Grammar Boosting: A New Technique for Proving Lower Bounds for Computation over Compressed Data[*]

Rajat De[†] Dominik Kempa[‡]

Abstract

Computation over compressed data is a new paradigm in the design of algorithms and data structures that can reduce space usage and speed up computation by orders of magnitude. One of the most frequently employed compression frameworks, capturing many practical compression methods (such as the Lempel–Ziv family, dictionary methods, and others), is grammar compression. In this framework, a string T of length N is represented as a context-free grammar of size n whose language contains only the string T. In this paper, we focus on studying the limitations of these techniques. Previous work focused on proving lower bounds for algorithms and data structures operating over grammars constructed using algorithms that achieve the approximation ratio $\rho = \mathcal{O}(\text{polylog}\, N)$ (since finding the smallest grammar representation is NP-hard, every polynomial-time grammar compressor can be viewed as an approximation algorithm). Unfortunately, for many grammar compressors we either have $\rho = \omega(\text{polylog}\, N)$ or it is not known whether $\rho = \mathcal{O}(\text{polylog}\, N)$ holds. In their seminal paper, Charikar, Lehman, Liu, Panigrahy, Prabhakaran, Sahai, and Shelat [IEEE Trans. Inf. Theory 2005] studied seven popular grammar compression algorithms: RePair, Greedy, LongestMatch, Sequential, Bisection, LZ78, and α-Balanced. Only one of them (α-Balanced) is known to achieve $\rho = \mathcal{O}(\text{polylog}\, N)$.

In this paper, we develop the first technique for proving lower bounds for data structures and algorithms on grammars that is fully general and does not depend on the approximation ratio ρ of the used grammar compressor. Our first set of results concerns *compressed data structures*. In 2013, Verbin and Yu proved that implementing random access to T using a grammar constructed by an algorithm with $\rho = \mathcal{O}(\text{polylog}\, N)$ requires $\Omega(\log N / \log\log N)$ time in the worst case. This lower bound applies to any structure using $\mathcal{O}(n\, \text{polylog}\, N)$ space and matches the existing upper bounds. We prove that this lower bound holds also for RePair, Greedy, LongestMatch, Sequential, and Bisection, while $\Omega(\log\log N)$ time is required for random access to LZ78. Our lower bounds apply to any structure using $\mathcal{O}(n\, \text{polylog}\, N)$ space and match the existing upper bounds. Moreover, we show that our technique generalizes to the class of *global* algorithms (that includes, e.g., the RePair algorithm), i.e., the lower bound $\Omega(\log N / \log\log N)$ applies to the whole class. This makes a significant step forward in a long-standing open problem of analyzing global algorithms.

Our second set of results concerns *compressed computation*, i.e., computation that runs in time that depends on the size of the input in compressed form. Recently, Abboud, Backurs, Bringmann, and Künnemann [FOCS 2017 and NeurIPS 2020] proved numerous limitations of compressed computation under popular conjectures (such as SETH, k-Clique, k-OV, and k-SUM). Similarly as above, however, their framework also displays a dependence on ρ. For example, their results imply that, assuming the k-Clique Conjecture, there is no algorithm to solve CFG Parsing (for which the best algorithm has a time complexity of $\mathcal{O}(N^{\omega})$, where ω is the exponent of matrix multiplication) on grammars constructed using Bisection (which satisfies $\rho = \widetilde{\Theta}(N^{1/2})$) that runs in $\mathcal{O}(n^c \cdot N^{\omega-\epsilon})$ time for constants $\epsilon > 0$ and $c < 2\epsilon$. Using our new techniques, we improve these and other conditional lower bounds. For example, for CFG parsing on Bisection, we rule out algorithms with runtime $\mathcal{O}(n^c \cdot N^{\omega-\epsilon})$ for *all* constants $\epsilon > 0$ and $c > 0$.

1 Introduction

Modern applications produce textual data at a rate not seen before. During 2004–2015, the cost of sequencing the DNA of a single person has decreased from \$20 million to around \$1000, i.e., by a factor of 2×10^5 [48]. This resulted in projects like the 100,000 Genome Project [44], which during 2013-2018, produced around 75 terabytes of text. The efforts to sequence even larger populations are now underway, e.g., in 2018, 26 countries started the

[*]The full version of the paper can be accessed at https://arxiv.org/abs/2307.08833.

[†]Stony Brook University, NY, USA. Email: rde@cs.stonybrook.edu.

[‡]Stony Brook University, NY, USA. Email: kempa@cs.stonybrook.edu. Supported by a Simons Foundation Junior Faculty Fellowship.

ongoing 1+ Million Genomes Initiative [25]. It is predicted that genomics research will generate between 2 and 40 exabytes of data within the next decade [77, 92]. Other sources of massive textual datasets include versioned text documents (such as Wikipedia) and source code repositories (such as Github) [74, 53].

This explosion of data has not been matched by the corresponding increase in computational power. One ray of hope in being able to handle such massive datasets is that they are highly repetitive [89, 26, 11, 46, 74]. This has been the driving force behind the development of *compressed algorithms and data structures* [72, 73, 74], which combine aspects of information theory, lossless data compression, and combinatorial pattern matching, to perform various queries or even run complex computation directly on data in compressed form [37, 53, 59, 1].

One of the most general frameworks for storing highly repetitive strings is *grammar compression* [62, 20, 91], in which we represent a string using a *straight-line program (SLP)*, i.e., a context-free grammar, whose language contains only the input string. On the one hand, this framework is easy to work with and can succinctly encode even complex structure of repetitions. On the other hand, it comes with solid mathematical foundations – as shown in [20, 91, 36, 59, 53, 65, 90, 60], grammar compression is up to logarithmic factors equivalent to LZ77 [100], LZ-End [66], RLBWT [18], macro schemes [93], collage systems [61], string attractors [59], and substring complexity [90, 65], and it is at least as powerful as automata [14], Byte-Pair [32], and several other LZ-type compressors [101, 98, 94, 30]. For this reason, grammar compression has been a very popular framework in numerous previous studies. This includes pattern matching [50, 38, 1, 42, 15, 19], sequence similarity [95, 47, 1, 41], context-free grammar (CFG) parsing, RNA folding, disjointness [1], and compressed linear algebra [2, 29]. Grammars are also the key component in algorithms converting between different compressed representations [53, 58]. We refer to surveys in [74, 73, 69] and discussion in [1, 37, 59, 53] for more details.

The central component in many of the above applications, and a useful structure on its own, is a *compressed index* – a data structure requiring small space (close to the size of SLP representing the text) that supports various queries over the underlying (uncompressed) text. Nowadays, SLP-based indexes supporting random-access [13, 39, 9, 59, 65, 56], rank/select [88, 86, 10, 9], longest common extension (LCE) [49, 43, 78, 56], pattern matching [22, 23, 24, 34, 33, 27, 21, 64, 43, 65], suffix array queries [56], and various geometric queries [17, 16] are available. Despite these advances, our understanding of lower bounds on SLP-compressed indexes remains an open problem, even for random access (the most basic query and a building block for more complex queries).

- On the one hand, Bille et al. [13] proved that for any grammar compression algorithm that reduces a length-N string T into a representation of size n, we can build a structure of size $\mathcal{O}(n)$ allowing decoding of any symbol of T in $\mathcal{O}(\log N)$ time. The latter result has recently been generalized by Ganardi et al. [39], who proved that for any size-n SLP encoding a length-N string, there exists a size-$\mathcal{O}(n)$ SLP encoding the same string but with height $\mathcal{O}(\log N)$. At the cost of increasing the space by a $\mathcal{O}(\log^{\epsilon} n)$ factor, it is possible to reduce the query time to $\mathcal{O}(\log N/\log\log N)$ [9]. To complement this, Verbin and Yu, proved that for every algorithm that achieves an $\mathcal{O}(\operatorname{polylog} N)$ approximation ratio[1], one cannot access symbols of T faster than $\mathcal{O}(\log N/\log\log N)$ time using a representation of size $\mathcal{O}(n \operatorname{polylog} N)$ [97].[2]
- On the other hand, for LZ78 [101] in $\mathcal{O}(n)$ space it is possible to implement random access in $\mathcal{O}(\log\log N)$ [28]. Consistent with the bound of Verbin and Yu, LZ78 achieves an approximation ratio of $\widetilde{\Omega}(N^{2/3})$ [20, 8].[3]

The above situation suggests that there exists a trade-off between the approximation ratio and the time required for random access. This, however, leads to two very serious issues:

1. The above techniques do not say anything about lower bounds on random access to grammars computed using algorithms with $\omega(\operatorname{polylog} N)$ approximation factor. This is problematic, since the majority of practical grammar compressors are in this category: Charikar et al. [20] prove that Sequitur [76], Sequential [99], Bisection [63, 75], LZ78 [101], LZW [98] all achieve $\Omega(N^{\epsilon})$ ratio (for some constant $\epsilon > 0$). Badkobeh et al. [7] prove analogous bound for LZD [45].
2. Even worse, for many grammar compressors, it is not known whether they achieve an approximation ratio of $\mathcal{O}(\operatorname{polylog} N)$. This includes Greedy [5, 6, 4], LongestMatch [67], and RePair [68] – the last being one of the most practical and widely studied compressors [12, 82, 35, 70, 71, 31, 40] which "(...)

[1]Finding the smallest grammar encoding of a given string is NP-hard [20]. Thus, every polynomial time grammar compression algorithm can be viewed as the approximation algorithm for the *smallest grammar problem* [20].

[2]Verbin and Yu [97] formulate this equivalently as stating that for any *universal* data structure (i.e., working for every grammar compressor), one cannot achieve $o(\log N/\log\log N)$ query time in $\mathcal{O}(n \operatorname{polylog} N)$ space.

[3]We assume that $\widetilde{\Omega}$ and $\widetilde{\mathcal{O}}$ suppresses factors polylogarithmic in the length N of the uncompressed text T.

consistently outperforms other grammar-based compressors, including those that offer theoretical guarantees of approximation." [73]

With current techniques, proving lower bounds for algorithms like RePair appears to be a hopeless task, since RePair has resisted all attempts to prove an upper bound on its approximation ratio for over 20 years. Given this situation, we ask:

Problem 1. *Can we prove lower bounds for data structures based on grammar compressors without first establishing their approximation ratio?*

Another application of data compression in the design of algorithms is *compressed computation*, where the goal is to develop algorithms whose runtime depends on the size of the input in the compressed form. For example, Tiskin [95] developed an algorithm that, given two strings $S_1 \in \Sigma^N$ and $S_2 \in \Sigma^N$, both in grammar-compressed form of total size n, computes the longest common subsequence $LCS(S_1, S_2)$ in $\widetilde{\mathcal{O}}(n \cdot N)$ time. For highly compressible strings (e.g., when $n = \mathcal{O}(N^{1/2})$), this is a significant improvement over the currently best general algorithm for LCS that runs in $\mathcal{O}(N^2)$ time (and is unlikely to be improved to $\mathcal{O}(N^{2-\epsilon})$ due to the recent conditional lower bound [3] based on the Strong Exponential Time Hypothesis (SETH)).

Abboud et al. [1, 2] recently asked whether algorithms like the above LCS algorithm can be improved, i.e., when restricting the set of inputs to those that are highly compressible, is the algorithm running in $\mathcal{O}((n \cdot N)^{1-\epsilon})$ time achievable. They proved that under popular hardness assumptions such as SETH, k-OV, k-Clique, or k-SUM, the currently best compressed algorithms for several problems are optimal. In particular, they showed that unless SETH fails, there is no algorithm for LCS that runs in $\mathcal{O}((n \cdot N)^{1-\epsilon})$ time, for any constant $\epsilon > 0$, even when restricted to highly compressible strings. They proved similar conditional lower bounds for CFG parsing, RNA folding, matrix-vector multiplication, inner product, and several other problems.

Similarly as the lower bound of Verbin and Yu, however, the techniques in [1, 2] exhibit a dependence on the approximation ratio ρ of the grammar compressor used to obtain the input grammar. For example, for the CFG parsing problem, where given a CFG Γ (for simplicity, let us assume $|\Gamma| = \widetilde{\mathcal{O}}(1)$) and a string $S \in \Sigma^N$, the goal is to check if $S \in L(\Gamma)$, the currently best algorithm runs in $\widetilde{\mathcal{O}}(N^\omega)$ time, where ω is the exponent of matrix multiplication [96]. Abboud et al. [1] proved that unless the k-Clique Conjecture fails, for S constructed using grammar compressors with $\rho = \mathcal{O}(\operatorname{polylog} N)$, there is no algorithm running $\mathcal{O}(\operatorname{poly}(n) \cdot N^{\omega-\epsilon})$ time. However, for larger ρ, e.g., $\rho = \Theta(N^\alpha)$, this technique excludes only the algorithms running in $\mathcal{O}(n^c \cdot N^{\omega-\epsilon})$, where $c < \epsilon/\alpha$. For example, if $\alpha = 1/2$, then there is no algorithm running in $\mathcal{O}(n^{3/2} \cdot N)$, but it leaves open whether there is an algorithm running in $\mathcal{O}(n^4 \cdot N)$ time. We thus ask:

Problem 2. *What are the limitations for compressed computation on grammars obtained using algorithms with large or unknown approximation ratios? Can we prove such lower bounds without first establishing those approximation ratios?*

Our Results We present a new technique for proving lower bounds on grammar-compressed strings called *Grammar Boosting* that does not require any knowledge about the approximation ratio of the algorithm and lets us answer both of the above questions.

New Lower Bounds for Data Structures. We prove that the lower bound of $\Omega(\log N / \log\log N)$ applies to nearly all of the classical and commonly used grammar compressors, including: RePair [68], Greedy [4, 5, 6], LongestMatch [67], Sequitur [76], Sequential [99], Bisection [63, 75], and LZD [45]. No lower bounds for random access on either of these grammars were known before. Our bound applies to any structure whose space is $\mathcal{O}(n \operatorname{polylog} N)$ (where n is the output size of any of the above algorithms), i.e., it is always as strong as the bound of Verbin and Yu [97]. Our result establishes the first query separation between algorithms like Sequitur or LZD, and LZ78 (which admits a random access solution with $\mathcal{O}(\log\log N)$ query time [28]). As an auxiliary result, we show (via a reduction from the *colored predecessor problem* [85]) that random access to LZ78 in $\mathcal{O}(\log\log N)$ time is optimal within near-linear space (i.e., $\mathcal{O}(n \operatorname{polylog} N)$), which is the case in [28].

Our technique applies not only to individual algorithms but is able to capture an entire class. Specifically, we show that the lower bound $\Omega(\log N / \log\log N)$ holds for all *global algorithms* [20] (which includes RePair, GREEDY, and LONGESTMATCH). This makes a significant step forward in a long-standing open problem postulated by Charikar et al. [20]: "Because they are so natural and our understanding is so incomplete, global algorithms are one of the most interesting topics related to the smallest grammar problem that deserves further investigation."

The key idea in the framework and Verbin and Yu [97] is to prove that given any collection $\mathcal{P}$ of n points on an $n \times n$ grid, we can construct a string $A(\mathcal{P})$ (called the *answer string*; see Definition 4.1) of length $|A(\mathcal{P})| = \Theta(n^2)$ that encodes answers to all possible parity range counting queries [84] on $\mathcal{P}$, and has a grammar of size $\mathcal{O}(n \operatorname{polylog} n)$. Since answering such queries in $\mathcal{O}(n \operatorname{polylog} n)$ space requires $\Omega(\log n/\log\log n)$ time [84] (Theorem 4.1), any universal structure that for a grammar G encoding $T \in \Sigma^N$ takes $\mathcal{O}(n \operatorname{polylog} N)$ space, must thus take $\Omega(\log N/\log\log N)$ time for access.

Let ALG be some fixed grammar compression algorithm. The issue with applying the above idea to ALG is that it would require proving a bound on the size of the output of ALG on $A(\mathcal{P})$, which for, e.g., REPAIR can be very difficult. We instead prove that for any string T for which there *exists* an SLP of size n that encodes T, we can construct a string T' such that:

(a) The length of T' is polynomial in the length of T,
(b) T can be quickly identified within T', e.g., $T[j] = T'[\alpha + \beta \cdot j]$ for some $\alpha \geq 0$ and $\beta > 0$,
(c) ALG compresses T' into size $\mathcal{O}(n)$. Note that this does not require proving anything about the compression of T by ALG.

In other words, we take a string T having a small grammar G and "boost" the performance of ALG by presenting T in a well-structured form of T' so that ALG compresses the string T' into size $\mathcal{O}(|G|)$. This still lets us utilize the reduction from parity range counting queries [84], since by Item (a), $\log|T'| = \Theta(\log|T|)$, and by Item (b), accessing symbols of T via T' does not incur any time penalty. To construct T', we typically first define an auxiliary grammar G' with the set of nonterminals similar to G, but including special sentinel symbols identifying the nonterminals. The string T' is then defined by listing expansions of all nonterminals of G' in the order of nondecreasing length, repeating each expansion twice, optionally separating with additional sentinel symbols. The crux of the analysis is to show that for any of the algorithms we studied, such structuring forces the algorithm to compress the string in a specific way. For global algorithms this is particularly hard, since their behavior is not very well understood. We manage, however, to fully characterize a class of all intermediate grammars that global algorithms can reach during the processing of T', and then prove that there is only one possible final grammar. We present more details in the Technical Overview (Section 4). As a result, we obtain a series of lower bounds. In a single theorem, we can summarize them as follows. For any grammar compression algorithm ALG, we denote the output of ALG on a string T by ALG(T).

THEOREM 1.1. *Let* ALG *be any global algorithm (e.g.,* REPAIR*,* GREEDY*, or* LONGESTMATCH*), or one of the following algorithms:* SEQUITUR*,* SEQUENTIAL*,* BISECTION*, or* LZD*. In the cell-probe model, there is no data structure that, for every string T of length N, achieves $\mathcal{O}(|\text{ALG}(T)| \log^c N)$ space (where $c = \mathcal{O}(1)$) and implements random access queries to T in $o(\log N/\log\log N)$ time.*

New Lower Bounds for Compressed Computation. Our second result is to demonstrate that the grammar boosting technique complements the framework of Abboud et al. [1, 2] for proving conditional lower bounds for compressed computation. We consider two problems: CFG Parsing (defined above), and Weighted RNA Folding (defined below). Among other results, the authors of [1] prove that unless the k-Clique Conjecture (resp. Combinatorial k-Clique Conjecture) fails (see Section 2.1), these problems essentially require $\Omega(N^\omega)$ (resp. $\Omega(N^3)$) time, even for highly compressible inputs. Here we extend these results to several grammar compressors with large or unknown approximation ratio ρ.

Consider the CFG Parsing problem. The key idea in the hardness proof presented in [1] is, given $k \geq 3$ and an undirected graph $G = (V, E)$, to construct a small CFG Γ and a highly compressible string S of length $|S| = \mathcal{O}(|V|^{k+2})$ (specifically, S has a grammar of size $\mathcal{O}(|V|^3)$), such that G has a $3k$-clique if and only if $S \in L(\Gamma)$. To adapt this reduction to a grammar compression algorithm ALG, we construct a "well-structured" string S' and a CFG Γ' such that:

(a) $|S'| = |V|^{k+\mathcal{O}(1)}$,
(b) $S' \in L(\Gamma')$ if and only if $S \in L(\Gamma)$,
(c) ALG compresses S' into size $|V|^{\mathcal{O}(1)}$.

The construction of S' for Sequential is similar to the random-access problem above. The CFG Γ' is obtained from Γ by ensuring that it ignores every second symbol as well as some sufficiently long prefix of S'. As a result, we obtain a series of lower bounds. In a single theorem, we can summarize them as follows (note that this is a simplified form, and in some cases, the lower bounds hold with stronger assumptions on $|\Sigma|$ and $|\Gamma|$).

Theorem 1.2. *Let* Alg *be any global algorithm (e.g.,* RePair*,* Greedy*, or* LongestMatch*), or one of the following algorithms:* Sequitur*,* Sequential*,* Bisection*, or* LZD*. Let $\delta \in (0, 1]$. Assuming the k-Clique Conjecture (resp. Combinatorial k-Clique Conjecture), there is no algorithm (resp. combinatorial algorithm) that, given $T \in \Sigma^N$ and a CFG Γ such that $|\Gamma| = \mathcal{O}(N^\delta)$, $|\Sigma| = \mathcal{O}(N^\delta)$, and $|\textsc{Alg}(T)| = \mathcal{O}(N^\delta)$, determines if $T \in L(\Gamma)$ in $\mathcal{O}(N^{\omega-\epsilon})$ (resp. $\mathcal{O}(N^{3-\epsilon})$) time, for any $\epsilon > 0$.*

In the RNA Folding problem, we are given a string $T \in \Sigma^N$, where Σ is augmented with a *match* operation such that for every $a \in \Sigma$, there exists a matching symbol $\overline{a} \in \Sigma$ satisfying $\overline{a} \neq a$ and $\overline{\overline{a}} = a$. The goal is to compute the cardinality $|R|$ of a largest set $R \subseteq [1\,..\,N]^2$ such that for every $(i, j) \in R$, the symbols $T[i]$ and $T[j]$ match, and every pair of intervals in R is either nested or disjoint. In the weighted variant, we are additionally given a weight function $w : \Sigma \to [0\,..\,M]$, and the goal is to compute the weight of a set R that maximizes the total weight of matching pairs. We denote it $\mathrm{WRNA}(T)$. The application of grammar boosting for this problem requires proving some auxiliary results about the studied grammar compression algorithms or the RNA Folding problem, but in its core, it uses a reduction from k-Clique [1] essentially in a black-box manner. As a result, we obtain a series of lower bounds. In a single theorem, we can summarize them as follows.

Theorem 1.3. *Let* Alg *be* Sequential*,* LZD*, or any global algorithm (e.g.,* RePair*,* Greedy*, or* LongestMatch*). Assuming the k-Clique Conjecture (resp. Combinatorial k-Clique Conjecture), there is no algorithm (resp. combinatorial algorithm) that, given $T \in \Sigma^N$ (where Σ is augmented with a match operation) and a weight function $w : \Sigma \to [0\,..\,M]$ such that $|\Sigma| = \mathcal{O}(N^\delta)$, $M = \mathcal{O}(\mathrm{poly}(N))$, and $|\textsc{Alg}(T)| = \mathcal{O}(N^\delta)$, computes $\mathrm{WRNA}(T)$ in $\mathcal{O}(N^{\omega-\epsilon})$ (resp. $\mathcal{O}(N^{3-\epsilon})$) time, for any $\epsilon > 0$.*

Related Work Another important compression method utilized in compressed indexing is the Run-Length Burrows–Wheeler Transform (RLBWT) [18]. Gagie et al. [37] demonstrated that it is possible to efficiently support suffix array and suffix tree queries using $\mathcal{O}(r \log n)$ space[4], where r is the size of RLBWT. On the other hand, Kempa and Kociumaka proved that for all strings, it always holds $r = \mathcal{O}(\delta \log \delta \max(1, \log \frac{n}{\delta \log \delta}))$ [53] (where $\delta \leq r$ is the *substring complexity* [65], a measure closely related to Lempel-Ziv and grammar compression), establishing the first link between RLBWT-based indexes, and LZ and grammar-based indexes. Moreover, the above gap between r and δ is asymptotically tight in the worst-case for all values of r, δ, and n [53]. This created a divide between indexes efficiently supporting easier queries like random access or pattern matching (which achieve $\mathcal{O}(\delta \log n)$ space [65]), and indexes supporting the powerful queries like suffix array (for which the best achievable space was $\mathcal{O}(r \log n)$ [37], i.e., up to $\Theta(\log^2 n)$ times larger). This apparent hierarchy was recently proved to be nonexistent: In [56], Kempa and Kociumaka described the first compressed index, called δ-SA, that supports the suffix array functionality in $\mathcal{O}(\delta \log n)$ space. More precisely, δ-SA uses $\mathcal{O}(\delta \log \frac{n \log \sigma}{\delta \log n})$ space. The significance of this result is that $\mathcal{O}(\delta \log \frac{n \log \sigma}{\delta \log n})$ is the asymptotically smallest space sufficient to represent any string with parameters n, σ, and δ [65]. This is also the space used by the smallest text indexes supporting basic queries such as random access [65]. Hence, [56] can be viewed as collapsing the hierarchy of compressed indexes to a point. As an auxiliary result, [56] also describes the first structure supporting efficient LCE queries in $\mathcal{O}(\delta \log \frac{n \log \sigma}{\delta \log n})$ space.

An important aspect of text indexes that also received significant attention is their efficient construction [83, 81, 53, 55, 87, 57, 52, 56]. Of particular interest here are the algorithms operating in *compressed time*, i.e., time proportional to the input in compressed form. Significant progress on this problem has recently been made by [56], which described the algorithm that, given any LZ77-like parsing of T consisting of f phrases, constructs δ-SA in $\mathcal{O}(f \,\mathrm{polylog}\, n)$ time. This is the first compressed-time algorithm for constructing a compressed index with suffix array functionality. They also developed the first such construction algorithms for the optimal-space indexes supporting LCE and random access queries.

Yet another aspect of (compressed) text indexes that has recently attracted a lot of attention is making them dynamic. Several of the above-mentioned operations can be supported in this setting [43, 54, 79, 78].

[4]Nishimoto and Tabei [80] showed how to reduce the space of [37] for LF and Φ^{-1} queries to $\mathcal{O}(r)$. These queries, however, are not sufficient to compute arbitrary suffix array values.

2 Preliminaries

Let $w \in \Sigma^n$ be a *string* (or *text*) of length n over *alphabet* Σ. Denote $\sigma = |\Sigma|$. We index strings starting from 1, i.e., $w = w[1]w[2]\cdots w[n]$. For $1 \le i \le j \le n$, we denote substrings of w as $w[i\,..\,j]$ and by $[i\,..\,j)$ we mean $[i\,..\,j-1]$. We denote the length of string w as $|w|$. The concatenation of strings u and v is written as $u\cdot v$ or uv, and the empty string is denoted ε. For every $u, v \in \Sigma^*$, we denote $\mathrm{Occ}(u, v) = \{i \in [1\,..\,|v|] : i+|u| \le |v|+1 \text{ and } v[i\,..\,i+|u|) = u\}$.

A *context-free grammar* (CFG) is a tuple $G = (V, \Sigma, R, S)$, where V is a finite set of *nonterminals* (or *variables*), Σ is a finite set of *terminals*, and $R \subseteq V \times (V \cup \Sigma)^*$ is a set of *productions* (or *rules*). We assume $V \cap \Sigma = \emptyset$ and $S \in V$. The nonterminal S is called the *starting nonterminal*. Nonterminals in $V \setminus \{S\}$ are called *secondary*. If $(N, \gamma) \in R$, then we write $N \to \gamma$. For $u, v \in (V \cup \Sigma)^*$, we write $u \Rightarrow v$ if there exist $u_1, u_2 \in (V \cup \Sigma)^*$ and a rule $N \to \gamma$ such that $u = u_1 N u_2$ and $v = u_1 \gamma u_2$. We say that u *derives* v and write $u \Rightarrow^* v$, if there exists a sequence $u_1, \ldots, u_k$, $k \ge 1$, such that $u = u_1$, $v = u_k$, and $u_i \Rightarrow u_{i+1}$ for $1 \le i < k$. The *language* of grammar G is the set $L(G) := \{w \in \Sigma^* \mid S \Rightarrow^* w\}$.

A grammar $G = (V, \Sigma, R, S)$ is called a *straight-line grammar* (SLG) if for any $N \in V$, there is exactly one production with N on the left side, and there exists a linear order $\prec$ on V such that for every $X, Y \in V$, if $(X, \gamma) \in R$ and Y occurs in γ, then $X \prec Y$. The unique γ such that $N \to \gamma$ is called the *definition* of N and denoted $\mathrm{rhs}_G(N)$. If G is clear from the context, we simply write $\mathrm{rhs}(N)$. In any SLG, for any $u \in (V \cup \Sigma)^*$, there exists exactly one $w \in \Sigma^*$ such that $u \Rightarrow^* w$. We call such w the *expansion* of u and denote $\exp_G(u)$ (or simply $\exp(u)$ when G is clear). Note that for any SLG G, $L(G) = \{\exp_G(S)\}$.

We say that two SLGs $G_1 = (V_1, \Sigma, R_1, S_1)$ and $G_2 = (V_2, \Sigma, R_2, S_2)$ are *isomorphic* if there exists a bijection $f : V_1 \cup \Sigma \to V_2 \cup \Sigma$ such that:

- $f(S_1) = S_2$,
- For every $c \in \Sigma$, $f(c) = c$,
- For every $N_1 \in V_1$, letting $N_2 = f(N_1)$, $u_1 = \mathrm{rhs}_{G_1}(N_1)$, and $u_2 = \mathrm{rhs}_{G_2}(N_2)$, it holds that $|u_1| = |u_2|$, and for every $j \in [1\,..\,|u_1|]$, $u_2[j] = f(u_1[j])$.

If G_1 is isomorphic to G_2, then $|V_1| = |V_2|$. Moreover, for every $N_1 \in V_1$, $\exp_{G_1}(N_1) = \exp_{G_2}(f(N_1))$. In particular, $L(G_1) = \{\exp_{G_1}(S_1)\} = \{\exp_{G_2}(f(S_1))\} = \{\exp_{G_2}(S_2)\} = L(G_2)$.

Let $G = (V, \Sigma, R, S)$ be an SLG. We define the *parse tree* of $A \in V \cup \Sigma$ as a rooted ordered tree $\mathcal{T}_G(A)$ (we omit G, whenever it is clear from the context), where each node v is associated with a symbol $s(v) \in V \cup \Sigma$. The root of $\mathcal{T}(A)$ is a node ρ such that $s(\rho) = A$. If $A \in \Sigma$ then ρ has no children. If $A \in V$ and $\mathrm{rhs}(A) = B_1 \cdots B_k$, then ρ has k children and the subtree rooted at the ith child is (a copy of) $\mathcal{T}(B_i)$. The parse tree $\mathcal{T}(G)$ of G is defined as the parse tree $\mathcal{T}(S)$ of the starting nonterminal S.

The idea of *grammar compression* is, given a string w, to compute a small SLG G such that $L(G) = \{w\}$. The *size* of the SLG G is defined as $|G| := \sum_{N \in V} |\mathrm{rhs}(N)|$. Clearly, it is easy to encode any G in $\mathcal{O}(|G|)$ space: pick an ordering of nonterminals and write down the definitions of all variables with nonterminals replaced by their number in the order.

The size of the smallest SLG generating w is denoted $g^*(w)$. The decision problem SmallestGrammar of determining whether for a given string w it holds $g^*(w) \le t$ is NP-hard [20]. The optimization version of the problem is APX-hard [20], but $\mathcal{O}(\log(n/g^*))$-approximations are known [91, 20, 51].

Definition 2.1. An SLG $G = (V, \Sigma, R, S)$ is *admissible* if for every $X \in V$, it holds $|\mathrm{rhs}_G(X)| = 2$ and X occurs in the parse tree $\mathcal{T}(S)$.

2.1 Hardness Assumptions

Conjecture 2.1. (k-Clique) *For all constant $k \ge 3$ and $\epsilon > 0$, there is no algorithm checking if an undirected graph $G = (V, E)$ has a k-clique that runs in $\mathcal{O}(|V|^{(k\omega/3)(1-\epsilon)})$ time.*

Conjecture 2.2. (Combinatorial k-Clique) *For all constant $k \ge 3$ and $\epsilon > 0$, there is no combinatorial algorithm checking if an undirected graph $G = (V, E)$ has a k-clique that runs in $\mathcal{O}(|V|^{k(1-\epsilon)})$ time.*

3 Grammar Compression Algorithms

3.1 Global Algorithms

Definition 3.1. Let $G = (V, \Sigma, R, S)$ be an SLG. A string $s \in (\Sigma \cup V)^+$ is called *maximal* (with respect to G) if it satisfies the following conditions:

1. $|s| \geq 2$,
2. The string s has at least two non-overlapping occurrences on the right-hand side of G (i.e., in the definitions of nonterminals of G),
3. There is no string $s' \in (\Sigma \cup V)^+$ such that $|s'| > |s|$ and s' has at least as many non-overlapping occurrences on the right-hand side of G as s (where the number of non-overlapping occurrences of a string x on the right-hand side of G is defined with a greedy search, i.e., we scan the definition of each nonterminal left-to-right, and once an occurrence of x is found at position i, we restart the search at position $i + |x|$).

Charikar et al. [20] defines the class of "global algorithms" as all grammar compression algorithms operating according to the following principle. Begin with a grammar having a single starting nonterminal S whose definition is w. Each iteration of the algorithm: (1) chooses a maximal string s (Definition 3.1), (2) creates a new nonterminal N and sets s as its definition, and (3) scans (left to right) the definition of every nonterminal except N, replacing every encountered occurrence of s with N. Note that all the affected occurrences of s are non-overlapping. Global algorithms differ only in the choice of the maximal string s at each step.

Charikar et al. [20] list the following global algorithms:

RePair [68]: In each round, the algorithm selects the maximal string s with the highest number of non-overlapping occurrences on the right-hand side of the current grammar. We remark that the original formulation of RePair [68] additionally requires $|s| = 2$.

Greedy [4, 5, 6]: In each round, the algorithm selects the maximal string that results in the largest reduction in the grammar size.

LongestMatch [67]: In each round, the algorithm selects the longest maximal string.

3.2 Nonglobal Algorithms

Sequential [99]: Process the input left-to-right. In each step, first compute the longest prefix of the remaining suffix of the input that is equal to $\exp(N)$ for some secondary nonterminal N existing in the grammar, and append N to the definition of the start rule. If there is no such prefix, append the next symbol from the input. If now there exists a pair of symbols AB on the right-hand side of the grammar with two non-overlapping occurrences (in [99], it is proved that there cannot be more such occurrences), create a new nonterminal M with $\text{rhs}(M) = AB$, and replace both the occurrences with M. Finally, if after this update there exists a nonterminal that is only used once on the right-hand side of the grammar, remove it from the grammar, replacing its occurrence with its definition.

Sequitur [76]: Process input string left-to-right. In each step, we first append the next symbol from the input into the definition of the start rule. We then apply the following reductions to the grammar as long as possible, each time choosing the reduction earliest in the list:

1. If the length-2 suffix AB of the definition of the starting nonterminal S is equal to the definition of some other nonterminal N, replace this length-2 suffix with N.
2. If the length-2 suffix AB of the definition of the starting nonterminal has another non-overlapping occurrence on the right-hand side of the grammar (in [76], it is proved that there cannot be more than one such occurrence), create a new nonterminal M with $\text{rhs}(M) = AB$, and replace both occurrences with M.
3. If there exists a nonterminal that is only used once on the right-hand side of the grammar, remove it from the grammar, replacing its occurrence with its definition.

Bisection [63, 75]: Let $w \in \Sigma^n$ be the input string with $n \geq 2$. The first step of the algorithm computes a set $\mathcal{S}$ of substrings of w as follows. First, we insert the string w itself into the set. If $n > 1$, we then compute the largest $k \geq 0$ such that $2^k < n$ and recursively process $w[1\,..\,2^k]$ and $w(2^k\,..\,n]$. After $\mathcal{S}$ is computed, we create a nonterminal for every $s \in \mathcal{S}$ satisfying $|s| \geq 2$. Each such nonterminal can be defined by a rule with exactly two symbols on the right.

LZ78 [101]: Let $w \in \Sigma^n$ be the input string. The LZ78 algorithm computes the factorization $w = f_1 f_2 \dots f_{z_{78}}$ (the elements of which are called *phrases*) such that for every $i \in [1\mathinner{.\,.} z_{78}]$, it holds either that $f_i \in \Sigma$ (if $w[|f_1 f_2 \dots f_{i-1}| + 1]$ is the leftmost occurrence of that symbol in w), or f_i is the longest prefix of $f_i \dots f_{z_{78}}$ such that there exists $i' \in [1\mathinner{.\,.} i)$ satisfying $f_{i'} c = f_i$ for some $c \in \Sigma$. This parsing can be easily encoded as an SLG of size $3z_{78}$.

LZD [45]: Let $w \in \Sigma^n$ be the input string. The LZD algorithm factorizes w into $f_1 f_2 \cdots f_m$ such that $f_0 = \varepsilon$, and for $1 \leq i \leq m$, $f_i = f_{i_1} f_{i_2}$ where f_{i_1} is the longest prefix of $w[k\mathinner{.\,.} n]$ with $f_{i_1} \in \{f_j : 1 \leq j < i\} \cup \Sigma$, f_{i_2} is the longest prefix of $w[k + |f_{i_1}|\mathinner{.\,.} n]$ with $f_{i_2} \in \{f_j : 0 \leq j < i\} \cup \Sigma$, and $k = |f_1 \cdots f_{i-1}| + 1$. Intuitively, at step i, $1 \leq i \leq m$, LZD computes f_{i_1} as the longest prefix of the unprocessed string among $f_1, ..., f_{i-1}$ or Σ. It then analogously computes f_{i_2} for remaining suffix of w (or sets $f_{i_2} = \varepsilon$ if the remaining suffix is empty). The ith phrase is then defined as $f_i = f_{i_1} f_{i_2}$. Note, that we can represent this factorization as an SLG by creating a nonterminal N_i for each factor f_i, and then creating the starting nonterminal S with $N_1 \cdots N_m$ as the definition. The size of this SLG is $3m$.

4 Technical Overview

Due to space constraints, here we present only the overview of the basic grammar boosting (for data structures).

4.1 The Framework of Verbin and Yu

The study of data structure lower bounds on grammar-compressed strings was pioneered by Verbin and Yu [97]. Below, we provide a summary of their techniques.

Definition 4.1. [Verbin and Yu [97]] Let $\mathcal{P} \subseteq [1\mathinner{.\,.} m]^2$ be a set of $|\mathcal{P}| = m$ points on an $m \times m$ grid. By $A(\mathcal{P})$, we denote a binary string of length m^2 defined such that for every $x, y \in [1\mathinner{.\,.} m]$,

$$A(\mathcal{P})[x + (y-1)m] = |\{(x', y') \in \mathcal{P} : x' \leq x \text{ and } y' \leq y\}| \bmod 2.$$

Verbin and Yu called $A(\mathcal{P})$ the *answer string* as it encodes the answers for all possible parity range counting queries on the set $\mathcal{P}$. Any such query, given $(x, y) \in [1\mathinner{.\,.} m]^2$, returns the parity of the number of points from $\mathcal{P}$ in the range $[1\mathinner{.\,.} x] \times [1\mathinner{.\,.} y]$ (note that rows are enumerated from bottom to top). Verbin and Yu proved the following result.

Lemma 4.1. (Verbin and Yu [97]) *Assume that m is a power of two. Let $\mathcal{P} \subseteq [1\mathinner{.\,.} m]^2$ be a set of $|\mathcal{P}| = m$ points on an $m \times m$ grid. There exists an admissible SLG $G = (V, \Sigma, R, S)$ (Definition 2.1) of height $\mathcal{O}(\log m)$ such that $L(G) = \{A(\mathcal{P})\}$ and $|G| = \mathcal{O}(m \log m)$.*

Theorem 4.1. (Pătraşcu [84]) *In the cell-probe model, there is no data structure that, for every set $\mathcal{P}$ of $|\mathcal{P}| = m$ points on an $m \times m$ grid, achieves $\mathcal{O}(m \log^c m)$ space (where $c = \mathcal{O}(1)$) and implements parity range counting queries on $\mathcal{P}$ in $o(\log m / \log\log m)$ time.*

Theorem 4.2. (Verbin and Yu [97]) *In the cell-probe model, there is no data structure that, for every string T of length N and every SLG G of T such that $L(G) = \{T\}$, achieves $\mathcal{O}(|G| \log^c N)$ space (where $c = \mathcal{O}(1)$) and implements random access queries to T in $o(\log N / \log\log N)$ time.*

The key idea in the proof of the above fact is as follows. Suppose that there exists a data structure D that, given any SLG G for a string $T \in \Sigma^N$, uses $\mathcal{O}(|G| \log^c N)$ space (where $c = \mathcal{O}(1)$) and answers random access queries on T in $o(\log N / \log\log N)$ time. Let $\mathcal{P} \subseteq [1\mathinner{.\,.} m]^2$ be any set of $|\mathcal{P}| = m$ points on an $m \times m$ grid. Assume for simplicity that m is a power of two (otherwise, letting m' be the smallest power of two satisfying $m' \geq m$, we apply the proof for $\mathcal{P}' = \mathcal{P} \cup \{(p, p)\}_{p \in (m\mathinner{.\,.} m']}$; note that the answer to any range counting query on $\mathcal{P}$ is equal to the answer on $\mathcal{P}'$). By Lemma 4.1, there exists an admissible SLG $G_{\mathcal{P}} = (V_{\mathcal{P}}, \{0, 1\}, R_{\mathcal{P}}, S_{\mathcal{P}})$ such that $L(G_{\mathcal{P}}) = \{A(\mathcal{P})\}$ is an answer string for $\mathcal{P}$ (Definition 4.1), and it holds $|G_{\mathcal{P}}| = \mathcal{O}(m \log m)$. Recall that $|A(\mathcal{P})| = m^2$. Let D' denote the structure D for $G_{\mathcal{P}}$. By $|G_{\mathcal{P}}| = \mathcal{O}(m \log m)$ and the assumption, D uses $\mathcal{O}(|G_{\mathcal{P}}| \log^c |A(\mathcal{P})|) = \mathcal{O}(m \log m \log^c(m^2)) = \mathcal{O}(m \log^{1+c} m)$ space, and implements random access to $A(\mathcal{P})$ in $o(\log(m^2)/\log\log(m^2)) = o(\log m/\log\log m)$ time. Given D' and any $(x, y) \in [1\,..\,m]^2$, we can thus answer in $o(\log m/\log\log m)$ the parity range query on $\mathcal{P}$ with arguments (x, y) by issuing a random access query on $A(\mathcal{P})$ with position $j = x + (y-1)m$. Thus, the existence of D' contradicts Theorem 4.1.

4.2 Grammar Boosting

The Main Idea We now describe our new technique. Consider any $T \in \Sigma^N$ and assume that there exists a grammar $G = (V, \Sigma, R, S)$ such that $L(G) = \{T\}$ and $|G| = n$. Assume that G is *admissible* (Definition 2.1). Most grammars that we start with will satisfy this property, but any grammar can be transformed into an admissible grammar generating the same string without asymptotically increasing its size. Consider any ordering $N_1, \ldots, N_{|V|}$ of nonterminals in V such that $|\mathrm{exp}_G(N_1)| \leq \cdots \leq |\mathrm{exp}_G(N_{|V|})|$. Let $\Sigma' = \Sigma \cup \{\$_i : i \in [1\,..\,|V|]\}$ and $G' = (V, \Sigma', R', S)$ be a grammar with the same set of nonterminals and the starting nonterminal as G, but with a unique sentinel symbol in every definition, i.e., such that for every $i \in [1\,..\,|V|]$, it holds $\mathrm{rhs}_{G'}(N_i) = A\$_iB$, where $A, B \in V \cup \Sigma$ are such that $\mathrm{rhs}_G(N_i) = AB$. Let $T' = \bigodot_{j=1,\ldots,|V|} \mathrm{exp}_{G'}(N_j) \cdot \#_{2j-1} \cdot \mathrm{exp}_{G'}(N_j) \cdot \#_{2j}$.

Observation: $|T'| = \mathcal{O}(|T|^2)$ and T is an easily identifiable subsequence of T'. Note that for every $X \in V$, we have $|\mathrm{exp}_{G'}(X)| = 2|\mathrm{exp}_G(X)| - 1$ and, letting $m = |\mathrm{exp}_G(X)|$, it holds $\mathrm{exp}_{G'}(X)[2j-1] = \mathrm{exp}_G(X)[j]$ for every $j \in [1\,..\,m]$. By definition of T', we therefore have $|T'| = 4\sum_{X \in V} |\mathrm{exp}_G(X)|$. Consequently, since for every $X \in V$ it holds $|\mathrm{exp}_G(X)| \leq |T|$, and $|V| \leq |T|$, we obtain $|T'| \leq 4|T|^2$. For the second claim, note that since there exists $j \in [1\,..\,|V|]$ such that $S = N_j$, it follows by the above that for some $\delta \geq 0$, we have $T[j] = T'[\delta + 2j - 1]$, where $j \in [1\,..\,|T|]$.

By the above observation, T' is plain enough that we can use it to access symbols of T without incurring any penalty in the runtime. We now outline how to prove that T' is simultaneously structured strongly enough, so that the algorithms studied in the paper compress it to size $\mathcal{O}(n)$.

Analysis of Nonglobal Algorithms As an illustration, we consider the processing of T' using SEQUENTIAL (see Section 3.2). Denote $\Sigma'' = \Sigma' \cup \{\#_j : j \in [1\,..\,2|V|]\}$. For every $k \in [0\,..\,|V|]$, let $G_k = (V_k, \Sigma'', R_k, S_k)$ be such that $V_k = \{N_1, \ldots, N_k, S_k\}$, for every $i \in [1\,..\,k]$, $\mathrm{rhs}_{G_k}(N_i) = \mathrm{rhs}_{G'}(N_i)$, and $\mathrm{rhs}_{G_k}(S_k) = \bigodot_{i=1,\ldots,k} N_i \cdot \#_{2i-1} \cdot N_i \cdot \#_{2i}$. We claim that after $8k$ steps of SEQUENTIAL, the algorithm has processed the prefix $\bigodot_{j=1,\ldots,k} \mathrm{exp}_{G'}(N_j) \cdot \#_{2j-1} \cdot \mathrm{exp}_{G'}(N_j) \cdot \#_{2j}$ of T', and the resulting grammar is isomorphic to G_k. We proceed by induction on k. The inductive base is easily verified. To show the inductive step, let $A, B \in V \cup \Sigma$ be such that $\mathrm{rhs}_{G'}(N_k) = A \cdot \$_k \cdot B$ and assume that SEQUENTIAL processed $\bigodot_{j=1,\ldots,k-1} \mathrm{exp}_{G'}(N_j) \cdot \#_{2j-1} \cdot \mathrm{exp}_{G'}(N_j) \cdot \#_{2j}$. Thus, $\mathrm{exp}_{G'}(A) \cdot \$_k \cdot \mathrm{exp}_{G'}(B) \cdot \#_{2k-1} \cdot \mathrm{exp}_{G'}(A) \cdot \$_k \cdot \mathrm{exp}_{G'}(B) \cdot \#_{2k}$ is the prefix of the remaining suffix. Note that during the next five steps, we process $\mathrm{exp}_{G'}(A) \cdot \$_k \cdot \mathrm{exp}_{G'}(B) \cdot \#_{2k-1} \cdot \mathrm{exp}_{G'}(A)$, and simply append five symbols $A'\$_kB'\#_{2k-1}A'$ (where A' and B', respectively, correspond to $\mathrm{exp}_{G'}(A)$ and $\mathrm{exp}_{G'}(B)$) to the definition of the starting nonterminal. Next, we create a new nonterminal X capturing the repetition of $A'\$_k$. In the seventh step, we again create a new nonterminal X' corresponding to the repetition of XB', and then remove X (it now occurs only once). Finally, we append $\#_{2k}$. The result is isomorphic to G_k. The high-level analysis of SEQUITUR is similar, except each step involves many smaller substeps (in which intermediate grammars are partially completed versions of G_k).

Analysis of Global Algorithms We now outline the proof that all global algorithms on T' output the same grammar as nonglobal algorithms. The key difficulty in the analysis, compared to nonglobal algorithms, is that replacements leading up to the grammar isomorphic with $G_{|V|}$ do not occur in order. We thus need to generalize the class of intermediate grammars. We show that it suffices to consider $2^{|V|}$ grammars. We define them as follows.

- Let $\{M_i : i \in [1\,..\,|V|]\}$ be a set of fresh variables, i.e., such that $\{M_i\}_{i\in[1..|V|]} \cap \{N_i\}_{i\in[1..|V|]} = \emptyset$. For every $X \in V \cup \Sigma$ and $\mathcal{I} \subseteq \{1, \ldots, |V|\}$, by $\mathrm{bexp}_{\mathcal{I}}(X)$ we denote a string obtained by starting with X and repeatedly expanding the nonterminals (according to their definition in G') until only symbols in $\Sigma' \cup \{N_i\}_{i\in\mathcal{I}}$ are left. Each occurrence of the remaining nonterminal from $\{N_i\}_{i\in\mathcal{I}}$ is then replaced with the matching symbol from $\{M_i\}_{i\in\mathcal{I}}$.
- For every $\mathcal{I} \subseteq \{1, \ldots, |V|\}$, we let $G_{\mathcal{I}} = (V_{\mathcal{I}}, \Sigma'', R_{\mathcal{I}}, S)$, where $V_{\mathcal{I}} = \{S\} \cup \{M_i\}_{i\in\mathcal{I}}$. For any $i \in \mathcal{I}$, we let $\mathrm{rhs}_{G_{\mathcal{I}}}(M_i) = \mathrm{bexp}_{\mathcal{I}}(X) \cdot \$_i \cdot \mathrm{bexp}_{\mathcal{I}}(Y)$, where $X, Y \in V \cup \Sigma$ are such that $\mathrm{rhs}_G(N_i) = XY$. We also set

$\text{rhs}_{G_{\mathcal{I}}}(S) = \bigodot_{i=1,\ldots,|V|} \text{bexp}_{\mathcal{I}}(N_i) \cdot \#_{2i-1} \cdot \text{bexp}_{\mathcal{I}}(N_i) \cdot \#_{2i}$. We denote $\mathbb{G} = \{G_{\mathcal{I}} : \mathcal{I} \subseteq \{1,\ldots,|V|\}\}$. Note that $|\mathbb{G}| = 2^{|V|}$ and $L(G_{\mathcal{I}}) = \{T'\}$ holds for every $\mathcal{I} \subseteq \{1,\ldots,|V|\}$. Observe also that the initial grammar that every global algorithm starts with when processing T' (see Section 3.1), is isomorphic with $G_{\emptyset}$.

Observation 1: If s is maximal with respect to $G_{\mathcal{I}}$, then $s = \text{bexp}_{\mathcal{I}}(N_i)$ for some $i \in \{1,\ldots,|V|\} \setminus \mathcal{I}$. Each symbol in $\{\#_i\}_{i\in[1\,..\,2|V|]}$ occurs once on the right-hand side of $G_{\mathcal{I}}$. On the other hand, every second symbol in the remaining substrings of $G_{\mathcal{I}}$ belongs to $\{\$_i\}_{i\in[1\,..\,|V|]}$. Thus, by $|s| \geq 2$, one of them occurs in s. It is easy to check that every symbol in $\{\$_i\}_{i\in\mathcal{I}}$ occurs once on the right-hand side of $G_{\mathcal{I}}$. Thus, s contains $\$_i$ for some $i \in \{1,\ldots,|V|\} \setminus \mathcal{I}$. An inductive argument shows that every occurrence of $\$_i$ for such i can be extended into an occurrence of $\text{bexp}_{\mathcal{I}}(N_i)$. String $\text{bexp}_{\mathcal{I}}(N_i)$ must therefore be a substring of s. Choosing as i the maximal $t \in \{1,\ldots,|V|\} \setminus \mathcal{I}$ such that $\$_t$ occurs in s, we thus have $s = \text{bexp}_{\mathcal{I}}(N_i)$, since every occurrence of $\text{bexp}_{\mathcal{I}}(N_i)$ on the right-hand side of $G_{\mathcal{I}}$ is surrounded by either $\$_{i'}$ with $i' > i$, or a symbol from $\{\#_i\}_{i\in[1\,..\,2|V|]}$.

Observation 2: The output of one step of every global algorithm on $G_{\mathcal{I}}$ with $\text{bexp}_{\mathcal{I}}(N_i)$ as a maximal string is isomorphic to $G_{\mathcal{I}\cup\{i\}}$. Denote $\mathcal{I}' = \mathcal{I} \cup \{i\}$ and $s = \text{bexp}_{\mathcal{I}}(N_i)$. First, we observe that by an inductive argument it follows that replacing every occurrence of M_i on the right-hand side of $G_{\mathcal{I}'}$ with s, and removing nonterminal M_i, results in $G_{\mathcal{I}}$. On the other hand, no two occurrences of s on the right-hand side of $G_{\mathcal{I}}$ are overlapping. These two together imply the claim, since the output of a single step of a global algorithm is then not determined by the order of replacements.

Observation 3: There exists a maximal string with respect to $G_{\mathcal{I}}$ if and only if $\mathcal{I} \neq \{1,\ldots,|V|\}$. The first implication follows from above. For the second implication, observe that if $\mathcal{I} \neq \{1,\ldots,|V|\}$, then, letting $i \in \{1,\ldots,|V|\} \setminus \mathcal{I}$ and $s = \text{bexp}_{\mathcal{I}}(N_i)$, we have $|s| \geq 2$, and s has at least two non-overlapping occurrences on the right-hand side of $G_{\mathcal{I}}$ (in the definition of S). Thus, either s is maximal, or it can be extended into a maximal string (Definition 3.1).

By the above, the intermediate grammars computed by every global algorithm on T' are isomorphic to a chain $G_{\emptyset}, G_{\mathcal{I}_1}, \ldots, G_{\mathcal{I}_{|V|}}$ such that $\mathcal{I}_1 \subsetneq \mathcal{I}_2 \subsetneq \cdots \subsetneq \mathcal{I}_{|V|}$. Thus, $\mathcal{I}_{|V|} = \{1,\ldots,|V|\}$, and hence the final grammar is isomorphic to $G_{\{1,\ldots,|V|\}}$, which has size $\mathcal{O}(|V|) = \mathcal{O}(n)$.

Putting Everything Together Theorem 1.1 follows from the above analysis as follows. Suppose that for some Alg as in Theorem 1.1, there exists a structure D that for any $T \in \Sigma^N$ uses $\mathcal{O}(|\text{Alg}(T)| \log^c N)$ space (where $c = \mathcal{O}(1)$) and answers random access queries on T in $o(\log N/\log\log N)$ time. Let $\mathcal{P} \subseteq [1\,..\,m]^2$ be any set of $|\mathcal{P}| = m$ points on an $m \times m$ grid. By Lemma 4.1, there exists an admissible grammar $G_{\mathcal{P}} = (V_{\mathcal{P}}, \{0,1\}, R_{\mathcal{P}}, S_{\mathcal{P}})$ such that $L(G_{\mathcal{P}}) = \{A(\mathcal{P})\}$ is the answer string for $\mathcal{P}$ (Definition 4.1), and it holds $|G_{\mathcal{P}}| = \mathcal{O}(m \log m)$. Let us now consider the string T' (defined as in the beginning of Section 4.2) for $T = A(\mathcal{P})$ and $G = G_{\mathcal{P}}$. As noted in the initial observation of Section 4.2, it holds $|T'| = \mathcal{O}(|T|^2) = \mathcal{O}(m^4)$, and there exists $\delta \geq 0$, such that $T[j] = T'[\delta + 2j - 1]$, for every $j \in [1\,..\,|T|]$. Let $G_{T'} = \text{Alg}(T')$ be the output of Alg on T'. By the above discussion, we have $|G_{T'}| = \mathcal{O}(|G_{\mathcal{P}}|) = \mathcal{O}(m \log m)$. Let D' denote a data structure consisting of the following two components:

1. The structure D for string T'. By $|G_{T'}| = \mathcal{O}(m \log m)$ and the above assumption, the structure D uses $\mathcal{O}(|\text{Alg}(T')| \log^c |T'|) = \mathcal{O}(m \log m \log^c(m^4)) = \mathcal{O}(m \log^{1+c} m)$ space, and implements random access to T' in $o(\log|T'|/\log\log|T'|) = o(\log(m^4)/\log\log(m^4)) = o(\log m/\log\log m)$ time,
2. The position $\delta \geq 0$, as defined above.

Observe that given the structure D' and any $(x,y) \in [1\,..\,m]^2$, we can answer in $o(\log m/\log\log m)$ the parity range query on $\mathcal{P}$ with arguments (x,y) by a random access query to T' with position $j = \delta + 2j' - 1$, where $j' = x + (y-1)m$. Thus, the existence of D' contradicts Theorem 4.1.

References

[1] Amir Abboud, Arturs Backurs, Karl Bringmann, and Marvin Künnemann. Fine-grained complexity of analyzing compressed data: Quantifying improvements over decompress-and-solve. In Chris Umans, editor, *58th IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 192–203. IEEE Computer Society, 2017. doi:10.1109/FOCS.2017.26.

[2] Amir Abboud, Arturs Backurs, Karl Bringmann, and Marvin Künnemann. Impossibility results for grammar-compressed linear algebra. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Annual Conference on Neural Information Processing Systems 2020 (NeurIPS)*, 2020. URL: https://proceedings.neurips.cc/paper/2020/hash/645e6bfdd05d1a69c5e47b20f0a91d46-Abstract.html.

[3] Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. Tight hardness results for LCS and other sequence similarity measures. In Venkatesan Guruswami, editor, *IEEE 56th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 59–78. IEEE Computer Society, 2015. doi:10.1109/FOCS.2015.14.

[4] Alberto Apostolico and Stefano Lonardi. Some theory and practice of greedy off-line textual substitution. In *Data Compression Conference (DCC)*, pages 119–128. IEEE Computer Society, 1998. doi:10.1109/DCC.1998.672138.

[5] Alberto Apostolico and Stefano Lonardi. Compression of biological sequences by greedy off-line textual substitution. In *Data Compression Conference (DCC)*, pages 143–152. IEEE Computer Society, 2000. doi:10.1109/DCC.2000.838154.

[6] Alberto Apostolico and Stefano Lonardi. Off-line compression by greedy textual substitution. *Proceedings of the IEEE*, 88(11):1733–1744, 2000. doi:10.1109/5.892709.

[7] Golnaz Badkobeh, Travis Gagie, Shunsuke Inenaga, Tomasz Kociumaka, Dmitry Kosolobov, and Simon J. Puglisi. On two LZ78-style grammars: Compression bounds and compressed-space computation. In Gabriele Fici, Marinella Sciortino, and Rossano Venturini, editors, *24th International Symposium on String Processing and Information Retrieval (SPIRE)*, pages 51–67. Springer, 2017. doi:10.1007/978-3-319-67428-5_5.

[8] Hideo Bannai, Momoko Hirayama, Danny Hucke, Shunsuke Inenaga, Artur Jez, Markus Lohrey, and Carl Philipp Reh. The smallest grammar problem revisited. *IEEE Trans. Inf. Theory*, 67(1):317–328, 2021. doi:10.1109/TIT.2020.3038147.

[9] Djamal Belazzougui, Manuel Cáceres, Travis Gagie, Pawel Gawrychowski, Juha Kärkkäinen, Gonzalo Navarro, Alberto Ordóñez Pereira, Simon J. Puglisi, and Yasuo Tabei. Block trees. *Journal of Computer and System Sciences*, 117:1–22, 2021. doi:10.1016/j.jcss.2020.11.002.

[10] Djamal Belazzougui, Travis Gagie, Pawel Gawrychowski, Juha Kärkkäinen, Alberto Ordóñez Pereira, Simon J. Puglisi, and Yasuo Tabei. Queries on LZ-bounded encodings. In Ali Bilgin, Michael W. Marcellin, Joan Serra-Sagristà, and James A. Storer, editors, *2015 Data Compression Conference (DCC)*, pages 83–92. IEEE, 2015. doi:10.1109/DCC.2015.69.

[11] Bonnie Berger, Noah M. Daniels, and Y. William Yu. Computational biology in the 21st century: Scaling with compressive algorithms. *Communication of the ACM*, 59(8):72–80, jul 2016. doi:10.1145/2957324.

[12] Philip Bille, Inge Li Gørtz, and Nicola Prezza. Space-efficient Re-Pair compression. In Ali Bilgin, Michael W. Marcellin, Joan Serra-Sagristà, and James A. Storer, editors, *2017 Data Compression Conference (DCC)*, pages 171–180. IEEE, 2017. doi:10.1109/DCC.2017.24.

[13] Philip Bille, Gad M. Landau, Rajeev Raman, Kunihiko Sadakane, Srinivasa Rao Satti, and Oren Weimann. Random access to grammar-compressed strings and trees. *SIAM Journal on Computing*, 44(3):513–539, 2015. doi:10.1137/130936889.

[14] Anselm Blumer, Janet A. Blumer, David Haussler, Ross M. McConnell, and Andrzej Ehrenfeucht. Complete inverted files for efficient text retrieval and analysis. *Journal of the ACM*, 34(3):578–595, 1987. doi:10.1145/28869.28873.

[15] Karl Bringmann, Philip Wellnitz, and Marvin Künnemann. Few matches or almost periodicity: Faster pattern matching with mismatches in compressed texts. In Timothy M. Chan, editor, *30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1126–1145. SIAM, 2019. doi:10.1137/1.9781611975482.69.

[16] Nieves R. Brisaboa, Adrián Gómez-Brandón, Miguel A. Martínez-Prieto, and José R. Paramá. 3DGraCT: A grammar-based compressed representation of 3D trajectories. In Travis Gagie, Alistair Moffat, Gonzalo Navarro, and Ernesto Cuadros-Vargas, editors, *25th International Symposium on String Processing and Information Retrieval (SPIRE)*, pages 102–116. Springer, 2018. doi:10.1007/978-3-030-00479-8_9.

[17] Nieves R. Brisaboa, Adrián Gómez-Brandón, Gonzalo Navarro, and José R. Paramá. GraCT: A grammar based compressed representation of trajectories. In Shunsuke Inenaga, Kunihiko Sadakane, and Tetsuya Sakai, editors, *23rd International Symposium on String Processing and Information Retrieval (SPIRE)*, pages 218–230, 2016. doi:10.1007/978-3-319-46049-9_21.

[18] Michael Burrows and David J. Wheeler. A block-sorting lossless data compression algorithm. Technical Report 124, Digital Equipment Corporation, Palo Alto, California, 1994. URL: https://www.hpl.hp.com/techreports/Compaq-DEC/SRC-RR-124.pdf.

[19] Panagiotis Charalampopoulos, Tomasz Kociumaka, and Philip Wellnitz. Faster approximate pattern matching: A unified approach. In Sandy Irani, editor, *61st IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 978–989. IEEE, 2020. doi:10.1109/FOCS46700.2020.00095.

[20] Moses Charikar, Eric Lehman, Ding Liu, Rina Panigrahy, Manoj Prabhakaran, Amit Sahai, and Abhi Shelat. The smallest grammar problem. *IEEE Transactions on Information Theory*, 51(7):2554–2576, 2005. doi:10.1109/TIT.2005.850116.

[21] Anders Roy Christiansen, Mikko Berggren Ettienne, Tomasz Kociumaka, Gonzalo Navarro, and Nicola Prezza. Optimal-time dictionary-compressed indexes. *ACM Transactions on Algorithms*, 17(1):8:1–8:39, 2021. doi:10.1145/3426473.

[22] Francisco Claude and Gonzalo Navarro. Self-indexed grammar-based compression. *Fundamenta Informaticae*, 111(3):313–337, 2011. doi:10.3233/FI-2011-565.

[23] Francisco Claude and Gonzalo Navarro. Improved grammar-based compressed indexes. In Liliana Calderón-Benavides, Cristina N. González-Caro, Edgar Chávez, and Nivio Ziviani, editors, *19th International Symposium on String Processing and Information Retrieval (SPIRE)*, pages 180–192. Springer, 2012. doi:10.1007/978-3-642-34109-0_19.

[24] Francisco Claude, Gonzalo Navarro, and Alejandro Pacheco. Grammar-compressed indexes with logarithmic search time. *Journal of Computer and System Sciences*, 118:53–74, 2021. doi:10.1016/j.jcss.2020.12.001.

[25] European Commission. 1+ Million Genomes Initiative. https://digital-strategy.ec.europa.eu/en/policies/1-million-genomes.

[26] Sebastian Deorowicz, Agnieszka Danek, and Heng Li. AGC: Compact representation of assembled genomes. *bioRxiv*, 2022. doi:10.1101/2022.04.07.487441.

[27] Diego Díaz-Domínguez, Gonzalo Navarro, and Alejandro Pacheco. An LMS-based grammar self-index with local consistency properties. In Thierry Lecroq and Hélène Touzet, editors, *28th International Symposium on String Processing and Information Retrieval (SPIRE)*, pages 100–113. Springer, 2021. doi:10.1007/978-3-030-86692-1_9.

[28] Akashnil Dutta, Reut Levi, Dana Ron, and Ronitt Rubinfeld. A simple online competitive adaptation of Lempel-Ziv compression with efficient random access support. In Ali Bilgin, Michael W. Marcellin, Joan Serra-Sagristà, and James A. Storer, editors, *2013 Data Compression Conference (DCC)*, pages 113–122. IEEE, 2013. doi:10.1109/DCC.2013.19.

[29] Paolo Ferragina, Giovanni Manzini, Travis Gagie, Dominik Köppl, Gonzalo Navarro, Manuel Striani, and Francesco Tosoni. Improving matrix-vector multiplication via lossless grammar-compressed matrices. *Proc. VLDB Endow.*, 15(10):2175–2187, 2022. URL: https://www.vldb.org/pvldb/vol15/p2175-tosoni.pdf.

[30] Edward R Fiala and Daniel H Greene. Data compression with finite windows. *Communications of the ACM*, 32(4):490–505, 1989. doi:10.1145/63334.63341.

[31] Isamu Furuya, Takuya Takagi, Yuto Nakashima, Shunsuke Inenaga, Hideo Bannai, and Takuya Kida. MR-RePair: Grammar compression based on maximal repeats. In Ali Bilgin, Michael W. Marcellin, Joan Serra-Sagristà, and James A. Storer, editors, *Data Compression Conference (DCC)*, pages 508–517. IEEE, 2019. doi:10.1109/DCC.2019.00059.

[32] Philip Gage. A new algorithm for data compression. *C Users Journal*, 12(2):23–38, feb 1994. URL: https://dl.acm.org/doi/abs/10.5555/177910.177914.

[33] Travis Gagie, Pawel Gawrychowski, Juha Kärkkäinen, Yakov Nekrich, and Simon J. Puglisi. A faster grammar-based self-index. In Adrian-Horia Dediu and Carlos Martín-Vide, editors, *6th International Conference on Language and Automata Theory and Applications (LATA)*, pages 240–251. Springer, 2012. doi:10.1007/978-3-642-28332-1_21.

[34] Travis Gagie, Pawel Gawrychowski, Juha Kärkkäinen, Yakov Nekrich, and Simon J. Puglisi. LZ77-based self-indexing with faster pattern matching. In Alberto Pardo and Alfredo Viola, editors, *11th Latin American Symposium on Theoretical Informatics (LATIN)*, pages 731–742. Springer, 2014. doi:10.1007/978-3-642-54423-1_63.

[35] Travis Gagie, Tomohiro I, Giovanni Manzini, Gonzalo Navarro, Hiroshi Sakamoto, and Yoshimasa Takabatake. Rpair: Rescaling RePair with Rsync. In Nieves R. Brisaboa and Simon J. Puglisi, editors, *26th International Symposium on String Processing and Information Retrieval (SPIRE)*, pages 35–44. Springer, 2019. doi:10.1007/978-3-030-32686-9_3.

[36] Travis Gagie, Gonzalo Navarro, and Nicola Prezza. On the approximation ratio of Lempel-Ziv parsing. In Michael A. Bender, Martin Farach-Colton, and Miguel A. Mosteiro, editors, *13th Latin American Symposium on Theoretical Informatics (LATIN)*, pages 490–503. Springer, 2018. doi:10.1007/978-3-319-77404-6_36.

[37] Travis Gagie, Gonzalo Navarro, and Nicola Prezza. Fully functional suffix trees and optimal text searching in BWT-runs bounded space. *Journal of the ACM*, 67(1):1–54, 2020. doi:10.1145/3375890.

[38] Moses Ganardi and Pawel Gawrychowski. Pattern matching on grammar-compressed strings in linear time. In Joseph (Seffi) Naor and Niv Buchbinder, editors, *2022 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2833–2846. SIAM, 2022. doi:10.1137/1.9781611977073.110.

[39] Moses Ganardi, Artur Jez, and Markus Lohrey. Balancing straight-line programs. *Journal of the ACM*, 68(4):27:1–27:40, 2021. doi:10.1145/3457389.

[40] Michal Ganczorz and Artur Jez. Improvements on Re-Pair grammar compressor. In Ali Bilgin, Michael W. Marcellin, Joan Serra-Sagristà, and James A. Storer, editors, *2017 Data Compression Conference (DCC)*, pages 181–190. IEEE, 2017. doi:10.1109/DCC.2017.52.

[41] Arun Ganesh, Tomasz Kociumaka, Andrea Lincoln, and Barna Saha. How compression and approximation affect efficiency in string distance measures. In Joseph (Seffi) Naor and Niv Buchbinder, editors, *2022 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2867–2919. SIAM, 2022. doi:10.1137/1.9781611977073.112.

[42] Pawel Gawrychowski. Optimal pattern matching in LZW compressed strings. In Dana Randall, editor, *22nd Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 362–372. SIAM, 2011. doi:10.1137/1.9781611973082.29.

[43] Pawel Gawrychowski, Adam Karczmarz, Tomasz Kociumaka, Jakub Lacki, and Piotr Sankowski. Optimal dynamic strings. In Artur Czumaj, editor, *29th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1509–1528. SIAM, 2018. Full version: arxiv.org/abs/1511.02612. doi:10.1137/1.9781611975031.99.

[44] Genomics England. The 100,000 Genomes Project. https://www.genomicsengland.co.uk/about-genomics-england/the-100000-genomes-project/.

[45] Keisuke Goto, Hideo Bannai, Shunsuke Inenaga, and Masayuki Takeda. LZD factorization: Simple and practical online grammar compression with variable-to-fixed encoding. In Ferdinando Cicalese, Ely Porat, and Ugo Vaccaro, editors, *26th Annual Symposium on Combinatorial Pattern Matching (CPM)*, pages 219–230. Springer, 2015. doi:10.1007/978-3-319-19929-0_19.

[46] Dan Greenfield, Vaughan Wittorff, and Michael Hultner. The importance of data compression in the field of genomics. *IEEE Pulse*, 10(2):20–23, 2019. doi:10.1109/MPULS.2019.2899747.

[47] Danny Hermelin, Gad M. Landau, Shir Landau, and Oren Weimann. Unified compression-based acceleration of edit-distance computation. *Algorithmica*, 65(2):339–353, 2013. doi:10.1007/s00453-011-9590-6.

[48] Mikel Hernaez, Dmitri Pavlichin, Tsachy Weissman, and Idoia Ochoa. Genomic data compression. *Annual Review of Biomedical Data Science*, 2:19–37, 2019. doi:10.1146/annurev-biodatasci-072018-021229.

[49] Tomohiro I. Longest common extensions with recompression. In Juha Kärkkäinen, Jakub Radoszewski, and Wojciech Rytter, editors, *28th Annual Symposium on Combinatorial Pattern Matching (CPM)*, pages 18:1–18:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPICS.CPM.2017.18.

[50] Artur Jeż. Faster fully compressed pattern matching by recompression. *ACM Transactions on Algorithms*, 11(3):20:1–20:43, 2015. doi:10.1145/2631920.

[51] Artur Jeż. A really simple approximation of smallest grammar. *Theoretical Computer Science*, 616:141–150, 2016. doi:10.1016/j.tcs.2015.12.032.

[52] Dominik Kempa. Optimal construction of compressed indexes for highly repetitive texts. In Timothy M. Chan, editor, *30th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1344–1357. SIAM, 2019. doi:10.1137/1.9781611975482.82.

[53] Dominik Kempa and Tomasz Kociumaka. Resolution of the Burrows-Wheeler transform conjecture. In Sandy Irani, editor, *61st IEEE Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1002–1013. IEEE, 2020. doi:10.1109/FOCS46700.2020.00097.

[54] Dominik Kempa and Tomasz Kociumaka. Dynamic suffix array with polylogarithmic queries and updates. In Stefano Leonardi and Anupam Gupta, editors, *54th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 1657–1670. ACM, 2022. doi:10.1145/3519935.3520061.

[55] Dominik Kempa and Tomasz Kociumaka. Breaking the O(n)-barrier in the construction of compressed suffix arrays and suffix trees. In Nikhil Bansal and Viswanath Nagarajan, editors, *2023 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 5122–5202. SIAM, 2023. doi:10.1137/1.9781611977554.CH187.

[56] Dominik Kempa and Tomasz Kociumaka. Collapsing the hierarchy of compressed data structures: Suffix arrays in optimal compressed space. In *IEEE 64th Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE Computer Society, 2023. URL: https://doi.org/10.48550/arXiv.2308.03635.

[57] Dominik Kempa and Dmitry Kosolobov. LZ-End parsing in compressed space. In Ali Bilgin, Michael W. Marcellin, Joan Serra-Sagristà, and James A. Storer, editors, *2017 Data Compression Conference (DCC)*, pages 350–359. IEEE, 2017. doi:10.1109/DCC.2017.73.

[58] Dominik Kempa and Ben Langmead. Fast and space-efficient construction of AVL grammars from the LZ77 parsing. In Petra Mutzel, Rasmus Pagh, and Grzegorz Herman, editors, *29th Annual European Symposium on Algorithms (ESA)*, pages 56:1–56:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.ESA.2021.56.

[59] Dominik Kempa and Nicola Prezza. At the roots of dictionary compression: String attractors. In Ilias Diakonikolas, David Kempe, and Monika Henzinger, editors, *50th Annual ACM SIGACT Symposium on Theory of Computing (STOC)*, pages 827–840. ACM, 2018. doi:10.1145/3188745.3188814.

[60] Dominik Kempa and Barna Saha. An upper bound and linear-space queries on the LZ-End parsing. In Joseph (Seffi) Naor and Niv Buchbinder, editors, *2022 ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 2847–2866. SIAM, 2022. doi:10.1137/1.9781611977073.111.

[61] Takuya Kida, Tetsuya Matsumoto, Yusuke Shibata, Masayuki Takeda, Ayumi Shinohara, and Setsuo Arikawa. Collage system: A unifying framework for compressed pattern matching. *Theoretical Computer Science*, 298(1):253–272, 2003. doi:10.1016/S0304-3975(02)00426-7.

[62] John C. Kieffer and En-Hui Yang. Grammar-based codes: A new class of universal lossless source codes. *IEEE Transactions on Information Theory*, 46(3):737–754, 2000. doi:10.1109/18.841160.

[63] John C. Kieffer, En-Hui Yang, Gregory J. Nelson, and Pamela C. Cosman. Universal lossless compression via multilevel pattern matching. *IEEE Transactions on Information Theory*, 46(4):1227–1245, July 2000. doi:10.1109/18.850665.

[64] Tomasz Kociumaka, Gonzalo Navarro, and Francisco Olivares. Near-optimal search time in δ-optimal space. In Armando Castañeda and Francisco Rodríguez-Henríquez, editors, *15th Latin American Symposium on Theoretical Informatics (LATIN)*, pages 88–103. Springer, 2022. doi:10.1007/978-3-031-20624-5_6.

[65] Tomasz Kociumaka, Gonzalo Navarro, and Nicola Prezza. Toward a definitive compressibility measure for repetitive sequences. *IEEE Trans. Inf. Theory*, 69(4):2074–2092, 2023. doi:10.1109/TIT.2022.3224382.

[66] Sebastian Kreft and Gonzalo Navarro. LZ77-like compression with fast random access. In James A. Storer and Michael W. Marcellin, editors, *2010 Data Compression Conference (DCC)*, pages 239–248. IEEE Computer Society, 2010. doi:10.1109/DCC.2010.29.

[67] J. Kevin Lanctôt, Ming Li, and En-Hui Yang. Estimating DNA sequence entropy. In David B. Shmoys, editor, *11th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 409–418. ACM/SIAM, 2000. URL: http://dl.acm.org/citation.cfm?id=338219.338586.

[68] N. Jesper Larsson and Alistair Moffat. Off-line dictionary-based compression. *Proceedings of the IEEE*, 88(11):1722–1732, 2000. doi:10.1109/5.892708.

[69] Markus Lohrey. Algorithmics on SLP-compressed strings: A survey. *Groups Complexity Cryptology*, 4(2):241–299, 2012. doi:10.1515/gcc-2012-0016.

[70] Markus Lohrey, Sebastian Maneth, and Roy Mennicke. XML tree structure compression using RePair. *Information Systems*, 38(8):1150–1167, 2013. doi:10.1016/j.is.2013.06.006.

[71] Takuya Mieno, Shunsuke Inenaga, and Takashi Horiyama. RePair grammars are the smallest grammars for Fibonacci words. In Hideo Bannai and Jan Holub, editors, *33rd Annual Symposium on Combinatorial Pattern Matching (CPM)*, pages 26:1–26:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPICS.CPM.2022.26.

[72] Gonzalo Navarro. *Compact data structures: A practical approach*. Cambridge University Press, Cambridge, UK, 2016. doi:10.1017/cbo9781316588284.

[73] Gonzalo Navarro. Indexing highly repetitive string collections, part I: Repetitiveness measures. *ACM Comput. Surv.*, 54(2):29:1–29:31, 2021. doi:10.1145/3434399.

[74] Gonzalo Navarro. Indexing highly repetitive string collections, part II: Compressed indexes. *ACM Comput. Surv.*, 54(2):26:1–26:32, 2021. doi:10.1145/3432999.

[75] Greg Nelson, John Kieffer, and Pamela Cosman. An interesting hierarchical lossless data compression algorithm. In *IEEE Information Theory Society Workshop*, 1995.

[76] Craig G. Nevill-Manning and Ian H. Witten. Identifying hierarchical structure in sequences: A linear-time algorithm. *Journal of Artificial Intelligence Research*, 7:67–82, 1997. doi:10.1613/jair.374.

[77] National Human Genome Research Institute (NIH). Genomic data science. https://www.genome.gov/about-genomics/fact-sheets/Genomic-Data-Science.

[78] Takaaki Nishimoto, Tomohiro I, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Fully dynamic data structure for LCE queries in compressed space. In Piotr Faliszewski, Anca Muscholl, and Rolf Niedermeier, editors, *41st International Symposium on Mathematical Foundations of Computer Science (MFCS)*, pages 72:1–72:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. doi:10.4230/LIPICS.MFCS.2016.72.

[79] Takaaki Nishimoto, Tomohiro I, Shunsuke Inenaga, Hideo Bannai, and Masayuki Takeda. Dynamic index and LZ factorization in compressed space. *Discret. Appl. Math.*, 274:116–129, 2020. doi:10.1016/j.dam.2019.01.014.

[80] Takaaki Nishimoto and Yasuo Tabei. Optimal-time queries on BWT-runs compressed indexes. In Nikhil Bansal, Emanuela Merelli, and James Worrell, editors, *48th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 101:1–101:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.ICALP.2021.101.

[81] Takaaki Nishimoto and Yasuo Tabei. R-enum: Enumeration of characteristic substrings in BWT-runs bounded space. In Pawel Gawrychowski and Tatiana Starikovskaya, editors, *32nd Annual Symposium on Combinatorial Pattern Matching (CPM)*, pages 21:1–21:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.CPM.2021.21.

[82] Carlos Ochoa and Gonzalo Navarro. RePair and all irreducible grammars are upper bounded by high-order empirical entropy. *IEEE Transactions on Information Theory*, 65(5):3160–3164, 2019. doi:10.1109/TIT.2018.2871452.

[83] Tatsuya Ohno, Kensuke Sakai, Yoshimasa Takabatake, Tomohiro I, and Hiroshi Sakamoto. A faster implementation of online RLBWT and its application to LZ77 parsing. *J. Discrete Alg.*, 52-53:18–28, 2018. doi:10.1016/j.jda.2018.11.002.

[84] Mihai Patrascu. Lower bounds for 2-dimensional range counting. In David S. Johnson and Uriel Feige, editors, *39th Annual ACM Symposium on Theory of Computing (STOC)*, pages 40–46. ACM, 2007. doi:10.1145/1250790.1250797.

[85] Mihai Patrascu and Mikkel Thorup. Time-space trade-offs for predecessor search. In Jon M. Kleinberg, editor, *38th Annual ACM Symposium on Theory of Computing (STOC)*, pages 232–240. ACM, 2006. doi:10.1145/1132516.1132551.

[86] Alberto Ordóñez Pereira, Gonzalo Navarro, and Nieves R. Brisaboa. Grammar compressed sequences with rank/select support. *Journal of Discrete Algorithms*, 43:54–71, 2017. doi:10.1016/j.jda.2016.10.001.

[87] Alberto Policriti and Nicola Prezza. From LZ77 to the run-length encoded Burrows-Wheeler transform, and back. In Juha Kärkkäinen, Jakub Radoszewski, and Wojciech Rytter, editors, *28th Annual Symposium on Combinatorial Pattern Matching (CPM)*, pages 17:1–17:10. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPICS.CPM.2017.17.

[88] Nicola Prezza. Optimal rank and select queries on dictionary-compressed text. In Nadia Pisanti and Solon P. Pissis, editors, *30th Annual Symposium on Combinatorial Pattern Matching (CPM)*, pages 4:1–4:12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICS.CPM.2019.4.

[89] Molly Przeworski, Richard R. Hudson, and Anna Di Rienzo. Adjusting the focus on human variation. *Trends in Genetics*, 16(7):296–302, 2000. doi:10.1016/S0168-9525(00)02030-8.

[90] Sofya Raskhodnikova, Dana Ron, Ronitt Rubinfeld, and Adam D. Smith. Sublinear algorithms for approximating string compressibility. *Algorithmica*, 65(3):685–709, 2013. doi:10.1007/s00453-012-9618-6.

[91] Wojciech Rytter. Application of Lempel–Ziv factorization to the approximation of grammar-based compression. *Theoretical Computer Science*, 302(1–3):211–222, 2003. doi:10.1016/S0304-3975(02)00777-6.

[92] Zachary D Stephens, Skylar Y Lee, Faraz Faghri, Roy H Campbell, Chengxiang Zhai, Miles J Efron, Ravishankar Iyer, Michael C Schatz, Saurabh Sinha, and Gene E Robinson. Big data: astronomical or genomical? *PLoS biology*, 13(7):e1002195, 2015. doi:10.1371/journal.pbio.1002195.

[93] James A. Storer and Thomas G. Szymanski. The macro model for data compression (extended abstract). In Richard J. Lipton, Walter A. Burkhard, Walter J. Savitch, Emily P. Friedman, and Alfred V. Aho, editors, *10th Annual ACM Symposium on Theory of Computing (STOC)*, pages 30–39. ACM, 1978. doi:10.1145/800133.804329.

[94] James A. Storer and Thomas G. Szymanski. Data compression via textual substitution. *Journal of the ACM*, 29(4):928–951, 1982. doi:10.1145/322344.322346.

[95] Alexander Tiskin. Fast distance multiplication of unit-monge matrices. *Algorithmica*, 71(4):859–888, 2015. doi:10.1007/s00453-013-9830-z.

[96] Leslie G. Valiant. General context-free recognition in less than cubic time. *J. Comput. Syst. Sci.*, 10(2):308–315, 1975. doi:10.1016/S0022-0000(75)80046-8.

[97] Elad Verbin and Wei Yu. Data structure lower bounds on random access to grammar-compressed strings. In Johannes Fischer and Peter Sanders, editors, *24th Annual Symposium on Combinatorial Pattern Matching (CPM)*, pages 247–258. Springer, 2013. doi:10.1007/978-3-642-38905-4_24.

[98] Terry A. Welch. A technique for high-performance data compression. *Computer*, 17(6):8–19, 1984. doi:10.1109/MC.1984.1659158.

[99] En-Hui Yang and John C. Kieffer. Efficient universal lossless data compression algorithms based on a greedy sequential grammar transform - part one: Without context models. *IEEE Transactions on Information Theory*, 46(3):755–777, 2000. doi:10.1109/18.841161.

[100] Jacob Ziv and Abraham Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, 23(3):337–343, 1977. doi:10.1109/TIT.1977.1055714.

[101] Jacob Ziv and Abraham Lempel. Compression of individual sequences via variable-rate coding. *IEEE Transactions on Information Theory*, 24(5):530–536, 1978. doi:10.1109/TIT.1978.1055934.