An Algorithmic Framework for Compression and Text Indexing *

Roberto Grossi† Ankur Gupta‡ Jeffrey Scott Vitter§

Abstract

We present a unified algorithmic framework to obtain nearly optimal space bounds for text compression and compressed text indexing, apart from lower-order terms. For a text T of n symbols drawn from an alphabet Σ, our bounds are stated in terms of the hth-order empirical entropy of the text, H_h. In particular, we provide a tight analysis of the Burrows-Wheeler transform (BWT) establishing a bound of $nH_h + M(T, \Sigma, h)$ bits, where $M(T, \Sigma, h)$ denotes the asymptotical number of bits required to store the empirical statistical model for contexts of order h appearing in T. Using the same framework, we also obtain an implementation of the compressed suffix array (CSA) which achieves $nH_h + M(T, \Sigma, h) + O(n \lg \lg n / \lg_{|\Sigma|} n)$ bits of space while still retaining competitive full-text indexing functionality.

The novelty of the proposed framework lies in its use of the finite set model instead of the empirical probability model (as in previous work), giving us new insight into the design and analysis of our algorithms. For example, we show that our analysis gives improved bounds since $M(T, \Sigma, h) \leq \min\{g'_h \lg(n/g'_h + 1), H^*_h n + \lg n + g''_h\}$, where $g'_h = O(|\Sigma|^{h+1})$ and $g''_h = O(|\Sigma|^{h+1} \lg |\Sigma|^{h+1})$ do not depend on the text length n, while $H^*_h \geq H_h$ is the modified hth-order empirical entropy of T. Moreover, we show a strong relationship between a compressed full-text index and the succinct dictionary problem. We also examine the importance of lower-order terms, as these can dwarf any savings achieved by high-order entropy. We report further results and tradeoffs on high-order entropy-compressed text indexes in the paper.

1 Introduction

The world is drowning in data. Classic algorithms are greedy in terms of their space usage and often cannot perform computations on all of the data. This trend has not gone unnoticed by researchers, as evidenced by the recent issues in data streaming [Mut03] and sublinear algorithms [Cha04]. Unlike these cases, many problems require the entire dataset to be stored in compressed format but still need it to be queried quickly. In fact, compression may have a more far-reaching impact than simply storing data succinctly: "That which we can compress we can understand, and that which we can understand we can predict," as observed in [Aar05]. Much of what we call "insight" or "intelligence" can be thought of as simply finding succinct representations of sensory data [Bau04]. For instance, we are far from fully understanding the intrinsic structure of biological sequences, and in fact, we cannot compress them well either.

*The results on text indexing were presented in preliminary form at the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms [GGV03].

†Dipartimento di Informatica, Università di Pisa, Largo Bruno Pontecorvo 3, 56127 Pisa (grossi@di.unipi.it). Support was provided in part by Italian Ministry of Education, University and Research (MIUR).

‡Center for Geometric and Biological Computing, Department of Computer Science, Duke University, Durham, NC 27708–0129 (agupta@cs.duke.edu). Support was provided in part by the Army Research Office (ARO) through grant DAAD20–03–1–0321 and by the National Science Foundation through research grant ISS–0415097.

§Department of Computer Science, Purdue University, West Lafayette, IN 47097–2066 (jsv@purdue.edu). Support was provided in part by Army Research Office MURI grants DAAH04–96–1–0013 and DAAD19–01–1–0725 and by National Science Foundation research grant CCR–9877133.

Researchers have considered these issues in several algorithmic contexts, such as the design of efficient algorithms for managing highly-compressible data structures. They have carefully studied the exact resources needed to represent trees [BDMR99, GRR04, MRS01a, MRS01b, MR02], graphs [Jac89a, BBK03], sets and dictionaries [BB04, BM99, Pag01, RR03, RRR02], permutations and functions [MRRR03, MR04], and text indexing structures [FM05, GV05, GGV04, FGGV04, Sad02, Sad03]. The goal is to design algorithms with tight space complexity $s(n)$. The Kolmogorov complexity for representing this data provides a lower bound on the value of $s(n)$ for each representation studied. (Kolmogorov complexity essentially defines compression in terms of the size of the smallest program that can generate the input provided [LV97].) However, Kolmogorov complexity is undecidable for arbitrary data, so any compression method is known to be sub-optimal in this sense.[1] The hope is to achieve $s(n) + o(s(n))$ bits, with nearly-optimal asymptotic time bounds, i.e. $O(t(n))$ time, while remaining competitive with state-of-the-art (uncompressed) data structures [Jac89a].

Providing the exact analysis of space occupancy (up to lower-order terms) is motivated by the above theoretical issues as well as the following technological issues. Space savings can translate into faster processing (since packing data can reduce disk accesses), which results in shorter seek times or allows data storage on faster levels of the cache. A recent line of research uses the I/O computation model to take into account some of these issues, such as cache-oblivious algorithms and data structures [AV88, BDFC05]. Some algorithms exploit data compression to achieve provably better time bounds [RC93, KV98, VK96]. From an economical standpoint, compressed data would require less media to store (such as RAM chips in search engines or portable computing devices) or less time to transmit over regulated bandwidth models (such as transmissions by cell phones).

Similar goals are difficult to achieve when analyzing time bounds due to the complexity of modern machines, unless some simple computation model (such as one reminiscent of the comparison model) is used. Some sources of imprecision include cache hits/misses, dynamic re-ordering of instructions to maximize instruction parallelism, disk scheduling issues, and latency of disk head movements. Space bounds, on the other hand, are more easily predicted and can often be validated experimentally. This concrete verification is an important component of research due to technological advances which may affect an otherwise good bound: 64-bit CPUs are on the market (increasing the pointer size), Unicode text is becoming more commonplace (requiring more than 8 bits per symbol as in ASCII text), and XML databases are encoding more data as well (adding a non-trivial amount of formatting data to the "real" information). We need to squeeze all this data and provide fast access to its compressed format. For a variety of data structures, therefore, the question remains: Can we achieve a near-optimum compression and support asymptotically fast queries?

1.1 Entropy, Text Compression, and Compressed Text Indexing

In this paper, we focus on text data, which are used to encode a wide variety of raw text, literary works, digital databases, product catalogues, genomic databases, etc. In this context, the tight space complexity $s(n)$ is better expressed in terms of the *entropy* of the particular text at hand. (See [Sha48] for the definition of entropy and [CT91] for the relation between entropy and Kolmogorov complexity.) We want to develop tight space bounds for *text compression*, i.e. storing a text in a compressed binary format. We additionally want to design *compressed text indexes* to decode any small portion of the text or search for any pattern as a substring of the text, without decompressing the binary format entirely. In particular, we study how to obtain a compressed representation of the text that is a *self-index*, namely, we obtain a compressed binary format that is also an index for the text itself.

[1] Extrapolating from [Aar05, Bau04], the undecidability of Kolmogorov complexity implies that there is a computational limit on finding succinct representations for our sensory data.

We consider the text T as a sequence of n symbols, where each symbol is drawn from the alphabet Σ. For ease of exposition, we "number" the symbols in alphabet Σ from 1 to $\sigma = |\Sigma|$, such that the renumbered symbol y is also the yth lexicographically ordered symbol in $\Sigma = \{1, 2, \ldots, \sigma\}$. Without loss of generality, we can assume that $\sigma \leq n$, since we only need to consider those symbols that actually occur in T. We also denote a substring $T[i]T[i+1]\cdots T[j]$ of contiguous text symbols by $T[i, j]$. Since the raw text T occupies $n \lg \sigma$ bits of storage, T is compressible if it can be represented in fewer than $n \lg \sigma$ bits.[2] We trivially know that no encoding of T can take fewer bits than the entropy of T, which measures how much randomness is in T. (Here, entropy is related to the size of the smallest program which generates T, according to the Kolmogorov complexity.) So, we expect that the entropy of T is a lower bound to the space complexity $s(n)$ for compressed data structures that store T.

Ideally, we would like to achieve the entropy bound, but all we can quantitatively analyze is an approximation of it, namely,

$$nH_h + M(T, \Sigma, h) \tag{1}$$

bits of space. In formula (1), $H_h \leq \lg \sigma$ is the *hth-order empirical entropy* of T, which captures the dependence of symbols on their *context*, made up of the h adjacent symbols in the text T. As n increases, $M(T, \Sigma, h)$ denotes the asymptotical number of bits used to store the empirical probabilities for the corresponding statistical model in T: informally, $M(T, \Sigma, h)$ represents the number of bits required to store the number of occurrences of yx as a substring of the text T, for each context x of length h and each symbol $y \in \Sigma$. (These quantities are discussed formally in Sections 2 and 3.) As h increases, nH_h is non-increasing and $M(T, \Sigma, h)$ is non-decreasing. Thus, carefully tuning the context length h gives the best choice for minimizing space. An interesting problem is how to obtain nearly optimal space bounds where $s(n)$ is approximated by formula (1) for the best choice of h. In practice, English text is often compressible by a factor of 3 or 4, and the best choice for h is usually about 4 or 5. Lempel and Ziv have provided an encoding such that $h \leq \alpha \lg n + O(1)$ (where $0 < \alpha < 1$) is sufficiently good for approximating the entropy; Luczak and Szpankowski prove a sufficient approximation for ergodic sources when $h = O(\lg n)$ in [LS97].

1.2 Our Results

In this paper, we introduce a *unified algorithmic framework* for achieving the first *nearly optimal* space bounds for *both* text compression and compressed text indexing. We provide a new tight analysis of text compression [BW94] based on the *Burrows-Wheeler transform* (BWT). We also provide a new implementation of compressed text indexing [FM05, GV05, Sad03] based on the *compressed suffix array* (CSA). A key point of our unified approach is the use of the *finite set model* instead of the empirical probability model adopted in previous work, giving us new insight into the analysis. As we show in the paper, we capture the empirical probabilities encoded in $M(T, \Sigma, h)$ bits (see formula (1)) by employing a two-dimensional conceptual organization which groups contexts x from the text by their predicted symbols y. This scheme can be seen as an alternative way to model an arbitrary partition of the BWT. We then restructure each context accordingly, encoding each group with an algorithm that stores t items out of a universe of size n in the information theoretic minimum space $\lceil \lg \binom{n}{t} \rceil$ bits (since there are $\binom{n}{t}$ subsets of t items out of n). In Sections 1.3–1.4, we detail our results for text compression and text indexing, which reach nearly optimal space bounds for both areas.

[2] In this paper, we use the notation $\lg_b^c a = (\lg_b a)^c = (\lg a / \lg b)^c$ to denote the cth power of the base-b logarithm of a. If no base is specified, the implied base is 2.

1.3 Text Compression

In this section, we discuss our results for text compression, which are based on the Burrows-Wheeler transform (BWT). Simply put, the BWT rearranges the text T so that it is easily compressed by other methods. In practice, the compressed version of this transformed text is quite competitive with other methods [Fen96, Fen02, FTL03]. The BWT is at the heart of compressors based on block-sorting (such as `bzip2`) that outperform Lempel-Ziv-based compressors (such as `gzip`). We provide a method for representing the BWT in compressed format using well-known results from combinatorial enumeration [Knu05, Rus05] in an unusual way, exploiting the functionality of ranking and unranking t-subsets for compressing and decompressing the t items thus stored.[3] We collect and store this information in our new *wavelet tree*, a novel data structure that we use to represent the LF mapping (from the BWT and used in the FM-index [FM05]) and the neighbor function Φ (at the heart of the CSA [GV05]). Our framework-based analysis gives a bound of $nH_h + M(T, \Sigma, h)$ bits for any given h as input, thus matching formula (1). The best value of h can be found using the optimal partitioning of the BWT as given in [FGMS05], so that our bound holds for any h (simply because formula (1) cannot be smaller for the other values of h). For comparison purposes, we give an upper bound on the number of bits needed to encode the statistical model,

$$M(T, \Sigma, h) \leq \min\left\{g'_h \lg(n/g'_h + 1),\, H^*_h n + \lg n + g''_h\right\}, \tag{2}$$

where $g'_h = O(\sigma^{h+1})$ and $g''_h = O(\sigma^{h+1} \lg \sigma^{h+1})$ do not depend on the text length n.

In formula (2), $H^*_h \geq H_h$ is the *modified* hth-order empirical entropy (see Section 2) introduced in [Man01] to show that the BWT can be represented in at most $(5+\epsilon)nH^*_h + \lg n + g_h$ bits, where $\epsilon \approx 10^{-2}$ and $g_h = O(\sigma^{h+1} \lg \sigma)$. The latter bound is important for low-entropy texts, since the compression ratio scales with high-order entropy; this bound cannot be attained when replacing H^*_h by H_h. We refer the reader to [Man01] for previous literature on the subject. In contrast, the compression ratio of Lempel-Ziv algorithm [ZL77] does not scale for low-entropy texts: although its output is bounded by $nH_h + O(n \lg\lg n / \lg n)$ bits, it cannot be smaller than $2.5nH_0$ bits for some strings [KM99]. Note that the BWT is a booster for 0th-order compressors as shown in [FGMS05], where a close connection of the optimal partition of the BWT with the suffix tree [McC76] attains the best known space bounds for the analysis of the BWT, namely, $2.5nH^*_h + \lg n + g_h$ bits and $nH_h + n + \lg n + g_h$ bits. The related compression methods do not require the knowledge of the order h and take $O(n \lg \sigma)$ time for general alphabets.

Using (2), we can compare our analysis with the best bounds from previous work. When compared to the additive term of $O(n \lg\lg n / \lg n)$ in the analysis of the Lempel-Ziv method in [KM99], we obtain an $O(\log n)$ additive term for $\sigma = O(1)$ and $h = O(1)$, giving strong evidence why the BWT is better than the Lempel-Ziv method. Indeed, since $M(T, \Sigma, h) \leq g'_h \lg(n/g'_h + 1)$, our bound in (1) becomes $nH_h + O(\lg n)$ when $h = O(1)$ and $\sigma = O(1)$, thus exponentially reducing the additive term of n of the H_h-based analysis in [FGMS05]. In this case, our bound closes the gap in the analysis of BWT, since it matches the lower bound of $nH_h + \Omega(\lg\lg n)$, up to lower-order terms. The latter comes from the lower bound of $nH^*_0 + \Omega(\lg\lg n)$ bits, holding for a large family of compressors (not necessarily related to BWT), as shown in [FGMS05]; the only (reasonable) requirement is that any such compressor must produce a codeword for the text length n when it is fed with an input text consisting of the same symbol repeated n times. Since $H_h \leq H^*_0$, we easily derive the lower bound of $nH_h + \Omega(\lg\lg n)$ bits, but a lower bound of $nH_h + \Omega(\lg n)$ probably exists since $nH^*_0 \geq \lg n$ while nH_h can be zero.

As for the modified hth-order empirical entropy, we show that our analysis in (1) can be upper bounded by $n(H_h + H^*_h) + \lg n + g''_h$ bits using (2). Since $H_h \leq H^*_h$, our bound in (1) is strictly

[3] We use the term t-subset instead of the more usual k-subset terminology, because we use k to denote the levels of our compressed suffix array (described later). A similar observation holds for entropy H_h, which is often referred to as H_k in the literature.

smaller than $2.5nH_h^* + \lg n + g_h$ bits in [FGMS05], apart from the lower-order terms. Actually, our bound is definitively smaller in some cases. For example, while a bound of the form $nH_h^* + \lg n + g_h$ bits is not always possible [Man01], there are an infinite number of texts for which $nH_h = 0$ while $nH_h^* \neq 0$. In these cases, our bound from (1) is $nH_h^* + \lg n + g_h''$ bits.

1.4 Compressed Text Indexing

In this section, we discuss our analysis with respect to text indexing based on the compressed suffix array (CSA). Text indexing data structures preprocess a text T of n symbols drawn from an alphabet Σ such that any query pattern P of m symbols can be answered quickly without requiring an entire scan of the text itself. Depending on the type of query, we may want to know if P occurs in T (occurrence or search query), how many times P occurs in T (counting query), or the locations where P occurs in T (enumerative query). An occurrence of pattern P at position i identifies a substring $T[i, i+m-1]$ equal to P. Because a text index is a preprocessed structure, a reasonable query time should have no more than a polylg(n) cost plus an output sensitive cost $O(occ)$, where occ is the number of occurrences retrieved (which is crucial for large-scale processing).

Until recently, these data structures were greedy of space and also required a separate (original) copy of the text to be stored. Suffix trees [McC76, Ukk95, Wei73] and suffix arrays [GBS92, MM93] are prominent examples. The suffix tree is a compact trie whose leaves store each of the n suffixes contained in the text T, namely, $T[1, n], T[2, n], \ldots, T[n, n]$, where suffix $T[i, n]$ is uniquely identified by its starting position i. Suffix trees [McC76, MM93] allow fast queries of substrings (or patterns) in T in $O(m \lg \sigma + occ)$ time, but require at least $4n \lg n$ bits of space, in addition to keeping the text. The suffix array SA is another popular index structure. It maintains the permuted order of $1, 2, \ldots, n$ that corresponds to the locations of the suffixes of the text in lexicographically sorted order, $T\big[SA[1], n\big]$, $T\big[SA[2], n\big]$, $\ldots$, $T\big[SA[n], n\big]$. Suffix arrays [GBS92, MM93] (that store the length of the longest common prefix) are nearly as good at searching as are suffix trees. Their time for finding occurrences is $O(m + \lg n + occ)$ time, but the space cost is at least $n \lg n$ bits, plus the cost of keeping the text.

A new trend in the design of modern indexes for full-text searching is addressed by the CSA [GV05, Rao02, Sad03, Sad02] and the opportunistic FM-index [FM05], the latter making the very strong intuitive connection between the power of the BWT and suffix arrays. They support the functionalities of suffix arrays and overcome the aforementioned space limitations. In our framework, we implement the CSA by replacing the basic t-subset encoding with succinct dictionaries supporting constant-time *rank* and *select* queries. The *rank* query returns the number of entries in the dictionary that are less than or equal to the input entry; the *select* query returns the ith entry in the dictionary for the input i. Succinct dictionaries store t keys over a bounded universe n in the information theoretically minimum space $\lceil \lg \binom{n}{t} \rceil$ bits, plus lower-order terms $O(n \lg \lg n / \lg n) = o(n)$ [RRR02]. We show a close relationship between compressing a full-text index with high-order entropy to the succinct dictionary problem. Prior to this paper, the best space bound was $5nH_h + O\big(n\frac{\sigma + \lg \lg n}{\lg n} + n^\epsilon \sigma 2^{\sigma \lg \sigma}\big)$ bits for the FM-index, supporting a new backward search algorithm in $O(m + occ \times \lg^{1+\epsilon} n)$ time for any $\epsilon > 0$ [FM05]. We refer the reader to the survey in [NM05] for a discussion of more recent work in this area.

We obtain several tradeoffs between time and space as shown in Tables 1 and 2. For example, Theorem 10 gives a self-index requiring $nH_h + O(n \lg \lg n / \lg_\sigma n)$ bits of space (where $h+1 \leq \alpha \lg_\sigma n$ for an arbitrary positive constant $\alpha < 1$) that allows searching for patterns of length m in $O(m \lg \sigma + occ \times \text{polylg}(n))$ time. Thus, using our new analysis of the BWT, our implementation provides the first self-index reaching the high-order empirical entropy nH_h of the text with a multiplicative constant of 1; moreover, we conjecture that $g_h' \lg(n/g_h' + 1)$ additional bits are not achievable for text indexing. If true, this claim would imply that adding self-indexing capabilities to a compressed text requires more space than $M(T, \Sigma, h)$, the number of bits encoding the empirical statistical

	bits of space	*lookup* & *lookup*$^{-1}$	*substring*	notes
Thm.6	$nH_h \lg \lg_\sigma n + o(n \lg \sigma) + O\big(\sigma^h(n^\beta + \sigma)\big)$	$O(\lg \lg_\sigma n)$	$O(c/\lg_\sigma n + \lg \lg_\sigma n)$	any $0 < \beta < 1$
Thm.7	$\epsilon^{-1} nH_h + O\big(n \lg \lg n/\lg^\epsilon_\sigma n + \sigma^h(n^\beta + \sigma)\big)$	$O\big((\lg_\sigma n)^{\epsilon/1-\epsilon} \lg \sigma\big)$	$O\big(c/\lg_\sigma n + (\lg_\sigma n)^{\epsilon/1-\epsilon} \lg \sigma\big)$	any $0 < \beta < 1$, $0 < \epsilon \leq 1/2$
Cor.3	$\epsilon^{-1} nH_h + O(n) + O\big(\sigma^h(n^\beta + \sigma)\big)$	$O\big((\lg_\sigma n)^{\epsilon/1-\epsilon}\big)$	$O\big(c/\lg_\sigma n + (\lg_\sigma n)^{\epsilon/1-\epsilon}\big)$	$n = o(n \lg \sigma)$ for $\sigma = \omega(1)$
Thm.8	$nH_h + O\big(n \lg \lg n/\lg_\sigma n + \sigma^{h+1} \lg(1 + n/\sigma^{h+1})\big)$	$O(\lg^2 n/\lg \lg n)$	$O(c \lg \sigma + \lg^2 n/\lg \lg n)$	any $0 < \beta < 1$

Table 1: Trade-offs between time and space for the implementation of CSA and its supported operations. (See Definition 2.) The lower-order terms in the space complexity are all $o(n \lg \sigma)$ bits except $\sigma^h(n^\beta + \sigma)$ (because of $M(T, \Sigma, h)$), which is $o(n \lg \sigma)$ when $h + 1 \leq \alpha \lg_\sigma n$ for any arbitrary positive constant $\alpha < 1$ (we fix β such that $\alpha + \beta < 1$). In all cases, *compress* requires $O(n \lg \sigma + \sigma^h(n^\beta + \sigma))$ time.

	bits of space	search/count time	enumerative time (per item)	notes
Thm.9	$\epsilon^{-1} nH_h + O(n \lg \lg n/\lg^\epsilon_\sigma n)$	$O\big(m/\lg_\sigma n + (\lg n)^{(1+\epsilon)/(1-\epsilon)} (\lg \sigma)^{(1-3\epsilon)/(1-\epsilon)}\big)$	$O\big((\lg n)^{(1+\epsilon)/(1-\epsilon)} (\lg \sigma)^{(1-3\epsilon)/(1-\epsilon)}\big)$	any $0 < \epsilon \leq 1/2$
Thm.10	$nH_h + O(n \lg \lg n/\lg_\sigma n)$	$O(m \lg \sigma + \lg^4 n/(\lg^2 \lg n \lg \sigma))$	$O(\lg^4 n/(\lg^2 \lg n \lg \sigma))$	$1 > \omega \geq 2\epsilon/(1 - \epsilon)$
Thm.11	$\epsilon^{-1} nH_h + O(n \lg \lg n/\lg^\epsilon_\sigma n)$	$O(m/\lg_\sigma n + \lg^\omega n \lg^{1-\epsilon} \sigma)$	$O(\lg^\omega n \lg^{1-\epsilon} \sigma)$	$0 < \epsilon \leq 1/3$

Table 2: Trade-offs between time and space for the compressed text indexing based on the CSA, under the assumption that $h + 1 \leq \alpha \lg_\sigma n$ for any arbitrary positive constant $\alpha < 1$. The lower-order terms in the space complexity are all $o(n \lg \sigma)$ bits. In all cases, the construction takes $O(n \lg \sigma)$ time and uses a temporary area of $O(n \lg n)$ bits of space.

model for the BWT. Actually, we also conjecture that the $O(n \lg\lg n / \lg_\sigma n)$ term is the minimum additional cost for obtaining the $O(m \lg \sigma)$-time search bound. Bro Miltersen [Mil05] proved a lower bound of $\Omega(n \lg\lg n / \lg n)$ bits for constant-time *rank* and *select* queries on an explicit bitvector (i.e. $\sigma = 2$). (Other tradeoffs for the lower bounds on size are reported in [Mil05, DLO03, GM03].) While this result does not directly imply a lower bound for text indexing, it remains as strong evidence of the difficulty of improving the lower-order terms in our framework since it is heavily based on *rank* and *select* queries.

As another example, let us consider Theorem 11, where we develop an hybrid implementation of the CSA, occupying $\epsilon^{-1} n H_h + O(n \lg\lg n / \lg^\epsilon_\sigma n)$ bits ($0 < \epsilon \leq 1/3$), so that searching is very fast and takes $O(m / \lg_\sigma n + occ \times \lg^\omega n \lg^{1-\epsilon} \sigma)$ time ($1 > \omega > 2\epsilon/(1-\epsilon) > 0$). For low-entropy text over an alphabet of size $\sigma = O(1)$, we obtain the first self-index that *simultaneously* exhibits sublinear size $o(n)$ in bits and sublinear search and counting query time $o(m)$; reporting the occurrences takes $o(\lg n)$ time per occurrence.

Also, due to the ambivalent nature of our wavelet tree, we can obtain an implementation of the LF mapping for the FM-index as a byproduct of our method. (See Section 4.3 for more details.) We obtain an $O(m \lg \sigma)$ search/count time by using the backward search algorithm in [FM05] in $nH_h + O(n \lg\lg n / \lg_\sigma n)$ bits. We also get $O(m)$ time in $nH_h + O(n) = nH_h + o(n \lg \sigma)$ bits when σ is not a constant. This avenue has been explored in [FMMN04], showing how to get $O(m)$ time in $nH_h + O(n \lg\lg n / \lg_\sigma n)$ bits when $\sigma = O(\text{polylg}(n))$, using a wavelet tree with a fanout of $O(\lg^\eta n)$ for some constant $0 < \eta < 1$. All these results together imply that the FM-index can be implemented with $O(m)$ search time using nearly optimal space, $nH_h + O(n \lg\lg n / \lg_\sigma n)$ bits, when either $\sigma = O(\text{polylg}(n))$ or $\sigma = \Omega(2^{O(\lg n / \lg\lg n)})$. The space is still $nH_h + O(n) = nH_h + o(n \lg \sigma)$ for the other values of σ, but we do not know if the lower-order term $O(n)$ can be reduced.

1.5 Outline of Paper

The rest of the paper is organized as follows. In Section 2, we describe the differences between various notions of empirical entropy and propose a new definition based on the finite set model. In Sections 3–4, we describe our algorithmic framework, showing a tighter analysis of the BWT and detailing our new wavelet tree. In Section 5, we use this framework to achieve high-order entropy compression in the CSA. In Section 6, we apply our CSA to build self-indexing data structures that support fast searching. In Section 7, we give some final considerations and open problems.

2 High-Order Empirical Entropy

In this section, we formulate our analysis of the space complexity in terms of the high-order empirical entropy of a text T of n symbols drawn from alphabet $\Sigma = \{1, 2, \ldots, \sigma\}$. In particular, we discuss various notions of entropy from both an empirical probability model and a finite set model. In Section 2.1, we consider classic notions of entropy according to the empirical probability model. We describe a new definition based on the finite set model in Section 2.2.

2.1 Empirical Probabilistic High-Order Entropy

We provide the necessary terminology for the analysis and explore empirical probability models. For each symbol $y \in \Sigma$, let n^y be the number of its occurrences in text T. With symbol y, we associate its empirical probability, $\text{Prob}[y] = n^y/n$, of occurring in T. (Note that by definition, $n = \sum_{y \in \Sigma} n^y$, so the empirical probability is well defined.) Following Shannon's definition of entropy [Sha48], the 0*th-order empirical entropy* is

$$H_0 = H_0(T) = \sum_{y \in \Sigma} -\text{Prob}[y] \times \lg \text{Prob}[y]. \tag{3}$$

Since $nH_0 \leq n \lg \sigma$, expression (3) simply states that an efficient variable-length coding of text T would encode each symbol y based upon its frequency in T rather than simply using $\lg \sigma$ bits. The number of bits assigned for encoding an occurrence of y would be $-\lg \text{Prob}[y] = \lg(n/n^y)$.

We can generalize the definition to higher-order empirical entropy, so as to capture the dependence of symbols upon their context, made up of the h previous symbols in the text. For a given h, we consider all possible h-symbol sequences x that appear in the text. (They are a subset of Σ^h, the set of all possible h-symbol sequences over the alphabet Σ.) We denote the number of occurrences in the text of a particular context x by n^x, with $n = \sum_{x \in \Sigma^h} n^x$ as before, and we let $n^{x,y}$ denote the number of occurrences in the text of the concatenated sequence yx (meaning that y precedes x).[4] Then, the *hth-order empirical entropy* is defined as

$$H_h = H_h(T) = \sum_{x \in \Sigma^h} \sum_{y \in \Sigma} -\text{Prob}[y, x] \times \lg \text{Prob}[y|x], \tag{4}$$

where $\text{Prob}[y, x] = n^{x,y}/n$ represents the empirical *joint* probability that the symbol y occurs in the text immediately before the context x of h symbols and $\text{Prob}[y|x] = n^{x,y}/n^x$ represents the empirical *conditional* probability that the symbol y occurs immediately before context x, given that x occurs in the text. (We refer the interested reader to [CT91] for more details on conditional entropy.) Setting $h = 0$, we obtain H_0 as defined previously. In words, expression (4) is similar to (3), except that we partition the probability space further according to contexts of length h in order to capture statistically significant patterns from the text.

An important observation to note is that $H_{h+1} \leq H_h \leq \lg \sigma$ for any integer $h \geq 0$. Hence, expression (4) states that a better variable-length coding of text T would encode each symbol y based upon the joint and conditional empirical frequency for any context x of y.

Manzini [Man01] gives an equivalent definition of (4) in terms of H_0. For any given context x, let w_x be the concatenation of the symbols y that appear in the text immediately before context x. We denote its length by $|w_x|$ and its 0th-order empirical entropy by $H_0(w_x)$, thus defining H_h as

$$H_h = \frac{1}{n} \sum_{x \in \Sigma^h} |w_x| H_0(w_x). \tag{5}$$

One potential difficulty with the definition of H_h is that the inner terms of the summation in (5) could equal 0 (or an arbitrarily small constant), which can be misleading when considering the encoding length of a text T. (One relatively trivial case is when the text contains n equal symbols, as no symbol needs to be "predicted".) Manzini introduced *modified* high-order empirical entropy H_h^* to address this point and capture the constraint that the encoding of the text must contain at least $\lg n$ bits for coding its length n. Using a modified

$$H_0^* = H_0^*(T) = \max\{H_0, (1 + \lfloor \lg n \rfloor)/n\} \tag{6}$$

to make the change, he writes

$$\hat{H}_h = \frac{1}{n} \sum_{x \in \Sigma^h} |w_x| H_0^*(w_x). \tag{7}$$

Unfortunately, $\hat{H}_{h+1} \leq \hat{H}_h$ does not necessarily hold in (7) as it did for H_h. To solve this problem, let P_h be a *prefix cover*, namely, a set of substrings having length at most h such that

[4] The standard definition of conditional probability for text documents considers the symbol y immediately *after* the sequence x, though it makes no meaningful difference since we could simply use this definition on the reversed text as discussed in [FGMS05].

every string from Σ^h has a unique prefix in P_h. Manzini then defines the *modified hth-order empirical entropy* as

$$H_h^* = H_h^*(T) = \frac{1}{n} \min_{P_h} \sum_{x \in P_h} |w_x| H_0^*(w_x). \tag{8}$$

so that $H_{h+1}^* \leq H_h^*$ *does* hold in (8). Other immediate consequences of this encoding-motivated entropy measure are that $H_h^* \geq H_h$ and $nH_h^* \geq \lg n$, but nH_h can be a small constant. Let the *optimal prefix cover* P_h^* be the prefix cover which minimizes H_h^* in (8). Thus, (8) can be equivalently stated as $H_h^* = \frac{1}{n} \sum_{x \in P_h^*} |w_x| H_0^*(w_x)$.

The empirical probabilities that are employed in the definition of the high-order empirical entropy can be obtained from the number of occurrences $n^{x,y}$, where $\sum_{x \in P_h^*, y \in \Sigma} n^{x,y} = n$. Indeed, $n^y = \sum_{x \in P_h^*} n^{x,y}$ and $n^x = \sum_{y \in \Sigma} n^{x,y}$. This motivates the following definition, which will guide us through our high-order entropy analysis.

Definition 1 The *empirical statistical model* for a text T is composed of two parts:

i. The partition of Σ^h induced by the contexts of the prefix cover P_h^*.
ii. The sequence of non-negative integers, $n^{x,1}, n^{x,2}, \ldots, n^{x,\sigma}$, where $x = 1, 2, \ldots, |P_h^*|$. (Recall that $n^{x,y}$ is the number of occurrences of yx as a substring of T.)

We denote the asymptotical number of bits used to store the information in parts (i)–(ii) by $M(T, \Sigma, h)$, as n increases.

2.2 Finite Set High-Order Entropy

We provide a new definition of high-order empirical entropy H'_h, based on the finite set model rather than on conditional probabilities. We use this definition to avoid dealing with empirical probabilities explicitly. We show that our new definition is $H'_h \leq H_h \leq H_h^*$, so that we can provide upper bounds in terms of H'_h in our analysis. These bounds immediately translate into upper bounds in terms of H_h and H_h^* as well.

For ease of exposition, we "number" the lexicographically ordered contexts x as $1 \leq x \leq \sigma^h$. Let the *multinomial* coefficient $\binom{n}{m_1, m_2, \ldots, m_p} = \frac{n!}{m_1! \, m_2! \cdots m_p!}$ represent the number of partitions of n items into p subsets of size $m_1, m_2, \ldots, m_p$. In this paper, we define $0! = 1$. (Note again that $n = m_1 + m_2 + \cdots + m_p$.) When $m_1 = t$ and $m_2 = n - t$, we get precisely the binomial coefficient $\binom{n}{t}$. We define

$$H'_0 = H'_0(T) = \frac{1}{n} \lg \binom{n}{n^1, n^2, \ldots, n^\sigma}, \tag{9}$$

which simply counts the number of possible partitions of n items into σ unique buckets, i.e. the alphabet size. We use the optimal prefix cover P_h^* in (8) to give the definition of our *alternative high-order empirical entropy*[5]

$$H'_h = H'_h(T) = \frac{1}{n} \sum_{x \in P_h^*} \lg \binom{n^x}{n^{x,1}, n^{x,2}, \ldots, n^{x,\sigma}}. \tag{10}$$

For example, consider the text $T = \texttt{mississippi\#}$. Fixing $h = 1$ and taking $P_h^* = \Sigma^h$, we have that all contexts are of length 1. For context $x = \texttt{i}$ occurring $n^{\texttt{i}} = 4$ times in T, we have the symbols $y = \texttt{m}, \texttt{p}$, and $\texttt{s}$ appearing $n^{\texttt{i},\texttt{m}} = n^{\texttt{i},\texttt{p}} = 1$ and $n^{\texttt{i},\texttt{s}} = 2$ times in T. Thus, the contribution of context $x = \texttt{i}$ to $nH'_1(T)$ is $\lg \binom{4}{1,1,2} = \lg 12$ bits. In fact, our definitions (9) and (10) are quite liberal, as shown below.

[5] Actually, it can be defined for any prefix cover P_h, including $P_h = \Sigma^h$.

Theorem 1 *For any given text T and context length $h \geq 0$, we have $H'_h \leq H_h$.*

Proof: It suffices to show that $nH'_0 \leq nH_0$ for all alphabets Σ, since then $\lg \binom{n^x}{n^{x,1}, n^{x,2}, \ldots, n^{x,\sigma}} \leq |w_x| H_0(w_x)$. By setting $P_h^* = \Sigma^h$ in (10) and applying Manzini's definition of entropy in (5) naturally leads to the claim.

The bound $nH'_0 \leq nH_0$ trivially holds when $\sigma = 1$. We first prove this bound for an alphabet Σ of $\sigma = 2$ symbols. Let t and $n - t$ denote the number of occurrences of the two symbols in T. We want to show that $nH'_0 = \lg \binom{n}{t} \leq nH_0 = t \lg(n/t) + (n-t) \lg(n/(n-t))$ by (9). The claim is true by inspection when $n \leq 4$ or $t = 0, 1, n-1$. Let $n > 4$ and $2 \leq t \leq n-2$. We apply Stirling's double inequality [Fel68] to obtain

$$\frac{n^n \sqrt{2\pi n}}{e^{n - \frac{1}{12n+1}}} < n! < \frac{n^n \sqrt{2\pi n}}{e^{n - \frac{1}{12n}}}. \tag{11}$$

Taking logarithms and focusing on the right-hand side of (11), we see that

$$\lg n! < n \lg \frac{n}{e} + \frac{1}{2} \lg n + \frac{1}{12n} \lg e + \lg \sqrt{2\pi}. \tag{12}$$

Similarly to (12), we take the left-hand side of (11), and obtain

$$\lg n! > n \lg \frac{n}{e} + \frac{1}{2} \lg n + \frac{1}{12n+1} \lg e + \lg \sqrt{2\pi}. \tag{13}$$

Applying (12) and (13) to $\lg \binom{n}{t} = \lg(n!) - \lg(t!) - \lg((n-t)!)$, we have

$$nH'_0 = \lg \binom{n}{t} < nH_0 - \frac{1}{2} \lg \frac{t(n-t)}{n} - \left[\frac{1}{12t+1} + \frac{1}{12(n-t)+1} - \frac{1}{12n} \right] - \lg \sqrt{2\pi}. \tag{14}$$

Since $t(n-t) \geq n$ and $1/(12t+1) + 1/(12(n-t)+1) \geq 1/(12n)$ by our assumptions on n and t, it follows that $nH'_0 \leq nH_0$, proving the result when $\sigma = 2$.

Next, we show the claimed bound for the general alphabet ($\sigma \geq 2$ and $h = 0$) and by using induction on the alphabet size (with the base case $\sigma = 2$ as detailed before). We write

$$\lg \binom{n}{n^1, n^2, \ldots, n^\sigma} = \lg \left[\binom{n - n^\sigma}{n^1, n^2, \ldots, n^{\sigma-1}} \times \binom{n}{n^\sigma} \right]. \tag{15}$$

We use induction for the right-hand side of (15) to get

$$\lg \binom{n - n^\sigma}{n^1, n^2, \ldots, n^{\sigma-1}} \leq \sum_{y=1}^{\sigma-1} n^y \lg \frac{n - n^\sigma}{n^y}, \tag{16}$$

$$\lg \binom{n}{n^\sigma} \leq n^\sigma \lg \frac{n}{n^\sigma} + (n - n^\sigma) \lg \frac{n}{n - n^\sigma}. \tag{17}$$

Summing (16) and (17), we obtain $\sum_{y=1}^{\sigma} n^y \lg \frac{n}{n^y} = nH_0$, thus proving the claim for any alphabet size σ.

□

The above discussion now justifies the use of H'_h in our later analysis, but we continue to state bounds in terms of H_h as it represents more standard notation. The key point to understand is that we can derive equations in terms of multinomial coefficients without worrying about the empirical probability of symbols appearing in the text T.

Original	*Sorted*			*Mappings*			*Suffix Array*	
Q	F		L	i	$LF(i)$	$\Phi(i)$	$SA[i]$	
mississippi#	i	ppi#missis	s	1	8	7	8	ippi#
#mississippi	i	ssippi#mis	s	2	9	10	5	issippi#
i#mississipp	i	ssissippi#	m	3	5	11	2	ississippi#
pi#mississip	i	#mississip	p	4	6	12	11	i#
ppi#mississi	m	ississippi	#	5	12	3	1	mississippi#
ippi#mississ	p	i#mississi	p	6	7	4	10	pi#
sippi#missis	p	pi#mississ	i	7	1	6	9	ppi#
ssippi#missi	s	ippi#missi	s	8	10	1	7	sippi#
issippi#miss	s	issippi#mi	s	9	11	2	4	sissippi#
sissippi#mis	s	sippi#miss	i	10	2	8	6	ssippi#
ssissippi#mi	s	sissippi#m	i	11	3	9	3	ssissippi#
ississippi#m	#	mississipp	i	12	4	5	12	#

Table 3: Matrix Q for the BWT containing the cyclic shifts of text $T =$ `mississippi#` (column '*Original*'). Sorting of the rows of Q, in which the first (F) and last (L) symbols in each row are separated (column '*Sorted*'). Functions LF and Φ for each row of the sorted Q (column '*Mappings*'). Suffix array SA for T (column '*Suffix Array*').

3 The Unified Algorithmic Framework: Tighter Analysis for the BWT

The characterization of the high-order empirical entropy in terms of the multinomial coefficients given in Section 2.2 drives our analysis in a unified framework for text compression and compressed text indexing. In this section, we begin with a simple, yet nearly optimal analysis of the Burrows-Wheeler transform (BWT). Section 3.1 formally defines the BWT and highlights its connection to (compressed) suffix arrays. Our key partitioning scheme is described in Section 3.2; it serves as the critical foundation in achieving a high-order entropy analysis for the BWT. Sections 3.3–3.4 motivate and develop our multi-use *wavelet tree* data structure, which serves as a flexible tool in both compression and text indexing. We finish the BWT analysis in Section 3.5.

3.1 The BWT and (Compressed) Suffix Arrays

We now give a short description of the BWT in order to explain its salient features. Consider the text $T =$ `mississippi#` in the example shown in Table 3, where `i` $<$ `m` $<$ `p` $<$ `s` $<$ `#` and `#` is an end-of-text symbol. The BWT forms a conceptual matrix Q whose rows are the cyclic (forward) shifts of the text in sorted order and stores the last column $L =$ `ssmp#pissiii` written as a contiguous string. Note that L is an invertible permutation of the symbols in T. In particular, $LF(i) = j$ in Table 3 indicates for any symbol $L[i]$, the corresponding position j in F where $L[i]$ appears. For instance, $LF(3) = 5$ since $L[3] =$ `m` occurs in position 5 of F; $LF(8) = 10$ since $L[8] =$ `s` occurs in position 10 of F (as the third `s` among the four appearing consecutively in F). Using L and LF, we can recreate the text T in reverse order by starting at the last position n (corresponding to `#mississippi`), writing its value from F, and following the LF function to the next value of F. Continuing the example from before, we follow the pointers from $LF(n)$: $LF(12) = 4$, $F[4] =$ `i`; $LF(4) = 6$, $F[6] =$ `p`; $LF(6) = 7$, $F[7] =$ `p`; and so on. In other words, the LF function gives the position in F of the preceding symbol from the original text T. Thus one could store L and recreate T, since we can obtain F by sorting L and the LF function can be derived by inspection. Note that L is compressible using 0th-order compressors, boosting them to attain high-order entropy [FGMS05]. In the following, we connect the BWT with L.

Clearly, the BWT is related to suffix sorting, since the comparison of any two circular shifts must stop when the end marker # is encountered. The corresponding suffix array is a simple way to store the sorted suffixes. The suffix array SA for a text T maintains the permuted order of $1, 2, \ldots, n$ that corresponds to the locations of the suffixes of the text in lexicographically sorted order, $T\big[SA[1], n\big]$, $T\big[SA[2], n\big]$, $\ldots$, $T\big[SA[n], n\big]$. By dropping the symbols *after* # in the sorted matrix Q (column '*Sorted*' in Table 3), we obtain the sequence of sorted suffixes represented by SA (column '*Suffix Array*' in Table 3). In the example above, $SA[6] = 10$ because the sixth largest lexicographically ordered suffix, `pi#`, begins at position 10 in the original text.

We make the connection between the BWT and SA more concrete by describing the neighbor function Φ, introduced to represent the CSA in [GV05]. In particular, the Φ function indicates, for any position i in SA, the corresponding position j in SA such that $SA[j] = SA[i]+1$ (a sort of suffix link similar to that of suffix trees [McC76]). For example in Table 3, $\Phi(6) = 4$ since $SA[6] = 10$ and $SA[4] = 11$. The Φ function can be implemented by using Σ *lists* as shown in [GV05]. Given a symbol $y \in \Sigma$, the list y is the set of positions from the suffix array such that for any position p in list y, $T[SA[p]]$ is *preceded* by y.[6] In words, it collects the positions where y occurs in the text based upon information from the suffix array. The fundamental property of these Σ lists is that each list is an *increasing* series of positions. For instance, list `i` from our example is $\langle 7, 10, 11, 12\rangle$ since for each entry, $T[SA[p]]$ is preceded by an `i`. The concatenation of the lists y for $y = 1, 2, \ldots, \sigma$ gives Φ. Going on in the example, list `m` is $\langle 3\rangle$; list `p` is $\langle 4, 6\rangle$; list `s` is $\langle 1, 2, 8, 9\rangle$, and list `#` is $\langle 5\rangle$. Their concatenation yields the Φ function shown in Table 3. Thus, the value of $\Phi(i)$ is just the ith nonempty entry in the concatenation of the lists, and belongs to some list y.

We can reconstruct SA and the BWT by using Φ and the position f of the last suffix $SA[f] = n$, where $\Phi(f)$ is the position in SA containing the first suffix. Continuing the example from before (where $f = 12$) we can recreate SA by iterating Φ as $\Phi(f) = 5$, $SA[5] = 1$; $\Phi(5) = 3$, $SA[3] = 2$; $\Phi(3) = 11$, $SA[11] = 3$, and so on. In general, we compute $\Phi(f)$, $\Phi(\Phi(f))$, $\ldots$, so that the rank j in SA for the ith suffix in T $(1 \leq i, j \leq n)$ is obtained as $j = \Phi^{(i)}(f)$ by i iterations of Φ on f. However, this process not only recovers the values of SA, but also the corresponding lists y (which provide the symbols for the BWT by the definition of Σ lists). In particular, symbol y occurs in the jth position of the BWT, where $j = \Phi^{(i)}(f)$. In the example, symbol $y =$ `#` is in position $\Phi(f) = 5$ of the BWT because the fth entry in Φ is in list `#`; symbol $y =$ `m` is in position $\Phi(5) = 3$ because the fifth entry is in list `m`; symbol $y =$ `i` is in position $\Phi(3) = 11$, and so on.

Hence, the Φ function is also an invertible representation of the BWT. As can be seen from Table 3, $LF(\Phi(i)) = \Phi(LF(i)) = i$ for $1 \leq i \leq n$; thus, these functions are *inverses* of each other. (The Φ function can also be thought of as the FL mapping while the LF mapping can be thought of as the encoding of inverse suffix links.) Encoding the Φ function is no harder than encoding LF. In the following, we make use of this connection to achieve a high-order empirical entropy analysis of the BWT.

3.2 Context-Based Partitioning of the BWT

We now show our major result for this section; we describe a nearly optimal analysis of the compressibility of the Burrows-Wheeler transform with respect to high-order empirical entropy, exploiting the relationship between the BWT and suffix arrays illustrated in Section 3.1.

Let P_h^* be the optimal prefix cover as defined in Section 2, and let $n^{x,y}$ be the corresponding values in equation (10), where $x \in P_h^*$ and $y \in \Sigma$ (see also Definition 1). We denote by $|P_h^*|$ the number of contexts in P_h^*, where $|P_h^*| \leq \sigma^h$. The following theorem formalizes the bounds that we anticipated in formulas (1) and (2) for our analysis.

[6] Specifically, $y = T[SA[p] - 1]$ for $SA[p] > 1$, and $y = T[n]$ when $SA[p] = 1$.

context x	**list** `i`	**list** `m`	**list** `p`	**list** `s`	**list** `#`
`i`	$\emptyset$	$\langle 3 \rangle$	$\langle 4 \rangle$	$\langle 1, 2 \rangle$	$\emptyset$
`m`	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\langle 5 \rangle$
`p`	$\langle 7 \rangle$	$\emptyset$	$\langle 6 \rangle$	$\emptyset$	$\emptyset$
`s`	$\langle 10, 11 \rangle$	$\emptyset$	$\emptyset$	$\langle 8, 9 \rangle$	$\emptyset$
`#`	$\langle 12 \rangle$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

Table 4: An example of our conceptual table $\mathcal{T}$, where each sublist $\langle x, y \rangle$ contain $n^{x,y}$ entries. The contexts x are associated with rows and the lists y are associated with columns.

context x	n^x	$\#x$	**list** `i`	**list** `m`	**list** `p`	**list** `s`	**list** `#`
`i`	4	0	$\emptyset$	$\langle 3 \rangle$	$\langle 4 \rangle$	$\langle 1, 2 \rangle$	$\emptyset$
`m`	1	4	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\langle 1 \rangle$
`p`	2	5	$\langle 2 \rangle$	$\emptyset$	$\langle 1 \rangle$	$\emptyset$	$\emptyset$
`s`	4	7	$\langle 3, 4 \rangle$	$\emptyset$	$\emptyset$	$\langle 1, 2 \rangle$	$\emptyset$
`#`	1	11	$\langle 1 \rangle$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$

Table 5: The sublists of Table 4 in normalized form. The value of n^x is defined as in equation (10) and indicates the interval length in the row for context x. The value $\#x$ should be added to the sublists' entries in row x to obtain the same entries in Table 4.

Theorem 2 (Space-Optimal Burrows-Wheeler Transform) *The Burrows-Wheeler transform for a text T of n symbols drawn from an alphabet Σ can be compressed using*

$$nH_h + M(T, \Sigma, h) \tag{1}$$

bits for the best choice of context length h and prefix cover P_h^*, where the number of bits required for encoding the empirical statistical model behind P_h^* (see Definition 1) is

$$M(T, \Sigma, h) \leq \min\left\{ g'_h \lg(1 + n/g'_h),\ H_h^* n + \lg n + g''_h \right\}, \tag{2}$$

where $g'_h = O(\sigma^{h+1})$ and $g''_h = O(\sigma^{h+1} \lg \sigma^{h+1})$ do not depend on the text length n.

We devote the rest of Section 3 to the proof of Theorem 2. We describe our analysis for an *arbitrary* prefix cover P_h, so it also holds also for the optimal prefix cover P_h^* as in equation (10). Since every string in Σ^h has a unique prefix in P_h, we have that P_h induces a partition of the suffixes stored in the suffix array SA (or the corresponding circular shifts of T). In particular, the suffixes starting with a given context $x \in P_h$ occupy contiguous positions in SA. In the example of Table 3, the positions $1, \ldots, 4$ in SA corresponds to the suffixes starting with context $x =$ `i`.

Our basic idea is to apply context partitioning to the Σ *lists* discussed in Section 3.1. We implement our idea by partitioning each list y further into sublists $\langle x, y \rangle$ by contexts $x \in P_h$. Intuitively, sublist $\langle x, y \rangle$ stores the suffixes in SA that start with x and are preceded by y. Thus, each item p in sublist $\langle x, y \rangle$ indicates that $T\big[SA[p] - 1, SA[p] + h\big] = yx$. For context length $h = 1$, if we continue the example in Table 3, we break the Σ lists by context (in lexicographical order `i`, `m`, `p`, `s`, and `#`, and numbered from 1 up to $|P_h|$). The list for $y =$ `i` is $\langle 7, 10, 11, 12 \rangle$, and is broken into sublist $\langle 7 \rangle$ for context $x =$ `p`, sublist $\langle 10, 11 \rangle$ for context $x =$ `s`, and sublist $\langle 12 \rangle$ for $x =$ `#`. We recall that the fundamental property of Σ lists is that each list is an increasing series of positions. Thus, each sublist $\langle x, y \rangle$ we have created is also *increasing* and contains $n^{x,y}$ entries, where $n^{x,y}$ is defined as in equation (10) and Definition 1.

We build a conceptual 2-dimensional table $\mathcal{T}$ that follows Definition 1. (Each row x implicitly represents the suffixes in SA that start with context x and the columns y are the symbols "predicted"

in each context.) The contexts $x \in P_h$ correspond to the rows and the Σ lists y are stored in the columns x. The columns of $\mathcal{T}$ are partitioned by row according to the contexts. For our running example (with $h = 1$) see Table 4 for an instance of $\mathcal{T}$. Our table $\mathcal{T}$ has some nice properties if we consider its rows and columns as follows:

- We can implement the Φ function by accessing the sublists in $\mathcal{T}$ in *column major order*, as discussed in Section 3.1.
- We have a strong relationship with the high-order empirical entropy in equation (10) and the statistical empirical model of Definition 1, if we encode these sublists in *row major order*.

For any context $x \in P_h$, if we encode the sublists in row x using nearly $\lg \binom{n^x}{n^{x,1}, n^{x,2}, \ldots, n^{x,\sigma}}$ bits, we automatically achieve the hth-order empirical entropy when summing over all the contexts as required in equation (10). For example, context $x = \texttt{i}$ should be represented with nearly $\lg \binom{4}{1,1,2}$ bits, since two sublists contain one entry each and one sublist contains two entries. The empirical statistical model should record the partition induced by P_h and which sublists are empty, and should encode the lengths of the nine nonempty sublists in Table 4, using $M(T, \Sigma, h)$ bits.

The crucial observation to make is that all entries in the row corresponding to a given context x create a *contiguous* sequence of positions. For instance, along the first row of Table 4 for $x = \texttt{i}$, there are four entries that are in the range $1 \ldots 4$. Similarly, row $x = \texttt{s}$ contains the four entries in the range $8 \ldots 11$; row `s` should be encoded with $\lg \binom{4}{2,2}$ bits. We represent this range as an interval $[1, 4]$ with the offset $\#x = 7$. We call this representation a *normalization*, which subtracts the value of $\#x$ from each entry p of the sublists $\langle x, y \rangle$ for $y \in \Sigma$. In words, we normalize the sublists in Table 4 by *renumbering each element based on its order within its context* and obtain the context information shown in Table 5. Here, n^x is the number of elements in each context x, and $\#x$ represents the partial sum of all prior entries; that is, $\#x = \sum_{x' < x} n^{x'}$. (Note that the values of n^x and $\#x$ are easily computed from the set of sublist lengths $n^{x,y}$.) For example, the first entry in sublist $\langle \texttt{s}, \texttt{i} \rangle$, 10, is written as 3 in Table 5, since it is the third element in context `s`. We can recreate entry 10 from $\#x$ by adding $\#\texttt{s} = 7$ to 3. As a result, each sublist $\langle x, y \rangle$ is a subset of the range implicitly represented by interval $[1, n^x]$ with the offset $\#x$. We exploit this organization to encode the BWT.

Encoding: We run the boosting algorithm from [FGMS05] on the BWT to find the optimal value of context order h and the optimal prefix cover P_h^* using the cost of $nH_h' + M(T, \Sigma, h)$ according to equation (10). (Recall that $H_h' \leq H_h$ by Theorem 1.) Once we know h and set $P_h = P_h^*$, we can cleanly separate the contexts and encode the Φ function as described in our table $\mathcal{T}$. Thus, we follow the two steps below, storing the following components of $\mathcal{T}$:

1. We encode the empirical statistical model given in Definition 1.
2. For each context $x \in P_h$, we separately encode the sublists $\langle x, y \rangle$ for $y \in \Sigma$ to capture high-order entropy. Each of these sublists is a subset of the integers in the range $[1, n^x]$ with offset $\#x$. These sublists form a *partition* of the integers in the interval $[1, n^x]$.

The storage for step 1 is $M(T, \Sigma, h)$, the asymptotical number of bits required for encoding the model (see Definition 1). The storage required for step 2 should use nearly $\lg \binom{n^x}{n^{x,1}, n^{x,2}, \ldots, n^{x,\sigma}}$ bits per context x, and should not exceed a total of nH_h bits plus lower-order terms, once we determine P_h^*, as stated in Theorem 2.

Decoding: We retrieve the empirical statistical model encoded in step 1 above, which allows us to infer the number of rows and columns of our table $\mathcal{T}$, and which sublists are nonempty and their lengths. (Note that the values of n, n^x and $\#x$ can be obtained from these lengths.) Next, we retrieve the sublists encoded in step 2 since we know their lengths. At this point, we have recovered the content of $\mathcal{T}$, allowing us to implement the Φ function with the columns of $\mathcal{T}$ as discussed before. Given Φ, we can decode BWT as described at the end of Section 3.1.

The rest of the section discusses how to encode the sublists and the empirical statistical model, so as to complete the proof of Theorem 2.

3.3 Removing Space Redundancy of Multiple Sublists

We motivated and built a partitioning scheme that allows us to consider each context independently, as emphasized at the end of Section 3.2. For any given context $x \in P_h$, we now focus on the problem of encoding the sublists $\langle x, y \rangle$ for $y \in \Sigma$ (i.e. step 2 of encoding). We recall that the latter sublists form a partition of the integers in the range represented by the interval $[1, n^x]$ with offset $\#x$. In the following, we assume that $\#x = 0$ without loss of generality, since the values of $\#x$ can be inferred (by Definition 1).

One simple method would be to simply encode each sublist $\langle x, y \rangle$ as a subset of $t = n^{x,y}$ items out of a universe of $n' = n^x$ items. We can use *t-subset encoding*, requiring the information-theoretic minimum of $\lceil \lg \binom{n'}{t} \rceil$ bits, with $O(t)$ operations, according to [Knu05, Rus05]. All the t-subsets are enumerated in some canonical order (e.g. lexicographic order) and the subset occupying rank r in this order is encoded by the value of r itself, which requires $\lceil \lg \binom{n'}{t} \rceil$ bits. In this way, the t-subset encoding can also be seen as the compressed representation of an *implicit bitvector* of length n': If the subset contains $1 \leq s_1 < s_2 < \cdots < s_t \leq n'$, the s_ith entry in the bitvector is $\mathbf{1}$, for $1 \leq i \leq t$; the remaining $n' - t$ bits are $\mathbf{0}$s. Thus, we can use subset rank (and unrank) primitives for encoding (and decoding) sublist $\langle x, y \rangle$ as a sequence r of $\lceil \lg \binom{n^x}{n^{x,y}} \rceil$ bits.

Unfortunately, the fact that our subset encoding is *locally* optimal for sublist $\langle x, y \rangle$ does not imply that it is *globally* optimal for all the sublists. Indeed, summing over the contexts x shows that the total space bound is more than the entropy term nH_h. Actually, this sum adds a linear term of $O(n)$, which prevents us to reach a nearly optimal analysis for low-entropy texts. In order to see why, let t^x be the number of nonempty sublists contained in a given context x and, without loss of generality, let the number of entries in the nonempty sublists be $n^{x,1}$, $n^{x,2}$, ..., n^{x,t^x}, where $\sum_{1 \leq y \leq t^x} n^{x,y} = n^x$.

Lemma 1 *Given context x, the following relation holds,*

$$\sum_{1 \leq y \leq t^x} \lg \binom{n^x}{n^{x,y}} = \lg \binom{n^x}{n^{x,1}, n^{x,2}, \ldots, n^{x,t^x}} + O(n^x). \tag{18}$$

Proof: When $t^x = 2$, $\sum_{1 \leq y \leq t^x} \lg \binom{n^x}{n^{x,y}} = \lg \binom{n^x}{n^{x,1}, n^{x,2}, \ldots, n^{x,t^x}}$ and the lemma is trivially proved. Thus, let $t^x > 2$, so that the following holds.

$$\begin{aligned}
\sum_{1 \leq y \leq t^x} \lg \binom{n^x}{n^{x,y}} &= \lg \left[\frac{1}{n^{x,1}!\, n^{x,2}! \ldots n^{x,t^x}!} \times \frac{(n^x!)^{t^x}}{(n^x - n^{x,1})!\, (n^x - n^{x,2})! \ldots (n^x - n^{x,t^x})!} \right] \\
&\leq \lg \left[\frac{1}{n^{x,1}!\, n^{x,2}! \ldots n^{x,t^x}!} \, (n^x)^{(n^x)} \right] \\
&= \lg \left[\frac{1}{n^{x,1}!\, n^{x,2}! \ldots n^{x,t^x}!} \right] + n^x \lg n^x
\end{aligned}$$

Since $\lg \binom{n^x}{n^{x,1}, n^{x,2}, \ldots, n^{x,t^x}} = \lg \left[\frac{n^x!}{n^{x,1}!\, n^{x,2}! \ldots n^{x,t^x}!} \right]$ and $\lg n^x! \leq n^x \lg n^x - n^x \lg e + 1/2 \lg n^x + 1/12n \lg e + \lg \sqrt{2\pi}$ by Stirling's inequality [Fel68], the claim is proved. The additional term of $O(n^x)$ in equation (18) is tight in several cases; for example, when $t^x = n^x > 2$ and each $n^{x,y} = 1$. □

The apparent paradox implied by equation (18) can be resolved by noting that each subset encoding only represents the entries of one particular sublist; that is, there is a separate subset encoding for each symbol y in context x. In the multinomial coefficient of equations (10) and (18), all the sublists are encoded together as a multiset. Thus, it is more expensive to have a subset encoding of *each sublist individually* rather than having a single encoding for the entire context. In

Lemma 1, the $O(n^x)$ additive term in the number of bits is the extra cost incurred by encoding all positions where each of the sublists does *not* occur. When summed over all n entries in all sublists and all contexts, this term gives an $O(n)$ contribution to the total space bound.

We can remove this undesired space term by encoding each of our sublists in terms of the locations not occupied by prior sublists within the same context. We perform a *scaling* of the universe to this end. This is better explained in terms of the implicit bitvectors related to the subset encodings. In our example of Table 5, for context $x = \mathtt{i}$, the $\langle x, \mathtt{m}\rangle$ sublist contains the third position in the interval $[1, 4] = \{1, 2, 3, 4\}$, so the corresponding bitvector is **0010**, encoded in $\lceil \lg \binom{4}{1} \rceil$ bits. When we encode the $\langle x, \mathtt{p}\rangle$ sublist, we only encode the positions that the $\langle x, \mathtt{p}\rangle$ sublist occupies in terms of positions *not* used by the $\langle x, \mathtt{m}\rangle$ sublist, namely, we are left with positions $\{1, 2, 4\}$ corresponding to the **0**s in the bitvector. So entry 4 in the $\langle x, \mathtt{p}\rangle$ sublist corresponds to position 3 in $\{1, 2, 4\}$ (out of three items in the scaled universe) and the resulting bitvector, **001**, is encoded in $\lceil \lg \binom{3}{1} \rceil$ bits. After that, the positions available are in $\{1, 2\}$ (the two remaining **0**s in the scaled universe). When we encode later sublists, we encode only those positions not used by the $\langle x, \mathtt{m}\rangle$ and $\langle x, \mathtt{p}\rangle$ sublists. In our example, the $\langle x, \mathtt{s}\rangle$ sublist contains the remaining two positions thus available, yielding the bitvector **11** encoded in $\lceil \lg \binom{2}{2} \rceil = 0$ bits. The total number of bits for context x is $\lceil \lg \binom{4}{1} \rceil + \lceil \lg \binom{3}{1} \rceil + \lceil \lg \binom{2}{2} \rceil < \lg \binom{4}{1,1,2} + 3$ as required.

In general, we apply the scaling of the universe for a context x as follows. When we encode the $\langle x, y\rangle$ sublist, we only encode the positions that the $\langle x, y\rangle$ sublist occupies in terms of positions *not* used by the $\langle x, y'\rangle$ sublists for $1 \le y' < y$. (These positions are those corresponding to the remaining **0**s in the resulting bitvector.) In this way, we iterate the scaling to the sublists:

1. We represent sublist $\langle x, 1\rangle$ using $n^{x,1}$-subset encoding in a universe of size n^x, with $\lceil \lg \binom{n^x}{n^{x,1}} \rceil$ bits.
2. For $y = 2, 3, \ldots, t^x$, we represent sublist $\langle x, y\rangle$ using $n^{x,y}$-subset encoding in a scaled universe of size $n' = n^x - \sum_{y'=1}^{y-1} n^{x,y'}$, with $\lceil \lg \binom{n'}{n^{x,y}} \rceil$ bits.

Note that the last sublist, $\langle x, t^x\rangle$, is encoded using $\lceil \lg \binom{n^{x,t^x}}{n^{x,t^x}} \rceil = 0$ bits. We introduce here the notion of *depth* of a context x, which measures the maximum number of sublists in context x that must be examined to recover the entries of any sublist of x. As we shall see, the depth is related to decompression time; in the above scheme, the depth is t^x.

Lemma 2 (Incremental Representation of Sublists) *For each context x, using the incremental representation of sublists by scaling the universe, we can encode the t^x nonempty sublists for that context in fewer than $\lg \binom{n^x}{n^{x,1}, n^{x,2}, \ldots, n^{x,\sigma}} + t^x$ bits, so that the depth is t^x.*

Proof: It suffices to consider the representation for a single context x. We show that in the encoding of the sublists for x, the information theoretically minimum space required for these sublists is

$$\begin{aligned}
&\left\lceil \lg \binom{n^x}{n^{x,1}} \right\rceil + \left\lceil \lg \binom{n^x - n^{x,1}}{n^{x,2}} \right\rceil + \left\lceil \lg \binom{n^x - n^{x,1} - n^{x,2}}{n^{x,3}} \right\rceil + \cdots + \left\lceil \lg \binom{n^{x,t^x}}{n^{x,t^x}} \right\rceil \\
&< \lg \left[\binom{n^x}{n^{x,1}} \binom{n^x - n^{x,1}}{n^{x,2}} \binom{n^x - n^{x,1} - n^{x,2}}{n^{x,3}} \cdots \binom{n^{x,t^x}}{n^{x,t^x}} \right] + t^x \\
&= \lg \left[\frac{n^x!}{n^{x,1}!\, n^{x,2}! \cdots n^{x,t^x}!} \right] + t^x = \lg \binom{n^x}{n^{x,1}, n^{x,2}, \ldots, n^{x,t^x}} + t^x.
\end{aligned}$$

We can replace t^x by σ in the multinomial coefficient of the above formula because the empty sublists do not contribute. The ramifications of the above approach in terms of the depth is that the recovering of the entries of a sublist of context x would be somewhat sequential in terms of t^x, the number of nonempty sublists within x, in order to decompose the relative positions that are stored. For instance, to recover the ith position in the $\langle x, \mathtt{p}\rangle$ sublist, we have to find the position j of the ith non-position in the $\langle x, \mathtt{m}\rangle$ sublist (i.e. the position j of the ith **0** in its corresponding

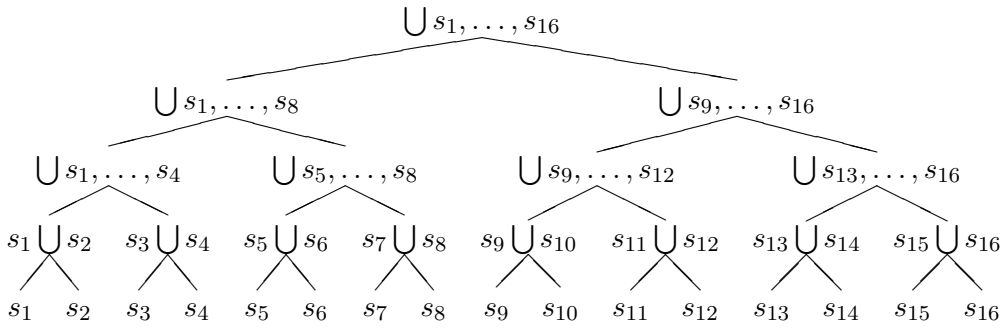

Figure 1: A wavelet tree

bitvector). Then we have to find the position of the jth non-position in the $\langle x, \mathtt{i} \rangle$ sublist, and so on. Thus, the depth is t^x, since we potentially have to backtrack through each nonempty sublist to recover the entries of the last sublist in the context. □

3.4 The Wavelet Tree

As we saw in Section 3.3, the main difficulty with the approach of Lemma 2 is the linear depth, which gives an overhead associated with backtracking through all the sublists of a given context. We present instead the *wavelet tree* data structure, which is of independent interest, to reduce the depth to nearly $\lg t^x \leq \lg \sigma$, while still encoding in the desired space.

We can consider the scheme discussed in Section 3.3 as a sequential representation of the sublists for context x. In particular, scaling the universe can be seen as an associative operation on these sublists, and the sequential representation can be thought of as one of the possible associations. Others are clearly possible, such as the hierarchical representation. Our idea is that of applying subset encoding to groups of sublists, instead of individual sublists. Going on in our example, we could group sublists $\langle x, \mathtt{m} \rangle$ and $\langle x, \mathtt{p} \rangle$ together, thus obtaining positions $\langle 3, 4 \rangle$ for them. At this point, the corresponding bitvector would be **0011**, represented with $\lceil \lg \binom{4}{2} \rceil$ bits. Then, the $\langle x, \mathtt{s} \rangle$ list would be represented as before, with $\lceil \lg \binom{2}{2} \rceil$ bits. We need a further subset encoding to distinguish between $\langle x, \mathtt{m} \rangle$ and $\langle x, \mathtt{p} \rangle$, but on a scaled universe with bitvectors **10** and **1**, respectively, using $\lceil \lg \binom{2}{1} \rceil$ and $\lceil \lg \binom{1}{1} \rceil$ bits. Note that total space is still bounded as before, namely, $\lceil \lg \binom{4}{2} \rceil + \lceil \lg \binom{2}{2} \rceil + \lceil \lg \binom{2}{1} \rceil + \lceil \lg \binom{1}{1} \rceil < \lg \binom{4}{1,1,2} + 3$, since the terms of the form $\lceil \lg \binom{k}{k} \rceil = 0$ do not contribute.

We can generalize this approach to any wavelet tree of binary degree, including the balanced wavelet tree in Figure 1, which is of interest in this paper as its tree shape can be kept implicitly. In order to explain its salient features, it suffices to consider the representation for a single context x. For ease of presentation, let s_y denote the $\langle x, y \rangle$ sublist of $n^{x,y}$ entries. We do not actually store the t^x leaves, but they are drawn for clarity. Each leaf represents a sublist as shown in Figure 1. We implicitly consider each left branch to be associated with a **0** and each right branch to be associated with a **1**. Each internal node is the subset encoding with the elements in its left subtree stored as **0**, and the elements in its right subtree stored as **1**. For instance, the root node of the tree in Figure 1 represents each of the positions of $s_1, \ldots, s_8$ as a **0**, and each of the positions of $s_9, \ldots, s_{16}$ as a **1**. Any grouping of sublists is possible, not necessarily involving those associated with consecutive leaves.

The major point here is that each of the $t^x - 1$ internal nodes represents elements relative to its subtrees. Rather than the linear relative encoding of sublists we had in Section 3.3, we use a tree structure to exploit exponential relativity, thus reducing the depth significantly from t^x to $O(\lg t^x)$. In some sense, the earlier approach corresponds to a completely skewed wavelet tree, as opposed

to the balanced structure now. Recovering the entries of any sublist s_y proceeds exactly as in Section 3.3, except that we start from the leaf corresponding to s_y and examine only the subsets in its ancestors. This motivates the reduction in the depth. Interestingly, any shape of the wavelet tree gives the same upper bounds on space; what changes is the depth.

Lemma 3 (Wavelet Tree Compression) *Using a wavelet tree for each context x, we can encode the t^x nonempty sublists for that context in fewer than $\lg \binom{n^x}{n^{x,1}, n^{x,2}, \ldots, n^{x,\sigma}} + t^x$ bits, so that the depth is $O(\lg t^x)$.*

Proof: We analyze the space required in terms of the contribution of each internal node's t-subset encoding. We prove that this cost is exactly the logarithm of the multinomial coefficient in equation (10) for the high-order empirical entropy.[7] Note that the leaves of the wavelet tree do not contribute to the cost since they generate terms of the form $\lceil \lg \binom{s_y}{s_y} \rceil = 0$ in the calculations for the number of required bits. By induction, it is simple to verify that the space required among all the $t^x - 1$ internal nodes is

$$\begin{aligned}
&\lg \binom{n^{x,1} + n^{x,2}}{n^{x,2}} + \lg \binom{n^{x,3} + n^{x,4}}{n^{x,4}} + \cdots + \lg \binom{n^{x,t^x-1} + n^{x,t^x}}{n^{x,t^x}} \\
&\quad + \lg \binom{n^{x,1} + \cdots + n^{x,4}}{n^{x,3} + n^{x,4}} + \lg \binom{n^{x,5} + \cdots + n^{x,8}}{n^{x,7} + n^{x,8}} + \cdots + \lg \binom{n^{x,t^x-3} + \cdots + n^{x,t^x}}{n^{x,t^x-1} + n^{x,t^x}} \\
&\quad \vdots \\
&\quad + \lg \binom{n^{x,1} + \cdots + n^{x,t^x}}{n^{x,1} + \cdots + n^{x,t^x/2}} = \lg \binom{n^x}{n^{x,1}, n^{x,2}, \ldots, n^{x,t^x}} = \lg \binom{n^x}{n^{x,1}, n^{x,2}, \ldots, n^{x,\sigma}}.
\end{aligned}$$

Hence, each wavelet tree encodes a particular context in precisely the high-order empirical entropy, which is what we wanted in equation (10). As in the proof of Lemma 2, the rounding due to the ceilings adds further t^x bits to the above bound. □

The advantage of using the wavelet tree will be clear in the rest of the paper, where we use the Φ function described in Section 3.1. In Section 4, we will replace the t-subset encodings with fully indexable dictionaries [RRR02] inside the nodes of the wavelet tree. We will exploit its organization for compressed text indexing, as we detail in Sections 5 and 6.

3.5 A Space-Optimal Burrows-Wheeler Transform

In this section, we pull together our techniques and summarize our analysis of the BWT for proving Theorem 2. Our proof constructs a compressed version of the BWT in $nH_h + M(T, \Sigma, h)$ bits. Looking at the encoding described at the end of Section 3.2, the storage required for step 1 is $M(T, \Sigma, h)$ bits, while the storage for step 2 uses fewer than $\lg \binom{n^x}{n^{x,1}, n^{x,2}, \ldots, n^{x,\sigma}} + t^x$ bits for each context x by Lemma 3. The overall contribution for the contexts is $nH'_h + |P^*_h|\sigma \leq nH_h + \sigma^{h+1}$ bits, by equation (10), Theorem 1, and the fact that $\sum_{x \in P^*_h} t^x \leq |P^*_h|\sigma \leq \sigma^{h+1}$. (We will soon account for this term σ^{h+1} in $M(T, \Sigma, h)$.)

To make our analysis more concrete in comparison to previous work, we give some significant upper bounds on $M(T, \Sigma, h)$. Recall that the empirical statistical model relates to two items. Item (i) in Definition 1 needs to encode the partition of Σ^h induced by the contexts of P^*_h using subset encoding with $\lceil \lg \binom{\sigma^h}{|P^*_h|} \rceil \leq \sigma^h$ bits. So, we can assume that $M(T, \Sigma, h) \geq \sigma^{h+1}$ in the worst

[7] In some sense, we are calculating the space requirements for each s_y, propagated over the entire tree. For instance, in the example above, s_y is implicitly stored in each node of its root-to-leaf path. We could analyze it this way, and show that the two notions are the same, though we defer the argument in the interest of brevity.

case. Item (ii) in Definition 1 encodes the sequence of lengths of the sublists, $n^{x,y}$. We devote the rest of this section to the encoding of these lengths using Elias' gamma and delta codes [Eli75]. [8]

We recall that the gamma code for a positive integer ℓ represents ℓ in two parts: the first encodes $1 + \lfloor \lg \ell \rfloor$ in unary, followed by the value of $\ell - 2^{\lfloor \lg \ell \rfloor}$ encoded in binary, for a total of $1 + 2\lfloor \lg \ell \rfloor$ bits. For example, the gamma codes for $\ell = 1, 2, 3, 4, 5, \ldots$ are $\mathbf{1}, \mathbf{01\,0}, \mathbf{01\,1}, \mathbf{001\,00}, \mathbf{001\,01}, \ldots$, respectively. The delta code requires fewer bits asymptotically by encoding $1 + \lfloor \lg \ell \rfloor$ via the gamma code rather than in unary. For example, the delta codes for $\ell = 1, 2, 3, 4, 5, \ldots$ are $\mathbf{1}, \mathbf{010\,0}, \mathbf{010\,1}, \mathbf{011\,00}, \mathbf{011\,01}, \ldots$, and require $1 + \lfloor \lg \ell \rfloor + 2\lfloor \lg \lg 2\ell \rfloor$ bits. Other choices for integer encoding are possible but here we give a couple of examples leading to the bound given in formula (2). In the first example, we obtain the following upper bound.

Lemma 4 $M(T, \Sigma, h) = O\left(\sigma^{h+1} \lg(1 + n/\sigma^{h+1})\right)$.

Proof: In this encoding, we choose to represent the lengths using the gamma code. We obtain a bitvector, Z, obtained by concatenating the gamma codes for $n^{x,1}, n^{x,2}, \ldots, n^{x,\sigma}$ for $x = 1, 2, \ldots, |P_h^*|$. Note that Z contains $O(\sum_{x \in P_h^*, y \in \Sigma} \lg n^{x,y})$ bits. This quantity is maximized when all lengths $n^{x,y}$ are equal to $\Theta(n/(|P_h^*| \times \sigma) + 1)$ by Jensen's inequality [CT91], and thus can be bounded by $O\big((|P_h^*| \times \sigma) \lg\big(1 + n/(|P_h^*| \times \sigma)\big)\big) = O\big(\sigma^{h+1} \lg(1 + n/\sigma^{h+1})\big)$ since $|P_h^*| \leq \sigma^h$. We do not need to encode n as it can be recovered from the sum of the sublists lengths. □

In the second example, we show how to obtain an alternative upper bound in terms of the modified entropy, H_h^*, as given in Section 2.1.

Lemma 5 $M(T, \Sigma, h) \leq nH_h^* + \lg n + O\left(\sigma^{h+1} \lg \sigma^{h+1}\right)$.

In order to prove Lemma 5, we need some technical facts.

Fact 1 *For $1 \leq t \leq n'/2$, we have $\lg \binom{n'}{t} >= 2t - \lg t - 1$.*

Proof: $\lg \binom{n'}{t} \geq \lg \binom{2t}{t} \geq \lg(2^{2t}/2t) = 2t - \lg t - 1$. □

Fact 2 *For any arbitrarily fixed constant γ with $0 < \gamma < 1$, there exist constants $t_\gamma, n_\gamma > 0$ such that the following inequalities hold for any integers $t > t_\gamma$ and $n' > n_\gamma$:*

i. $2 \lg \lg(2t) + 1 < \gamma \lg t$,

ii. $\lg t + 2 \lg \lg(2t) + 1 < \gamma(2t - \lg t - 1)$,

iii. $\binom{n'}{t} > (n')^{\gamma^{-1}}$ *and* $t \leq n'/2$.

Proof: Points (i) and (ii) follow from the asymptotics as t increases. For point (iii), we have $\binom{n'}{t} \geq \frac{n'(n'-1)\cdots(n'-\lceil \gamma^{-1} \rceil)}{(\lceil \gamma^{-1} \rceil + 1)!} > (n')^{\gamma^{-1}}$ when $t > \lceil \gamma^{-1} \rceil$ for sufficiently large n' (where $t \leq n'/2$). □

Fact 3 *The sequence of lengths n^x for $x = 1, 2, \ldots, |P_h^*|$ can be stored using*

$$\sum_{x \in P_h^*} \lg n^x + \lg n + O(\sigma^{h+1} \lg \sigma^{h+1})$$

bits of space.

[8] In a certain sense, the contribution of Theorem 2 is that of restricting the analysis of the BWT to the integer encoding problem for the lengths $n^{x,y}$ in the empirical statistical model.

Proof: With each context x with n^x entries, we associate its length n^x encoded in binary using $b(x) = \lfloor \lg n^x \rfloor + 1$ bits. These $b(x)$ bits alone do not permit a decoding of n^x, as they are not prefix codes and so we need to know the value of $b(x)$. We describe how to fix this problem. We permute the contexts x so that they are sorted by the $b(x)$ values. Note that the contexts requiring the same amount of bits for their lengths are contiguous now, and there are at most $\lg n + 1$ distinct values of $b(x)$ since $1 \leq b(x) \leq \lg n + 1$. A bitvector of $\lg n + 1$ bits can record which are the distinct values of $b(x)$ for all contexts x. All what remains to record is when $b(x') < b(x)$ for any two consecutive contexts x' and x in the sorted order. (Recall that either $b(x') < b(x)$ or $b(x') = b(x)$ after the sorting.) We use further $|P_h^*|$ bits to mark this situation. Given the permutation of the contexts sorted by $b(x)$, we require $O(\lg |P_h^*|!) = O(\sigma^{h+1} \lg \sigma^{h+1})$ bits to restore their original order. □

To prove Lemma 5, we have to encode the sequence of the sublists' lengths, $n^{x,1}, n^{x,2}, \ldots, n^{x,\sigma}$, where $x = 1, 2, \ldots, |P_h^*|$ and $\sum_{x \in P_h^*, y \in \Sigma} n^{x,y} = n$. We do not encode the sequence as is, but we perform the following preprocessing where we use constants γ and δ such that $0 < \gamma = 2\delta < 1$.

If context x contains just a single nonempty sublist, we use σ bits to mark that nonempty sublist among the σ sublists for x. We also encode n^x in $\lg n^x$ bits by Fact 3. Since $n^x H_0^*(w_x) = \lg n^x + O(1)$ in this case (see Section 2.1), we obtain a total contribution of $n^x H_0^*(w_x) + \sigma$ bits in Lemma 5.

If context x contains two or more nonempty sublists, we still use σ bits to mark the nonempty sublists among the σ sublists for x. However, we need a more careful analysis to prove our claimed bound for an equivalent representation of the sequence $n^{x,1}, n^{x,2}, \ldots, n^{x,\sigma}$ that we describe next. For ease of analysis, recall from Section 3.3 that we can alternatively obtain the logarithm of the multinomial coefficient in formula (10) by using $\lceil \lg \binom{n'}{n^{x,y}} \rceil$ bits for the sublist $\langle x, y \rangle$, in a scaled universe of size $n' = n^x - \sum_{y'=1}^{y-1} n^{x,y'}$. (We use these bounds just for the purpose of bounding the cost of the encoding, as we do not require to follow the approach of Section 3.3.)

- We use σ bits for context x, one bit per sublist. The bit for the sublist $\langle x, y \rangle$ is **1** if and only if $n^{x,y} > n'/2$ and, in this case, we set $t = n' - n^{x,y}$. Otherwise, we set the bit to **0** and $t = n^{x,y}$. Now, $t \leq n'/2$ in both cases (so as we verify the hypothesis of Facts 1 and 2(iii)). Note that given n', t and the corresponding bit, we can recover the value of $n^{x,y}$, as expected.
- We just use the delta code of t, since the value of n' can be inferred recursively from the previous sublists. Note that we also need to encode n^x in this scheme, to be able to start with the decoding of $n^{x,1}$. This motivates the use of Fact 3.
- The delta codes for the sublists yielding $t \leq t_\gamma$ contribute $O(\lg t_\gamma) = O(1)$ bits each, totalizing $O(\sigma \lg t_\gamma) = O(\sigma)$ bits for context x. An analogous argument holds for n^x when $n^x \leq n_\gamma$.
- For the sublists yielding $t > t_\gamma$ (recall that $t \leq n'/2$), we apply first Fact 2(ii) to show that the delta code of t requires less than $\gamma(2t - \lg t - 1)$ bits, and then Fact 1 to show that the latter quantity is upper bounded by $\gamma \lg \binom{n'}{t}$.

Hence, the delta code of $t \geq 1$ requires less than $\delta \lg \binom{n'}{n^{x,y}}$ bits (recall that $\gamma < \delta$). This implies that the sum of these delta codes for context x can be upper bounded by $\delta \lg \binom{n^x}{n^{x,1}, n^{x,2}, \ldots, n^{x,\sigma}} + t^x$.

However, we need the value of n^x to be able to decode $n^{x,1}$ from t, as previously mentioned. Hence, we are left with the evaluation of the term $\lg n^x$ coming from Fact 3, as it is now required in our scheme. This is a subtle point as it can increase significantly the final cost if not analyzed properly. We need to slightly change the scheme discussed above.

- Instead of encoding $n^{x,1}$ as first value in the sequence for context x, we use σ bits to indicate that we encode $n^{x,y}$ as a first value, such that $t = \min\{n^{x,y}, n^x - n^{x,y}\}$ satisfies the condition $\binom{n'}{t} > (n')^{\gamma^{-1}}$ (see Fact 2(iii), where $n' = n^x$, and $t \leq n'/2$). While the overall cost for the sum of the logarithms do not change and give the logarithm of the multinomial coefficient, as observed in Section 3.3, we obtain the further property that $\lg n^x = \lg n' < \gamma \lg \binom{n'}{t}$, the same upper bound on the amount of bits required by the delta code of t. Hence, the code for both n^x and $n^{x,y}$ takes overall $2\gamma \lg \binom{n'}{t} = \delta \lg \binom{n'}{t}$ bits. The rest of the encoding is unchanged.

- If no such $n^{x,y}$ exists, this means that each $t \leq t_\gamma = O(1)$ by Fact 2(iii) and so its encoding takes $O(\lg t_\gamma) = O(1)$ bits. The overall cost for the sublists' lengths of the statistical model is $O(\sigma \lg t_\gamma) = O(\sigma)$ bits. Further, the $\lg n^x$ bits for encoding n^x by Fact 3 can be bounded as $n^x H_0^*(w_x)$ since $n^x H_0^*(w_x) \geq \lg n^x$ in this case. Hence, the total contribution is $n^x H_0^*(w_x) + O(\sigma)$ bits in Lemma 5.

In summary, for each context x, the encoding of the sublists lengths requires $O(\sigma)$ bits plus either $\delta \lg \binom{n^x}{n^{x,1}, n^{x,2}, \ldots, n^{x,\sigma}} + t^x$ or $n^x H_0^*(w_x)$ bits. The former term is simply upper bounded by $\delta n^x H_0^*(w_x) + t^x$ by Theorem 1. We must add further $O(\sigma(\lg t_\gamma + \lg n_\gamma)) = O(\sigma)$ bits for the bitvectors and the small lengths as discussed before. The sum of the bits required by all contexts $x \in P_h^*$ is therefore upper bounded by $nH_h^* + \lg n + O(\sigma^{h+1} \lg \sigma^{h+1})$. Note that we can infer the value of n by using the lengths encoded as given in Fact 3. This completes the proof of Lemma 5.

A final word is on the encoding and decoding time. Subset encoding in $\lg \binom{n'}{t}$ bits requires $O(t)$ operations on integers [Knu05, Rus05]. We can upper bound this cost by $O\big(t(\lg \binom{n'}{t} / \lg n + 1)\big)$ time. Hence, the cost of encoding the sublists is upper bounded by $O\big(\sum_{x \in P_h^*} n^x \big(\lg \binom{n^x}{n^{x,1}, n^{x,2}, \ldots, n^{x,\sigma}} / \lg n + 1\big)\big) = O\big(n(nH_h / \lg n + 1)\big)$. The latter term can be $O(n^2)$ in the worst case but it can be $O(n)$ for low-entropy texts. We also have to add a cost of $O(g_h'')$ for setting up the rest of the encoding (e.g. the empirical statistical model). Decoding has a similar cost.

4 Random Access Compressed Representation of *LF* and Φ

In Section 3, we have described the importance of the LF mapping and the Φ function for compressing the BWT. As we shall see, these functions are also essential to performing compressed text indexing. However, we need more functionality since we need random access to their compressed values with a small cost for decoding. With the techniques discussed so far, computing the ith value of LF or Φ, for $1 \leq i \leq n$, has two major drawbacks:

- We need to decompress all the information, even if we need a single value of LF or Φ.
- The decompression is sequentially performed even though the required access is random.

We circumvent the two drawbacks above by using succinct dictionaries and compressed directories for speeding up the access and avoiding to decompress all the data while keeping the space occupancy entropy-bound. The main contribution of this section is to show how to store LF and Φ in compressed format so that each call decompresses just a small portion of their format:

- Each call takes $O(\lg \sigma)$ time using further $O(n \lg \lg n / \lg_\sigma n) = o(n \lg \sigma)$ bits of space for storing the compressed auxiliary data structures.
- Each call takes $O(1)$ time using further $O(n)$ bits for the compressed auxiliary data structures (i.e. $o(n \lg \sigma)$ bits when σ is not a constant).

We proceed in the rest of the section as follows. In Section 4.1, we describe how to extend the functionalities of the wavelet trees to succinct dictionaries. We then show how to use wavelet trees and some auxiliary data structures to get the random access to the compressed representation of Φ in Section 4.2 and to that of LF in Section 4.3.

4.1 Wavelet Trees as Succinct Dictionaries

Our compressed directories hinge on constant-time *rank* and *select* data structures [Jac89b, Mun96, Pag01, RRR02]. For a bitvector B of size n, the function $rank_{\mathbf{1}}(B, i)$ returns the number of $\mathbf{1}$s in B up to (and including) position i. The function $select_{\mathbf{1}}(B, i)$ returns the position of the ith $\mathbf{1}$ in B. We can also define $rank_{\mathbf{0}}$ and $select_{\mathbf{0}}$ in terms of the $\mathbf{0}$s in B. As previously mentioned in Section 3.3, subset encoding can implicitly represent B as a subset of the elements from $1 \ldots n$, associating

each **1**, say in position j in B, with element j in the subset.[9] Letting t be the number of elements thus implicitly represented (the number of **1**s in the bitvector), we can replace bitvector B supporting $rank_{\mathbf{1}}$ and $select_{\mathbf{1}}$ with the constant-time *indexable dictionaries* developed by Raman, Raman, and Rao [RRR02], requiring $\lceil \lg \binom{n}{t} \rceil + O(t \lg\lg t / \lg t) + O(\lg\lg n)$ bits. As can be seen, the bound of subset encoding, $\lceil \lg \binom{n}{t} \rceil$, has an additional term for the fast-access directories, $O(t \lg\lg t / \lg t) + O(\lg\lg n)$. Moreover, $rank_{\mathbf{1}}(B, i) = -1$ if $B[i] \neq \mathbf{1}$ in indexable dictionaries. If we wish to support the full functionalities of $rank_{\mathbf{1}}$, $select_{\mathbf{1}}$, $rank_{\mathbf{0}}$, and $select_{\mathbf{0}}$, we need to use the *fully-indexable* version of their structure, called an FID.

Theorem 3 (Raman, Raman, and Rao [RRR02]) *An* FID *storing t items out of a universe of n items, requires*

$$\left\lceil \lg \binom{n}{t} \right\rceil + O\left(\frac{n \lg\lg n}{\lg n} \right)$$

bits of space. Each call to $rank_{\mathbf{1}}$, $select_{\mathbf{1}}$, $rank_{\mathbf{0}}$, and $select_{\mathbf{0}}$ takes $O(1)$ time.

Note that the additional term of $O(n \lg\lg n / \lg n)$ in Theorem 3 is related to the universe size n, instead of the subset size t. Analogously to what done with subset encoding, since $\lg \binom{n}{t} \leq n$, we will use FIDs as space-efficient replacements of bitvectors of length n with t **1**s (alternatively, with $n - t$ **0**s) supporting *rank* and *select* on both **0**s and **1**s. [10] In this way, we can successfully reuse part of the analysis given in Section 3.

Let us now consider the wavelet trees as defined in Section 3.4. What we obtain by replacing the subset encodings in the nodes with FIDs, is a generalization of *rank* and *select* operations from binary to σ-ary vectors as discussed next. Let us adopt the notation introduced in Section 3.4, so that s_y denotes the $\langle x, y \rangle$ sublist of $n^{x,y}$ entries, where $1 \leq y \leq t^x$ (recall that $t^x \leq \sigma$ is the number of nonempty sublists for context x and that the symbols from Σ for these sublists are renumbered from 1 to t^x without loss of generality). Recall that the wavelet tree for context x stores the symbols belonging to the contiguous portion of the BWT corresponding to context x and we denote this portion by $w_x = w_x[1 \ldots n^x]$. To make the discussion a bit more general, we define two primitives, where $1 \leq y \leq t^x$ and $1 \leq i \leq n^x$:

- For each symbol y, function $rank'_y(w_x, i)$ returns the number of occurrences of y in w_x up to (and including) position i.
- For each symbol y, function $select'_y(w_x, i)$ returns the position of the ith occurrence of y in w_x.

As can be seen, when $w_x = B$ and $y \in \{\mathbf{0}, \mathbf{1}\}$, we obtain the classical *rank* and *select* operations on bitvectors B. We show how wavelet trees can support $rank'$ and $select'$ efficiently using FIDs.

Lemma 6 *Using a wavelet tree for context x, we can encode the t^x nonempty sublists for that context in fewer than*

$$\lg \binom{n^x}{n^{x,1}, n^{x,2}, \ldots, n^{x,\sigma}} + O\left(t^x + n^x \frac{\lg\lg n^x}{\lg_{t^x} n^x} \right)$$

bits, so that $rank'$ and $select'$ take $O(\lg t^x)$ time.

To begin with, we augment our wavelet tree by replacing the t-subset encoding of [Knu05, Rus05] with the FID structure from [RRR02]. To resolve query $select'_y(w_x, i)$ on our new wavelet tree for w_x, we follow these steps.

▷ $select'_y(w_x, i)$:

1. Set $s = s_y$.
2. If s is the left child, search for the ith **0** in s's parent dictionary: set $i = select_{\mathbf{0}}(i)$.

[9] Note that ranking/unranking a subset refers to the lexicographic generation of subsets mentioned in Section 3.3, not to be confused with the *rank* function defined here.

[10] In this paper, we write $rank(i)$ or $select(i)$ to denote the appropriate function on **1**s when there is no confusion.

3. If s is the right child, search for the ith $\mathbf{1}$ in s's parent dictionary: set $i = \mathit{select}_{\mathbf{1}}(i)$.
4. Set $s = \mathit{parent}(s)$.
5. Recurse to step 2, unless $s = \mathit{root}$.
6. Return i as the answer to the query select' in sublist s_y.

This query trivially requires $O(\lg t^x)$ time since *select* takes constant time and the depth of the wavelet tree is $O(\lg t^x)$ as shown in Lemma 3. The other query can be performed analogously.

▷ $\mathit{rank}'_y(w_x, i)$:

1. Set $s = \mathit{root}$.
2. If s_y is a descendent of the left child, set $i = \mathit{rank}_{\mathbf{0}}(i)$ in s's dictionary.
3. If s_y is a descendent of the right child, set $i = \mathit{rank}_{\mathbf{1}}(i)$ in s's dictionary.
4. Set $s =$ the child of s that is ancestor of leaf s_y.
5. Recurse to step 2, unless $s = s_y$.
6. Return i as the answer to the query rank' in sublist s_y.

This query also requires $O(\lg t^x)$ time. The space analysis of the new wavelet tree is similar to that of the unaugmented wavelet tree in Lemma 3, except that we must sum the costs of the lower-order terms for the FIDs. Specifically, there are $O(\lg t^x)$ levels in the wavelet tree and, for each such level, there are universe sizes $u_1, u_2, \ldots, u_r$, such that $r < t^x$ and $\sum_{j=1}^{r} u_j \leq n^x$. Each FID gives an extra contribution of at most $cu_j \lg\lg u_j / \lg u_j$ bits to the analysis in Lemma 3, for a constant $c > 0$. For a given level in the wavelet tree, we claim that the additional number of bits is

$$\sum_{j=1}^{r} cu_j \lg\lg u_j / \lg u_j = O(n^x \lg\lg n^x / \lg n^x). \tag{19}$$

Hence, we get a total of $O(n^x \lg\lg n^x / \lg_{t^x} n^x)$ bits of space for all the levels. In order to prove our claim (19), first note that there exists a constant $\kappa_0 > 1$ such that the function $f(\kappa) = \kappa \lg\lg \kappa / \lg \kappa$ is concave for any $\kappa > \kappa_0$. We then split the sum in equation (19) in two parts. The first part involves the terms such that $u_j \leq \kappa_0$, giving a total contribution of $O(r)$, since κ_0 is constant with respect to n^x and r, the number of nonempty sublists in the given level of the wavelet tree. The second part involves only the terms such that $u_j > \kappa_0$, for which the concavity of $f(\kappa)$ holds. Multiplying by r/r and applying Jensen's inequality [CT91], we obtain

$$\frac{r}{r} \times \sum_{j=1}^{r} c \frac{u_j \lg\lg u_j}{\lg u_j} = O\left(r \times \frac{(\sum_{j=1}^{r} \frac{u_j}{r}) \lg\lg(\sum_{j=1}^{r} \frac{u_j}{r})}{\lg(\sum_{j=1}^{r} \frac{u_j}{r})} \right) = O\left(\frac{n^x \lg\lg \frac{n^x}{r}}{\lg \frac{n^x}{r}} \right).$$

Note the sum over the r values on all levels of the wavelet tree is $t^x - 1$ (i.e. the number of internal nodes), so that the total is $O(t^x + n^x \lg\lg n^x / \lg_{t^x} n^x)$ additional bits, thus completing the proof of Lemma 6. This term seems difficult to improve due to strong evidence from Miltersen [Mil05]. In the following, when we invoke the *rank* and *select* operations, we specify the dictionary they refer to unless this is clear from the context.

4.2 Getting Random Access to the Compressed Representation of Φ

We now describe how to store, in compressed format, the Φ function described in Section 3.1, so as we can quickly compute any value $\Phi(i)$, for $1 \leq i \leq n$, by decompressing a small portion of the format. We employ the conceptual table $\mathcal{T}$ described in Section 3.2, and adopt $\mathcal{T}$'s encoding for the BWT given at the end of Section 3.2, except that the wavelet trees are now augmented with FIDs as discussed in Section 4.1 (cf. Theorem 3). Recall that in order to support a query for $\Phi(i)$, we need to decompress the ith nonempty entry in the concatenation in column major order of the sublists in $\mathcal{T}$. (We refer to Table 5 for an example.) We need the following basic information: the

list y containing entry $\Phi(i)$; the context x such that the $\langle x, y\rangle$ sublist contains $\Phi(i)$; the element z stored explicitly in the *normalized* $\langle x, y\rangle$ sublist (see Table 5); the number of elements $\#x$ in all contexts prior to x. In the example for $\Phi(2) = 10$, we have $y = \texttt{i}$, $x = \texttt{s}$, $\#x = 7$, and $z = 3$. The value for $\Phi(i)$ is then $\#x + z$ because of the normalization of the sublists described in Section 3.2. We execute five main steps to answer a query.

▷ *Query* $\Phi(i)$:

1. Consult a directory G to determine $\Phi(i)$'s list y and the number of elements in all prior lists, $\#y$. (We now know that $\Phi(i)$ is the $(i - \#y)$th element in list y.) In the example above, we consult G to find $y = \texttt{i}$ and $\#y = 0$.
2. Consult a list L^y to determine the context x of the $(i-\#y)$th element in list y. For example, we consult $L^{\texttt{i}}$ to determine $x = \texttt{s}$. We identify the $\langle x, y\rangle$ sublist and $\#p$, the number of entries in previous sublists $\langle x, y'\rangle$ with $y' < y$.
3. Look up the appropriate entry in $\langle x, y\rangle$ to find z. This entry occupies position $i - \#y - \#p$ inside $\langle x, y\rangle$; hence, $z = \mathit{select}'_y(i - \#y - \#p)$ for context x. In the example, we look for the first entry in the $\langle \texttt{s}, \texttt{i}\rangle$ sublist and determine $z = 3$.
4. Consult a directory F to determine $\#x$, the number of elements in all prior contexts. In the example, after looking at F, we determine $\#x = 7$.
5. Return $\#x + z$ as the solution to $\Phi(i)$. The example would then return $\Phi(i) = \#x + z = 7 + 3 = 10$.

We now detail some of the steps given above, describing the set of auxiliary data structures.

4.2.1 Directories G and F

We describe the details of the directory G (and the analogous structure F), which determines $\Phi(i)$'s list y and the number of elements in all prior lists $\#y$. We can think of G conceptually as a bitvector of length n. For each nonempty list y (considered in lexicographical order) containing $n^y = \sum_{x \in P_h^*} n^{x,y}$ elements (where P_h^* is the optimal prefix cover defined in Section 2), we write a **1**, followed by $(n^y - 1)$ **0**s. Intuitively, each **1** represents the first element of a list. Since there are as many **1**s in G as nonempty lists, G cannot have more than $l = \sigma$ **1**s. To retrieve the desired information in constant time, we compute $y = \mathit{rank}(G, i)$ and $\#y = \mathit{select}(G, y) - 1$. The F directory is similar, where each **1** denotes the start of a context x (considered in lexicographical order), rather than the start of a list, followed by $(n^x - 1)$ **0**s. Since there are at most $c = |P_h^*| \le \sigma^h$ possible contexts, we have at most that many **1**s. We use FIDs to store these directories.

Lemma 7 *We can store G using*

$$\left\lceil \lg \binom{n}{l} \right\rceil + O\left(\frac{n \lg\lg n}{\lg n}\right) = O\left(\sigma \lg\left(1 + \frac{n}{\sigma}\right) + \frac{n \lg\lg n}{\lg n}\right)$$

bits of space, and F using space

$$\left\lceil \lg \binom{n}{c} \right\rceil + O\left(\frac{n \lg\lg n}{\lg n}\right) = O\left(|P_h^*| \lg\left(1 + \frac{n}{|P_h^*|}\right) + \frac{n \lg\lg n}{\lg n}\right).$$

4.2.2 List-Specific Directory L^y

Once we know which list y our query $\Phi(i)$ is in, we must find its context x. We create a directory L^y for each list y, exploiting the fact that the entries are grouped into $\langle x, y\rangle$ sublists as follows. We can think of L^y conceptually as a bitvector of length n^y, the number of items indexed in list y. For each nonempty $\langle x, y\rangle$ sublist (in lexicographical order by x) containing $n^{x,y}$ elements, we write a **1**, followed by $(n^{x,y} - 1)$ **0**s. Intuitively, each **1** represents the first element of a sublist. Since

there are as many **1**s in L^y as nonempty sublists in list y, that directory cannot have more than $\min\{|P_h^*|, n^y\}$ **1**s. Directory L^y is made up of two distinct components:

The first component is a FID that produces a nonempty context number $p > 0$. In the example, the same context $x =$ `p` has $p = 1$ in list `i` while has $p = 2$ in list `p`. It also produces the number $\#p$ of items in all prior sublists. In the example, context $x =$ `p` has $\#p = 0$ in list `i`, and $\#p = 1$ in list `p`. To retrieve the desired information in constant time, we compute $p = rank(L^y, i - \#y)$ and $\#p = select(L^y, p) - 1$.

In order to save space, we actually store a single directory shared by all lists y. For each list y, we can retrieve the list's p and $\#p$ values. Conceptually, we represent this global directory L as a simple concatenation (in lexicographical order by y) of the list-specific bitvectors described above. The only additional information we need is the starting position of each of the above bitvectors, which is easily obtained by computing $start = \#y$. We compute $p = rank(i) - rank(start)$ and $\#p = select(p + rank(start)) - start - 1 = select(rank(i)) - start - 1$. We implement L by a single FID storing s entries in a universe of size n, where $s = \sum_{x \in P_h^*} t^x$ is the number of nonempty sublists.

Lemma 8 *We can compute p and $\#p$ in constant time, and the space used (in bits) is*

$$\left\lceil \lg \binom{n}{s} \right\rceil + O\left(\frac{n \lg\lg n}{\lg n}\right) = O\left(s \lg\left(1 + \frac{n}{s}\right) + \frac{n \lg\lg n}{\lg n}\right).$$

The second component maps p, the local context number for list y, into the global one x. Since there are at most $|P_h^*|$ different contexts x for nonempty sublists $\langle x, y\rangle$ and at most σ nonempty lists y, we use the concatenation of σ bitvectors of $|P_h^*|$ bits each, where bitvector b^y corresponds to list y and its **1**s correspond to the nonempty sublists of list y. We represent the concatenation of bitvectors b^y (in lexicographical order by y) using a single FID. Mapping a value p to a context x for a particular list y is equivalent to identifying the position of the pth **1** in b^y. This can be done by a constant number of *rank* and *select* queries.

Lemma 9 *We can map p to x in constant time, and the space used (in bits) is*

$$\left\lceil \lg \binom{|P_h^*|\sigma}{s} \right\rceil + O\left(\frac{(|P_h^*|\sigma) \lg\lg(|P_h^*|\sigma)}{\lg(|P_h^*|\sigma)}\right) = o\left(\sigma^{h+1}\right).$$

4.2.3 Time and Space Complexity

Theorem 4 *The Φ function can be represented in compressed format for a text of n symbol over the alphabet Σ using $nH_h + O(n \lg\lg n / \lg_\sigma n) + g'_h \lg(1 + n/g'_h)$ bits of space, where $g'_h = O(\sigma^{h+1})$, so that each call to Φ takes $O(\lg \sigma)$ time.*

Proof: The space occupancy is that indicated by Theorem 2, except that Lemma 3 should be replaced by Lemma 6 plus the additional terms indicated in Lemma 7, Lemma 8 and Lemma 9, where $s = \sum_{x \in P_h^*} t^x$ is bounded by g'_h. The time cost is constant except for the wavelet tree, as stated in Lemma 6, where $t^x \le \sigma$. □

Theorem 5 *The Φ function can be represented in compressed format using $nH_h + O(n) + g'_h \lg(1 + n/g'_h)$ bits of space, so that each call to Φ takes $O(1)$ time.*

Proof: The proof is analogous to that of Theorem 4, except that for each context x, the wavelet tree is replaced by a set of t^x *indexable dictionaries* [RRR02] representing sublists $\langle x, y\rangle$ for $1 \le y \le t^x$ with $\lceil \lg \binom{n^x}{n^{x,y}} \rceil + O(n^{x,y} \lg\lg n^{x,y} / \lg n^{x,y})$ bits (since we only need *select* operations on them, there is no need to use an FID). When we need to perform $select'_y$ for context x, we just run the *select* operation on the indexable dictionary for $\langle x, y\rangle$. By Lemma 1, using indexable dictionaries adds a

term that sums up to $O(n)$ in the bound of Theorem 4, but it requires to perform only a number of $O(1)$ constant-time queries to a single dictionary, totalizing $O(1)$ time. This scheme may pay when σ is not a constant, since it requires additional $O(n) = o(n \lg \sigma)$ bits of space for the auxiliary data structures. □

4.3 Getting Random Access to the Compressed Representation of *LF*

The machinery for the compressed representation of Φ can be reused also for the LF mapping. In [FM05], it is shown that $LF(i) = C[L[i]] + Occ(i, L[i])$ for any $1 \leq i \leq n$. Here, for any $y \in \Sigma$, vector $C[y]$ counts the number of occurrences of symbols $y' < y$ appearing in the text T, and $Occ(i, y)$ is the number of occurrences of y appearing in the first i positions of the BWT (here it is identified with L). It turns out that, given i, we can compute the context x and the list $y = L[i]$ as described for *Query* $\Phi(i)$ in Section 4.2. Then, we can obtain $Occ(i, y)$ as the value of $rank'_y(i - \#y - \#p) + \#y$ for context x. The following are corollaries of Theorems 4 and 5.

Corollary 1 *The LF function can be represented in compressed format for a text of n symbol over the alphabet Σ using $nH_h + O(n \lg \lg n / \lg_\sigma n) + g'_h \lg(1 + n/g'_h)$ bits of space, where $g'_h = O(\sigma^{h+1})$, so that each call to LF takes $O(\lg \sigma)$ time.*

In particular, we note that in Corollary 2, we can use the indexable dictionaries since we invoke $rank(i)$ for a suitable sublist $\langle x, y \rangle$, such that $y = L[i]$, the ith symbol in the BWT. This corresponds to the weak form of *rank* supported by indexable dictionaries.

Corollary 2 *The LF function can be represented in compressed format using $nH_h + O(n) + g'_h \lg(1 + n/g'_h)$ bits of space, so that each call to LF takes $O(1)$ time.*

5 Using the Framework for Compressed Suffix Arrays

In this section, we use the machinery developed so far to achieve text indexing, showcasing the insights we obtained in our prior investigation. In the remainder of this paper, we will detail the results of our CSA, though analogous methods hold for the FM-index implemented with the wavelet tree. In fact, there are a whole host of methods now that use Φ or the LF mapping (see the survey in [NM05]).

5.1 Compressed Suffix Arrays (CSAs)

To recap, a standard suffix array [GBS92, MM93] is an array containing the position of each of the n suffixes of text T in lexicographical order. In particular, $SA[i]$ is the starting position in T of the ith suffix in lexicographical order, $T[SA[i], n]$. The size of a suffix array is $\Theta(n \lg n)$ bits, as each of the positions stored uses $\lg n$ bits. A suffix array allows constant time *lookup* to $SA[i]$ for any i. In order to achieve self-indexing, we also use the notion of the *inverse* suffix array SA^{-1}, such that $SA^{-1}[j] = i$ if and only if $SA[i] = j$. In other words, $SA^{-1}[j]$ gives the rank in the lexicographic order of suffix $T[j, n]$ among the suffixes of T.

The CSA contains the same information as a standard (inverse) suffix array, though it operates only on a compressed format. For the rest of the paper, we assume that the CSA supports the following set of operations as given in [GV05, Sad03, Sad02].

Definition 2 Given a text T of length n, a *compressed suffix array* (CSA) for T supports the following operations without requiring explicit storage of T or its (inverse) suffix arrays, SA, SA^{-1}:

- *compress*(T) produces a compressed representation, Z, that encodes (i) text T, (ii) its suffix array SA, and (iii) its inverse suffix array SA^{-1};

- $lookup_Z(i)$ returns the value of $SA[i]$, the position of the ith suffix in lexicographical order, for $1 \leq i \leq n$;
- $lookup_Z^{-1}(j)$ returns the value of $SA^{-1}[j]$, the rank of the jth suffix in T, for $1 \leq j \leq n$;
- $substring_Z(i, c)$ decompresses the first c symbols (a prefix) of the suffix $T\big[SA[i], n\big]$, for $1 \leq i \leq n$ and $1 \leq c \leq n - SA[i] + 1$.

We drop some of the parameters from the operations listed in Definition 2 whenever their usage is clear from the context. For example, if we wish to decompress the $c = 6$ symbols belonging to the text substring $T[18, 25]$, we indicate the corresponding operations as follows. First we find the lexicographic position, $lookup^{-1}(18) = 16$, of its corresponding suffix and then we execute $substring(16, c)$.

The structure of a CSA is recursive in nature, where each of the $\ell = \lg \lg_\sigma n$ levels indexes half the elements of the previous level. Hence, the kth level indexes $n_k = n/2^k$ elements. We review and use this recursive decomposition given below:[11]

1. Start with $SA_0 = SA$, the suffix array for text T.
2. For each $0 \leq k < \ell$, transform SA_k into a more succinct representation through the use of a bitvector B_k, function $rank(B_k, i)$, neighbor function Φ_k, and SA_{k+1} (representing the recursion).
3. The final level, $\ell = \lg \lg_\sigma n$ is written explicitly.

SA_k is not explicitly stored (except at the last level ℓ), but we refer to it for the sake of explanation. B_k is a bitvector such that $B_k[i] = \mathbf{1}$ if and only if $SA_k[i]$ is even. Even-positioned suffixes are represented in SA_{k+1} with their positions divided by 2. In order to retrieve odd-positioned suffixes, we employ the neighbor function Φ_k, which maps a position i in SA_k containing the value p into the position j in SA_k containing the value $p + 1$. In words, Φ_k is the Φ function described in Section 3.1 applied to SA_k instead of SA. Hence, we can equivalently describe Φ_k by the following formula (also handling the case when $SA_k[i] = n$):

$$\Phi_k(i) = \left\{ \; j \quad \text{such that} \quad SA_k[j] = (SA_k[i] \bmod n) + 1 \; \right\}.$$

A *lookup* for $SA_k[i]$ can be answered in the following way:

$$SA_k[i] = \begin{cases} 2 \cdot SA_{k+1}\big[rank(B_k, i)\big] & \text{if } B_k[i] = \mathbf{1} \\ SA_k\big[\Phi_k(i)\big] - 1 & \text{if } B_k[i] = \mathbf{0}. \end{cases}$$

An example of the recursion for a text T is given below, where `a` < `b` < `#` and `#` is a special end-of-text symbol. (The text T is borrowed from [GV05], but note that the Φ_k function is used instead.) No further levels are needed, since the four suffix array pointers at level 3 are stored explicitly.

	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*	*16*	*17*	*18*	*19*	*20*	*21*	*22*	*23*	*24*	*25*	*26*	*27*	*28*	*29*	*30*	*31*	*32*
T:	a	b	b	a	b	b	a	b	b	a	b	b	a	b	a	a	a	b	a	b	a	b	b	a	b	b	b	a	b	b	a	#
SA_0:	15	16	13	17	19	10	7	4	1	21	28	24	31	14	12	18	9	6	3	20	27	23	30	11	8	5	2	26	22	29	25	32
B_0:	**0**	**1**	**0**	**0**	**0**	**1**	**0**	**1**	**0**	**0**	**1**	**1**	**0**	**1**	**1**	**1**	**0**	**1**	**0**	**1**	**0**	**0**	**1**	**0**	**1**	**0**	**1**	**1**	**1**	**0**	**0**	**1**
$rank(B_0, i)$:	0	1	1	1	1	2	2	3	3	3	4	5	5	6	7	8	8	9	9	10	10	10	11	11	12	12	13	14	15	15	15	16
Φ_0:	2	4	14	16	20	24	25	26	27	29	30	31	32	1	3	5	6	7	8	10	11	12	13	15	17	18	19	21	22	23	28	9

	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*	*16*
SA_1:	8	5	2	14	12	7	6	9	3	10	15	4	1	13	11	16
B_1:	**1**	**0**	**1**	**1**	**1**	**0**	**1**	**0**	**0**	**1**	**0**	**1**	**0**	**0**	**0**	**1**
$rank(B_1, i)$:	1	1	2	3	4	4	5	5	5	6	6	7	7	7	7	8
Φ_1:	8	7	9	11	14	1	6	10	12	15	16	2	3	4	5	13

[11] We use the neighbor function Φ_k to emphasize its importance to our methods; for the full level approach, Grossi and Vitter use the partial function Ψ_k in their exposition.

	1	*2*	*3*	*4*	*5*	*6*	*7*	*8*
SA_2:	4	1	7	6	3	5	2	8
B_2:	**1**	**0**	**0**	**1**	**0**	**0**	**1**	**1**
$rank(B_2, i)$:	1	1	1	2	2	2	3	4
Φ_2:	6	7	8	3	1	4	5	2

	1	*2*	*3*	*4*
SA_3:	2	3	1	4

Here, $\Phi_0(4) = 16$, since $SA_0[4] = 17$ and $SA_0[16] = 17 + 1 = 18$. For this example, suppose we already know SA_1. To retrieve $SA_0[16]$, since $B_0[16] = \mathbf{1}$, we compute $2 \cdot SA_1[rank(B_0, 16)] = 2 \cdot SA_1[8] = 2 \cdot 9 = 18$. To retrieve $SA_0[4]$, since $B_0[4] = \mathbf{0}$, we compute $SA_0[\Phi_0(4)] - 1 = SA_0[16] - 1 = 18 - 1 = 17$.

The CSA has two incarnations which show some inherent tradeoffs of space versus time. The first (time-efficient) version reduces the space requirement to $O(n \lg \sigma \lg \lg_\sigma n)$ bits, while *lookup* takes only $O(\lg \lg_\sigma n)$ time. This version explicitly uses the recursive structure explained above. The second (space-efficient) version skips all but a constant fraction ϵ of these levels, for some $0 < \epsilon \leq 1$, relying on a succinct dictionary D_k to perform the task of B_k, but instead mapping elements several levels away. This scheme reduces the space requirement to $O(\epsilon^{-1} n \lg \sigma)$, however *lookup* now takes $O(\lg^\epsilon_\sigma n)$ time. In practice, the second scheme is much better, as the slowdown in searching is reasonable. We remark that Sadakane [Sad03] has shown that the space complexity can be restated in terms of the order-0 entropy $H_0 \leq \lg \sigma$, giving as a result $O(\epsilon^{-1} H_0 n)$ bits.

In order to compress SA^{-1} along with SA, it suffices to keep SA^{-1}_ℓ in the last level ℓ, as the rest of the machinery for compressing SA and SA^{-1} is identical [Sad03, Sad02]. Hence the cost of $lookup^{-1}$ is the same as that for *lookup*, and it suffices to discuss the latter only. Moreover, it is not difficult to extend the *substring* operation using Φ_k for any value of k, such that each application of Φ_k decompresses $\Theta(2^k)$ symbols at a time, for a total cost of $O(c/2^k)$ time plus the cost of a *lookup*. We use the inverse suffix array and this extended version of *substring* in Section 6.

5.2 High-Order Entropy-Compressed Suffix Arrays

We consider the task of attaining entropy bounds for the usage of space in the CSA by using our unified algorithmic framework for Φ_k at each level k, which contributes the bulk of the space that the CSA uses. In the rest of this section, we prove the tradeoffs shown in Table 1 for the space and time complexity of a CSA and its supported operations as given in Definition 2.

Theorem 6 (Time-Efficient Entropy-Compressed Suffix Arrays) *Implementing a* CSA *uses* $nH_h \lg \lg_\sigma n + O\big(n(\lg \lg \lg_\sigma n / \lg \lg_\sigma n + \lg \lg n / \lg_\sigma n + \lg \sigma / \lg_\sigma n) + \sigma^h(n^\beta + \sigma)\big)$ *bits and* $O\big(n \lg \sigma + \sigma^h(n^\beta + \sigma)\big)$ *preprocessing time for compress, for any arbitrarily small constant* $0 < \beta < 1$. *(The space increases to* $O(n) = o(n \lg \sigma)$ *when* σ *is non-constant.) Each lookup takes* $O(\lg \lg_\sigma n)$ *time and each substring call for* c *symbols takes the cost of lookup plus* $O(c / \lg_\sigma n)$ *time.*

It is worth noting that the space bound in Theorem 6 is $nH_h \lg \lg_\sigma n + o(n \lg \sigma)$ bits when $h + 1 \leq \alpha \lg_\sigma n$ for any arbitrary positive constant $\alpha < 1$ (we fix β such that $\alpha + \beta < 1$). [12] When $\lg \sigma = \Theta(\lg n)$, the space bound reduces to $O(nH_h) + o(n \lg \sigma)$ bits and *lookup* time is $O(1)$. A better space usage can be obtained with the following tradeoff.

Theorem 7 (Space-Efficient Entropy-Compressed Suffix Arrays) *Implementing a* CSA *uses* $\epsilon^{-1} nH_h + O\big(n \lg \lg n / \lg^\epsilon_\sigma n + \sigma^h(n^\beta + \sigma)\big)$ *bits and* $O\big(n \lg \sigma + \sigma^h(n^\beta + \sigma)\big)$ *preprocessing time for compress, for any arbitrarily small constants* $0 < \beta < 1$ *and* $0 < \epsilon \leq 1/2$. *Each lookup takes* $O\big((\lg_\sigma n)^{\epsilon/1-\epsilon} \lg \sigma\big)$ *time and each substring call for* c *symbols takes the cost of lookup plus* $O(c / \lg_\sigma n)$ *time.*

[12] The assumption on $h + 1 \leq \alpha \lg_\sigma n$ is reasonable since Luczak and Szpankowski [LS97] show that the average phrase length of the Lempel-Ziv encoding for ergodic sources is $O(\lg n)$ bits.

For an alphabet of non-constant size, it can be useful to use a corollary of Theorem 7:

Corollary 3 (Space-Efficient Entropy-Compressed Suffix Arrays) *Implementing a* CSA *uses* $\epsilon^{-1}nH_h + O(n + \sigma^h(n^\beta + \sigma))$ *bits and* $O\big(n \lg \sigma + \sigma^h(n^\beta + \sigma)\big)$ *preprocessing time for compress, for any arbitrarily small constants* $0 < \beta < 1$ *and* $0 < \epsilon \leq 1/2$. *Each lookup takes* $O\big((\lg_\sigma n)^{\epsilon/1-\epsilon}\big)$ *time and each substring call for* c *symbols takes the cost of lookup plus* $O(c/\lg_\sigma n)$ *time.*

Again, note that the space bound in Theorem 7 and Corollary 3 is $\epsilon^{-1}nH_h + o(n \lg \sigma)$ when $h+1 \leq \alpha \lg_\sigma n$ for $\sigma = \omega(1)$ and any arbitrary positive constant $\alpha < 1$ (we fix β such that $\alpha+\beta < 1$). A special case gives the best space bound in this paper:

Theorem 8 (Nearly Space-Optimal Entropy-Compressed Suffix Arrays) *Implementing a* CSA *uses* $nH_h + O\big(n \lg\lg n/\lg_\sigma n + \sigma^h(n^\beta + \sigma)\big)$ *bits and* $O(n \lg \sigma + \sigma^h(n^\beta + \sigma))$ *preprocessing time for compress, for any arbitrarily small constant* $0 < \beta < 1$. *Each lookup takes* $O(\lg^2 n/\lg\lg n)$ *time and each substring call for* c *symbols takes the cost of lookup plus* $O(c \lg \sigma)$ *time.*

The CSA in Theorem 8 is a nearly space-optimal self-index in that it uses the same space as the *compressed* text—nH_h bits—plus the lower-order terms for the text indexing directories. For example, we get $nH_h + O(n \lg\lg n/\lg n)$ bits when $\sigma = O(1)$ and $h + 1 \leq \alpha \lg_\sigma n$ for any arbitrary constant $\alpha < 1$ (we fix β such that $\alpha+\beta < 1$). All space bounds mentioned above include implicitly the cost $M(T, \Sigma, h)$ of the statistical model, which is dominated by the other lower-order terms.

5.2.1 Compressed Representation of the Neighbor Function Φ_k

We now show how to obtain entropy bounds for implementing Φ_k at each level k of a CSA. We refer to the machinery discussed for the implementation of Φ in Section 4.2. Since $\Phi = \Phi_0$ and $n = n_0$, we can use either of Theorems 4 and 5 for level $k = 0$. Hence we restrict our focus on level $k > 0$, for which we are interested in extending the bounds of Theorem 5. We introduce some useful notation to this end. We denote the number of elements at level k by $n_k = n/2^k$, and the number of elements at level k that are in context x by n_k^x. Similarly, we define n_k^y as the number of elements at level k in list y; and $n_k^{x,y}$ as the number of elements at level k that are in both context x and list y, that is, the size of sublist $\langle x, y\rangle$. Note that $n_k = \sum_x n_k^x = \sum_y n_k^y = \sum_{x,y} n_k^{x,y}$.

Lemma 10 *For any level* k, *the* Φ_k *function can be represented in compressed format using* $nH_h + O(n_k + \sigma^{2^k+h})$ *bits of space, so that each call to* Φ_k *takes* $O(1)$ *time.*

Proof: We conceptually partition the symbols of the text T into $n/2^k$ non-overlapping segments of 2^k symbols each, assuming without loss of generality that n is a multiple of 2^k. We refer to each segment as a "meta-symbol" and we can regard the text T as a new text T_k consisting of $n/2^k$ meta-symbols over the alphabet $\Sigma' = \Sigma^k$. (These meta-symbols are precisely those corresponding to the Σ' lists at level k. We still draw contexts of length h from the original text T.) Note that SA_k is the suffix array for T_k and Φ_k is the corresponding Φ function at level k. Consequently, we can implement Φ_k along the lines described in Section 4.2. However, a direct application of Theorem 5 to T_k for the analysis of the space usage requires some observations to obtain the claimed bounds.

First, we need to refine the analysis by reviewing the space complexity of the auxiliary data structures in Section 4.2, indexing them by k to denote their use at level k. Directories G_k and F_k require $O(n_k)$ bits of space by Lemma 7 (where $l = l_k, n = n_k$), using the fact that $\lg \binom{a}{b} \leq a$. Directories L_k^y, for all lists y at level k, occupy a total of $O(n_k + \sigma^{2^k+h})$ bits by Lemma 8 and Lemma 9 (where $n = n_k$ and $s \leq n_k$ is an upper bound on the number of sublists at level k).

Second, we need to relate the high-order entropy of T_k with H_h in our analysis. The current T_k is built on all of the *even* text positions of T_{k-1}. Similarly, there is also text built on *odd* positions.

Let $T_k^e = T_k$ and T_k^o denote the two different ways of merging every two symbols of T_{k-1}. When reflected to T, note that T_k^o and T_k^e overlap in T except for $O(2^k)$ initial or final symbols in T. Hence, they essentially encode the same information. We bound the entropy of T_k^o and T_k^e together, showing that their total entropy is no more than $nH'_h + nH'_{h+1}$ bits, which can be bounded by $2nH_h$ by Theorem 1. Hence, representing any of the two requires at most $nH_h + O(2^k \lg \sigma)$ bits, proving the lemma (since we need that bound for T_k^e). For the sake of clarity, let $n_o^{x,yz}$ denote the number of occurrences in T_k^o of the concatenated sequence yzx, where $y, z \in \sigma^{2^k}$ and $x \in P_h^*$ (where $n_o^{x,yz} = 0$ when x is not aligned to a position of T_k^o reflected in T). We similarly define $n_e^{x,yz}$ for T_k^e. Then, their entropy is

(20)
$$nH'_h(T_k^o) + nH'_h(T_k^e) = \sum_{x \in P_h^*} \lg \binom{n_o^x}{n_o^{x,11}, n_o^{x,12}, \ldots, n_o^{x,\sigma^{2^k}\sigma^{2^k}}} + \sum_{x \in P_h^*} \lg \binom{n_e^x}{n_e^{x,11}, n_e^{x,12}, \ldots, n_e^{x,\sigma^{2^k}\sigma^{2^k}}}$$

Using equation (15), we separate the terms in (20) fully into a product of binomial coefficients with $\sigma^{2^{k+1}}$ total terms. Then, since $\binom{a}{b}\binom{c}{d} \le \binom{a+c}{b+d}$ for all positive $a \ge b, c \ge d$, we simplify by combining the respective terms in (20) to get

$$\begin{aligned}
\sum_{x \in P_h^*} \lg \binom{n^x}{n^{x,11}, n^{x,12}, \ldots, n^{x,\sigma^{2^k}\sigma^{2^k}}} &= \sum_{x \in P_h^*} \lg \left(\frac{n^x!}{\prod_{y,z \in \sigma^{2^k}} n^{x,yz}!} \right) \\
&= \sum_{x \in P_h^*} \lg \left(\frac{n^x!}{\prod_{y,z \in \sigma^{2^k}} n^{x,yz}!} \right) \left(\frac{\prod_{z \in \sigma^{2^k}} n^{x,z}!}{\prod_{z \in \sigma^{2^k}} n^{x,z}!} \right) \\
&= \sum_{x \in P_h^*} \lg \left(\frac{n^x!}{\prod_{z \in \sigma^{2^k}} n^{x,z}!} \right) \left(\frac{\prod_{z \in \sigma^{2^k}} n^{zx}!}{\prod_{y,z \in \sigma^{2^k}} n^{zx,y}!} \right) \\
&= nH'_h + nH'_{h+1}
\end{aligned}$$

by the definition of high-order empirical entropy H'_h from equation (10) and multinomial coefficients. □

5.2.2 Bounds for the Entropy-Compressed Suffix Array

We have almost all of the pieces we need to prove Theorems 6–8 for CSA. We begin with the proof of Theorem 6. We define $\ell = \lg \lg_\sigma n$ to be the last level in the CSA, as given in Section 5.1. We introduce a special level $\ell' = \ell - O(1)$, such that $\sigma^{2^{\ell'}} = O(n^\beta)$ for any arbitrary constant $0 < \beta < 1$. Our choice of ℓ' implies that $2^{\ell'} = \Theta(\lg_\sigma n)$ and $2^{\ell - \ell'} = O(1)$.

Instead of storing all levels as discussed in Section 5.1, we only store the levels $k = 0, \lg \ell', \lg \ell' + 1, \lg \ell' + 2, \ldots, \ell' - 1, \ell'$ of the recursion in the CSA. (Notice the gap between 0 and $\lg \ell'$, and the gap between ℓ' and ℓ.) For each of these levels up to ℓ', we store a bitvector B_k and a neighbor function Φ_k as described in Section 5.1, with their space detailed in the points below:

1. Bitvector B_0 stores $n_{\lg \ell'}$ entries out of a universe of size n, implemented as an indexable dictionary [RRR02] using $O(n_{\lg \ell'} \lg(n/n_{\lg \ell'})) = O(n \lg \lg \lg_\sigma n / \lg \lg_\sigma n)$ bits. For $\lg \ell' \le k \le \ell' - 1$, bitvector B_k stores $n_k/2$ entries out of a universe of size n_k, implemented as an indexable dictionary requiring $O(n_k)$ bits. Hence, the total contribution is $O(n \lg \lg \lg_\sigma n / \lg \lg_\sigma n)$ bits.
2. Neighbor function Φ_k is implemented as described in Section 5.2.1. The space bounds are stated in Theorems 4–5 when $k = 0$, namely, either $nH_h + O(n \lg \lg n / \lg_\sigma n) + g'_h \lg(1 + n/g'_h)$ or $nH_h + O(n) + g'_h \lg(1 + n/g'_h)$ bits of space, where $g'_h = O(\sigma^{h+1})$. For $k > 0$, we use Lemma 10, which gives $\sum_{k=\lg \ell'}^{\ell'} (nH_h + O(n_k + \sigma^{2^k + h})) < nH_h(\lg \lg_\sigma n - 1) + O(n_{\lg \ell'} + \sigma^{2^{\ell'} + h})$ bits, where the second term can be bounded as $O(n_{\lg \ell'} + \sigma^{2^{\ell'} + h}) = O(n / \lg \lg_\sigma n + \sigma^h n^\beta)$.

3. Level $k = \ell$ should explicitly store the suffix array SA_ℓ and the inverted suffix array SA_ℓ^{-1}, according to what we described in Section 5.1. To significantly reduce the space usage, we now store the arrays at level $\ell + \lg t(n)$ where we fix $t(n) = \lg \lg_\sigma n$. Hence we store $SA_{\ell+\lg t(n)}$, $SA^{-1}_{\ell+\lg t(n)}$, along with an array $LCP_{\ell+\lg t(n)}$ for the longest common prefix information [MM93] to allow fast searching in $SA_{\ell+\lg t(n)}$, with a total space contribution of $O(n \lg \sigma / \lg \lg_\sigma n)$ bits for level $\ell + \lg t(n)$.

Summing up the bounds in points 1–3, we obtain $nH_h \lg \lg_\sigma n + O\big(n\big(\lg \lg \lg_\sigma n / \lg \lg_\sigma n + \lg \lg n / \lg_\sigma n + \lg \sigma / \lg_\sigma n\big) + \sigma^h(n^\beta + \sigma)\big)$ bits of space required for the CSA, for any arbitrarily small constant $0 < \beta < 1$. Note that the latter bound is $nH_h \lg \lg_\sigma n + o(n \lg \sigma) + O(\sigma^h(n^\beta + \sigma))$. The space has an additional term $O(n) = o(n \lg \sigma)$ when σ is non-constant, since we use Theorem 5 for level $k = 0$.

Building the above data structures is a variation of what was done in [GV05]; thus it takes $O\big(n \lg \sigma + \sigma^h(n^\beta + \sigma)\big)$ time to *compress* (as given in Definition 2). The *lookup* operation requires $O(2^{\lg \ell'} + \ell' + 2^{\ell + \lg t(n) - \ell'}) = O(\lg \lg_\sigma n)$ time because accessing any of the data structures in any level requires constant time. (Note that, for level $k = 0$, we use Theorem 4 if $\sigma = O(1)$ or Theorem 5 otherwise). A *substring* query for c symbols requires $O(c/\lg_\sigma n + \lg \lg_\sigma n)$ time since $\Phi_{\ell'}$ decompresses $2^{\ell'} = \Theta(\lg_\sigma n)$ symbols at a time, as we remarked in Section 5.1. This completes the proof of Theorem 6.

We now discuss the complexity of CSA that leads to Theorem 7. We keep a constant number $1/\epsilon$ of the levels as in [GV05], where $0 < \epsilon \le 1/2$. In particular, we store level 0, level ℓ', and then one level every other $\lambda\ell'$ levels; in sum, $1 + 1/\lambda = 1/\epsilon$ levels, where $\lambda = \epsilon/(1-\epsilon)$ with $0 < \lambda < 1$. Each such level $k \le \ell'$ stores the following data structures:

- A directory D_k (in place of B_k in point 1 above) storing the $n_{k+\lambda\ell'}$ (or n_ℓ when $k = \ell'$) entries of the next sampled level. Note that D_0, which stores $n_{\lambda\ell'}$ entries out of a universe of size n, requires just $O(n_{\lambda\ell'}\ell') = O(n \lg \lg_\sigma n / \lg^\lambda_\sigma n)$ bits by using an indexable dictionary [RRR02]. Each of the other D_k's add a geometrically decreasing contribution upper bounded by the cost of D_0.
- A neighbor function Φ_k implemented as given in point 2 above. For all levels $k = \lambda\ell', 2\lambda\ell', \ldots$, neighbor function Φ_k contributes a (geometrically decreasing) total of $O(n_{\lambda\ell'}) = O(n/\lg^\lambda_\sigma n)$ bits, in addition to the term of $O(\sigma^{2^{\ell'}+h}) = O(\sigma^h n^\beta)$ as before. Note that the analysis for Φ_0 is as given in point 2 above. The total required space is therefore (where $\lambda > \epsilon$)

$$\epsilon^{-1} nH_h + O\left(\frac{n \lg \lg_\sigma n}{\lg^\lambda_\sigma n} + \frac{n \lg \lg n}{\lg_\sigma n} + \sigma^h n^\beta\right) = \epsilon^{-1} nH_h + O\left(\frac{n \lg \lg n}{\lg^\epsilon_\sigma n} + \sigma^h n^\beta\right). \tag{21}$$

- The arrays mentioned in point 3 above, except that we now fix $t(n) = \lg^\lambda_\sigma n \lg \sigma$. Thus, we obtain a total space contribution of $O(n \lg \sigma / t(n)) = O(n/\lg^\lambda_\sigma n)$ bits.

In sum, we obtain a total space complexity that is bounded by equation (21). Thus, we are able to save space at a small cost to *lookup*, namely, $O(2^{\lambda\ell'} \lg \sigma + (1/\epsilon - 1)2^{\lambda\ell'} + 2^{\ell + \lg t(n) - \ell'})$ time, where the $\lg \sigma$ factor in the first term is due to the implementation of Φ_0 with the bounds of Theorem 4. Simplifying, we obtain $O(\lg^\lambda_\sigma n \lg \sigma + t(n)) = O(\lg^\lambda_\sigma n \lg \sigma) = O((\lg_\sigma n)^{\epsilon/1-\epsilon} \lg \sigma)$. The *substring* operation for c symbols requires an additional $O(c/\lg_\sigma n)$ time. We can drop the $\lg \sigma$ factor to $O(1)$ in Corollary 3 by using Theorem 5 for the analysis of Φ_0. Building the above data structures is again a variation of what was done for Theorem 6, so *compress* requires $O(n \lg \sigma + \sigma^h n^\beta)$ time, thus proving Theorem 7.

Finally, we prove Theorem 8, which is an interesting special case by a simple modification of the scheme described above. Here we just keep levels 0 and $\ell + \lg t(n)$ where $t(n) = \lg n / \lg \lg n$. We store the following data structures:

- Dictionary D_0 stores $n_{\ell+\lg t(n)}$ entries over a universe of size n, requiring $O(n_{\ell+\lg t(n)}(\ell + \lg t(n))) = O(n(\lg \lg_\sigma n + \lg t(n))/(t(n) \lg_\sigma n))$ bits using an indexable dictionary [RRR02].

- The neighbor function Φ_0 as given in point 2 above, with the bounds of Theorem 4.
- The three arrays as given in point 3 above, using $O(n \lg \sigma / t(n))$ bits.

Thus, the total space is $nH_h + O(n(\lg \lg_\sigma n + \lg t(n))/(t(n) \lg_\sigma n) + n \lg \lg n / \lg_\sigma n + n \lg \sigma / t(n)) = nH_h + O(n \lg \lg n / \lg_\sigma n)$ bits. We also have to add $O\big(\sigma^{h+1} \lg(1 + n/\sigma^{h+1})\big)$ bits for the statistical model. The *lookup* cost is bounded by $O(2^{\ell + \lg t(n)} \lg \sigma) = O(t(n) \lg_\sigma n \lg \sigma) = O(\lg^2 n / \lg \lg n)$, where the $\lg \sigma$ factor comes from the cost of a call to Φ_0 (with the bounds of Theorem 4). Similarly, decompressing each symbol in *substring* has a $O(\lg \sigma)$ cost.

6 Applications to Text Indexing

We use the CSA as an integral component of an efficient text indexing structure that attains the hth-order entropy for a text T of n symbols over alphabet Σ. *Throughout this section, we assume that $h + 1 \leq \alpha \lg_\sigma n$ for any arbitrary constant $\alpha < 1$ to guarantee that the encoding of the empirical statistical model requires $o(n)$ bits.* [13] Our high-order entropy-compressed text indexes support fast searching of a pattern P of length m in $O(m + \text{polylg}(n))$ time with only $nH_h + o(n)$ bits, where nH_h is the information-theoretic upper bound on the number of bits required to encode the text T of length n (cf. Section 2). We also describe a text index that takes $o(m)$ search time *and* uses $o(n)$ bits on highly compressible texts with a small-sized alphabet Σ. The full list of tradeoffs for the space and time complexity of compressed text indexing is shown in Table 2.

6.1 High-Order Entropy-Compressed Text Indexing

We now present our search of a pattern P of length m in the CSA for T. We need the following pattern matching tool to search for P in a sequence of contiguous suffixes stored in the CSA, in compressed form, where the proof of Lemma 11 is given in Section 6.2.

Lemma 11 (Pattern matching tool) *Given a sequence of r consecutive suffixes stored in the CSA, we can search for the leftmost and the rightmost of these suffixes having P as a prefix, in $O(m + r)$ symbol comparisons plus $O(r)$ lookup and substring operations.*

We show how to search P using the CSA and the tool in Lemma 11. We first perform a binary search of P in $SA_{\ell + \lg t(n)}$, which is stored explicitly along with $LCP_{\ell + \lg t(n)}$, the longest common prefix information required in [MM93]. (The term $t(n)$ depends on the implementation of the CSA as described in Section 5.2.2.) Because we have the longest common prefix information, the binary search requires only $O(m)$ symbol comparisons plus $O(\lg n)$ *lookup* and *substring* operations. At that point, we locate $r = 2^{\ell + \lg t(n)} = O(t(n) \lg_\sigma n)$ contiguous suffixes stored, in compressed form, in the CSA. We run the pattern matching tool in Lemma 11 on these r suffixes, at the cost of $O(m + t(n) \lg_\sigma n)$ symbol comparisons and $O(t(n) \lg_\sigma n)$ calls to *lookup* and *substring*, which is also the asymptotic cost of the whole search. The following results show several tradeoffs that we obtain with the simple search scheme described so far.

Theorem 9 *Given a text T of n symbols over an alphabet Σ, we can replace T by a CSA occupying $\epsilon^{-1} nH_h + O(n \lg \lg n / \lg^\epsilon_\sigma n)$ bits, so that searching a pattern of length m takes $O(m/\lg_\sigma n + (\lg n)^{(1+\epsilon)/(1-\epsilon)} (\lg \sigma)^{(1-3\epsilon)/(1-\epsilon)})$ time, for any fixed value of $0 < \epsilon \leq 1/2$. Reporting each occurrence of the pattern takes $O((\lg n)^{(1+\epsilon)/(1-\epsilon)} (\lg \sigma)^{(1-3\epsilon)/(1-\epsilon)})$ time.*

Proof: Using Theorem 7, we have $t(n) = \lg^\lambda_\sigma n \lg \sigma$, where $\lambda = \epsilon/(1-\epsilon)$. The $O(m + t(n) \lg_\sigma n)$ symbol comparisons give a contribution of $O((m + \lg^{1+\lambda}_\sigma n \lg \sigma)/\lg_\sigma n) = O(m/\lg_\sigma n + \lg^\lambda_\sigma n \lg \sigma)$,

[13] This condition is not satisfied if keeping the suffix array *uncompressed* for the text T requires nearly the same space as encoding the hth-order empirical statistics of T. Hence T is not a low-entropy text.

since we can decompress and compare $\Theta(\lg_\sigma n)$ adjacent symbols with $O(1)$ RAM operations. The $O(t(n)\lg_\sigma n) = O(\lg_\sigma^{1+\lambda} n \lg \sigma)$ calls to *lookup* and *substring* (see Lemma 11) give a contribution of $O(\lg_\sigma^{1+2\lambda} n \lg^2 \sigma) = O((\lg n)^{(1+\epsilon)/(1-\epsilon)}(\lg \sigma)^{(1-3\epsilon)/(1-\epsilon)})$. □

For example, fixing $\epsilon = 1/2$ in Theorem 9 when $\sigma = O(1)$, we obtain a search time of $O(m/\lg n + occ \times \lg^3 n)$ with a self-index occupying $2nH_h + O(n \lg\lg n/\sqrt{\lg n})$ bits, where *occ* is the number of occurrences reported. We can reduce the space to nH_h bits plus a lower-order term, obtaining the first nearly space-optimal self-index with polylg(n) reporting time.

Theorem 10 *Given a text of n symbols over an alphabet Σ, we can replace it by a* CSA *occupying nearly optimal space, i.e., $nH_h + O(n \lg\lg n/\lg_\sigma n)$ bits, so that searching a pattern of length m takes $O(m \lg \sigma + \lg^4 n/(\lg^2 \lg n \lg \sigma))$ time. Reporting each pattern occurrence takes $O(m \lg \sigma + \lg^4 n/(\lg^2 \lg n \lg \sigma))$ time.*

Proof: Using Theorem 8, we have $t(n) = \lg n/\lg\lg n$. The $O(m + t(n)\lg_\sigma n)$ symbol comparisons contribute $O(m \lg \sigma + \lg^2 n/\lg\lg n)$, while the $O(t(n)\lg_\sigma n) = O(\lg^2 n/(\lg\lg n \lg \sigma))$ calls to *lookup* and *substring* give a contribution of $O(\lg^4 n/(\lg^2 \lg n \lg \sigma))$. □

If we augment the CSA to obtain the hybrid multi-level data structure in [GV05], we can improve the lower-order terms in the search time of Theorem 9, where $t(n) = \lg_\sigma^\lambda n \lg \sigma$ and $\lambda = \epsilon/(1-\epsilon) > \epsilon$. We use a sparse suffix tree storing every other $(t(n)\lg n)$th suffix using $O(n/t(n)) = O(n/\lg_\sigma^\epsilon n)$ bits to locate a portion of the (compressed) suffix array storing $O(t(n)\lg n)$ suffixes. However, we do not immediately run our pattern matching tool from Lemma 11; instead, we employ a nested sequence of space-efficient Patricia tries [MRS98] of size $\lg^{\omega-\lambda} n$ until we are left with segments of $r = \lg_\sigma^\lambda n$ adjacent suffixes in the CSA, for any fixed value of $1 > \omega \geq 2\lambda > 0$. This scheme adds $O(n/r) = O(n/\lg_\sigma^\epsilon n)$ bits to the self-index, allowing us to restrict the search of pattern P to a segment of r consecutive suffixes in the CSA. At this point, we run our pattern matching tool from Lemma 11 on these r suffixes to identify the leftmost occurrence of the pattern.

Theorem 11 *Given a text of n symbols over an alphabet Σ, we can replace it by a hybrid* CSA *occupying $\epsilon^{-1}nH_h + O(n \lg\lg n/\lg_\sigma^\epsilon n)$ bits, so that searching a pattern of length m takes $O(m/\lg_\sigma n + \lg^\omega n \lg^{1-\epsilon} \sigma)$ time, for any fixed value of $1 > \omega \geq 2\epsilon/(1-\epsilon) > 0$ and $0 < \epsilon \leq 1/3$.*

Proof: Searching in the sparse suffix tree takes $O(m/\lg_\sigma n + \lg_\sigma^\lambda n \lg \sigma)$ time as in [GV05], where the second term is our *lookup* cost in Theorem 7 with $\lambda = \epsilon/(1-\epsilon)$. Then, the search goes through a constant number of space-efficient Patricia tries with $O(\lg^{\omega-\lambda} n)$ calls to *lookup* and *substring*, each of $O(\lg_\sigma^\lambda n \lg \sigma)$ time, requiring a total of $O(\lg^\omega n \lg^{1-\epsilon} \sigma)$ time by Theorem 7. Finally, the pattern matching tool is run on a segment of $r = O(\lg_\sigma^\lambda n)$ suffixes, in $O(\lg_\sigma^{2\lambda} n \lg \sigma) = O(\lg^\omega n \lg^{1-\epsilon} \sigma)$ time. The cost of comparing $\Theta(\lg_\sigma n)$ symbols at a time and decompressing them sums to $O(m/\lg_\sigma n)$, where the additional cost of *substring* is accounted for above. □

We provide the first self-index with small alphabets that is sublinear both in space and in search time.

Corollary 4 *For any low-entropy text over an alphabet of size $\sigma = O(1)$, the self-index in Theorem 11 occupies just $o(n)$ bits and requires $o(m)$ search time. Reporting each occurrence takes $o(\lg n)$ time.*

6.2 A Pattern Matching Tool

In this section, we prove Lemma 11 by describing the implementation of the following pattern matching tool. Given a list of r sequences $S_1 \leq \cdots \leq S_r$ in lexicographical order, the pattern matching tool identifies the least sequence S_i having P as a prefix in $O(m+r)$ time. (Identifying the greatest such sequence is analogous.) We first assume that these r suffixes are explicitly given. Next, we show how to adapt the tool when these suffixes are stored, in compressed form, in the CSA.

Our search tool is reminiscent of the Patricia search [Mor68], the Hirschberg's sequential search [Hir78], and the Bit-Tree search [Fer92], as we only need one full comparison of P against a suffix. Our tool examines the sequences $S_1, \ldots, S_r$ in left-to-right order. We start out by comparing the symbols of P against the symbols of S_1 consecutively until there is a mismatch. We then find the first match in S_2 starting with the symbol that caused the mismatch with S_1. We repeat this process starting at S_2. We stop when we have examined all the sequences unsuccessfully (declaring that there is no occurrence of P), or we succeed in matching the symbols of P at sequence S_i. The steps are detailed below, where we denote the kth symbol of a sequence S by $S[k]$:

1. Set $i = 1$ and $k = 1$.
2. Increment k until either $k > m$ or $S_i[k] \neq P[k]$. If $k > m$, go to step 4; otherwise, find the smallest $j > i$ such that $S_j[k] = P[k]$.
3. If such j does not exist, declare that P is not the prefix of any sequence and quit with a failure. Otherwise, assign the value of j to i.
4. If $k \leq m$, go to step 2. Otherwise, check whether S_i has P as a prefix, returning S_i as the least sequence in case of success; declare a failure otherwise.

Denoting the positions assigned to i in step 3 with $i_1 < i_2 < \cdots < i_k$, we observe that we do not access the first $k-1$ symbols of $S_{i_{k-1}+1}, \ldots, S_{i_k}$, which could be potential mismatches. In general, we compare only a total of $O(i_k + k)$ symbols of $S_{i_1}, \ldots, S_{i_k}$ against those in P, where $i_k \leq r$. Only when we have reached the end of the pattern P (i.e. $k > m$) do we set $i = i_m$ and perform a full comparison of P against S_i in order to determine if there is really a match. This results in a correct method notwithstanding potential mismatches.

Lemma 12 *Given a list of r sequences $S_1, \ldots, S_r$ in lexicographical order, let S_i be the sequence identified by our search tool. If P is a prefix of S_i, then S_i is the least sequence with this property. Otherwise, no sequence in $S_1, \ldots, S_r$ has P as a prefix. The cost of the search is $O(m+r)$ time, where m is the length of P.*

Proof: Suppose P is a prefix of S_i, where S_i was identified by our search tool. We first show that P is not a prefix of $S_1, \ldots, S_{i-1}$. Suppose by contradiction that a sequence S_f has P as a prefix, where $f < i$. Suppose that we are matching the kth symbol of P at the time we examine S_f. Since P is a prefix of S_f, we have a match and our search tool scans the $(k+1)$st symbol in P, the $(k+2)$nd symbol in P and so on, matching all of them with S_f. Hence, our search tool identifies S_f with $f \neq i$, giving a contradiction. This logic proves the first part of the lemma; namely that S_i is the least sequence having P as a prefix, because we consider the sequences S_i in lexicographical order.

To prove the second part, we know that our search tool fails to match P. To see why no sequence in $S_1, \ldots, S_r$ has P as a prefix, note that $S_1, \ldots, S_{i-1}$ cannot have P as a prefix as shown in the previous paragraph. We also have to show this property for the remaining sequences $S_i, \ldots, S_r$. Suppose by contradiction that a sequence S_j, with $j \geq i$, has P as a prefix. Let k be the position of the rightmost symbol in P that we compare to S_j. Our method implies that the kth symbol in S_j is different from that of P. Hence, P cannot be a prefix of S_j, giving the contradiction.

Finally, the time required is $O(m+r)$, as each comparison in our method contributes to at most $2m$ matches and at most r mismatches. □

We now evaluate how the time complexity is affected if $S_1, \ldots, S_r$ are implicitly stored in the CSA, say, at consecutive positions $q+1, \ldots, q+r$ for a suitable value of q. To use our search tool, we need to decompress starting from the kth symbol of a suffix S_i by knowing its position $q+i$ in the CSA. (Recall that $SA[q+i]$ contains the starting position of S_i in the text.) To this end, it suffices to decompress the *first* symbols in the suffix at position $SA^{-1}\big[SA[q+i]+k-1\big]$ in the CSA, where SA and SA^{-1} denote the suffix array and its inverse (as mentioned in Definition 2). Equivalently, the latter suffix S_j can be obtained by removing the first $k-1$ symbols from S_i since $j = SA[q+i]+k-1$. This scheme only requires a constant number of *lookup* operations and a single *substring* operation, with a cost that is independent of the value of k, thus proving Lemma 11.

7 Conclusions

We have presented a unified algorithmic framework to obtain new results in the area of compression and text indexing. We described two techniques—a *context-sensitive partitioning scheme* and the *wavelet tree*—to provide the first optimal space bounds for the Burrows-Wheeler transform aside from lower-order terms. We then used this critical framework to develop a text indexing structure based on a high-order entropy-compressed suffix array that exhibit several tradeoffs between occupied space and search/decompression time. We described how to implement them as a *self-index* requiring $nH_h + O(n \lg\lg n / \lg_\sigma n)$ bits of space and allowing searches of patterns of length m in $O(m \lg \sigma + \text{polylg}(n))$ time. Our scheme provides the first self-index that asymptotically realizes the high-order entropy H_h per symbol of the text. We also proved how to achieve the first self-index with sublinear size $o(n)$ in bits and sublinear query time $o(m)$ for low-entropy texts over an alphabet of constant size.

The most immediate goal is to address whether a compressed full-text index with $nH_h + O(\text{polylg}(n))$ bits and $O(m + \text{polylg}(n))$ query time exists. If not, it would separate indexing from compression for low-entropy strings. Beyond that, we would like to achieve $nH_h + O(n \lg\lg n / \lg_\sigma n)$ bits with an optimal $O(m/\lg_\sigma n + occ)$ search time. A compelling problem is to improve the time for *lookup* so that each call takes constant time. Another interesting challenge would be to support approximate matches (those that match patterns with some threshold of error).

Acknowledgments

We would like to thank Rajeev Raman, Venkatesh Raman, S. Srinivasa Rao, and Kunihiko Sadakane for sending us a copy of the full journal version of their papers, and Rajeev Raman and S. Srinivasa Rao for clarifying some details on succinct data structures. We would also like to thank Paolo Ferragina, Raffaele Giancarlo, and Giovanni Manzini for fruitful discussions, and Frank Ruskey for sending us a chapter of [Rus05].

References

[Aar05] Scott Aaronson. NP-complete problems and physical reality. *SIGACT News*, 36(1):30, 2005.

[AV88] A. Aggarwal and J. S. Vitter. The Input/Output complexity of sorting and related problems. *Communications of the ACM*, 31(9):1116–1127, 1988.

[Bau04] Eric Baum. *What is Thought?* MIT Press, 2004.

[BB04] Daniel K. Blandford and Guy E. Blelloch. Compact representations of ordered sets. In *SODA '04: Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 11–19. Society for Industrial and Applied Mathematics, 2004.

[BBK03] Daniel K. Blandford, Guy E. Blelloch, and Ian A. Kash. Compact representations of separable graphs. In *SODA '03: Proceedings of the fourteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 679–688. Society for Industrial and Applied Mathematics, 2003.

[BDFC05] Michael A. Bender, Erik D. Demaine, and Martin Farach-Colton. Cache-oblivious B-trees. *SIAM J. Comput.*, 2005. (Also in IEEE FOCS 2000.).

[BDMR99] David Benoit, Erik D. Demaine, J. Ian Munro, and Venkatesh Raman. Representing trees of higher degree. In *Proceedings of the 6th International Workshop on Algorithms and Data Structures, WADS'99 (Vancouver, Canada, August 11-14, 1999)*, volume 1663 of *LNCS*, pages 169–180. Springer-Verlag, 1999.

[BM99] Andrej Brodnik and J. Ian Munro. Membership in constant time and almost-minimum space. *SIAM Journal on Computing*, 28(5):1627–1640, October 1999.

[BW94] M. Burrows and D.J. Wheeler. A block sorting data compression algorithm. Technical report, Digital Systems Research Center, 1994.

[Cha04] B. Chazelle. Who says you have to look at the input? The brave new world of sublinear computing, 2004. Plenary talk at at *the 15th Annual ACM-SIAM Symposium on Discrete Algorithms* (SODA 2004).

[CT91] T. M. Cover and J. A. Thomas. *Elements of Information Theory*. Wiley-Interscience, New York, 1991.

[DLO03] Erik D. Demaine and Alejandro López-Ortiz. A linear lower bound on index size for text retrieval. *J. Algorithms*, 48(1):2–15, 2003.

[Eli75] Peter Elias. Universal codeword sets and representations of the integers. *IEEE Transactions on Information Theory*, IT-21:194–203, 1975.

[Fel68] W. Feller. *An Introduction to Probability Theory and its Applications*, volume 1. John Wiley & Sons, New York, 3rd edition, 1968.

[Fen96] P. Fenwick. Punctured elias codes for variable-length coding of the integers. 1996. The University of Auckland, NZ. TR 137. ISSN 1173-3500.

[Fen02] P. Fenwick. Burrows-Wheeler compression with variable-length integer codes. In *Software–Practice and Experience*, volume 32, pages 1307–1316, 2002.

[Fer92] David E. Ferguson. Bit-Tree: a data structure for fast file processing. *Communications of the ACM*, 35(6):114–120, June 1992.

[FGGV04] L. Foschini, R. Grossi, A. Gupta, and J. S. Vitter. Fast compression with a static model in high-order entropy. In *Proceedings of the IEEE Data Compression Conference*, Snowbird, UT, March 2004.

[FGMS05] Paolo Ferragina, Raffaele Giancarlo, Giovanni Manzini, and Gabriella Sciortino. Boosting textual compression in optimal linear time. *Journal of the ACM*, 52(4):688–713, 2005. (Also in CPM 2003, ACM-SIAM SODA 2004.).

[FM05] Paolo Ferragina and Giovanni Manzini. On compressing and indexing data. *Journal of the ACM*, 52(4):552–581, 2005. (Also in IEEE FOCS 2000.).

[FMMN04] Paolo Ferragina, Giovanni Manzini, Veli Mäkinen, and Gonzalo Navarro. Succinct representation of sequences. Technical Report DCC-2004-5, Departamento de Ciencias de la Computación, Universidad de Chile, August 2004. (Also in SPIRE 2004.).

[FTL03] P. Fenwick, M. Titchener, and M. Lorenz. Burrows Wheeler – alternatives to move to front. *Data Compression Conference (DCC)*, 2003.

[GBS92] Gaston H. Gonnet, Ricardo A. Baeza-Yates, and Tim Snider. New indices for text: PAT trees and PAT arrays. In *Information Retrieval: Data Structures And Algorithms*, chapter 5, pages 66–82. Prentice-Hall, 1992.

[GGV03] Roberto Grossi, Ankur Gupta, and Jeffrey S. Vitter. High-order entropy-compressed text indexes. In *Proceedings of the Thitheenth Annual ACM-SIAM Symposium on Discrete Algorithms*. ACM/SIAM, January 2003.

[GGV04] Roberto Grossi, Ankur Gupta, and Jeffrey S. Vitter. When indexing equals compression: Experiments with compressing suffix arrays and applications. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, January 2004.

[GM03] Anna Gál and Peter Bro Miltersen. The cell probe complexity of succinct data structures. In *Automata, Languages and Programming, 30th International Colloquium (ICALP 2003)*, volume 2719 of *Lecture Notes in Computer Science*, pages 332–344. Springer-Verlag, 2003.

[GRR04] Richard F. Geary, Rajeev Raman, and Venkatesh Raman. Succinct ordinal trees with level-ancestor queries. In *SODA '04: Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1–10. Society for Industrial and Applied Mathematics, 2004.

[GV05] Roberto Grossi and Jeffrey S. Vitter. Compressed suffix arrays and suffix trees with applications to text indexing and string matching. *SIAM J. Comput.*, 2005. (Also in ACM STOC 2000.).

[Hir78] D. S. Hirschberg. A lower worst-case complexity for searching a dictionary. In *Proc. 16th Annual Allerton Conference on Communication, Contr ol, and Computing*, pages 50–53, 1978.

[Jac89a] Guy Jacobson. Space-efficient static trees and graphs. In *Proceedings of the 30th Annual IEEE Symposium on Foundations of Computer Science*, pages 549–554, 1989.

[Jac89b] Guy Jacobson. Succinct static data structures. Technical Report CMU-CS-89-112, Dept. of Computer Science, Carnegie-Mellon University, January 1989.

[KM99] S. Rao Kosaraju and Giovanni Manzini. Compression of low entropy strings with lempel-ziv algorithms. *SIAM J. Comput.*, 29(3):893–911, 1999.

[Knu05] Donald E. Knuth. *Combinatorial Algorithms*, volume 4 of *The Art of Computer Programming*. Addison-Wesley, Reading, MA, USA, 2005. In preparation.

[KV98] P. Krishnan and J. S. Vitter. Optimal prediction for prefetching in the worst case. *SIAM Journal on Computing*, 27(6):1617–1636, December 1998.

[LS97] T. Luczak and W. Szpankowski. A suboptimal lossy data compression based in approximate pattern matching. *IEEE Trans. Information Theory*, 43:1439–1451, 1997.

[LV97] M. Li and P. Vitanyi. *An Introduction to Kolmogorov Complexity and Its Applications*. Springer Verlag, 1997.

[Man01] Giovanni Manzini. An analysis of the Burrows-Wheeler transform. *Journal of the ACM*, 48(3):407–430, May 2001.

[McC76] Eduard M. McCreight. A space-economical suffix tree construction algorithm. *Journal of the ACM*, 23(2):262–272, 1976.

[Mil05] Peter Bro Miltersen. Lower bounds on the size of selection and rank indexes. In *Proc. the Sixteenth ACM-SIAM symposium on Discrete Algorithms (SODA05)*, pages 11–12, Philadelphia, PA, USA, 2005.

[MM93] Udi Manber and Gene Myers. Suffix arrays: a new method for on-line string searches. *SIAM Journal on Computing*, 22(5):935–948, 1993.

[Mor68] Donald R. Morrison. PATRICIA - Practical Algorithm To Retrieve Information Coded In Alphanumeric. *Journal of the ACM*, 15(4):514–534, October 1968.

[MR02] J. Ian Munro and Venkatesh Raman. Succinct representation of balanced parentheses and static trees. *SIAM Journal on Computing*, 31(3):762–776, June 2002.

[MR04] J. Ian Munro and S. Srinivasa Rao. Succinct representations of functions. In *Annual International Colloquium on Automata, Languages and Programming (CALP)*, volume 3142 of *Lecture Notes in Computer Science*, pages 1006–1015. Springer-Verlag, 2004.

[MRRR03] J. Ian Munro, Rajeev Raman, Venkatesh Raman, and S. Srinivasa Rao. Succinct representations of permutations. In *Annual International Colloquium on Automata, Languages and Programming (CALP)*, volume 2719 of *Lecture Notes in Computer Science*, pages 345–356. Springer-Verlag, 2003.

[MRS98] J. Ian Munro, Venkatesh Raman, and S. Srinivasa Rao. Space efficient suffix trees. In *Proceedings of Foundations of Software Technology and Theoretical Computer Science*, volume 1530 of *Lecture Notes in Computer Science*, pages 186–195. Springer-Verlag, 1998.

[MRS01a] J. Ian Munro, Venkatesh Raman, and S. Srinivasa Rao. Space efficient suffix trees. *Journal of Algorithms*, 39:205–222, 2001.

[MRS01b] J. Ian Munro, Venkatesh Raman, and Adam J. Storm. Representing dynamic binary trees succinctly. In *Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA-01)*, pages 529–536, New York, January 7–9 2001. ACM Press.

[Mun96] J. Ian Munro. Tables. *FSTTCS: Foundations of Software Technology and Theoretical Computer Science*, 16:37–42, 1996.

[Mut03] S. Muthukrishnan. Data streams: Algorithms and applications, 2003. Plenary talk at *the 14th Annual ACM-SIAM Symposium on Discrete Algorithms* (SODA 2003).

[NM05] Gonzalo Navarro and Veli Mäkinen. Compressed full-text indexes. Technical Report DCC-2005-7, Departamento de Ciencias de la Computación, Universidad de Chile, June 2005.

[Pag01] Rasmus Pagh. Low redundancy in static dictionaries with constant query time. *SIAM Journal on Computing*, 31:353–363, 2001.

[Rao02] S. Srinivasa Rao. Time-space trade-offs for compressed suffix arrays. *IPL*, 82(6):307–311, 2002.

[RC93] J. H. Reif and S. Chen. Using difficulty of prediction to decrease computation: Fast sort, priority queue and convex hull on entropy bounded inputs. In *Proceedings of the IEEE Symposium on Foundations of Computer Science*, volume 34, Palo Alto, 1993.

[RR03] Rajeev Raman and S. Srinivasa Rao. Succinct dynamic dictionaries and trees. In *Annual International Colloquium on Automata, Languages and Programming (CALP)*, volume 2719 of *Lecture Notes in Computer Science*, pages 357–368. Springer-Verlag, 2003.

[RRR02] Rajeev Raman, Venkatesh Raman, and S. Srinivasa Rao. Succinct indexable dictionaries with applications to encoding k-ary trees and multisets. In *ACM-SIAM Symposium on Discrete Algorithms*, pages 233–242, 2002.

[Rus05] Frank Ruskey. *Combinatorial Generation*. 2005. In preparation.

[Sad02] Kunihiko Sadakane. Succinct representations of *lcp* information and improvements in the compressed suffix arrays. In *Proceedings of the Thitheenth Annual ACM-SIAM Symposium on Discrete Algorithms*. ACM/SIAM, 2002.

[Sad03] Kunihiko Sadakane. New text indexing functionalities of the compressed suffix arrays. *J. Algorithms*, 48(2):294–313, 2003. (Also in ISAAC 2000.).

[Sha48] C. E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423, July 1948.

[Ukk95] Esko Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14(3):249–260, September 1995.

[VK96] J. S. Vitter and P. Krishnan. Optimal prefetching via data compression. *Journal of the ACM*, 43(5), September 1996.

[Wei73] Peter Weiner. Linear pattern matching algorithm. *Proc. 14th Annual IEEE Symposium on Switching and Automata Theory*, pages 1–11, 1973.

[ZL77] J. Ziv and A. Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23(3):337–343, 1977.