.e., for these data structures, we can achieve $q = O(t)$ by increasing the redundancy r by a mere factor of 2.

When Π is a *uniformly random* permutation, i.e., $H(\Pi) \approx n \lg n$, this condition implies that the lower bound holds for $t, q \leq O(\lg n / \lg \lg n)$. In contrast, the BWT permutation of X can have *at most* $n \lg |\Sigma| = O(n)$ bits of entropy for constant-size alphabets (as Π_X is determined by X itself), hence condition (3) does not yield *any* lower bound whatsoever for our problem.

To circumvent this obstacle, we prove an "entropy polarization" lemma for BWT: It turns out that for a random string X, while an *average* coordinate $\Pi_X(i)$ indeed carries only $O(1)$ bits of entropy, the entropy distribution has huge variance. In fact, we show that for any $\varepsilon > \tilde{\Omega}(1/\lg n)$, there is a subset $\mathcal{I}$ of only $(1-\varepsilon)\frac{n}{\lg n}$ coordinates in $[n]$, whose total entropy is $H(\Pi_X(\mathcal{I})) \geq (1-O(\varepsilon))n$, i.e., this small set of coordinates has maximal entropy ($\approx \lg n$ bits each), and essentially determines the entire BWT permutation[14]. This lemma (Lemma 3) is reminiscent of *wringing lemmas* in information theory [2], and may be a BWT property of independent interest in other applications.

The intuition behind the proof is simple: Consider dividing X into $s := \frac{n}{C \lg n}$ disjoint blocks of size $C \lg n$ each, and let $\mathcal{I} := \{I_i, \ldots, I_s\} \subset [n]$ denote the set of first coordinates in each block respectively. Since X is random, each of the s blocks is an *independent* random $(C \lg n)$-bit string, hence for a large constant C, with overwhelming probability these substrings will be distinct, and intuitively, conditioned on this likely event, their lexicographic ordering remains random, hence the BWT locations of these indices alone must recover this random ordering, which is worth $\Omega(s \lg s) = \Omega(n)$ bits of information. However, the birthday paradox requires that $C > 2$ to avoid collisions, in which case the above argument can only show that a small constant fraction ($< 0.5n$) of the total entropy can be "extracted" from this small set, while the remaining $n - o(n)$ coordinates possess most of the entropy. This fact completely dooms the subsequent "cell-elimination" argument, since these $\Omega(n)$ remaining coordinates cause the load (average number of queries) on each w-bit cell to become prohibitively large ($> w \approx \lg n$), trivializing the compression process (see Theorem 7).

Nevertheless, setting $C = (1+\varepsilon)$, the number of "colliding" blocks (i.e., non-distinct substrings of length $(1+\varepsilon)\lg n$) is still only $\tilde{O}(n^{1-\varepsilon}) \ll \varepsilon n / \lg n$ with very high probability. Moreover, we show that conditioned on this high-probability event $\mathcal{E}$, the lexicographic ordering among the remaining *distinct* $\approx (1-2\varepsilon)\frac{n}{\lg n}$ blocks remains random. (For uniform n-bit strings, we show that conditioning on $\mathcal{E}$ preserves *exact* uniformity of the ordering, by symmetry of $\mathcal{E}$ w.r.t block-permutation, but more generally, we note that for any prior distribution, conditioning on $\mathcal{E}$ does not "distort" the original distribution by more than $\approx \sqrt{\lg(1/\mathbf{Pr}[\mathcal{E}])} = o(1)$ in statistical distance, hence this argument can be generalized to nonuniform strings). Since, conditioned on $\mathcal{E}$, the BWT mapping on $\mathcal{I}$ determines the lexicographic ordering of the blocks, the data processing inequality (DPI) implies that the entropy of $\Pi_X(\mathcal{I})$ is at least $\approx (1-2\varepsilon)\frac{n}{\lg n} \cdot \lg \frac{n}{\lg n} \geq (1-3\varepsilon)n$, as claimed.

Applying the "entropy polarization" lemma with $\varepsilon = O(\lg \lg n / \lg n)$, we then show how to adapt [16]'s cell-elimination argument to nonuniform permutations, replacing 'unpopular' cells with an efficient encoding of the *partial* bijection $\Pi_X(\mathcal{I})$ induced on (forward and inverse) queries $\in \mathcal{I}$ probing these cells. The polarization lemma then ensures that the remaining map of Π_X on $\bar{\mathcal{I}} = [n] \setminus \mathcal{I}$ can be encoded directly using $H(\Pi_X(\bar{\mathcal{I}}) |\ \mathcal{I}, \Pi_X(\mathcal{I})) \leq O(\varepsilon n) = O(n \lg \lg n / \lg n)$ bits, which will be dominated by the redundancy we obtain from the compression argument (so long as $tq \lesssim \lg n / \lg \lg n$), thereby completing the proof.

4 A Locally Decodable MTF Code and Rank Data Structure

In this section, we prove the following theorem, which is a more formal version of Theorem 3.

[14]Note that here we view Π_X as a mapping from X to $L = \mathsf{BWT}(X)$ and not the other way around, but this is just for the sake of simplicity of exposition and looking at Π_X^{-1} is of course equivalent.

Theorem 5. *For any string $x \in \Sigma^n$ with $|\Sigma| = O(1)$, there is a succinct data structure that encodes x using at most*

$$H_0(\mathsf{MTF}(x)) + n \Big/ \left(\frac{\lg n}{\max(t, \lg\lg n)}\right)^t + n^{1-\Omega(1)}$$

bits of space, supporting Rank *and* Dictionary *queries in time $O(t)$, in the word-RAM model with word size $w = \Theta(\lg n)$.*

Setup and Notation. Let the alphabet $\Sigma = \{c_1, c_2, \cdots, c_{|\Sigma|}\}$, where $c_1 < c_2 < \cdots < c_{|\Sigma|}$ according to the lexicographical ordering on Σ. Let $S = (a_1, a_2, \cdots, a_{|\Sigma|})$ denote the MTF stack with a_1 at the top and $a_{|\Sigma|}$ at the bottom. For $j \in [|\Sigma|]$, let $S[j]$ denote the character at position j in S, starting from the top. Fix a string $x = (x_1, x_2, \cdots, x_n) \in \Sigma^n$. Let $m = \mathsf{MTF}(x) = (m_1, m_2, \cdots, m_n) \in \{0, 1, \cdots, |\Sigma| - 1\}^n$ be the Move-to-Front (MTF) encoding of x, with the initial MTF stack $S_0 := (c_1, c_2, \cdots, c_{|\Sigma|})$.

Given a MTF stack $S = (a_1, a_2, \cdots, a_{|\Sigma|})$ and a permutation $\pi \in \mathcal{S}_{|\Sigma|}$, let $S' = \pi \circ S$ be the stack such that $S'[\pi(j)] = S[j] = a_j$ for all $j \in [|\Sigma|]$. We also associate with S the permutation $\pi(S)$ which converts the initial stack S_0 to S, i.e., $S = \pi(S) \circ S_0$. In this sense, we say that S_0 corresponds to the identity permutation $\mathbf{Id}_{|\Sigma|}$ on $[|\Sigma|]$, as $S_0[j] = c_j$ for all $j \in [|\Sigma|]$. For $i \in [n]$, let S_i be the stack induced by simulating the MTF decoder on $m[1:i]$, starting from S_0. Equivalently, S_i is the stack induced by $\mathsf{MTF}(x[1:i])$, i.e., the stack just after encoding the first i characters of x, starting from S_0. For $0 \le i < j \le n$, let $\pi_{i,j} \in \mathcal{S}_{|\Sigma|}$ be the unique permutation induced by simulating the MTF decoder on $m[i+1:j]$, starting from S_i.

4.1 Properties of MTF Encoding

The following proposition shows that for any $0 \le i < j \le n$, the permutation $\pi_{i,j}$ is a *local* function of $m[i+1:j]$. So, these permutations $\pi_{i,j}$ are valid augmented values for an aB-tree built over $m = \mathsf{MTF}(x)$, without reference to the true MTF stacks S_i and S_j.

Proposition 1. *Fix $0 \le i < j \le n$, and let S_i, S_j and $\pi_{i,j} \in \mathcal{S}_{|\Sigma|}$ be as defined above. Then $\pi_{i,j}$ is independent of S_i and S_j, given $m[i+1:j]$. Hence, we can generate $\pi_{i,j}$ by simulating the MTF decoding algorithm on $m[i+1:j]$, starting from the identity stack S_0.*

Proof. We prove this proposition by induction on $j - i$. Consider the base case, when $j - i = 1$. Then by definition of a single MTF step, we have

$$\pi_{i,i+1}(k) = \begin{cases} k+1 & \text{if } k \le m_{i+1} \\ 1 & \text{if } k = m_{i+1} + 1 \\ k & \text{if } k > m_{i+1} + 1 \end{cases} \tag{4}$$

Clearly, $\pi_{i,i+1}$ is independent of S_i and S_{i+1} given m_{i+1}. This proves the base case.

Now, suppose the claim is true for all i, j such that $j - i = k \in \mathbb{N}$, and let i, j be such that $j - i = k + 1$. Then by the induction hypothesis, $\pi_{i,j-1}$ is independent of S_i and S_{j-1} given $m[i+1:j-1]$. Moreover, $\pi_{j-1,j}$ is independent of S_{j-1} and S_j given m_j. Due to the sequential nature of the MTF encoding, we clearly have

$$\pi_{i,j} = \pi_{j-1,j} \circ \pi_{i,j-1}$$

As both the permutations $\pi_{j-1,j}$ and $\pi_{i,j-1}$ are independent of stacks S_i, S_{j-1}, S_j given $m[i+1:j]$, the same must be true for $\pi_{i,j}$. □

The following expression captures the evolution of the MTF stack, for all $0 \le i < j \le n$:

$$S_j = \pi_{i,j} \circ S_i \tag{5}$$

We can also "reverse" the steps of the MTF encoding. For fixed $0 \le i < j \le n$, if we are given the final stack S_j and the permutation $\pi_{i,j}$, we can recover the initial stack S_i by inverting $\pi_{i,j}$:

$$S_i = \pi_{i,j}^{-1} \circ S_j$$

4.2 Locally Decodable MTF Code

We first describe the construction of a single aB-tree over the entire MTF encoding $m = \mathsf{MTF}(x) \in \{0, 1, \cdots, |\Sigma| - 1\}^n$, which supports "local decoding" (DICTIONARY) queries. Let $B \ge 2$ be the branching factor. Each node v will be augmented with a permutation $\varphi_\pi(v) \in \Phi_\pi = \mathcal{S}_{|\Sigma|}$. For $i \in [n]$, the leaf node v corresponding to m_i is augmented with the permutation $\varphi_\pi(v) = \pi_{i-1,i}$. Let v be an internal node with its children being $v_1, v_2, \cdots, v_B$ in order from left to right. Then v is augmented with the composition of permutations of its children, i.e.,

$$\varphi_\pi(v) = \varphi_\pi(v_B) \circ \varphi_\pi(v_{B-1}) \circ \cdots \circ \varphi_\pi(v_1). \tag{6}$$

It is easy to observe that a node v whose subtree $\mathcal{T}_v$ is built over the sub-array $m[i+1 : j]$ is augmented with the value $\varphi_\pi(v) = \pi_{i,j}$. Now, Proposition 1 ensures that this is a legitimate definition of an aB-tree, because the value of a leaf is a function of its array element, and the value of an internal node is a function of the values of its B children.

The query algorithm maintains a MTF stack S, which is initialized to the identity stack S_0 at the beginning of the array. Let $i \in [n]$ be the query index. The algorithm traverses down the tree, updating S at each level. It maintains the invariant that whenever it visits a node v whose sub-tree encompasses $m[j+1 : k]$, it updates S to the true stack S_j just before the beginning of $m[j+1 : k]$.

We describe how to maintain this invariant recursively. The base case is the root (at depth $d = 0$) whose subtree contains the entire array m. So, the query algorithm initializes $S = S_0$, which corresponds to the true initial MTF stack. Now, let v be a node at depth d whose sub-tree T_v encompasses $m[j+1 : k]$. Suppose the query algorithm has visited v, and S is the true MTF stack S_j. By assumption, $j+1 \le i \le k$. Let $v_1, v_2, \cdots, v_B$ be the children of v in order from left to right, and let v_{β^*} be the child of v whose sub-tree includes i. Then we update S as follows:

$$S \leftarrow \varphi_\pi(v_{\beta^*-1}) \circ \varphi_\pi(v_{\beta^*-2}) \circ \cdots \circ \varphi_\pi(v_1) \circ S. \tag{7}$$

The above procedure explains the update rule which maintains the invariant at a node at depth $d+1$, assuming the invariant was maintained at a node at depth d. Thus, the proof that the invariant is maintained follows by induction on d.

Eventually, the algorithm reaches the leaf node corresponding to m_i. At this point, the MTF stack S is the true stack S_{i-1}. Hence, it reports $x_i = S[m_i]$. The running time is $t = O(\lg_B n)$.

For the sake of simplicity, we have stated the update rule 7 purely in terms of forward compositions of permutations $\pi_{i,j}$. In practice, if $\beta^* > B/2$, one can equivalently update S by starting from $\varphi_v \circ S$ and composing the inverse permutations $\varphi_\pi^{-1}(v_\beta)$ for $\beta \ge \beta^*$:

$$S \leftarrow \varphi_\pi^{-1}(v_{\beta^*}) \circ \varphi_\pi^{-1}(v_{\beta^*+1}) \circ \cdots \circ \varphi_\pi^{-1}(v_\beta) \circ \varphi_\pi(v) \circ S.$$

However, since all permutations $\varphi_\pi(v_\beta), \beta \in [B]$ are stored in a word, both update rules take $O(1)$ time, and so the query time remains unaltered. Henceforth, we will continue to state the update rules in terms of forward compositions.

4.3 Extension to Rank Queries (over MTF)

The aB-tree $\mathcal{T}$ above only supports "local decoding" (DICTIONARY) queries over $m = \mathsf{MTF}(x)$, while our application requires answering RANK queries (over $\mathsf{MTF}(L^{-runs})$). We now show how the above aB-tree can indeed be extended, via an extra simple observation, to support RANK queries under the MTF encoding.

Let v be a node in the aB-tree $\mathcal{T}$, whose subtree $\mathcal{T}_v$ is built over the sub-array $m[i+1:j]$. We would like to augment v with a vector $\tilde{\varphi}_{rk}(v) = (\tilde{\varphi}_{rk}(v, c_\sigma))_{\sigma \in [|\Sigma|]} \in \{0, 1, \cdots, n\}^{|\Sigma|}$, such that $\tilde{\varphi}_{rk}(v, c_\sigma)$ is the frequency of the character $c_\sigma \in \Sigma$ in $x[i+1:j]$. However, as $\mathcal{T}$ is built over $m = \mathsf{MTF}(x)$, and two occurrences of the same character $c \in \Sigma$ can be assigned distinct symbols in the MTF encoding, these augmented values are not consistent with the definition of an aB-tree.

To resolve this difficulty, we again use the fact that the permutation $\pi_{i,j}$ depends only on the sub-array $m[i+1:j]$. Recall that $S_0 = \left(c_1, c_2, \cdots, c_{|\Sigma|}\right)$ corresponds to the identity permutation $\mathbf{Id}_{|\mathbf{\Sigma}|}$. For a node v, let $\varphi_{rk}(v) := (\varphi_{rk}(v, \sigma))_{\sigma \in [|\Sigma|]}$, where $\varphi_{rk}(v, \sigma)$ is the frequency of c_σ in the sub-array rooted at v, *assuming* the MTF stack at the beginning of this sub-array is S_0. For a leaf node v at $i \in [n]$, we have $\varphi_{rk}(v, \sigma) = 1$ if $m_i = \sigma - 1$, and 0 otherwise.

Now, let v be an internal node with children $v_1, v_2, \cdots, v_B$. Fix a character $c_\sigma \in \Sigma$. In general, the MTF stack at the beginning of the sub-array rooted at $\mathcal{T}_v$ will be different from the MTF stack at the beginning of the sub-array $\mathcal{T}_{v_\beta}$ rooted at each child $v_\beta, \beta > 1$. So, in order to express $\varphi_{rk}(v, \sigma)$ in terms of the values of its children, we need to add the entry of the vector $\varphi_{rk}(v_\beta)$ which corresponds to c_σ, for each $\beta \in [B]$. We do this using the permutations $\varphi_\pi(v_\beta), \beta \in [B]$. For $\beta \in [B]$, the true MTF stack at the beginning of the sub-array rooted at v_β, assuming the MTF stack at the beginning of the sub-array rooted at v is S_0, is given by Equation 7. So, we have

$$\varphi_{rk}(v, \sigma) = \sum_{\beta=1}^{B} \varphi_{rk}(v_\beta, \varphi_\pi(v_{\beta-1}) \circ \varphi_\pi(v_{\beta-2}) \circ \cdots \circ \varphi_\pi(v_1)(\sigma)) \tag{8}$$

Let $\Phi_{rk} = \{0, 1, \cdots, n\}^{|\Sigma|}$. We augment each node v with $\varphi_{rk}(v) \in \Phi_{rk}$. As we also encode the permutation $\varphi_\pi(v)$, the value at each internal node is a function of the values of its children, and hence this is a legitimate aB-tree.

The query algorithm, given $(c_\sigma, i) \in \Sigma \times [n]$, initializes a rank counter $rk = 0$, and traverses the same root-to-leaf path as before. Fix an internal node v, with children $v_1, v_2, \cdots, v_B$, in its path. Let $\beta^* \in [B]$ be such that the sub-array rooted at v_{β^*} contains the index i. The algorithm updates rk as follows:

$$rk \leftarrow rk + \sum_{\beta=1}^{\beta^*-1} \varphi_{rk}(v_\beta, \varphi_\pi(v_{\beta-1}) \circ \varphi_\pi(v_{\beta-2}) \circ \cdots \circ \varphi_\pi(v_1)(\sigma)) \tag{9}$$

Then it recurses to v_{β^*} and performs this step until it reaches the leaf and returns $rk_x(c_\sigma, i)$.

4.4 Compressing the MTF aB-tree

We now describe how to compress the aB-tree $\mathcal{T}$ defined above, using Theorem 4, to support RANK (and hence DICTIONARY) queries under the MTF (followed by arithmetic) encoding, with respect to the desired space bound $H_0(\mathsf{MTF}(x))$. Let $O(t)$ be the desired query time. Choose $B \geq 2$ such that $B \lg B = \frac{\varepsilon \lg n}{\max(t|\Sigma|, \lg \lg n)}$ for some small $\varepsilon > 0$. Let $r = B^t$. We divide m into n/r sub-arrays $A_1, A_2, \cdots, A_{n/r}$ of size r and build an aB-tree over each sub-array. We show how to support DICTIONARY and RANK queries within each sub-array in time $O(\lg_B r) = O(t)$.

For each $j \in [n/r]$, we store the true MTF stack at the beginning of the sub-array A_j, the frequency of each character $c \in \Sigma$ in the prefix $x[1:(j-1)r]$, and its index in memory.

Given a DICTIONARY query with index $i \in [n]$, the query algorithm determines the sub-array A_j ($j = \lceil i/r \rceil$) containing i, initializes S to the MTF stack $S_{(j-1)r}$ just before A_j, and performs the query algorithm described in Section 4.2 on the aB-tree over A_j, with query index $i-(j-1)r$.

Similarly, given a RANK query $(c_\sigma, i) \in \Sigma \times [n]$, the query algorithm determines the sub-array A_j containing i, reads $r' := rk_x(c_\sigma, (j-1)r)$, the rank of c_σ in the prefix $x[1:(j-1)r]$, and builds the permutation $\pi^* = \pi_{0,(j-1)r}$ corresponding to the MTF stack $S_{(j-1)r}$. Then, it performs the query algorithm described in Section 4.3 on the aB-tree over A_j, with query $\left(c_{\pi^*(\sigma)}, i-(j-1)r\right) \in \Sigma \times [r]$. Finally, it adds r' to this answer and returns the sum.

For a MTF character $\sigma \in \{0, 1, \cdots, |\Sigma|-1\}$, let f_σ be the frequency of σ in m. Following [31], we define a measure of "entropy per character". For $\sigma \in \{0, 1, \cdots, |\Sigma|-1\}$, we encode each occurrence of σ in m using $\lg \frac{n}{f_\sigma}$ bits, rounded up to the nearest multiple of $1/r$. We impose a *zeroth-order entropy constraint* by augmenting each node v with an additional augmented value $\varphi_0(v)$, which is the sum of the entropy of the symbols in its subtree. Then we have

$$H_0(m) = \sum_{\sigma=0}^{|\Sigma|-1} f_\sigma \lg \frac{n}{f_\sigma} = \sum_{i=1}^{n} \lg \frac{n}{f_{m_i}} = \sum_{j=1}^{n/r} \sum_{i \in A_j} \lg \frac{n}{f_{m_i}} = \sum_{j=1}^{n/r} H_0(A_j),$$

where $H_0(A_j)$ is the sum of entropy of the symbols in A_j. Note that the assigned entropy $\lg \frac{n}{f_\sigma}$ of each occurrence of a character σ is a function of its frequency in the *entire* array (not in A_j).

Let Φ_0 be the alphabet of these values. As the entropy of each occurrence of a character can attain one of $O(r \lg n)$ values and the subtree of each node has at most r leaves, we have $|\Phi_0| = O(r^2 \lg n)$.

Thus, for each node v, we encode the vector of values $\varphi(v) = (\varphi_\pi(v), \varphi_{rk}(v), \varphi_0(v))$. Now, for a given value of $\varphi = (\varphi_\pi, \varphi_{rk}, \varphi_0)$, the number of arrays A of length r with $H_0(A) = \varphi_0$ is at most 2^{φ_0} by a packing argument. So, we have $\mathcal{N}(r, \varphi) \le \mathcal{N}(r, \varphi_0) \le 2^{\varphi_0}$, and hence we can apply Theorem 4 to store an aB-tree of size r, having value $\varphi = (\varphi_\pi, \varphi_{rk}, \varphi_0)$ at the root, using $\varphi_0 + 2$ bits. Summing this space bound over all n/r sub-arrays A_j, we get that the space required to store the aB-trees is at most $\sum_{j=1}^{n/r} (H_0(A_j) + 2) = H_0(m) + 2n/r$ bits.

The additional space required to store the true MTF stack and the rank of each character $c \in \Sigma$ at the beginning of each sub-array $A_j, j \in [n/r]$, is at most $\frac{n}{r}(|\Sigma| \lg n + |\Sigma| \lg |\Sigma|)$.

Now we analyze the space required for the look-up tables. We have the alphabet size $|\Phi| = |\Phi_\pi| \cdot |\Phi_{rk}| \cdot |\Phi_0| \le O(|\Sigma|! \cdot (r+1)^{|\Sigma|} \cdot r^2 \lg n)$ with $r = B^t$. So the look-up tables occupy (in words)

$$O\left(|\Phi|^{B+1} + B \cdot |\Phi|^B\right) = 2^{O(B|\Sigma| \lg |\Sigma| + t|\Sigma| \cdot B \lg B + B \lg \lg n)} = 2^{O(\varepsilon \lg n)} = n^{O(\varepsilon)},$$

where the penultimate equality follows by considering the value of B in two cases:

- If $t|\Sigma| > \lg \lg n$, then $B \lg B = \frac{\varepsilon \lg n}{t|\Sigma|}$. So, $t|\Sigma| \cdot B \lg B = \varepsilon \lg n$, and $B \lg \lg n \le B \cdot t|\Sigma| \le \varepsilon \lg n$.
- Otherwise, $B \lg B = \frac{\varepsilon \lg n}{\lg \lg n}$. So, $B \lg \lg n \le \varepsilon \lg n$, and $t|\Sigma| \cdot B \lg B \le B \lg B \cdot \lg \lg n = \varepsilon \lg n$.

This space usage is negligible for small enough constant $\varepsilon > 0$. However, as $B \ge 2$, the minimum redundancy is (ignoring polylog(n)) terms

$$O\left(|\Phi|^3\right) = O_{|\Sigma|}\left(r^{3(|\Sigma|+2)}\right) = O_{|\Sigma|}\left(r^{3|\Sigma|+6}\right)$$

So, the redundancy is $O\left(\frac{n}{r} + r^{3|\Sigma|+6}\right)$. We balance the terms to get that the redundancy is $O\left(\max\left\{\frac{n}{r}, n^{1-1/(3|\Sigma|+7)}\right\}\right)$. We use the assumption that $|\Sigma| = O(1)$, and adjust t by a constant

factor, to get that the overall space requirement is

$$s = H_0(m) + n \Big/ \left(\frac{\lg n}{\max(t, \lg \lg n)} \right)^t + n^{1-\Omega(1)}.$$

This concludes the proof of Theorem 5.

5 Succinct Rank Data Structure over RLX

In this section, we prove Theorem 3.1, which is restated below:

Theorem 3.1. *There exists a small constant $\delta > 0$ such that for any $x \in \Sigma^n$ and $t \leq \delta \lg n$, there is a succinct data structure $\mathcal{D}_{rk}$ that supports* RANK *queries on $L = \mathsf{BWT}(x)$ in time $O(t')$, using at most $|\mathtt{RLX}(L)| + n \lg n / 2^{t'} + n^{1-\Omega(1)}$ bits of space, in the $w = \Theta(\lg n)$ word-RAM model.*

Setup and Notation. Recall that $L = \mathsf{BWT}(x) \in \Sigma^n$. Let $m = \mathsf{MTF}(L)$. Then m is a string of length n over the MTF alphabet $\{\mathbf{0}, \mathbf{1}, \mathbf{2}, \cdots, |\mathbf{\Sigma}| - \mathbf{1}\}$ (boldface symbols indicate MTF characters). Let $\bar{m}$ be the string obtained from m by replacing each run of 0's with a single character which represents its length. Thus, $\bar{m}$ is a string of length $\bar{N} \leq n$ over the expanded alphabet $\bar{\Sigma} := [|\mathbf{\Sigma}| - \mathbf{1}] \cup [n]$, where $[|\mathbf{\Sigma}| - \mathbf{1}] := \{\mathbf{1}, \mathbf{2}, \cdots, |\mathbf{\Sigma}| - \mathbf{1}\}$. The information-theoretic minimum space required to encode $\bar{m}$ using a *zeroth order prefix-free code* is

$$H_0(\bar{m}) = \sum_{\sigma \in \bar{\Sigma}} f_\sigma \lg \frac{\bar{N}}{f_\sigma},$$

where f_σ is the frequency of σ in $\bar{m}$, for all $\sigma \in \bar{\Sigma}$. Consider any code which converts x to $m = \mathsf{MTF}(L)$ using BWT followed by MTF encoding, and then compresses m using Run-length Encoding of 0-runs followed by prefix-free coding over the expanded alphabet $\bar{\Sigma} = [|\mathbf{\Sigma}| - \mathbf{1}] \cup [n]$. This code requires at least $H_0(\bar{m})$ bits of space. In particular, we have $|\mathtt{RLX}(L)| \geq H_0(\bar{m})$ by definition of the RLX encoding.

We will build an aB-tree over a slightly modified encoding of $\bar{m}$, which is quite similar to the one defined in Section 4 but is succinct with respect to $H_0(\bar{m})$ (and hence with respect to $|\mathtt{RLX}(L)|$).

Let $\varepsilon \in (0, 1)$ be a small constant. We divide each run of 0's of length $\ell_j > n^\varepsilon$ in m into $\lceil \frac{\ell_j}{n^\varepsilon} \rceil$ runs of length at most n^ε each. We then replace each run of 0's by a single character which represents its length. Thus, we get a new string m' of length $N \leq n$ over the alphabet $\Sigma' := [|\mathbf{\Sigma}| - \mathbf{1}] \cup [n^\varepsilon]$. This is done to minimize the space required for the additional look-up tables accompanying the aB-trees which is defined later. The following lemma ensures that this step increases the space usage of the aB-trees by at most an $\tilde{O}\left(n^{1-\varepsilon}\right)$ *additive* term.

Lemma 2. *Let $\bar{m}$ and m' be as defined above. Then*

$$H_0\left(m'\right) \leq H_0\left(\bar{m}\right) + O\left(n^{1-\varepsilon} \lg n\right).$$

Intuitively, this lemma holds because the process of division of large runs introduces at most $n^{1-\varepsilon}$ additional symbols in m' as compared to $\bar{m}$. Moreover, the *relative* frequency of any character $\sigma \in \Sigma'$ only changes slightly, which allows us to bound the difference in the contribution of σ to $H_0(\bar{m})$ and $H_0(m')$. We postpone the formal proof of this lemma to Section 5.4.

5.1 Succinct aB-tree over m', and additional data structures

Fix the branching factor $B \geq 2$ to be constant, and let $O(t')$ be the desired query time. Let $r = B^{t'}$. We divide m' into N/r sub-arrays $A'_1, A'_2, \cdots A'_{N/r}$ of length r, and build an aB-tree over each sub-array. We augment each node v with a value $\varphi(v) = (\varphi_\pi(v), \varphi_{rk}(v), \varphi_0(v))$, where $\varphi_\pi(v) \in \mathcal{S}_{|\Sigma|}$, $\varphi_{rk}(v) = (\varphi_{rk}(v, \sigma))_{\sigma \in [\ell]} \in \{0, 1, \cdots, n^\varepsilon r\}^{|\Sigma|}$, and $\varphi_0(v) \in [0, r \lg N]$. These augmented values have the same meaning as in Section 4. We define these values formally below.

For each node v, we would like $\varphi_\pi(v) \in \mathcal{S}_{|\Sigma|}$ to be the permutation induced by the MTF encoding on the sub-array of m' (which corresponds to a contiguous sub-array of $\mathsf{MTF}(L)$) over which the subtree $\mathcal{T}_v$ is built. Similarly, we would like $\varphi_{rk}(v, \sigma)$ to be the frequency of c_σ in the sub-array rooted at v, assuming the MTF stack at the beginning of this sub-array is $S_0 = (c_1, c_2, \cdots, c_{|\Sigma|})$.

First, we define the augmented values at leaf nodes. Let v be a leaf node corresponding to $m'_i \in \Sigma'$ for some $i \in [N]$. If m'_i is a MTF symbol, i.e., $m'_i \in [|\mathbf{\Sigma}| - \mathbf{1}]$, then we set $\varphi_\pi(v)$ and $\varphi_{rk}(v)$ exactly as defined in Section 4. In particular, we define $\varphi_{rk}(v, \sigma^*) = 1$ for $\sigma^* = m'_i + 1$, and $\varphi_{rk}(v, \sigma) = 0$ for all $\sigma \neq \sigma^*$. If m'_i corresponds to a run of 0's of length ℓ_j in $\mathsf{MTF}(L)$, then we define $\varphi_\pi(v) = \mathbf{Id}_{|\Sigma|}$ to be the identity permutation, $\varphi_{rk}(v, 1) = \ell_j$, and $\varphi_{rk}(v, \sigma) = 0$ for all $\sigma > 1$. Here, we use the fact that the MTF stack does not change within a run of 0's.

We now define the values $\varphi_\pi(v)$ and $\varphi_{rk}(v)$ at each internal node v recursively in terms of the values at its B children $v_1, v_2, \cdots, v_B$, as given by Equations 6 and 8 respectively.

Finally, we specify the entropy constraint φ_0. Recall that for $\sigma \in \Sigma' = [|\mathbf{\Sigma}| - \mathbf{1}] \cup [n^\varepsilon]$, f_σ denotes the frequency of σ in m'. For each $\sigma \in \Sigma'$, we encode each occurrence of σ in m' using $\lg \frac{N}{f_\sigma}$ bits, rounded up to the nearest multiple of $1/r$. We impose a *zeroth-order entropy constraint* by augmenting each node v with $\varphi_0(v)$, the sum of the entropy of the symbols in its subtree. By the same arguments as in Section 4.4, the space occupied by the aB-trees is at most $H_0(m') + 2N/r$.

Additionally, we store the following information, for each $j \in [N/r]$:

- The true MTF stack $S_{(j-1)r}$ at the beginning of the sub-array A'_j.
- The frequency of each character $c \in \Sigma$ in the prefix $m'[1 : (j-1)r] = \left(A'_1, \cdots, A'_{j-1}\right)$.
- The index $i_j \in [n]$ of the character in m corresponding to the first character $m'_{(j-1)r+1}$ of A'_j (if $m'_{(j-1)r+1}$ represents a run in m, then we store the starting index of the run), and its index in memory. Let $T = \{i_j \in [n] \mid j \in [N/r]\}$ be the set of indices.

We also store the map $h : T \to [N/r]$, given by $h(i_j) = j$ for all $j \in [N/r]$. Finally, we build a predecessor data structure D_{pred} over T. As there are at most $\frac{N}{r} \leq \frac{n}{r}$ keys from a universe of size n, there exists a data structure which can answer predecessor queries in time $O(t')$ using space $\frac{n}{r} \cdot r^{\Omega(1/t')} \cdot O(\lg n) = O\left(\frac{n \lg n}{B^{\Theta(t')}}\right)$ bits (for details, see [32]).

5.2 Query algorithm

Let the query be $(c, i) \in \Sigma \times [n]$.

- Compute $i' = D_{pred}(i) \in T$, and index $j = h(i') \in [n/r]$ of the corresponding sub-array in m'.
- Define and initialize the following variables:
  - An MTF stack S, initialized to $S_{(j-1)r}$, the true stack just before A'_j, as well as the corresponding permutation $\pi^* = \pi_{0,(j-1)r}$.
  - A rank counter rk, initialized to the frequency of c in the prefix $m'[1 : (j-1)r]$.

  - A partial sum counter PS, initialized to $i' - 1$. At any point, let v be the last node visited by the query algorithm. Then PS records the index in m corresponding to the left-most node in the sub-array rooted at v (for a run, we store its starting index).

- Start from the root node of the aB-tree built over A'_j and recursively perform the following for each node v (with children $v_1, v_2, \cdots, v_B$) in the path (adaptively defined below), until a leaf node is reached:
  - Let $\beta^* \in [B]$ be the largest index such that $PS + \sum_{\beta=1}^{\beta^*-1} \sum_{\sigma=1}^{|\Sigma|} \varphi_{rk}(v_\beta, c_\sigma) \leq i$.
  - Update S and rk as specified by 7 and 9 respectively, with σ replaced by $\pi^*(\sigma)$.
  - Set $PS \leftarrow PS + \sum_{\beta=1}^{\beta^*-1} \sum_{\sigma=1}^{|\Sigma|} \varphi_{rk}(v_\beta, c_\sigma)$.
  - Recurse to v_{β^*}.
- Let m'_k be the character at the leaf node. If m'_k represents a run of 0's, set $c' = S[1]$, the character at the top of the stack S. Otherwise, set $c' = S[m'_k + 1]$.
- If $c' = c$, set $rk \leftarrow rk + (i - PS)$.
- Return rk.

Now we analyze the query time. The initial predecessor query and computation of sub-array index j requires $O(t')$ time. Then, the algorithm spends $O(1)$ time per node in the aB-tree, which has depth t'. Hence, the overall query time is $O(t')$.

5.3 Space Analysis

Recall that we encoded an approximation of zeroth-order entropy constraint φ_0 as an augmented value in the aB-tree. Using Lemma 2 and arguments similar to those in Section 4.4, we have that the aB-trees occupy at most $H_0(\bar{m}) + \frac{2n}{r} + \tilde{O}(n^{1-\varepsilon})$ bits. The additional data structures require $O\left(\frac{n \lg n \cdot |\Sigma|}{r}\right)$ bits of space.

Now we analyze the space required for the look-up tables. Let $\Phi = (\Phi_\pi, \Phi_{rk}, \Phi_0)$, where Φ_π, Φ_{rk} and Φ_0 are the alphabets over which the augmented values φ_π, φ_{rk} and φ_0 are defined respectively. Then $|\Phi| = |\Phi_\pi| \cdot |\Phi_{rk}| \cdot |\Phi_0| \leq O\left(|\Sigma|! \cdot (n^\varepsilon \cdot r)^{|\Sigma|} \cdot r^2 \lg n\right) = O_{|\Sigma|}\left(n^{\varepsilon \cdot |\Sigma|} \cdot B^{t'(|\Sigma|+2)} \cdot \lg n\right)$, as $r = B^{t'}$. So the look-up tables occupy (in words)

$$O\left(|\Phi|^{B+1} + B \cdot |\Phi|^B\right) = O_{|\Sigma|}\left(n^{\varepsilon \cdot |\Sigma|(B+1)} \cdot B^{(B+1)t'(|\Sigma|+2)} \cdot \lg^{B+1} n\right) = n^{O(\varepsilon)},$$

as $B = \Theta(1)$ and $t' \leq \delta \lg n$ for a small constant $\delta > 0$. So, the look-up tables occupy negligible space for small enough $\varepsilon > 0$. Thus, the overall space required (in bits) is at most

$$H_0(\bar{m}) + \frac{n \lg n \cdot |\Sigma|}{2^{\Omega(t')}} + n^{1-\Omega(1)}.$$

As $|\Sigma| = O(1)$ and $|\texttt{RLX}(L)| \geq H_0(\bar{m})$, we can adjust t' by a constant factor to obtain Theorem 3.1.

5.4 Proof of Lemma 2

Proof. We assume m contains at least one run of 0's of length exceeding n^ε, since $H_0(m') = H_0(\bar{m})$ otherwise. Let κ and κ' be the number of characters representing 0-runs in $\bar{m}$ and m' respectively. Let n' be the number of non-zero MTF characters in m' (or $\bar{m}$). Then $N = n' + \kappa'$, and $\bar{N} = n' + \kappa$.

For $i \in [n]$, let g_i be the number of runs of length i in $\bar{m}$. For $i \in [n^\varepsilon]$, let $\tilde{g}_i$ be the *additional* number of runs of length i introduced in m' through this transformation. Let κ_{sm} be the number of characters representing runs of length at most n^ε in $\bar{m}$.

The following facts are immediate:

$$\sum_{i=1}^{n^\varepsilon} g_i = \kappa_{sm} \leq \kappa. \tag{10}$$

$$\sum_{i=1}^{n^\varepsilon} (g_i + \tilde{g}_i) = \kappa' \leq \kappa_{sm} + 2n^{1-\varepsilon} \leq \kappa + 2n^{1-\varepsilon}. \tag{11}$$

We use these facts along with the Log-Sum Inequality to prove the lemma below.

$$\begin{aligned}
&H_0\left(m'\right) - H_0\left(\bar{m}\right) \\
&= \sum_{\sigma \in [|\mathbf{\Sigma}|-\mathbf{1}]} f_\sigma \lg \frac{n'+\kappa'}{f_\sigma} + \sum_{i=1}^{n^\varepsilon} (g_i + \tilde{g}_i) \lg \frac{n'+\kappa'}{g_i + \tilde{g}_i} - \sum_{\sigma \in [|\mathbf{\Sigma}|-\mathbf{1}]} f_\sigma \lg \frac{n'+\kappa}{f_\sigma} - \sum_{i=1}^{n} g_i \lg \frac{n'+\kappa}{g_i} \\
&\leq \sum_{\sigma \in [|\mathbf{\Sigma}|-\mathbf{1}]} f_\sigma \lg \frac{n'+\kappa'}{n'+\kappa} + \sum_{i=1}^{n^\varepsilon} g_i \lg \frac{n'+\kappa'}{n'+\kappa} + \sum_{i=1}^{n^\varepsilon} \tilde{g}_i \lg \frac{n'+\kappa'}{g_i + \tilde{g}_i} \\
&\leq (n'+\kappa) \lg \frac{n'+\kappa'}{n'+\kappa} + \sum_{i=1}^{n^\varepsilon} \tilde{g}_i \lg \frac{\sum_{i=1}^{n^\varepsilon} n'+\kappa'}{\sum_{i=1}^{n^\varepsilon} (g_i + \tilde{g}_i)} && \text{(Log-Sum Inequality)} \\
&\leq (n'+\kappa) \lg \left(1 + \frac{2n^{1-\varepsilon}}{n'+\kappa}\right) + (\kappa' - \kappa_{sm}) \lg \frac{n^\varepsilon (n'+\kappa')}{\kappa'} && \left(\textstyle\sum_{i=1}^{n^\varepsilon} \tilde{g}_i = \kappa' - \kappa_{sm}\right) \\
&\leq 2n^{1-\varepsilon} \left[\lg(e) + O(\lg n)\right] && \left(\lg_2(1+x) \leq x \lg_2(e),\ n'+\kappa' \leq n,\ \kappa' \geq 1\right) \\
&= O\left(n^{1-\varepsilon} \lg n\right).
\end{aligned}$$

The last two inequalities above follow from 10 and 11. □

6 Reporting pattern occurrences

In this section, we prove the existence of a succinct data structure for reporting the positions of occurrences of a given pattern in a string. For $x \in \Sigma^n$ and a pattern $p \in \Sigma^\ell$, let $occ(p)$ be the number of occurrences of p as a contiguous substring of x.

Theorem 6. *Fix a string $x \in \Sigma^n$. For any t, there is a succinct data structure that, given a pattern $p \in \Sigma^\ell$, reports the starting positions of the $occ(p)$ occurrences of p in x in time $O(t \cdot occ(p) + \ell \cdot \lg t)$, using at most*

$$|\mathit{RLX}(\mathsf{BWT}(x))| + O\left(\frac{n \lg n \lg t}{t}\right) + n^{1-\Omega(1)}$$

bits of space, in the $w = \Theta(\lg n)$ word-RAM model.

For reporting queries, this is a quadratic improvement over the FM-Index [13].

Proof. Let $t', \tilde{t} < t$ be parameters to be determined shortly. Let D_{rk} be the data structure given by Theorem 3.1 which supports RANK queries on L in time $O(t')$. We divide the original string x into $\lceil n/T \rceil$ blocks of size $T := O\left(\frac{t}{t'+\tilde{t}}\right)$. Let $S = \{(j-1)T+1 \mid j \in [n/T]\}$ be the set of starting indices of blocks. Let F_S be the set of indices in the first column F of the BWT Matrix $\mathcal{M}$ corresponding to indices in S. We store the map $h : F_S \mapsto S$. Moreover, we store a membership data structure on $[n]$, which given a query $i \in [n]$, answers `Yes` iff $i \in F_S$. This can be done using the data structure in [31] which answers RANK queries over $\{0,1\}^n$[15] in time $O(\tilde{t})$ using space (in bits)

$$\lg \binom{n}{n/T} + \frac{n}{(\lg n/\tilde{t})^{\tilde{t}}} + \tilde{O}\left(n^{3/4}\right) \leq \frac{n}{T}\lg(eT) + \frac{n}{(\lg n/\tilde{t})^{\tilde{t}}} + \tilde{O}\left(n^{3/4}\right). \qquad \left(\tbinom{n}{k} \leq \left(\tfrac{en}{k}\right)^k\right)$$

Our algorithm will follow the high-level approach to reporting pattern occurrences given in [13], replacing each component data structure with the corresponding succinct data structure described above. We first use D_{rk} to count the number of occurrences $occ(p)$ with $O(\ell \cdot |\Sigma|)$ RANK queries on L, which requires $O(\ell\, t')$ time. We observe that the algorithm for counting occurrences specified in [13] actually provides a contiguous set of rows $[R : R + occ(p) - 1] \subset [n]$ in the Burrows-Wheeler matrix $\mathcal{M}$ which are prefixed by p. For each $i \in [R : R + occ(p) - 1]$, the algorithm starts from i and performs $\alpha \leq T$ iterations of the LF-mapping algorithm[16] from Section 2.1.1, until it reaches an index $i' \in F_S$ (which is verified using the membership data structure). Then it reports $h(i') + \alpha$. By Lemma 1, each step requires $O(|\Sigma|)$ RANK queries on L, each of which can be done using $\mathcal{D}_{rk}$ in $O(t')$ time, hence the overall running time of the reporting phase is $O(T \cdot (t'+\tilde{t}) \cdot occ(p)) = O(t \cdot occ(p))$ by definition of T. We can assume $T \leq t = o(n)$, because otherwise we can decompress the entire string x in time $O(n) = O(t)$. So $eT = o(n)$, and the total space required (in bits) is at most

$$|\mathtt{RLX}(\mathsf{BWT}(x))| + \frac{n \lg n}{2^{t'}} + \frac{n}{(\lg n/\tilde{t})^{\tilde{t}}} + \frac{n}{T}\lg n + n^{1-\Omega(1)}.$$

In order to balance the redundancy terms in the above expression, we set $t' = \Theta(\lg T)$. Using the fact that $\tilde{t} \leq t'$ if the first two redundancy terms are equal, we get $t = O(T \cdot (\tilde{t}+t')) = O(T \lg T)$, so $T = O(t/\lg t)$. Thus, we get a *succinct* data structure for reporting the starting positions of the $occ(p)$ occurrences of a pattern $p \in \Sigma^\ell$ in x in time $O(t \cdot occ(p) + \ell \cdot \lg t)$, using at most

$$|\mathtt{RLX}(\mathsf{BWT}(x))| + O\left(\frac{n \lg n \lg t}{t}\right) + n^{1-\Omega(1)}$$

bits of space. This concludes the proof of Theorem 6. □

7 Lower Bound for Symmetric Data Structures

In this section, we prove a cell-probe lower bound on the redundancy of any succinct data structure that locally decodes the BWT permutation $\Pi_x : \mathsf{BWT}(x) \mapsto x$, induced by a *uniformly random* n-bit string $x \in \{0,1\}^n$. While it is somewhat unnatural to consider uniformly random (context-free) strings in BWT applications, we stress that our lower bound is more general and can yield nontrivial lower bounds for *non-product* distribution μ on $\{0,1\}^n$ which satisfy the premise of our "Entropy-Polarization" Lemma 3 below, possibly with weaker parameters (we prove this lemma for the uniform distribution, but the proof can be generalized to distributions with "sufficient block-wise independence").

[15] The bit-string y is the indicator vector of F_S. For $i \in [n]$, we have $i \in F_S$ iff $rk_y(1,i) \neq rk_y(1,i-1)$.

[16] Note that the LF Mapping algorithm can equivalently be considered to be jumps in the first column F.

The lower bound we prove below applies to a somewhat stronger problem than Problem 1, in which the data structure must decode *both* forward and *inverse* evaluations $(\Pi_x(i), \Pi_x^{-1}(j))$ of the induced BWT permutation. The requirement that the data structure recovers $\Pi_x^{-1}(j)$, i.e., the position of x_j in L, when decoding x_j, is very natural (and, when decoding the entire input x, is in fact without loss of generality). The "symmetry" assumption, namely, the implicit assumption that any such data structure must also efficiently compute *forward* queries $\Pi_x(i)$ mapping positions of $i \in L$ to their corresponding index $j \in X$, is less obvious, but is also a natural assumption given the sequential nature of the BWT decoding process (LF property, Lemma 1). Indeed, both the FM-index [13] and our data structure from Theorem 1 are essentially symmetric.[17] We shall prove Theorem 3.2, restated below:

Theorem 3.2. *Let $X \in_R \{0,1\}^n$ and let $\Pi_X \in \mathcal{S}_n$ be the induced BWT permutation from indices in $L := \mathsf{BWT}(X)$ to indices in X. Then, any data structure that computes $\Pi_X(i)$ and $\Pi_X^{-1}(j)$ for every $i, j \in [n]$ in time t, q respectively, such that $t \cdot q \leq \delta \lg n / \lg\lg n$ (for some constant $\delta > 0$), in the cell-probe model with word size $w = \Theta(\lg n)$, must use $n + \Omega\left(n/tq\right)$ bits of space in expectation.*

When $q = \Theta(t)$, our result implies that obtaining an $r \ll n/t^2$ trade-off for Problem 1 is generally impossible, hence Theorem 3.2 provides an initial step in the lower bound study of Problem 1.

The data structure problem in Theorem 3.2 is a variant of the *succinct permutations* problem Perms [16, 28], in which the goal is to represent a random permutation $\Pi \in_R \mathcal{S}_n$ succinctly using $\lg n! + o(n \lg n)$ bits of space, supporting both forward and inverse evaluation queries ($\Pi(i)$ and $\Pi^{-1}(i)$), in query times t, q respectively. Golynski [16] proved a lower bound of $r \geq \Omega(n \lg n/tq)$ on the space redundancy of any such data structure, which applies whenever

$$t, q \leq O\left(\frac{H(\Pi)}{n \cdot \lg\lg n}\right). \tag{12}$$

When $\Pi \in_R \mathcal{S}_n$ is a *uniformly random* permutation, (12) implies that the bound holds for $t, q \leq O(\lg n / \lg\lg n)$. However, in the setting of Theorem 3.2, this result does not yield *any* lower bound, since in our setting $H(\Pi_X) \leq n$ (as the BWT permutation of X is determined by X itself), hence (12) gives a trivial condition on the query times t, q. More precisely, the fact that $H(\Pi_X) \leq n$ implies that *an average* query $i \in [n]$ only reveals $\frac{1}{n}\sum_{i=1}^{n} H\left(\Pi_X(i) \,|\Pi_X(i-1), \ldots, \Pi_X(1)\right) = O(1)$ bits of information on X, in sharp contrast to a *uniformly random* permutation, where an average query reveals $\approx \lg n$ bits. This crucial issue completely dooms the cell-elimination argument in [16], hence it fails to prove anything for Theorem 3.2.

In order to circumvent this problem, we first show that a variant of [16]'s argument in fact yields the following generalized lower bound on the Perms problem, which applies to arbitrary (i.e., nonuniform) random permutations $\Pi \sim \mu$, as long as there is a *restricted subset* of queries $S \subseteq [n]$ with large enough entropy:

Theorem 7 (Cell-Probe Lower Bound for Nonuniform Perms, essentially [16] Lemma 3.1)**.** *Let $\Pi \sim \mu$ be a permutation chosen according to some distribution μ over $\mathcal{S}_n$. Suppose that there exists a subset of coordinates $S = S(\Pi) \subseteq [n]$, $|S| = \gamma$, and $\alpha > 0, \varepsilon \leq 1/2$ such that $H(\Pi_S | S, \Pi(S)) = \gamma \cdot \alpha \geq (1-\varepsilon) H_\mu(\Pi)$ bits. Then any 0-error succinct data structure for Perms_n under μ, in the cell-probe model with word size w, with respective query times t, q satisfying*

$$t, q \leq \min\left\{2^{w/5}, \frac{1}{32} \cdot \frac{\alpha}{\lg w}\right\} \text{ and } tq \leq \delta \min\left(\frac{\alpha^2}{w \lg(en/\gamma)}, \frac{\alpha}{2\varepsilon\, w}\right)$$

[17] Indeed, we can achieve $q = O(t)$ by increasing the *redundancy* r by at most a factor of 2, for storing an additional "marking index" for the reverse permutation, see the first paragraph of Section 3.

for some constant $\delta > 0$, must use $s \geq H_\mu(\Pi) + r$ bits of space in expectation, where

$$r = \Omega\left(\frac{\alpha^2 \cdot \gamma}{w \cdot tq}\right) .$$

Here, $\Pi_S := (\Pi(S_{i_1}), \Pi(S_{i_2}), \ldots, \Pi(S_{i_s}))$ denotes the projection of Π to S, i.e., the *ordered set* (vector) of evaluations of Π on S, while $\Pi(S)$ denotes the image of Π under S (the unordered set), and $H_\mu(Z)$ is the Shannon entropy of $Z \sim \mu$.

Theorem 7 implies that in order to prove a nontrivial bound in Theorem 3.2, it is enough to prove that the BWT-induced permutation Π_X when applied on random X, has a (relatively) small subset of coordinates with near-maximal entropy (even though *on average* it is constant). Indeed, we prove the following key lemma about the entropy distribution of the BWT permutation on random strings. Informally, it states that when X is random, while the *average* entropy of a coordinate (query) of Π_X is indeed only $H_\mu(\Pi_X(i)|\Pi_X(< i)) = O(1)$, this random variable has a lot of *variance*: A small subset ($\sim n/\lg n$) of coordinates have very high ($\sim \lg n$) entropy, whereas the rest of the coordinates have $o(1)$ entropy on average. This is the content of the next lemma:

Lemma 3 (Entropy Polarization of the BWT Permutation)**.** *Let $X \in_R \{0,1\}^n$, and μ be the distribution on $\mathcal{S}_n$ induced by BWT on X. For any $\varepsilon \geq \Omega(\lg\lg n/\lg n)$, with probability at least $1 - \tilde{O}\left(n^{-(1-\varepsilon/3)}\right)$, there exists a set $S \subset [n]$ of size $|S| = (1-O(\varepsilon))n/\lg n$, such that $H_\mu(\Pi_S|S, \Pi(S)) \geq n(1-\varepsilon)$.*

We first prove Theorem 3.2, assuming Theorem 7 and Lemma 3. We set $\varepsilon = O(\lg\lg n/\lg n)$. Let $S \subset [n]$ be the set of size $\gamma = |S| = (1 - O(\varepsilon))n/\lg n$ obtained from Lemma 3. We invoke Theorem 7 with the set S, $\alpha = (1 - O(\varepsilon))\lg n$, and μ being the distribution on $\mathcal{S}_n$ induced by BWT on $X \in_R \{0,1\}^n$. As BWT is an invertible transformation on $\{0,1\}^n$ and X is a uniformly random bit-string, Π_X must be a uniformly random permutation over a subset of $\mathcal{S}_n$ of size 2^n. So, $H_\mu(\Pi) = n$.

Let $\mathcal{D}$ be any 0-error data structure that computes $\Pi_X(i)$ and $\Pi_X^{-1}(j)$ for every $i, j \in [n]$ in time t, q such that $t\,q \leq \delta \lg n/\lg\lg n$ for small enough $\delta > 0$. Then it is easy to see that with probability at least $1 - \tilde{O}\left(n^{-(1-\varepsilon)}\right)$, the conditions of Theorem 7 are satisfied, and we get that $\mathcal{D}$ must use

$$s \geq \left(1 - \tilde{O}\left(n^{-(1-\varepsilon/3)}\right)\right)(H_\mu(\Pi) + r) = n + r - \tilde{O}\left(n^{\varepsilon/3}\right) = n + r - o(r)$$

bits of space in expectation, where

$$r \geq \Omega\left(\frac{\alpha^2 \cdot \gamma}{w \cdot tq}\right) = \Omega\left(\frac{n}{tq}\right) .$$

This concludes the proof of Theorem 3.2. Now we prove the Entropy Polarization Lemma 3, and then prove Theorem 7.

7.1 Proof of Lemma 3

Let $X \in_R \{0,1\}^n$, and $L = \mathsf{BWT}(X)$. Let $\Pi_X : L \to X$ denote the BWT-permutation between indices of X (i.e., $[n]$) and L. For convenience, we henceforth think of Π_X as the permutation between X and the *first column* F of the BWT matrix $\mathcal{M}$ of X, i.e., $\Pi_X : F \to X$. Note that these permutations are equivalent up to a fixed rotation. We also denote Π_X by Π, dropping the subscript X.

Recall that for any subset of coordinates $S \subseteq [n]$, $\Pi(S)$ denotes the image (unordered set of coordinates) of S in F under Π, and Π_S denotes the *ordered* set (i.e., the projection of Π to S).

Let $\ell = (1+\varepsilon)\lg n$, $\tilde{s} := \lfloor n/\ell \rfloor$. For $i \in [\tilde{s}]$, let $Y_i = X[(i-1)\ell + 1 : i\ell]$ be the ith block of X of length ℓ. Let J_i be the index of the row in the BWT matrix $\mathcal{M}$ which starts with Y_i; note that $F[J_i]$ is the first character in Y_i. Let $\tilde{S} = \{J_i \,|\, i \in [\tilde{s}]\}$. So $|\tilde{S}| = \tilde{s} = n/((1+\varepsilon)\lg n)$.

We first show the existence of a large subset of blocks Y_i which are pairwise distinct, with high probability. The BWT ordering among the rows J_i corresponding to the cyclic shifts starting with these blocks is consistent with the (unique) lexicographical ordering among the blocks themselves. We then show that the lexicographical ordering among these blocks is uniform. As this ordering is determined by the BWT permutation restricted to the corresponding rows J_i, the permutation restricted to these rows J_i itself must have high entropy.

For $1 \le i < j \le \tilde{s}$, we say that there is a "collision" between blocks Y_i and Y_j if $Y_i = Y_j$, and define $Z_{i,j} \in \{0,1\}$ to be the indicator of this event. Let $Z = \sum_{1 \le i < j \le \tilde{s}} Z_{ij}$ be the number of collisions that occur among the disjoint blocks of length ℓ.

Claim 1. *Let E be the event that the number of collisions $Z \le n^{1-\varepsilon}/\lg^2 n$. Then $\mathbf{Pr}[E] \ge 1 - \tilde{O}\left(n^{-(1-\varepsilon)}\right)$.*

Proof. Fix $1 \le i < j \le \tilde{s}$. Then Y_i and Y_j are disjoint substrings of X of length $\ell = (1+\varepsilon)\lg n$. As X is a uniformly random string of length n, we have that Y_i and Y_j are independent and uniformly random strings of length ℓ. So,

$$\mathbb{E}[Z_{ij}] = \mathbf{Pr}[Y_i = Y_j] = 2^{-\ell} = n^{-(1+\varepsilon)}.$$

By linearity of expectation and the fact that $\tilde{s} \le n/\lg n$, we have

$$\mathbb{E}[Z] = \sum_{1 \le i < j \le \tilde{s}} \mathbb{E}[Z_{ij}] = \binom{\tilde{s}}{2} \frac{1}{n^{1+\varepsilon}} \le \frac{n^{1-\varepsilon}}{2\lg^2 n}.$$

Fix $1 \le i < j \le \tilde{s}$. As Z_{ij} is an indicator random variable, we have

$$\mathbf{Var}[Z_{ij}] = \mathbf{Pr}[Y_i = Y_j](1 - \mathbf{Pr}[Y_i = Y_j]) \le \mathbf{Pr}[Y_i = Y_j].$$

It is easy to see that the random variables Z_{ij} are pairwise independent. So, we have

$$\mathbf{Var}[Z] = \sum_{1 \le i < j \le \tilde{s}} \mathbf{Var}[Z_{ij}] \le \sum_{1 \le i < j \le \tilde{s}} \mathbf{Pr}[Y_i = Y_j] = \mathbb{E}[Z].$$

By Chebyshev's inequality and the fact that $\mathbb{E}[Z] \ge \frac{n^{1-\varepsilon}}{4\lg^2 n}$, we have

$$\mathbf{Pr}\left[Z > \frac{n^{1-\varepsilon}}{\lg^2 n}\right] \le \mathbf{Pr}[|Z - \mathbb{E}[Z]| > \mathbb{E}[Z]] \le \frac{\mathbf{Var}[Z]}{\mathbb{E}[Z]^2} \le \frac{1}{\mathbb{E}[Z]} \le \frac{4\lg^2 n}{n^{1-\varepsilon}}.$$

□

For the remainder of this proof, we assume that the event E holds. Let $T = \{i \in [\tilde{s}] \,|\, Y_i \ne Y_j \,\forall\, j \ne i\}$ be the set of indices of blocks Y_i which do not collide with any other block. Let $s = |T|$, and $S = \{J_i \in [n] \,|\, i \in T\} \subset \tilde{S}$ be the corresponding set of indices J_i. As E holds, we have

$$|S| = s \ge \tilde{s} - 2Z \ge \frac{n}{(1+\varepsilon)\lg n} - \frac{2n^{1-\varepsilon}}{\lg^2 n} \ge \frac{n}{(1+2\varepsilon)\lg n} \ge \frac{n(1-2\varepsilon)}{\lg n}.$$

Now, given E and T, we define an ordering among the blocks $Y_i, i \in T$. For $j \in [s]$, we define $L_j = i^* \in [s]$ if Y_{i^*} is the jth smallest string (lexicographically) among $Y_i, i \in T$. Finally, let $\mathcal{L} = (L_1, \cdots, L_s)$. By abuse of notation, we can identify $\mathcal{L}$ with a permutation on s elements.

Note that, given E and T, this order specified by $\mathcal{L}$ is consistent with the relative ordering of the corresponding cyclic shifts of X starting with $Y_i, i \in T$, in the BWT Matrix $\mathcal{M}$. More precisely, we have $L_j = i^*$ if J_{i^*} is the jth smallest row index among $S = \{J_i \,|\, i \in T\}$. However, we define $\mathcal{L}$'s in this manner to emphasize that it is well-defined *without* reference to S.

We will use the following fact about conditional entropy multiple times:

Fact 2. *Let A, B be random variables in the same probability space. Then $H(A|B) \geq H(A) - H(B)$.*

Proof. The proof is immediate using the chain rule:

$$H(A) \leq H(A, B) = H(B) + H(A|B).$$

□

The following claim allows us to remove the conditioning on the set $\Pi(S)$ at the cost of a negligible loss in entropy.

Claim 2. *Let $S, \Pi(S)$ and Π_S be as defined above. Then $H_\mu(\Pi_S|S, \Pi(S)) \geq H_\mu(\Pi_S|S) - O(s \cdot \lg(n/s))$.*

Proof. We use Fact 2 with $A = \Pi_S|S$ and $B = \Pi(S)$ to get that $H_\mu(\Pi_S|S, \Pi(S)) \geq H_\mu(\Pi_S|S) - H_\mu(\Pi(S))$. Now, as $\Pi(S)$ is a s-size subset of the set $\{(i-1)\ell + 1 \,|\, i \in [\tilde{s}]\}$, we have that

$$H_\mu(\Pi(S)) \leq \lg \binom{\tilde{s}}{s} = O\left(s \lg \frac{\bar{s}}{s}\right) \leq O\left(s \lg \frac{n}{s}\right).$$

□

We will invoke Claim 2 later with $s = \Theta(n/\lg n)$. Note that for any $\varepsilon \geq \Omega(\lg\lg n/\lg n)$, we have that $s \lg(n/s) \leq O(s \lg\lg n) \leq \varepsilon n$, so the loss in entropy is negligible.

Claim 3. *Let $S, \mathcal{L}, \Pi_S, E$ and T be as defined above. Then $H_\mu(\Pi_S|S) \geq H(\mathcal{L}|S, E, T)$.*

Proof. It is enough to show that given E and the set S, the permutation Π_S (restricted to S) determines T and the ordering $\mathcal{L}$ among the blocks $Y_i, i \in T$. But this is true because, for any $j \in S$, the index $\Pi_S(j)$ specifies the position of $F[j]$ in X. In particular (as $j \in S$), it specifies the block index i such that $F[j]$ is the first character of Y_i, i.e., $j = J_i$. Then T is the set of these block indices. As this mapping is specified for all indices $j \in S$, the set T and the ordering $\mathcal{L}$ on T are clearly determined by Π_S, given S. So, $H_\mu(\Pi_S|S) \geq H_\mu(\Pi_S|S, E) \geq H(\mathcal{L}|S, E, T)$. □

As described earlier, given E and T, $\mathcal{L}$ is well-defined without reference to S. We now combine the previous claims to reduce the problem of lower-bounding $H_\mu(\Pi_S|S, \Pi(S))$ to that of lower-bounding $H(\mathcal{L}|E, T)$, while losing negligible entropy.

Claim 4. *Let $E, T, \mathcal{L}, S, \Pi(S)$ and Π_S be as defined above. Then $H_\mu(\Pi_S|S, \Pi(S)) \geq H(\mathcal{L}|E, T) - O(s \cdot \lg\lg n)$.*

Proof. From Claims 2, 3, Fact 2 and the fact that $|\Pi(S)| = |S| = s = \Theta(n/\lg n)$, we have

$$\begin{aligned} H_\mu(\Pi_S|S,\Pi(S)) &\geq H_\mu(\Pi_S|S) - O\left(s \cdot \lg \frac{n}{s}\right) \\ &\geq H(\mathcal{L}|S,E,T) - O(s \cdot \lg\lg n) \\ &\geq H(\mathcal{L}|E,T) - O(s \cdot \lg\lg n) \end{aligned}$$

For the last inequality, we apply Fact 2 with $A = \mathcal{L}|E,T$ and $B = S$, and upper bound $H_\mu(S)$ by $O(s \lg\lg n)$ using the same argument that was used to bound $H_\mu(\Pi(S))$. □

Claim 5. *Let $E, \mathcal{L}, T$ be as defined above. Then $H(\mathcal{L}|E,T) = \lg(s!) \geq s\lg n - O(s\lg\lg n)$.*

Proof. For $X \in \{0,1\}^n$ and permutation $\tau \in \mathcal{S}_{\tilde{s}}$, define $X^\tau \in \{0,1\}^n$ to be the string obtained by shifting the entire ith substring Y_i of X to the $\tau(i)$th block, for all $i \in [\tilde{s}]$. Formally, $X^\tau[(\tau(i)-1)\ell+1 : \tau(i)\ell] = X[(i-1)\ell+1 : i\ell] = Y_i$. We observe that X satisfies E if and only if X^τ satisfies E. This is because E is determined purely by inequalities among pairs of substrings Y_i, Y_j, which are kept intact by τ as it permutes entire blocks.

Now, the event E is clearly determined by X. So, for any fixed string $x \in \{0,1\}^n$, we have that $\mathbf{Pr}[E|X=x]$ is either 0 or 1. This implies that $\mathbf{Pr}[X = x|E] = \frac{\mathbf{Pr}[X=x]}{\mathbf{Pr}[E]}$ if x satisfies E, and $\mathbf{Pr}[X = x|E] = 0$ otherwise.

Let x be a string which satisfies E. Let $\tau \in \mathcal{S}_{\tilde{s}}$ be any permutation. Then

$$\mathbf{Pr}[X = x \mid E] = \frac{\mathbf{Pr}[X=x]}{\mathbf{Pr}[E]} = \frac{\mathbf{Pr}[X=x^\tau]}{\mathbf{Pr}[E]} = \mathbf{Pr}[X = x^\tau \mid E].$$

The second equality follows from the fact that X is uniform and the last equality follows because x satisfies E if and only if x^τ satisfies E. Similarly, for any permutation $\tau \in \mathcal{S}_{\tilde{s}}$ and any string x that does not satisfy E, we have $\mathbf{Pr}[X = x|E] = 0 = \mathbf{Pr}[X = x^\tau|E]$. This shows that the conditional distribution of X, given E, is still uniform over a subset of $\{0,1\}^n$ of size $2^n(1-o(1))$.

We show that, given E and any fixed subset $T \subset [\tilde{s}]$ of size $s \geq n(1-2\varepsilon)/\lg n$ such that the ℓ-length substrings indexed by T are all distinct, all $s!$ orderings $\mathcal{L}$ on T are equally likely. As $\mathcal{L}$ is defined with respect to the lexicographical ordering among the blocks $Y_i, i \in T$, we only consider permutations $\tau \in \mathcal{S}_{\tilde{s}}$ which fix indices outside T, i.e., $\tau(i) = i$ for all $i \in [\tilde{s}] \setminus T$. Fix any two distinct orderings $\mathcal{L}_1, \mathcal{L}_2 \in \mathcal{S}_s$. For $i \in \{1,2\}$, let $\Lambda_i^T \subset \{0,1\}^n$ be the set of strings x that satisfy event E with respect to the set T (i.e., the set of substrings $x[(i-1)\ell+1 : i\ell]$) are distinct for all $i \in T$), and give rise to the ordering $\mathcal{L}_i$ on T. We show that $|\Lambda_1^T| = |\Lambda_2^T|$ by exhibiting a bijection $f : \Lambda_1^T \to \Lambda_2^T$.

Let $\tau \in \mathcal{S}_{\tilde{s}}$ be the unique permutation which converts $\mathcal{L}_1$ to $\mathcal{L}_2$, and fixes indices outside T. For $x \in \Lambda_1^T$, define $f(x) = x^\tau$. Then $f(x)$ satisfies E as x satisfies E. Moreover, as x induces the order $\mathcal{L}_1$ on T, $f(x)$ must induce the order $\mathcal{L}_2$ on T. Hence, $f(x) \in \Lambda_2^T$. Clearly, this map is one-one. We observe that the inverse map $f^{-1} : \Lambda_2^T \to \Lambda_1^T$ is given by $f^{-1}(x) = x^{\tau^{-1}}$. So, f is a bijection, and hence $|\Lambda_1^T| = |\Lambda_2^T|$. But any string in $\Lambda_1^T \cup \Lambda_2^T$ is equally probable under the distribution of X conditioned on E. So, the probability of the induced ordering $\mathcal{L}$ on T being $\mathcal{L}_1$ or $\mathcal{L}_2$ is equal, i.e., $\mathbf{Pr}[\mathcal{L} = \mathcal{L}_1 \mid E,T] = \mathbf{Pr}[\mathcal{L} = \mathcal{L}_2 \mid E,T]$. As this is true for all pairs of orderings on T, we have that $\mathcal{L}$ is uniform on $s!$ orderings, given E,T. As $s = \Theta(n/\lg n)$, we have

$$H(\mathcal{L}|E,T) = \lg(s!) \geq s\lg s - O(s) \geq s\lg n - O(s\lg\lg n).$$

□

We conclude the proof of Lemma 3 by combining Claims 4 and 5, and replacing ε by $\varepsilon/3$:

$$H_\mu(\Pi_S|S,\Pi(S)) \geq H(\mathcal{L}|E,T) - O(s\cdot\lg\lg n) \geq s\lg n - O(s\cdot\lg\lg n) \geq n(1-2\varepsilon) - O(s\cdot\lg\lg n) \geq n(1-3\varepsilon).$$

7.2 Proof of Theorem 7

Proof. Let $\Pi \sim_\mu \mathcal{S}_n$ and let $S = S(\Pi) \subseteq [n]$ be the subset of $|S| = \gamma$ queries satisfying the premise of the theorem. Let D be a data structure for $\textsc{Perms}_n$ under μ which correctly computes the (forward and inverse) answers with query times t, q respectively (satisfying the bounds in the theorem statement), and $s = h + r'$ words of space, where $h = H_\mu(\Pi)/w$ and $r' = r/w$.

The idea is to adapt Golynski's argument [16] to the restricted set of "high entropy" queries $(S, \Pi(S))$ and use D in order to efficiently encode the answers of Π_S, while the remaining answers $(\Pi_{\overline{S}})$ can be encoded in a standard way using $\sim H_\mu(\Pi_{\bar{S}}|S, \Pi_S)$ extra bits (as this conditional entropy is very small by the premise).

To this end, suppose Alice is given $\Pi \sim \mu$ and $S = S(\Pi)$. Alice will first use D_S to send Π_S, along with the sets $S, \Pi(S) \subset [n]$, to Bob. The subsequent encoding argument proceeds in stages, where in each stage we delete one "unpopular cell" of D_S, i.e., a cell which is accessed by *few* forward and inverse queries in $S, \Pi(S)$, and protect some other cells from deletion in future stages. The number of stages z is set so that at the end, the number of remaining unprotected cells (i.e., the number of cells which were not protected nor deleted in any stage) is at least $h/2$.

Following [16], for any cell $l \in [h + r']$ we let $F_S(l)$ and $I_{\Pi(S)}(l)$ denote the subset of forward and inverse queries (in $S, \Pi(S)$ respectively) which probe l. Fix a stage k. Let C_k^S denote the *remaining* cells at the end of stage k (i.e., the set of undeleted unprotected cells) probed by any query $i \in S \cup \Pi(S)$. So $|C_0| = h + r'$ and $|C_z^S| \geq \frac{h}{2}$. For a query $i \in S$, let $R_k(i)$ denote the set of remaining cells at stage k probed by i. The average number of forward queries in S that probe a particular cell is

$$\frac{1}{|C_k^S|} \sum_{l \in C_k^S} |F_S(l)| = \frac{1}{|C_k^S|} \sum_{i \in S} |R_k(i)| \leq \frac{|S|\, t}{|C_k^S|} = \frac{\gamma\, t}{|C_k^S|}.$$

As such, there are at most $|C_k^S|/3$ cells which are probed by at least $3\frac{\gamma\, t}{|C_k^S|}$ forward queries in S. By an analogous argument, there are at most $|C_k^S|/3$ cells which are probed by at least $3\frac{\gamma\, q}{|C_k^S|}$ inverse queries in $\Pi(S)$. Hence, we can find a cell $d_k \in C_k^S$ which is probed by at most $\beta := 3\frac{\gamma\, t}{|C_k^S|}$ forward queries in S and $\beta' := 3\frac{\gamma\, q}{|C_k^S|}$ inverse queries in $\Pi(S)$. As $|C_k^S| \geq h/2$, we have that $\beta \leq 6\frac{\gamma t}{h}$ and $\beta' \leq 6\frac{\gamma\, q}{h}$. We delete (the contents of) this cell d_k. Let

$$F'_S(d_k) := F_S(d_k) \cap \Pi^{-1}(I_{\Pi(S)}(d_k))$$

denote the set of forward queries $i \in S$ such that both i and the reciprocal $j := \Pi(i) \in \Pi(S)$ probe the deleted cell d_k (note that $F'_S(d_k)$ is well defined even when D_S is adaptive, given the input Π). Define $I'_{\Pi(S)}(d_k)$ analogously. We say that the queries in $F'_S(d_k)$ and $I'_{\Pi(S)}(d_k)$ are *non-recoverable*. Note that, by definition, Π defines a bijection between $F'_S(d_k)$ and $I'_{\Pi(S)}$, i.e.,

$$\Pi(F'_S(d_k)) = I'_{\Pi(S)}(d_k). \tag{13}$$

In order to "preserve" the information in d_k, Alice will send Bob an array A_{d_k} of β entries, each of size $\lg \beta'$ bits, encoding the bijection $\Pi_{F'_S(d_k)} : F'_S(d_k) \longleftrightarrow I'_{\Pi(S)}(d_k)$ so that Bob can recover the answer to a query pair (i, j) in the event that both i and $j = \Pi(i)$ access the deleted cell d_k. Let $P(d_k) := \bigcup_{i \in \Pi(F_S(d_k)) \cup \Pi^{-1}(I_{\Pi(S)}(d_k))} R_k(i) \setminus \{d_k\}$ denote the union of all remaining cells probed by "reciprocal" queries to queries that probe d_k, excluding d_k itself. To ensure the aforementioned bijection can be encoded "locally" (i.e., using only $\beta \lg \beta'$ bits instead of $\beta \lg n$), we *protect* the cells in $P(d_k)$ from any future deletion. This is crucial, as it guarantees that any query is associated with at most *one* deleted cell.

The number of protected cells is at most $|P(d_k)| \leq t|I_{\Pi(S)}(d_k)| + q|F_S(d_k)| \leq t\beta' + q\beta \leq O(\gamma\, tq/h)$, since $F_S(d_k) \cup I_{\Pi(S)}(d_k)$ contains at most $|F_S(d_k)| \leq \beta$ forward queries in S and $|I_{\Pi(S)}(d_k)| \leq \beta'$ inverse queries in $\Pi(S)$. This implies by direct calculation ([16]) that the number of stages z that can be performed while satisfying $|C_z^S| \geq h/2$, is

$$z = \Theta(h/q\beta) = \Theta(h/t\beta'),$$

since $q\beta = t\beta'$. Note that by (13), sending the bijection array A_{d_k} can be done using (at most) $\beta \lg \beta'$ bits, by storing, for each query in $F'_S(d_k)$, the corresponding index from $I'_{\Pi(S)}(d_k)$ (where the sets are sorted in lexicographical order), since every query is associated with at most one deleted cell. Let $\mathcal{A}$ denote the concatenation of all the arrays $\{A_{d_k}\}_{k\in[z]}$ of deleted cells in the z stages (occupying at most $z\beta \lg \beta'$ bits), $\mathcal{L}$ denote the locations of all deleted cells (which can be sent using at most $\lg \binom{h+r'}{z} \leq z \lg(O(h)/z) = O(z \lg(\beta q))$ bits, assuming $r' = O(h)$), and $\mathcal{R}$ denote the contents of remaining cells, which occupy

$$h + r' - z$$

words. Alice sends Bob $M := (\mathcal{A}, \mathcal{R}, \mathcal{L})$, along with the explicit sets $S, \Pi(S)$. Alice also sends Bob the answers to the forward queries outside S, conditioned on S and the answers Π_S on S, using at most $H_\mu(\Pi_{\bar{S}}|S, \Pi_S) + 1$ bits using standard Huffman coding. Let this message be $\mathcal{P}$.

Assume w.l.o.g that $t \leq q$, hence $\beta \leq \beta'$. Recalling that $\beta' \leq 6\frac{\gamma q}{h}$, the premises of the lemma

$$q \leq \min\left\{2^{w/5}, \frac{1}{32} \cdot \frac{\alpha}{\lg w}\right\}, \text{ and } \quad h \geq \frac{1}{w} H_\mu(\Pi_S|S, \Pi(S)) = \frac{\gamma \cdot \alpha}{w}$$

imply that $\beta' \leq \frac{6\, q\, \gamma\, w}{\gamma\, \alpha} \leq \frac{6\, \alpha\, w}{32\, \alpha\, \lg w} < \frac{w}{5 \lg w}$, hence the total cost of sending $\mathcal{A}$ is at most

$$z\beta \lg \beta' \leq z\beta' \lg \beta' \leq z \frac{w}{5 \lg w} \cdot \lg w = zw/5$$

bits. Since $\lg q \leq w/5$ by assumption, the cost of sending $\mathcal{L}$ is at most $z \lg(\beta q) \leq zw/5 + z \lg w + O(z) = zw/5 + o(zw)$ bits. Sending the sets $S, \Pi(S)$ can be done using at most $2|S| \lg(en/|S|) = 2\gamma \lg(en/\gamma)$ bits. Now, as $tq \leq \frac{\delta \cdot \alpha^2}{w \lg(en/\gamma)}$ for small enough $\delta > 0$, we have

$$zw = \Theta\left(\frac{hw}{q\beta}\right) \geq \Omega\left(\frac{h^2\, w}{\gamma\, t\, q}\right) \geq \Omega\left(\frac{\gamma\, \alpha^2}{w\, tq}\right) \geq 10\gamma \lg(en/\gamma).$$

So, Alice can send $S, \Pi(S)$ using at most $zw/5$ bits. Finally, the message $\mathcal{P}$ requires (up to 1 bit)

$$H_\mu(\Pi_{\bar{S}}|S, \Pi_S) \leq \varepsilon H_\mu(\Pi) \leq 2\varepsilon\alpha\gamma \leq \frac{\delta\, \alpha^2\, \gamma}{w\, tq} \leq \frac{zw}{5}$$

bits in expectation with respect to μ. Thus, together with the cost of $\mathcal{R}$ and $\mathcal{P}$, Alice's total expected message size is at most $(h + r' - z)w + 4zw/5$ bits, or $h + r' - z + 4z/5 = h + r' - z/5$ words (up to $o(z)$ terms). But the minimum expected length of Alice's message must be h words. This implies that the redundancy *in bits* must be at least

$$r = r'w \geq \Omega(zw)$$

in expectation, which, on substituting $z = \Theta(h/q\beta)$, yields the claimed lower bound on r, assuming Bob can recover Π from Alice's message, which we argue below.

To decode $\Pi_S := \{\Pi(i)\}_{i \in S}$ given S and Alice's message M, Bob proceeds as follows: He first fills an empty memory of $h + r'$ words with the contents of $\mathcal{R}$ he receives from Alice, leaving all deleted cells empty. Let A, A' denote the forward (resp. inverse) query algorithms of D_S (note that A, A' are defined for *all* queries in $[n]$). Bob simulates the forward query algorithm $A(i)$ on all forward queries in S and the inverse query algorithm $A'(j)$ on all inverse queries $j \in \Pi(S)$. If a query $i \in S$ *fails* (i.e., probes some deleted cell) but A' on some inverse query $j \in \Pi(S)$ does not fail and returns $A'(j) = i$, he can infer the answer $\Pi(i) = j$. Similarly, he can infer the answer to an inverse query which fails, if its corresponding forward query does not fail. So, we focus on the non-recoverable queries in S and $\Pi(S)$.

If a query $i \in S$ *fails*, he finds the *first* deleted cell d probed by $A(i)$, and lexicographically lists *all* the queries in $F'_S(d)$ for which d is the *first* deleted cell. Similarly, he lexicographically lists the set of all inverse queries in $I'_{\Pi(S)}(d)$ whose first deleted cell is d. He then uses the array A_d, which stores the bijection between the non-recoverable forward queries in $F'_S(d)$ and the non-recoverable inverse queries in $I'_{\Pi(S)}(d)$, to answer these non-recoverable forward queries. In an analogous manner, he answers the non-recoverable inverse queries. Note that we crucially use the fact that each query accesses at most one deleted cell.

Finally, Bob uses $\mathcal{P}$ along with the answers to queries in S and $\Pi(S)$ to answer all forward queries in $\bar{S}$. At this point, since he knows the answers to all forward queries, he can recover Π, and answer all inverse queries in $\bar{\Pi}(S)$ as well. □

References

[1] Donald Adjeroh, Timothy Bell, and Amar Mukherjee. *The Burrows-Wheeler Transform: Data Compression, Suffix Arrays, and Pattern Matching.* Springer Publishing Company, Incorporated, 1 edition, 2008. 1

[2] Rudolf Ahlswede. An elementary proof of the strong converse theorem for the multiple-access channel. *J. Combinatorics, Information and System Sciences*, 1982. 3.2

[3] Jon Louis Bentley, Daniel D. Sleator, Robert E. Tarjan, and Victor K. Wei. A locally adaptive data compression scheme. *Commun. ACM*, 29(4):320–330, April 1986. 1, 2.2.1

[4] Philip Bille, Patrick Hagge Cording, Inge Li GNørtz, Benjamin Sach, Hjalte Wedel VildhNøj, and SNøren Vind. Fingerprints in compressed strings. 1.1

[5] Andrej Brodnik and J. Ian Munro. Membership in constant time and almost-minimum space. *SIAM J. Comput.*, 28(5):1627–1640, May 1999. 1.1

[6] M. Burrows and D. J. Wheeler. A block-sorting lossless data compression algorithm. Technical Report 124, Digital Equipment Corporation, 1994. 1, 2.2.2

[7] Yevgeniy Dodis, Mihai Patrascu, and Mikkel Thorup. Changing base without losing space. In *Proceedings of the Forty-second ACM Symposium on Theory of Computing*, STOC '10, pages 593–602, New York, NY, USA, 2010. ACM. 1.1

[8] Akashnil Dutta, Reut Levi, Dana Ron, and Ronitt Rubinfeld. A simple online competitive adaptation of lempel-ziv compression with efficient random access support. In *Proceedings of the 2013 Data Compression Conference*, DCC '13, pages 113–122, Washington, DC, USA, 2013. IEEE Computer Society. 1, 1.1

[9] Michelle Effros, Karthik Visweswariah, Sanjeev R. Kulkarni, and Sergio Verdu. Universal lossless source coding with the burrows wheeler transform. *IEEE TRANSACTIONS ON INFORMATION THEORY*, 48(5):1061–1081, 2002. 1, B

[10] Martin Farach and Mikkel Thorup. String matching in lempel-ziv compressed strings. In *Proceedings of the Twenty-seventh Annual ACM Symposium on Theory of Computing*, STOC '95, pages 703–712, New York, NY, USA, 1995. ACM. 1, 1.1, B

[11] Paolo Ferragina, Raffaele Giancarlo, and Giovanni Manzini. The myriad virtues of wavelet trees. *Inf. Comput.*, 207(8):849–866, August 2009. B

[12] Paolo Ferragina, Raffaele Giancarlo, Giovanni Manzini, and Marinella Sciortino. Boosting textual compression in optimal linear time. *J. ACM*, 52(4):688–713, July 2005. B

[13] Paolo Ferragina and Giovanni Manzini. Indexing compressed text. *J. ACM*, 52(4):552–581, July 2005. 1, 4, 1, 1.1, 2.2.2, 3, 6, 6, 7, B

[14] Travis Gagie and Giovanni Manzini. Move-to-front, distance coding, and inversion frequencies revisited. *Theoretical Computer Science*, 411(31):2925 – 2944, 2010. B

[15] Anna Gál and Peter Bro Miltersen. The cell probe complexity of succinct data structures. In *In Automata, Languages and Programming, 30th International Colloquium (ICALP 2003*, pages 332–344. Springer-Verlag, 2003. 1.1

[16] Alexander Golynski. Cell probe lower bounds for succinct data structures. In *Proceedings of the Twentieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2009, New York, NY, USA, January 4-6, 2009*, pages 625–634, 2009. 1, 1.1, 3.2, 3.2, 7, 7, 7, 7.2, 7.2

[17] Alexander Golynski, Roberto Grossi, Ankur Gupta, Rajeev Raman, and S. Srinivasa Rao. On the size of succinct indices. In *Algorithms - ESA 2007, 15th Annual European Symposium, Eilat, Israel, October 8-10, 2007, Proceedings*, pages 371–382, 2007. 1, 1.1

[18] Alexander Golynski, Rajeev Raman, and S. Srinivasa Rao. On the redundancy of succinct data structures. In *Proceedings of the 11th Scandinavian Workshop on Algorithm Theory*, SWAT '08, pages 148–159, Berlin, Heidelberg, 2008. Springer-Verlag. 1.1

[19] Haim Kaplan, Shir Landau, and Elad Verbin. A simpler analysis of burrows–wheeler-based compression. *Theor. Comput. Sci.*, 387(3):220–235, November 2007. 1, B

[20] Haim Kaplan and Elad Verbin. Most burrows-wheeler based compressors are not optimal. In *Proceedings of the 18th Annual Conference on Combinatorial Pattern Matching*, CPM'07, pages 107–118, Berlin, Heidelberg, 2007. Springer-Verlag. 1, B

[21] S. Rao Kosaraju and Giovanni Manzini. Compression of low entropy strings with lempel–ziv algorithms. *SIAM J. Comput.*, 29(3):893–911, December 1999. B

[22] Udi Manber and Gene Myers. Suffix arrays: A new method for on-line string searches. In *Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '90, pages 319–327, Philadelphia, PA, USA, 1990. Society for Industrial and Applied Mathematics. 1.1

[23] Giovanni Manzini. An analysis of the burrows-wheeler transform. *J. ACM*, 48(3):407–430, May 2001. 1, 2, 2.2.2, B

[24] Giovanni Manzini and Paolo Ferragina. Engineering a lightweight suffix array construction algorithm. *Algorithmica*, 40(1):33–50, June 2004. 1

[25] Edward M. McCreight. A space-economical suffix tree construction algorithm. *J. ACM*, 23(2):262–272, April 1976. 1.1

[26] Peter Bro Miltersen. Lower bounds on the size of selection and rank indexes. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '05, pages 11–12, Philadelphia, PA, USA, 2005. Society for Industrial and Applied Mathematics. 1.1

[27] Alistair Moffat. Implementing the ppm data compression scheme, 1990. B

[28] J. Ian Munro, Rajeev Raman, Venkatesh Raman, and Srinivasa Rao S. Succinct representations of permutations and functions. *Theor. Comput. Sci.*, 438:74–88, June 2012. 1.1, 7

[29] Gonzalo Navarro and Veli Mäkinen. Compressed full-text indexes. *ACM Comput. Surv.*, 39(1), April 2007. 1, 1.1

[30] Rasmus Pagh. Low redundancy in static dictionaries with constant query time. *SIAM J. Comput.*, 31(2):353–363, February 2002. 1, 1.1

[31] Mihai Pătraşcu. Succincter. In *Proceedings of the 2008 49th Annual IEEE Symposium on Foundations of Computer Science*, FOCS '08, pages 305–313, Washington, DC, USA, 2008. IEEE Computer Society. 1, 1, 1, 1.1, 2.3, 4, 2.3, 3.1, 4.4, 6

[32] Mihai Pătraşcu and Mikkel Thorup. Time-space trade-offs for predecessor search. In *Proceedings of the Thirty-eighth Annual ACM Symposium on Theory of Computing*, STOC '06, pages 232–240, New York, NY, USA, 2006. ACM. 3.1, 5.1

[33] Mihai Pătraşcu and Emanuele Viola. Cell-probe lower bounds for succinct partial sums. In *Proceedings of the Twenty-first Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '10, pages 117–122, Philadelphia, PA, USA, 2010. Society for Industrial and Applied Mathematics. 1.1

[34] Kunihiko Sadakane and Roberto Grossi. Squeezing succinct data structures into entropy bounds. In *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2006, Miami, Florida, USA, January 22-26, 2006*, pages 1230–1239, 2006. 1.1

[35] Julian Seward. Bzip2. `www.bzip.org`. 1, B

[36] Jared T. Simpson and Richard Durbin. Efficient construction of an assembly string graph using the fm-index. *Bioinformatics [ISMB]*, 26(12):367–373, 2010. 1.1

[37] Emanuele Viola. Cell-probe lower bounds for prefix sums. *Electronic Colloquium on Computational Complexity (ECCC)*, 16:54, 2009. 1.1

[38] Emanuele Viola. Bit-probe lower bounds for succinct data structures. *SIAM Journal on Computing*, 41(6):1593–1604, 2012. 1.1

[39] Emanuele Viola. A sampling lower bound for permutations. *Electronic Colloquium on Computational Complexity (ECCC)*, 24:166, 2017. 1.1

[40] Emanuele Viola. Sampling lower bounds: boolean average-case and permutations. *Electronic Colloquium on Computational Complexity (ECCC)*, 25:60, 2018. 1.1

[41] J. Ziv and A. Lempel. Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory*, 24(5):530–536, September 1978. 6, B

A Proof of Fact 1 (LF Mapping Property)

We prove the first equality in Fact 1. The intuition is that the order among distinct occurrences of a character c in both F and L are decided by the character's *right-context* in x, and so they must be the same.

Proof. Fix $c \in \Sigma$. Consider any two distinct occurrences of c in x, at positions $k_1, k_2 \in [n]$ respectively. For $\alpha \in \{1,2\}$, let $i_\alpha, j_\alpha \in [n]$ be indices such that the BWT maps $x[k_\alpha]$ to position i_α in L and position j_α in F, i.e., $x[k_\alpha] = L[i_\alpha] = F[j_\alpha] = c$. Then it suffices to prove that $i_1 < i_2$ if and only if $j_1 < j_2$, because the ordering among all occurrences of c is determined by the relative ordering among all pairs of occurrences of c.

For $\alpha \in \{1,2\}$, let $y_\alpha = x[k_\alpha + 1, k_\alpha + 2, \cdots, n, 1, \cdots, k_\alpha - 1]$ denote the *right-context* of $x[k_\alpha]$ in x, which corresponds to the cyclic shift of x by α positions (and then excluding $x[k_\alpha]$). For $\beta \in [n]$, let $\mathcal{M}[\beta]$ denote row β of the BWT matrix $\mathcal{M}$. Then it is easy to see that $\mathcal{M}[i_\alpha] = (y_\alpha, c)$ and $\mathcal{M}[j_\alpha] = (c, y_\alpha)$, for $\alpha \in \{1,2\}$. We write $z_1 \prec z_2$ below to mean that z_1 is smaller than z_2 according to the lexicographical order on Σ.

Assume $i_1 < i_2$. From this assumption and the fact that the rows of $\mathcal{M}$ are sorted lexicographically, we have $(y_1, c) \prec (y_2, c)$. But this implies that $y_1 \prec y_2$ lexicographically, as both strings are of equal length and end with c. So, we have

$$\mathcal{M}[j_1] = (c, y_1) \prec (c, y_2) = \mathcal{M}[j_2].$$

We conclude that $j_1 < j_2$ if $i_1 < i_2$. Clearly, the converse also holds. Thus, the relative ordering between any two occurrences of c is the same in F and L. This concludes the proof of Fact 1. □

B The RLX Benchmark and Comparison to Other Compressors

A theoretical justification of the RLX space benchmark was first given by [13, 23], where it was proved that $\texttt{RLX}(\mathsf{BWT}(x))$ approaches the infinite-order empirical entropy of x (even under the weaker version of [13] where the final arithmetic coding stage (3) is excluded), namely, that for *any* constant $k \geq 0$,

$$|\texttt{RLX}(\mathsf{BWT}(x))| \leq 5 \cdot H_k(x) + O(\lg n).$$

Several other compression methods were subsequently proposed for compressing $\mathsf{BWT}(x)$ (e.g., [11, 12, 14]), some of which achieving better (essentially optimal) worst-case theoretical guarantees with respect to $H_k(x)$, albeit at the price of an $\Omega(n/\lg n)$ or even $\Omega(n)$ additive factor, which becomes the dominant term in the interesting regime of compressed text indexing [10, 21]. The same caveat holds for other entropy-coding methods such as LZ77, LZ78 and PPMC [27, 41, 41], confirming experimental results which demonstrated the superiority of BWT-based text compression [9, 19, 23]. Indeed, Kaplan et. al [19, 20] observed that, despite failing to converge *exactly* to H_k, distance-based and MTF compression of BWT such as RLX tend to outperform other entropy coding methods (especially in the presence of long-term correlations as in English text). RLX is the basis of the `bzip2` program [35]. For further elaboration we refer the reader to [12, 23].

ECCC ISSN 1433-8092
https://eccc.weizmann.ac.il