58th Annual IEEE Symposium on Foundations of Computer Science

Fine-Grained Complexity of Analyzing Compressed Data: Quantifying Improvements over Decompress-And-Solve

Amir Abboud
Computer Science Department
Stanford University
Palo Alto, CA, USA
abboud@cs.stanford.edu

Arturs Backurs
EECS
MIT
Cambridge, MA, USA
backurs@mit.edu

Karl Bringmann
Max Planck Institute for Informatics
Saarland Informatics Campus
Saarbrücken, Germany
kbringma@mpi-inf.mpg.de

Marvin Künnemann
Max Planck Institute for Informatics
Saarland Informatics Campus
Saarbrücken, Germany
marvin@mpi-inf.mpg.de

***Abstract*—Can we analyze data without decompressing it? As our data keeps growing, understanding the time complexity of problems on *compressed* inputs, rather than in convenient uncompressed forms, becomes more and more relevant. Suppose we are given a compression of size n of data that originally has size N, and we want to solve a problem with time complexity $T(\cdot)$. The naïve strategy of "decompress-and-solve" gives time $T(N)$, whereas "the gold standard" is time $T(n)$: to analyze the compression as efficiently as if the original data was small.**

We restrict our attention to data in the form of a string (text, files, genomes, etc.) and study the most ubiquitous tasks. While the challenge might seem to depend heavily on the specific compression scheme, most methods of practical relevance (Lempel-Ziv-family, dictionary methods, and others) can be unified under the elegant notion of *Grammar-Compressions*. A vast literature, across many disciplines, established this as an influential notion for Algorithm design.

We introduce a direly needed framework for proving (conditional) lower bounds in this field, allowing us to assess whether decompress-and-solve can be improved, and by how much. Our main results are:

- **The $O(nN\sqrt{\log N/n})$ bound for LCS and the $O(\min\{N \log N, nM\})$ bound for Pattern Matching with Wildcards are optimal up to $N^{o(1)}$ factors, under the Strong Exponential Time Hypothesis. (Here, M denotes the uncompressed length of the compressed pattern.)**
- **Decompress-and-solve is essentially optimal for Context-Free Grammar Parsing and RNA Folding, under the k-Clique conjecture.**
- **We give an algorithm showing that decompress-and-solve is *not* optimal for Disjointness.**

***Keywords*-grammar-compression; fine-grained complexity**

I. Introduction

Computer Science is often called the science of processing digital data. A central goal of theoretical CS is to understand the time complexity of the tasks we want to perform on data. *Data compression* has been one of the most important notions in CS and Information Theory for decades, and it is increasingly relevant in our current age of "Big Data" where it is hard to think of reasons why *not* to compress our data: smaller data can be stored more efficiently, transmitting it takes less resources such as energy and bandwidth, and perhaps it can even be processed faster. Since nowadays and for years to come nearly all of our data comes in compressed form, a central question becomes:

What is the time complexity of analyzing compressed data?

Say we have a piece of data of size N given in a compressed form of size n. For a problem with time complexity $T(\cdot)$, the naïve strategy of "decompress and solve" takes $\Theta(T(N))$ time, while the "gold standard" is $O(T(n))$ time: we want to solve the problem on the compression as efficiently as if the original data was small. To provide meaningful statements we need to decide on three things: What type of *data* is it? What *problem* do we want to solve? Which *compression scheme* is being used?

For the first two questions, the focus of this paper will be on the most basic setting. We consider data that comes as strings, i.e. sequences of symbols such as text, computer code, genomes, and so on. And we study natural and basic questions one could ask about strings such as Pattern Matching, Language Membership, Longest Common Subsequence, Parsing, and Disjointness.

For the third question, we restrict our attention to *lossless* compression and, even then, there are multiple natural settings that we do not find to be the most relevant. We could consider Kolmogorov complexity, giving us the best possible compression of our data: assume that a string T is given by a short bitstring $K(T)$ which is a pair of Turing machine M and input x such that running M on

0272-5428/17 $31.00 © 2017 IEEE
DOI 10.1109/FOCS.2017.26

x outputs T, i.e. $K(T) = \langle M, x\rangle$ such that $M(x) = T$. The issue with Kolmogorov-compressions is that none of our data comes in this form, for two good reasons: First, it is computationally intractable to compute $K(T)$ given T, not even approximately. And second, analyzing arbitrary Turing machines without just running them is an infamously hopeless task. Thus, while studying the time complexity of analyzing Kolmogorov-compressed strings is natural, it might not be the most relevant for computer science applications. Another option is to consider the mathematically simplest forms of compression such as *Run-Length Encoding* (RLE): we compress x consecutive letters σ into σ^x, so the compression has the form $0^{x_1}1^{x_2}0^{x_3}\cdots 1^{x_\ell}$, and we only need $n = O(\ell \cdot \log N)$ bits to describe the potentially exponentially longer string of length N. This compression is at the other extreme of the spectrum: it is trivial to compute and easy to analyze, but it is far less "compressing" than popular schemes like Lempel-Ziv-compressions.

Instead, we consider what has proven to be one of the most influential kinds of compression for Algorithm design, namely *Grammar-Compressions*, a notion that has all the right properties. First, it is mathematically elegant and quite fun to reason about for theoreticians (as evidenced by the many pages of our paper). Second, it is equivalent [50] up to low order terms (moderate constants and log factors) to popular schemes like the Lempel-Ziv-family (LZ77, LZ78, LZW, etc.) [39], [64], [59], Byte-Pair Encoding [54], dictionary methods, and others [45], [41]. These compressions are used in ubiquitous applications such as the built-in Unix utility `compress`, zip, GIF, PNG, and even in PDF. Third, it is generic and likely to capture compression schemes that will be engineered in the future (after all, there is a whole industry on the topic and the quest might never be over). Fourth, we can compute the optimal such compression (up to log factors) in linear time [50], [20], [35]. And last but not least, ingenious algorithmic techniques have shown that it is possible to computationally *analyze* grammar-compressed data, beating the "decompress and solve" bound for many important problems.

A grammar compression of a string X is simply a context-free grammar, whose language is exactly $\{X\}$, that is, the only string the grammar can produce is X. For the purposes of this paper, it is enough to focus on a restricted form of grammars, known as *Straight Line Programs* (SLP). An SLP is defined over some alphabet Σ, say $\{0, 1\}$, and it is a set of replacement rules (or productions) of a very simple form: a rule is either a symbol in Σ or it is the concatenation of two previous rules (under some fixed ordering of the rules). The last replacement rule is the sequence defined by the SLP. For example, we can compress the sequence 01011 with the rules $S_1 \to 0;\ S_2 \to 1;\ S_3 \to S_1\,S_2;\ S_4 \to S_3\,S_3;\ S_5 \to S_4\,S_2$ and S_5 corresponds to the sequence 01011. For some strings this can give an exponential compression. A more formal definition and a figure are given in Section II.

To learn more about the remarkable success of grammar-compressions, we refer the reader to the surveys [61], [38], [26], [53], [28], [49], [51], [42], [52]. As a side remark, one of the exciting developments in this context was the surprising observation that a "compress and solve" strategy could actually lead to theoretically new algorithms for some problems, e.g. [46], [36].

Thus, we focus on what we find the most important interpretation of the central question above:

What is the time complexity of basic problems on grammar-compressed strings?

A. Previous Work

As a motivating example, consider the Longest Common Subsequence (LCS) problem. Given two uncompressed strings of length N we can find the length of the longest common (not necessarily contiguous) subsequence in $O(N^2)$ time using dynamic programming, and there are almost-matching $N^{2-o(1)}$ conditional lower bounds [2], [15], [3]. Throughout the paper we mostly ignore log factors, and so we think of LCS as a problem with $\tilde{\Theta}(N^2)$ time complexity (on uncompressed data). Now, assume our sequences are given in compressed form of size n. A natural setting to keep in mind is where $n \approx N^{1/2}$. How much time do we need to solve LCS on these compressed strings? The naïve upper bound gives $O(N^2)$ and the gold standard is $O(n^2) \approx O(N)$, so which is it?

Besides being a very basic question, LCS and the closely related Edit Distance are a popular theoretical modeling of sequence alignment problems that are of great importance in Bioinformatics[1]. Thus, this is a relatively faithful modeling of the question whether "compress-and-solve" can speed up genome analysis tasks, a question which has received extensive attention throughout the years [30], [45], [41], [29], [28].

A long line of work [16], [43], [6], [7], [23], [55], [56], [31] has shown that we can do *much* better than $O(N^2)$. The current best algorithm has the curious runtime $O(nN\sqrt{\log N/n})$ [27] which is tantalizingly close to a conjectured bound of $O(nN)$ from the seminal paper of Lifshits [40]. In our candidate setting of $n \approx N^{1/2}$, this is $\tilde{O}(N^{1.5})$. This is major speedup over the $\Omega(N^2)$ decompress-and-solve bound, but is still far away from the gold standard of $O(n^2)$ which in this case would be $O(N)$. Can we do better? For example, an $O(n^2 \cdot N^{0.1})$ bound could lead to major real-world improvements.

While there is a huge literature on the topic, both from the Algorithms community and from applied areas, in addition to the potential for real-world impact, studying these questions has not become a mainstream topic in the top algorithms conferences. In one of the only STOC/FOCS

[1]The *heuristic* algorithm BLAST for a generalized version of the problem has received sixty-thousand citations.

papers on the topic, Charikar et al. [20] write "*In short, the smallest grammar problem has been considered by many authors in many disciplines for many reasons over a span of decades. Given this level of interest, it is remarkable that the problem has not attracted greater attention in the general algorithms community.*"

We believe that one key reason for this is the lack of a relevant *complexity theory* and tools for proving *lower bounds*, leaving a confusing state of the art in which it is hard to distinguish algorithms providing fundamental new insights from *ad hoc* solutions. Most importantly, previous work has not given us the tools to know, when we encounter a data analysis problem in the real-world, what kind of upper bound we should expect. Instead, researchers have been proving P vs. NP-hard results, classifying problems into ones solvable in $\text{poly}(n, \log N)$ time and ones that probably require time $N^{\Omega(1)}$. In fact, even LCS is NP-hard [40]. This means that even if we have a compression of very small size $n = O(\log N)$ then we cannot solve LCS in $\text{poly}(n)$ time, unless P = NP. Dozens of such negative results have been proven (see [42]), and it has long been clear that almost any task of interest is "NP-hard", including the basic $\text{poly}(N)$ time solvable problems we discuss in this paper. However, this is hardly relevant to the questions *we* ask since it does not address the possibility of highly desirable bounds such as $n^2 \cdot N^{0.1}$. What we would really like to know is whether the bound should be $\text{poly}(n) \cdot N^{\varepsilon}$, or $\text{poly}(n) \cdot N$, or even higher: could it be that decompress-and-solve is impossible to beat for some problems?

B. Our Work

In this work, we introduce a framework for showing lower bounds on the time complexity of problems on grammar-compressed strings. Our lower bounds are based on popular conjectures from *Hardness in P* and Fine-Grained Complexity. This is perhaps surprising since the problems we consider are technically NP-hard. Our new complexity theoretic study of this field leads to three exciting developments: First, we resolve the exact time complexity up to $N^{o(1)}$ factors of some of the most classical problems such as LCS *on compressed data*. Second, we discover problems that *cannot be solved faster than the decompress-and-solve bound* by any N^{ε} factor. Third, we *fail* at proving tight lower bounds for some classical problems, which hints to us that known algorithms might be suboptimal. Indeed, in this paper we also find *new algorithms* for fundamental problems. We hope that our work will inspire increased interest in this important topic.

Longest Common Subsequence: Our first result is a resolution of the time complexity of LCS on compressed data, up to $N^{o(1)}$ factors, under the Strong Exponential Time Hypothesis[2] (SETH). We complement the $O(nN\sqrt{\log N/n})$ upper bound of Gawrychowski [27] with an $(nN)^{1-o(1)}$ lower bound.

Theorem I.1. *Assuming SETH, there is no $(nN)^{1-\varepsilon}$-time algorithm for LCS for any $\varepsilon > 0$. This even holds restricted to instances with $n = \Theta(N^{\alpha_n})$ for any $0 < \alpha_n < 1$, and an alphabet of constant size.*

Thus, in the natural setting $n \approx N^{1/2}$ from above, we should indeed be content with the $\tilde{O}(N^{1.5})$ upper bound since we will not be able to get much closer to the gold standard, unless SETH fails. Assuming SETH, our result confirms the conjecture of Lifshits, up to $N^{o(1)}$ factors.

One way to view this result is as an *Instance Optimality* result for LCS. The exact complexity of LCS on two strings is precisely proportional to the product of the decompressed size N and the instance-inherent measure n of how compressible they are.

RNA Folding and CFG Parsing: Next, we turn our attention to two other fundamental problems: Context-Free Grammar Recognition (aka Parsing) and RNA Folding. Parsing is the core computer science problem in which we want to decide whether a given string (e.g. computer code) can be derived from a given grammar (e.g. the grammar of a programming language). Having the ability to efficiently parse a *compressed* file is certainly desirable. In RNA Folding we are given a string over some alphabet (e.g. $\{A, C, G, T\}$) with a fixed pairing between its symbols (e.g. $A-T$ match and $C-G$ match), and the goal is to compute the maximum number of non-crossing arcs between matching letters that one can draw above the string (which corresponds to the minimum energy folding in two dimensions). RNA Folding is one of the most central problems in bioinformatics, and as we have discussed above, the ability to analyze compressed data is important in this field. How fast can we solve these problems?

Given an uncompressed string of size N, classical dynamic programming algorithms, such as the CYK parser [22], [63], [37], solve RNA Folding in $O(N^3)$ time and Parsing in $O(N^3 \cdot g)$ time if the grammar has size g. Wikipedia lists twenty-four parsing algorithms designed throughout the years, all of which take cubic time in the worst case. A theoretical breakthrough of Leslie Valiant [58] in 1975 showed that there are truly sub-cubic $O(gN^{\omega})$ parsing algorithms, where $\omega < 2.38$ is the fast matrix multiplication (FMM) exponent. However, Valiant's algorithm has not been used in practice due the inefficiency of FMM algorithms, and obtaining a *combinatorial*[3] sub-cubic time algorithm would

[2]SETH is the pessimistic version of P $\neq$ NP, stating that we cannot solve k-SAT in $O((2-\varepsilon)^n)$ time, for some $\varepsilon > 0$ independent of and for all constant k [33], [17].

[3]For the purposes of this paper, "combinatorial" should be interpreted as any *practically efficient* algorithm that does not suffer from the issues of FMM such as large constants and inefficient memory usage.

be of major interest. Alas, it was recently proved [1] that any improvement over these bounds implies breakthrough k-Clique algorithms: either finding such a combinatorial subcubic algorithm *or* getting any $O(N^{\omega-\varepsilon})$ time algorithm, for any $\varepsilon > 0$, would refute the k-Clique Conjecture[4]. The situation for RNA is even more interesting since Valiant's sub-cubic algorithm does not generalize to this case. Under the k-Clique conjecture, the same lower bounds still apply [1], [19], implying that any improvement will have to use FMM. Indeed, an $O(N^{2.82})$ algorithm using FMM was recently achieved [13].

Cubic time is a real bottleneck when analyzing large genomic data. One would hope that if we are able to compress the data down to size n we could solve problems like RNA Folding and Parsing in time that is much faster than the N^3 lower bounds (to simplify the discussion we focus on combinatorial algorithms), such as $n^3 \cdot N^{o(1)}$ or at least $n^{1.5}N^{1.5}$, in certain analogy the LCS case. No such algorithms were found to date, and we provide an explanation: Decompress-and-solve *cannot be beaten* for Parsing and (essentially) for RNA Folding, under the k-Clique Conjecture. For both problems we prove a conditional lower bound of $N^{\omega-o(1)}$ for any kind of algorithm, and $N^{3-o(1)}$ for combinatorial algorithms, even restricted to $n = O(N^{\varepsilon})$ for any $\varepsilon > 0$.

Theorem I.2. *Assuming the k-Clique conjecture, there is no $O(N^{\omega-\varepsilon})$ time algorithm for CFG Recognition or RNA Folding for any $\varepsilon > 0$. Assuming the combinatorial k-Clique conjecture, there is no combinatorial $O(N^{3-\varepsilon})$ time algorithm for CFG Recognition or RNA Folding for any $\varepsilon > 0$. These results hold even restricted to instances with $n = O(N^{\varepsilon})$, and, for CFG Recognition, for a grammar size of $|\Gamma| = O(\log N)$.*

Approximate Pattern Matching: We continue our quest for quantifying the possible improvements over decompress-and-solve for basic problems. Consider the following compressed versions of important primitives in text analysis known as *Approximate Pattern Matching* problems. In all these problems we assume that we are given a compressed text T of size n (and decompressed size N), and a compressed pattern P of size m (and decompressed size M), both over some constant size alphabet.

- **Pattern Matching with Wildcards:** In this problem, the strings contain wildcard symbols that can be replaced by any letter, and our goal is to decide if P appears in T.
- **Substring Hamming Distance:** Compute the smallest Hamming distance of any substring of T to P.

And a problem that generalizes both is:

- **Generalized Pattern Matching:** Given some cost function on pairs of alphabet symbols, find the substring T' of T minimizing the total cost of all pairs $(T'[i], P[i])$.

The above problems have been extensively studied both in the uncompressed (see [21]) and in the compressed [40], [11], [25] settings. All three problems can be solved in time $O(\min\{N \log N, nM\})$ (see the full version of this paper). Note that this bound beats the decompress-and-solve bound when the pattern is small, but can we avoid decompressing the pattern? We show a completely tight SETH-based lower bound of $\min\{N, nM\}^{1-o(1)}$ for all three problems, even for constant size alphabets and in all settings where the parameters are polynomially related.

Theorem I.3. *Assuming SETH, Substring Hamming Distance and Pattern Matching with Wildcards over alphabet $\{0, 1\}$ (plus wildcards $*$) take time $\min\{N, nM\}^{1-o(1)}$. This holds even restricted to instances with $n = \Theta(N^{\alpha_n})$, $M = \Theta(N^{\alpha_M})$ and $m = \Theta(N^{\alpha_m})$ for any $0 < \alpha_n < 1$ and $0 < \alpha_m \leq \alpha_M \leq 1$.*

Language Membership: Consider the compressed version of the most basic language membership problems. Assume we are given a compressed string T (again, from size N to n).

- **DFA Acceptance:** Given T and a DFA F with q states, decide whether F accepts T.
- **NFA Acceptance:** Given T and a NFA F with q states, decide whether F accepts T.

Classic algorithms solve the DFA Acceptance problem in time $O(\min\{nq, N+q\})$ [47], [32], and we prove a matching SETH-based lower bound of $\min\{nq, N+q\}^{1-o(1)}$.

Theorem I.4. *Assuming SETH, there is no algorithm for DFA Acceptance in time $O(\min\{nq, N\}^{1-\varepsilon})$ for any $\varepsilon > 0$. This holds even restricted to instances with constant alphabet size, $N = \Theta(n^{\alpha_N})$ and $q = \Theta(n^{\alpha_q})$ for any $\alpha_N > 1$ and $\alpha_q > 0$.*

For the NFA problem, the classic algorithms give $O(\min\{nq^{\omega}, Nq^2\})$ [44], [47], [32]. For combinatorial algorithms, we prove a matching lower bound of $\min\{nq^3, Nq^2\}^{1-o(1)}$, under the (combinatorial) k-Clique conjecture.

Theorem I.5. *Assuming the combinatorial k-Clique conjecture, there is no combinatorial algorithm for NFA Acceptance in time $O(\min\{nq^3, Nq^2\}^{1-\varepsilon})$ for any $\varepsilon > 0$. This holds even restricted to instances with constant alphabet size, $n = \Theta(q^{\alpha_n})$ and $N = \Theta(q^{\alpha_N})$ for any $\alpha_N \geq \alpha_n > 0$.*

Disjointness, Hamming Distance, and Subsequence: Could it be that for other, even more basic problems the decompress-and-solve bound cannot be beaten? One candidate might be Disjointness, the canonical hard problem in Communication Complexity.

[4]Given a graph on n nodes, the k-Clique conjecture [1] is in fact two independent conjectures: The first one states that we cannot solve k-clique in $O(n^{(1-\varepsilon)\cdot\omega k/3})$, for any $\varepsilon > 0$. The second one states that we cannot solve k-Clique combinatorially in $O(n^{(1-\varepsilon)k})$ time, for any $\varepsilon > 0$.

- **Disjointness:** Given two equal-length bit-strings, is there a coordinate in which both are 1?

The following two natural problems are at least as hard as Disjointness (for a proof, see the full version).

- **Hamming Distance:** Compute the Hamming Distance of two strings.
- **Subsequence:** Decide if a pattern of length M is a subsequence of a text of length N.

Note that all these problems can be solved trivially in $O(N)$ time if our strings are uncompressed. Could it be that we cannot solve them without decompressing our data? We are not aware of any known algorithms solving any of these problems in $O(N^{1-\varepsilon})$ time, for any $\varepsilon > 0$, even when our strings are compressed into size $n = O(N^{\alpha})$ for some small constant $\alpha > 0$. The only exceptions are the known $\tilde{O}(M)$ time algorithms [24], [18], [55], [62], [57], [10] for the Subsequence problem, which beat the decompress-and-solve bound when the pattern is significantly smaller than the text. However, in the case $M = \Theta(N)$ no improvements seem to be known.

We summarize our attempts at proving a matching lower bound. For combinatorial algorithms and a certain parameter setting, we obtain a $N^{1-o(1)}$ lower bound for the Subsequence problem under the combinatorial k-Clique conjecture.

Theorem I.6. *The Subsequence problem has no combinatorial $O(N^{1-\varepsilon})$ time algorithm for any $\varepsilon > 0$ in the setting $N = \Theta(M) = \Theta(n^2) = \Theta(m^2)$ and $|\Sigma| = O(N^{\varepsilon})$, assuming the combinatorial k-Clique conjecture.*

Under the Strong k-SUM Conjecture[5] we are able to prove an $N^{1/3-o(1)}$ for Disjointness even when $n = O(N^{\varepsilon})$ for any $\varepsilon > 0$.

Theorem I.7. *Let $k \geq 1$ be an integer. Consider the Disjointness problem with $N = M = \Theta(n^{3k+1}) = \Theta(m^{3k+1})$. Solving the Disjointness problem in this setting requires $N^{\frac{1}{3}+\frac{2}{9k+3}-o(1)}$ time assuming the Strong $(2k+1)$-SUM conjecture.*

The same lower bounds follow for Hamming Distance and Subsequence by the following reductions.

Theorem I.8. *The Disjointness problem can be reduced to the Subsequence problem and the Hamming Distance problem. These reductions lose at most constant factors in the length of compressed and decompressed sequences.*

The gap from the $O(N)$ upper bound is still large.

Motivated by our inability to prove tight lower bounds for these basic problems, despite seemingly having the right framework, we have turned our attention to upper bounds. In particular, we obtain the first improvement over the decompress-and-solve bound for Disjointness, Hamming Distance, and Subsequence. Our algorithms solve all these problems in $O(n^{1.410} \cdot N^{0.593})$ time.

Theorem I.9. *The Subsequence problem and the Hamming Distance problem can be solved in time*

$$\tilde{O}\left(\max(m,n)^{2-1/\log_2(2\varphi)} \cdot N^{1/\log_2(2\varphi)}\right) = \tilde{O}\left(\max(m,n)^{1.409\ldots} \cdot N^{0.592\ldots}\right),$$

where $\varphi = \frac{1+\sqrt{5}}{2}$ is the golden ratio.

The same result for the Disjointness problem follows from Theorem I.8. As a side result, we also design a very simple algorithm for the Subsequence problem with $O((n|\Sigma|+M)\log N)$ runtime (provided in the full version), which is comparable to the known but more involved algorithms [10].

One of the biggest benefits of having complexity theoretic results is that algorithm designers know what to focus on. We believe that these upper bounds can be improved further and suggest it as an interesting open question: *What is the time complexity of computing Disjointness on two grammar-compressed strings?*

C. Technical Overview

From a technical perspective, our paper is most related to the conditional lower bounds for sequence similarity measures on strings and curves that have been shown in recent years, specifically, the SETH-based lower bounds for edit distance [8], longest common subsequence [2], [15], Fréchet distance [12], and others [4], [9], [14], [48].

These results all proceed as follows. Let ϕ be a given k-SAT instance on $\tilde{n}$ variables and clauses $C_1, \ldots, C_{\tilde{m}}$. We can assume that $\tilde{m} = O(\tilde{n})$ by the Sparsification Lemma [34]. Split the $\tilde{n}$ variables into two halves X_1 and X_2 of size $\tilde{n}/2$. Enumerate all assignments $\alpha_1, \ldots, \alpha_{2^{\tilde{n}/2}}$ of the variables in X_1. For any assignment α_i and any clause C_ℓ, denote by $\text{sat}(\alpha_i, C_\ell)$ whether α_i satisfies C_ℓ, i.e., whether some variable in X_1 appears in C_ℓ (negated or unnegated) and is set by α_i so that C_ℓ is satisfied. Similarly, consider the assignments $\beta_1, \ldots, \beta_{2^{\tilde{n}/2}}$ of X_2. By construction, we can solve the k-SAT instance ϕ by testing whether there are α_i, β_j such that $\text{sat}(\alpha_i, C_\ell) \vee \text{sat}(\beta_j, C_\ell)$ holds for all $\ell \in [\tilde{m}]$. Making use of this fact, all previous conditional lower bounds for sequence similarity measures essentially construct the following natural sequence:

$$\begin{aligned} W &= \text{sat}(\alpha_1, C_1) \ldots \text{sat}(\alpha_1, C_{\tilde{m}}) \\ &\qquad \ldots \text{sat}(\alpha_{2^{n/2}}, C_1) \ldots \text{sat}(\alpha_{2^{n/2}}, C_{\tilde{m}}) \\ &= \bigcirc_{i \in [2^{\tilde{n}/2}]} \bigcirc_{\ell \in [\tilde{m}]} \text{sat}(\alpha_i, C_\ell). \end{aligned}$$

One typical variation of this string is to replace the bits $\{0,1\}$, indicating whether $\text{sat}(\alpha_i, C_\ell)$ holds, by two short strings $\{B(0), B(1)\}$. Other typical variations are

[5]The conjecture that, given a set of n integers bounded by $n^{\lceil k/2 \rceil}$, finding k integers that sum to zero requires time $n^{\lceil k/2 \rceil - o(1)}$.

to add appropriate padding strings around the substrings $\bigcirc_{\ell\in[\tilde{m}]}\mathrm{sat}(\alpha_i, C_\ell)$ or around the whole sequence W. These paddings typically only depend on n and $\tilde{m}$. Constructing a second sequence W' with α_i replaced by β_i, one can then try to emulate the search for the half-assignments α_i, β_j by a similarity measure on W, W'. All previous reductions follow this recipe, and thus construct a sequence like W.

Is $\boldsymbol{W}$ compressible?: For our purposes we need to construct compressible strings. Considering the entropy, the string W is very well compressible, since it only depends on the $\tilde{O}(\tilde{n})$ input bits of the sparse k-SAT instance ϕ. This entropy $\tilde{O}(\tilde{n})$ is extremely small compared to the length $O(\tilde{n}2^{\tilde{n}/2})$ of W. However, considering grammar-compression, the sequence W is a bad representation, since *W is not generated by any SLP of size $o(2^{\tilde{n}/2}/\tilde{n})$ in general!* To see this, first observe that all substrings $\bigcirc_{\ell\in[\tilde{m}]}\mathrm{sat}(\alpha_i, C_\ell)$ of W can potentially be different, meaning that W can have $2^{\tilde{n}/2}$ different substrings of length $\tilde{m}$. This happens e.g. if for each variable $x_i \in X$ there is a clause C_i consisting only of x_i (which makes the k-SAT instance trivial, but shows that W may have many different substrings in general). Second, observe that for any SLP $\mathcal{T}$ consisting of n non-terminals $S_1 \dots S_n$ and for any length $L \geq 1$ the generated string $\mathrm{eval}(\mathcal{T})$ has at most $n \cdot L$ different substrings of length L. Indeed, a rule $S_i \to S_\ell S_r$ can only create a new substring, that is not already contained in $\mathrm{eval}(S_\ell)$ or $\mathrm{eval}(S_r)$, if this substring overlaps the boundary between $\mathrm{eval}(S_\ell)$ and $\mathrm{eval}(S_r)$ in $\mathrm{eval}(S_i)$. Hence, the rule $S_i \to S_\ell S_r$ can contribute at most L new substrings of length L, amounting to at most nL different substrings overall. Combining these two facts, with $L = \tilde{m} = O(\tilde{n})$, we see that W in general has no SLP of size $o(2^{\tilde{n}/2}/\tilde{n})$.

Hence, the standard approach to conditional lower bounds for sequence similarity measures fails in the compressed setting, and it might seem like (SETH-based) conditional lower bounds are not applicable here.

A compressible sequence $\boldsymbol{T}$: On the contrary, we show that by simply inverting the ordering we obtain a very well compressible string:

$$\begin{aligned} T &= \mathrm{sat}(\alpha_1, C_1)\dots\mathrm{sat}(\alpha_{2^{n/2}}, C_1) \\ &\qquad \dots \mathrm{sat}(\alpha_1, C_{\tilde{m}})\dots\mathrm{sat}(\alpha_{2^{n/2}}, C_{\tilde{m}}) \\ &= \bigcirc_{\ell\in[\tilde{m}]}\bigcirc_{i\in[2^{\tilde{n}/2}]}\mathrm{sat}(\alpha_i, C_\ell). \end{aligned}$$

The difference between W and T might seem negligible, but it greatly changes the game of emulating k-SAT by a sequence similarity measure: In W we are looking for a local structure (a small substring) that "fits together" with a local structure in a different string W'. In T we have to ensure the choice of a consistent offset $\Delta \in [n]$ and "read" the symbols $T[\Delta], T[\Delta + 2^{\tilde{n}/2}], \dots, T[\Delta + (\tilde{m}-1)2^{\tilde{n}/2}]$, which seems much more complicated.

T is compressible to an SLP $\mathcal{T}$ of size $O(\tilde{n}^2)$, which is much smaller than the $\Omega(2^{\tilde{n}/2}/\tilde{n})$ bound for W. Indeed, consider a substring $\bigcirc_{i\in[2^{\tilde{n}/2}]}\mathrm{sat}(\alpha_i, C_\ell)$. We may assume that no variable appears more than once in C_ℓ. Consider the following SLP rules, for $1 \leq i \leq \tilde{n}/2$,

$$\begin{aligned} &A_0 \to 1, \\ &A_i \to A_{i-1}A_{i-1}, \\ &S_0 \to 0, \end{aligned} \qquad S_i \to \begin{cases} S_{i-1}A_{i-1} & \text{if } x_i \text{ appears in } C_\ell \\ A_{i-1}S_{i-1} & \text{if } \neg x_i \text{ appears in } C_\ell \\ S_{i-1}S_{i-1} & \text{otherwise} \end{cases}$$

We clearly have $\mathrm{eval}(A_i) = 1^{2^i}$. Moreover, if $\neg x_i$ appears in C_ℓ, then for $x_i = 0$, no matter what we choose for $x_1, \dots, x_{i-1}$, we have $\mathrm{sat}(\alpha_j, C_\ell) = 1$, and thus we may write A_{i-1}. For $x_i = 1$ we note that the value $\mathrm{sat}(\alpha_j, C_\ell)$ only depends on the remaining variables $x_1, \dots, x_{i-1}$, and thus we may write S_{i-1}. Along these lines, one can check that $\mathrm{eval}(S_{\tilde{n}/2}) = \bigcirc_{i\in[2^{\tilde{n}/2}]}\mathrm{sat}(\alpha_i, C_\ell)$. Creating such an SLP for each $\ell \in [\tilde{m}]$ and constructing their concatenation, we obtain an SLP of size $O(\tilde{m}\tilde{n}) = O(\tilde{n}^2)$ generating T.

Example Lower Bound: Pattern Matching with Wildcards: In the remainder of this section, we present an easy example for a conditional lower bound on compressed strings, namely for the problem Pattern Matching with Wildcards. Here we consider an alphabet Σ and we say that symbols $\sigma, \sigma' \in \Sigma \cup \{*\}$ *match* if $\sigma = *$ or $\sigma' = *$ or $\sigma = \sigma'$. We say that two equal-length strings X, Y (over alphabet $\Sigma \cup \{*\}$) match if $X[i]$ and $Y[i]$ match for all i. Given a text T of length N and a pattern P of length $M \leq N$, the task is to decide whether P matches some length-M substring of T.

Let ϕ be a k-SAT instance as above, but this time let $\alpha_1, \dots, \alpha_{2^{\tilde{n}}}$ be *all* the assignments of the $\tilde{n}$ variables in ϕ. We define the text T and pattern P by

$$T = \bigcirc_{\ell\in[\tilde{m}]}\bigcirc_{i\in[2^{\tilde{n}}]}\mathrm{sat}(\alpha_i, C_\ell) \qquad P = 1(*^{2^{\tilde{n}}-1}1)^{\tilde{m}-1}.$$

Note that P matches some substring of T if and only if there is an offset $\Delta \in [2^{\tilde{n}}]$ such that $T[\Delta] = T[\Delta + 2^{\tilde{n}}] = \dots = T[\Delta + (\tilde{m}-1)2^{\tilde{n}}] = 1$, which happens if and only if α_Δ is a satisfying assignment of ϕ. Hence, we constructed an equivalent instance of Pattern Matching with Wildcards.

Analogously to above, one can show that T is generated by an SLP $\mathcal{T}$ of size $n = O(\tilde{n}^2)$ that can be computed in time $O(\tilde{n}^2)$. Similarly, it is easy to see that P is generated by an SLP $\mathcal{P}$ of size $O(\tilde{n})$ that can be computed in time $O(\tilde{n})$. Hence, the reduction runs in time $O(\tilde{n}^2)$. We stress that we define strings T, P of exponential length in $\tilde{n}$, but in the reduction we never explicitly write down any such string, but we simply construct compressed representations. Since the resulting strings have length $O(2^{\tilde{n}}\tilde{n})$, any $O(N^{1-\varepsilon})$ time algorithm for Pattern Matching with Wildcards would imply an algorithm for k-SAT in time $O(2^{(1-\varepsilon)\tilde{n}}\mathrm{poly}(\tilde{n}))$, contradicting the Strong Exponential Time Hypothesis (SETH).

Note that this conditional lower bound of $N^{1-o(1)}$ holds even for strings compressible to size polylog(N).

In Section III we analyze Pattern Matching with Wildcards in more detail and show that the optimal running time, conditional on SETH, is $\min\{N, nM\}^{1\pm o(1)}$, and this holds for all settings of the text length N, the compressed text size n, the pattern length M, and the compressed pattern size m.

In Pattern Matching with Wildcards, we got a consistent choice of an offset Δ for free. It is much more complicated to achieve this for other problems such as Longest Common Subsequence, CFG Parsing, or RNA Folding. This overview summarized the main technical contributions of this paper, but left out many problem-specific tricks that can be found in the subsequent proofs, and that we think will find more applications for analyzing problems on compressed strings.

II. Preliminaries

Here we give general preliminaries on strings, straight-line programs, and hardness assumptions. For a positive integer n we let $[n] = \{1, \dots, n\}$, while for a proposition A we let $[A]$ be 1 if A is true and 0 otherwise.

Strings: Let Σ be a finite alphabet. In most parts of this paper we assume that $|\Sigma| = O(1)$, but in exceptional cases we allow the alphabet to grow with the input size. For a string T over alphabet Σ, we write $|T|$ for its length, $T[i]$ for its i-th symbol, and $T[i..j]$ for the substring from position i to position j. For two strings T, T' we write $T \circ T'$, or simply $T\,T'$, for their concatenation. For $k \geq 1$ we let $T^k := \bigcirc_{i=1}^{k} T$.

Straight-Line Programs (SLPs): An SLP $\mathcal{T}$ is a set of non-terminals $S_1, \dots, S_n$, each equipped with a rule of the form (1) $S_i \to \sigma$ for some $\sigma \in \Sigma$ or (2) $S_i \to S_{\ell(i)}, S_{r(i)}$ with $\ell(i), r(i) < i$. The string T generated by SLP $\mathcal{T}$ is recursively defined as follows. For a rule $S_i \to \sigma$ we let $\text{eval}(S_i) := \sigma$, and for a rule $S_i \to S_{\ell(i)}, S_{r(i)}$ we let $\text{eval}(S_i) := \text{eval}(S_{\ell(i)}) \circ \text{eval}(S_{r(i)})$. Then $T = \text{eval}(\mathcal{T}) := \text{eval}(S_n)$ is the string generated by SLP $\mathcal{T}$. Note that an SLP is a context-free grammar describing a unique string; so $\mathcal{T}$ is a *grammar-compressed* representation of T. We call $|\mathcal{T}| = n$ the *size* of $\mathcal{T}$. See Figure 1 for the depiction of an SLP; in particular note the difference between the *directed acyclic graph* that is the compressed representation $\mathcal{T}$ and the *parse tree* that we obtain by decompressing $\mathcal{T}$ to a tree whose leaves spell the decompressed text T.

Observation II.1. *For any string T and $k \geq 1$, there is an SLP of size $O(|T| + \log k)$ generating the string T^k.*

In all problems considered in this paper, the input contains a text T given by a grammar-compressed representation $\mathcal{T}$, such that $T = \text{eval}(\mathcal{T})$. We always denote by $N = |T|$ the length of the text and by $n = |\mathcal{T}|$ the size of its representation. Sometimes we are additionally given a pattern P by a grammar-compressed representation $\mathcal{P}$, and we denote the pattern length by $M = |P|$ and its representation size by $m = |\mathcal{P}|$.

A. Hardness Assumptions

The Strong Exponential Time Hypothesis (SETH) was introduced by Impagliazzo, Paturi, and Zane [34] and asserts that the central NP-hard satisfiability problem has no algorithms that are much faster than exhaustive search.

Conjecture II.2 (SETH)**.** *There is no $\varepsilon > 0$ such that for all $k \geq 3$, k-SAT on n variables can be solved in time $O(2^{(1-\varepsilon)n})$.*

Effectively all known SETH-based lower bounds for polynomial-time problems use reductions via the *Orthogonal Vectors problem* (OV): Given sets $\mathcal{A}, \mathcal{B} \subseteq \{0,1\}^d$ of size $|\mathcal{A}| = A$, $|\mathcal{B}| = B$, determine whether there exist vectors $a \in \mathcal{A}$, $b \in \mathcal{B}$ with $\sum_{i=1}^{d} a[i] \cdot b[i] = 0$. Simple algorithms solve OV in time $O(2^d(A+B))$ and $O(dAB)$. For $A = B$ and $d = c(A)\log A$ the fastest known algorithm runs in time $A^{2-1/O(\log c(A))}$ [5], which is only slightly subquadratic for $d \gg \log A$. This has led to the following conjecture, which follows from SETH [60].

Conjecture II.3 (OV)**.** *For any $\varepsilon > 0$ and $\beta > 0$, on instances with $B = \Theta(A^\beta)$ OV has no $O(A^{1+\beta-\varepsilon}\text{poly}(d))$ time algorithm.*

It is known that if this conjecture holds for some $\beta > 0$ then it holds for all $\beta > 0$, see e.g. [15].

More generally, for $k \geq 2$ we say that a tuple $(a_1, \dots, a_k)$ with $a_i \in \{0,1\}^d$ is *orthogonal* if for all $\ell \in [d]$ there exists an $i \in [k]$ such that $a_i[\ell] = 0$. In the k-OV problem we are given a set $\mathcal{A} \subseteq \{0,1\}^d$ of size A and want to determine whether there is an orthogonal tuple $(a_1, \dots, a_k)$ with $a_i \in \mathcal{A}$. The fastest known algorithm for k-OV is to run an easy reduction to OV and then solve OV. The following conjecture follows from SETH.

Conjecture II.4 (k-OV)**.** *For any $\varepsilon > 0$ and $k \geq 2$, k-OV is not in time $O(A^{k-\varepsilon}\text{poly}(d))$.*

For details on the k-Clique and (Strong) k-SUM conjecture see the full version.

III. Approximate Pattern Matching and Substring Hamming Distance

We study the following generalization of pattern matching.

Problem III.1 (Generalized Pattern Matching)**.** *Given a text T of length N by an SLP $\mathcal{T}$ of size n, a pattern P of length M by an SLP $\mathcal{P}$ of size m, both over some alphabet Σ, and given a cost function $\text{cost}\colon \Sigma \times \Sigma \to \mathbb{N}$, compute $\min_{0 \leq i \leq N-M} \sum_{j=1}^{M} \text{cost}(P[j], T[i+j])$, i.e., the minimum total cost of any alignment.*

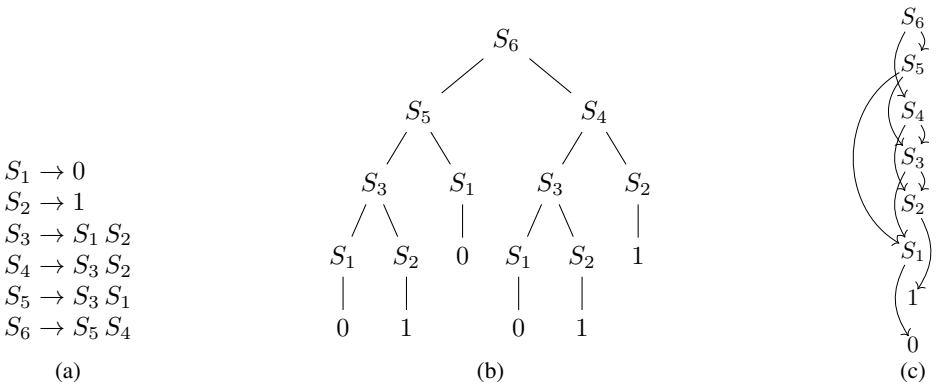

Figure 1: (a) An SLP generating the sequence 010011. (b) The corresponding parse tree. (c) The acyclic graph corresponding to the SLP.

In other words, we want to compute the length-M substring T' of T minimizing the total cost of aligned symbols in P and T'. This problem has two important special cases: (1) We obtain *Substring Hamming Distance* when $\mathrm{cost}(\sigma, \sigma') = [\sigma \neq \sigma']$ for any $\sigma, \sigma' \in \Sigma$. (2) We obtain *Pattern Matching with Wildcards* when T is over alphabet Σ and P is over alphabet $\Sigma \cup \{*\}$, we have $\mathrm{cost}(*, \sigma) = 0$ for any $\sigma \in \Sigma$ and $\mathrm{cost}(\sigma, \sigma') = [\sigma \neq \sigma']$ for any $\sigma, \sigma' \in \Sigma$, and the task is to decide whether the minimum total cost of any alignment is 0.

In this section, for all three problems we show an upper bound of $O(\min\{|\Sigma| N \log N, nM\})$ and a SETH-based lower bound of $\min\{N, nM\}^{1-o(1)}$. This yields a tight bound in case of constant alphabet size, as the lower bound constructs constant-alphabet strings. We leave it as an open problem to get tight bounds for larger alphabet size.

Note that it suffices to prove the upper bound for Generalized Pattern Matching and the lower bound for the special cases Substring Hamming Distance and Pattern Matching with Wildcards. The following upper bound follows from standard arguments and is proven in the full version.

Lemma III.2. *Generalized Pattern Matching can be solved in time* $O(\min\{|\Sigma| N \log N, nM\})$.

It remains to prove the SETH-based lower bound of $\min\{N, nM\}^{1-o(1)}$ for Substring Hamming Distance and Pattern Matching with Wildcards.

Recall the intuition given in Section I-C. We aim to make it more formal, however, to increase clarity of presentation and the generality of the result, we do not reduce directly from SAT but from k-OV. We start by designing a text T that enumerates all combinations of k vectors in a given k-OV instance, while still being well compressible. As usual, we consider k as a constant.

Lemma III.3. *Consider a* k-OV *instance* $\mathcal{A} = \{a_1, \ldots, a_A\} \subseteq \{0,1\}^d$. *Let* $b \in \{0,1\}^d$ *be an additional vector, and let* $S(0), S(1)$ *be strings of length* γ *(*$S(i)$ *is a sequence that represents an entry that is equal to* i*). We define the tuplified representation as follows:*

$$V = \mathrm{tuplify}\big(\mathcal{A}, k, b, S(0), S(1)\big) := \bigcirc_{\ell=1}^{d} \bigcirc_{i_1,\ldots,i_k \in [A]} S\Big(b[\ell] \cdot a_{i_1}[\ell] \cdots a_{i_k}[\ell]\Big),$$

where the second $\bigcirc$ *goes over all tuples* $(i_1, \ldots, i_k) \in [A]^k$ *in lexicographic order. This representation satisfies the following properties.*

1) *We can compute, in linear time in the output size, an SLP* $\mathcal{V}$ *generating* V *of size* $O(dA + \gamma)$ *or, when given SLPs* $\mathcal{S}(0), \mathcal{S}(1)$ *generating* $S(0), S(1)$*, of size* $O(dA + |\mathcal{S}(0)| + |\mathcal{S}(1)|)$.
2) *Write* $V = \bigcirc_{i=1}^{dA^k} V_i$ *with* $V_i \in \{S(0), S(1)\}$. *Then there exist* $i_1, \ldots, i_k \in [A]$ *such that* $(b, a_{i_1}, \ldots, a_{i_k})$ *is orthogonal if and only if there is an* offset $1 \leq \Delta \leq A^k$ *such that*

$$V_\Delta = V_{\Delta + A^k} = \ldots = V_{\Delta + (d-1)A^k} = S(0).$$

Proof: For the second property, note that by definition $V_\Delta, V_{\Delta+A^k}, \ldots, V_{\Delta+(d-1)A^k}$ are all equal to $S(0)$ for $\Delta \in [A^k]$ if and only if the Δ-th tuple $(i_1, \ldots, i_k) \in [A]^k$ in the lexicographic ordering of $[A]^k$ satisfies

$$b[\ell] \cdot a_{i_1}[\ell] \cdots a_{i_k}[\ell] = 0 \qquad \text{for all } \ell \in [d].$$

This condition is equivalent to $(b, a_{i_1}, \ldots, a_{i_k})$ being an orthogonal pair, so the claim follows.

It remains to construct a short SLP $\mathcal{V}$ generating V. We construct non-terminals $P_{S(0)}, P_{S(1)}$ with $\mathrm{eval}(P_{S(i)}) = S(i)$ by an SLP of size $\gamma_S = O(\gamma)$ as in Observation II.1, or of size $\gamma_S = O(|\mathcal{S}(0)| + |\mathcal{S}(1)|)$ by using given SLPs $\mathcal{S}(0), \mathcal{S}(1)$. We can extend this, using Observation II.1, to a slightly larger SLP of size $O(\log A + \gamma_S)$ that includes, for every $1 \leq j \leq k$, a non-terminal $P^j_{S(0)}$ with $\mathrm{eval}(P^j_{S(0)}) = S(0)^{A^j}$.

The crucial observation is the following: for any tuple $(i_1, \ldots, i_k) \in [A]^k$, let $p_\ell(i_1, \ldots, i_k) = a_{i_1}[\ell] \cdots a_{i_k}[\ell]$.

Then for any $\ell \in [d], j \in [k]$ and $(i_1, \ldots, i_j) \in [A]^j$, we have that $a_{i_j}[\ell] = 0$ implies $p_\ell(i_1, \ldots, i_j, i'_{j+1}, \ldots, i'_k) = 0$ for *all* $(i'_{j+1}, \ldots, i'_k) \in [A]^{k-j}$. We now define the final SLP using the starting non-terminal S_0 and the following productions

$$S_0 \to \mathrm{Test}_1 \ldots \mathrm{Test}_d$$

$$\mathrm{Test}_\ell \to \begin{cases} P^k_{S(0)} & \text{if } b[\ell] = 0 \\ \mathrm{List}^{(1)}_\ell & \text{otherwise} \end{cases} \qquad \ell \in [d],$$

$$\mathrm{List}^{(j)}_\ell \to \bigcirc_{i \in [A]} \begin{cases} P^{k-j}_{S(0)} & \text{if } a_i[\ell] = 0, \\ \mathrm{List}^{(j+1)}_\ell & \text{otherwise} \end{cases} \qquad \ell \in [d], j \in [k],$$

$$\mathrm{List}^{(k+1)}_\ell \to P_{S(1)}.$$

It is straight-forward to verify that $\mathrm{eval}(S_0) = V$. Note that the size of this SLP, i.e., the total number of non-terminals on the right hand side of the above rules, is bounded by $O(\gamma_S + dA)$. Moreover, the SLP can be constructed in linear time in its size. ∎

After this preparation, we can prove the lower bound of Theorem I.3 for Pattern Matching with Wildcards.

Theorem III.4. *Assuming the k-OV conjecture, Pattern Matching with Wildcards over alphabet $\{0,1\}$ (plus wildcards $*$) takes time $\min\{N, nM\}^{1-o(1)}$. This holds even restricted to instances with $n = \Theta(N^{\alpha_n})$, $M = \Theta(N^{\alpha_M})$ and $m = \Theta(N^{\alpha_m})$ for any $0 < \alpha_n < 1$ and $0 < \alpha_m \leq \alpha_M \leq 1$.*

Before we prove Theorem III.4, let us sketch the main idea by providing a simple $N^{1-o(1)}$-time conditional lower bound in the setting $n, m = O(N^\varepsilon)$ and $N = \Theta(M)$. Let $\mathcal{A} \subseteq \{0,1\}^d$ of size A be an arbitrary k-OV instance with $k > 1/\varepsilon$, and assume for simplicity $d \leq A^{o(1)}$. Using Lemma III.3 on $\mathcal{A}$, k, $S(0) = 0, S(1) = 1$ and $b = (1, \ldots, 1) \in \{0,1\}^d$, we compute an SLP $\mathcal{T}$ for

$$T = \mathrm{tuplify}(\mathcal{A}, k, b, S(0), S(1)).$$

We define the pattern P as

$$P = 0(*^{A^k - 1}0)^{d-1}.$$

Note that Pattern Matching with Wildcards on instance T, P checks whether for some offset Δ we have $T[\Delta] = T[\Delta + A^k] = \ldots = T[\Delta + (d-1)A^k] = 0$. Hence, by Lemma III.3, pattern P matches T if and only if there is an orthogonal tuple $(a_1, \ldots, a_k) \in \mathcal{A}^k$, showing correctness of the reduction.

Note that we have $N = \Theta(M) = \Theta(dA^k)$. By Lemma III.3, T has an SLP of size $O(dA)$, and by Observation II.1, P has an SLP of size $O(d \log A)$. By $d \leq A^{o(1)}$ and $k > 1/\varepsilon$, we are indeed in the setting $n, m = O(N^\varepsilon)$ and $N = \Theta(M)$. An $O(N^{1-\varepsilon})$ algorithm for Pattern Matching with Wildcards would now imply an $O(A^{k(1-\varepsilon)}\mathrm{poly}(d))$ for k-OV, contradicting the k-OV conjecture.

We now give the slightly more involved general construction.

Proof of Theorem III.4: For $k \geq 2$, let $\mathcal{A} = \{a_1, \ldots, a_A\}$ be a k-OV instance in d dimensions, and let $k_1, k_2 \geq 1$ with $k_1 + k_2 = k$. We will construct an equivalent instance of Pattern Matching with Wildcards with $N = O(dA^k)$, $M = O(dA^{k_1})$, $n = O(dA^{k_2+1})$, and $m = O(d \log A)$. Any $O(\min\{N, nM\}^{1-\varepsilon})$ algorithm for Pattern Matching with Wildcards would then imply an algorithm for k-OV in time $O(A^{(k+1)(1-\varepsilon)}\mathrm{poly}(d)) = O(A^{k(1-\varepsilon/2)}\mathrm{poly}(d))$ for $k \geq 2/\varepsilon$, contradicting the k-OV conjecture. Below we strengthen this statement to hold restricted to instances with $n = \Theta(N^{\alpha_n})$, $M = \Theta(N^{\alpha_M})$ and $m = \Theta(N^{\alpha_m})$ for any $0 < \alpha_n < 1$ and $0 < \alpha_m \leq \alpha_M \leq 1$.

To give such a reduction, we define the text as

$$T = \bigcirc_{\substack{(j_1, \ldots, j_{k_2}) \\ \in [A]^{k_2}}} 1^{A^{k_1}} \circ \mathrm{tuplify}(\mathcal{A}, k_1, \min(a_{j_1}, \ldots, a_{j_{k_2}}), 0, 1),$$

where $\min(b_1, \ldots, b_\ell)$ denotes the component-wise minimum of $b_1, \ldots, b_\ell$.

We define the pattern P as

$$P = 0(*^{A^{k_1} - 1}0)^{d-1}.$$

Correctness: Observe that P cannot overlap any $1^{A^{k_1}}$-block, since never more than $A^{k_1} - 1$ wildcards are followed by a 0 in P. Thus, P matches T if and only if there is a tuple $(j_1, \ldots, j_{k_2}) \in [A]^{k_2}$ such that P matches $T((j_1, \ldots, j_{k_2})) := \mathrm{tuplify}(\mathcal{A}, k_1, \min(a_{j_1}, \ldots, a_{j_{k_2}}), 0, 1)$. By the structure of the pattern, P matches any string S if and only if there is an offset Δ such that $S[\Delta] = S[\Delta + A^{k_1}] = \cdots = S[\Delta + (d-1)A^{k_1}] = 0$. Thus, by Lemma III.3, P matches $T((j_1, \ldots, j_{k_2}))$ if and only if there are vectors $a_1, \ldots, a_{k_1} \in \mathcal{A}$ for which $(a_1, \ldots, a_{k_1}, \min(a_{j_1}, \ldots, a_{j_{k_2}}))$ is an orthogonal tuple. The latter condition is equivalent to $(a_1, \ldots, a_{k_1}, a_{j_1}, \ldots, a_{j_{k_2}})$ being an orthogonal tuple. Since $k_1 + k_2 = k$ and T contains $T((j_1, \ldots, j_{k_2}))$ for all $(j_1, \ldots, j_{k_2}) \in [A]^{k_2}$, this proves that P matches T if and only if there is an orthogonal k-tuple in the instance $\mathcal{A}$.

Size Bounds: Note that $N = |T| = O(dA^k)$. By Lemma III.3 and Observation II.1, we can compute an SLP $\mathcal{T}$ of size $n = O(dA^{k_2+1})$ generating T, in linear time. Similarly, note that $M = |P| = O(dA^{k_1})$. By Observation II.1, we can compute an SLP $\mathcal{P}$ of length $m = O(d \log A)$ generating P, in linear time. This proves the claimed bounds.

Strengthening the Statement: We now prove the lower bound restricted to instances with $n = \Theta(N^{\alpha_n})$, $M = \Theta(N^{\alpha_M})$ and $m = \Theta(N^{\alpha_m})$ for any $0 < \alpha_n < 1$ and $0 < \alpha_m \leq \alpha_M \leq 1$. Let $\varepsilon > 0$ and set $\beta := \min\{1, \alpha_M + \alpha_n\}$. We choose $k_1, k_2 \geq 1$ such that $k_1 + k_2 = k$ and $k_1 \approx \min\{\alpha_M, 1 - \alpha_n\}k/\beta$ and $k_2 \approx \alpha_n k/\beta$. Note that k_1, k_2 are restricted to be integers, however, for sufficiently large k depending only on $\varepsilon, \alpha_M, \alpha_n$, we

can ensure $k_1 \leq (1+\varepsilon/4)\min\{\alpha_M, 1-\alpha_n\}k/\beta$ and $k_2+1 \leq (1+\varepsilon/4)\alpha_n k/\beta$. Note that for the dimension d we can assume $d \leq A$, since otherwise an $O(A^{k-\varepsilon}\text{poly}(d))$ algorithm clearly exists. In particular, for sufficiently large k we have $d \leq A^{(\varepsilon/4)\cdot\min\{\alpha_M,\alpha_m,\alpha_n,1-\alpha_n\}k/\beta}$. This yields

$$\begin{aligned} N &= O(dA^k) = O(A^{(1+\varepsilon/2)k/\beta}),\\ M &= O(dA^{k_1}) = O(A^{(1+\varepsilon/2)\alpha_M k/\beta}),\\ n &= O(dA^{k_2+1}) = O(A^{(1+\varepsilon/2)\alpha_n k/\beta}),\\ m &= O(d\log A) = O(A^{(1+\varepsilon/2)\alpha_m k/\beta}). \end{aligned}$$

Standard padding[6] of these four parameters allows us to achieve equality, up to constant factors, in the above inequalities, which yields the desired $n = \Theta(N^{\alpha_n})$, $M = \Theta(N^{\alpha_M})$ and $m = \Theta(N^{\alpha_m})$. Any $O(\min\{N, nM\}^{1-\varepsilon})$ algorithm for Pattern Matching with Wildcards in this setting would now imply an algorithm for k-OV in time $O(\min\{A^{(1+\varepsilon/2)k/\beta}, A^{(1+\varepsilon/2)(\alpha_M+\alpha_n)k/\beta}\}^{1-\varepsilon}) = O(A^{(1+\varepsilon/2)(1-\varepsilon)\min\{1,\alpha_M+\alpha_n\}k/\beta}) = O(A^{(1-\varepsilon/2)k})$, where we used the definition of β and $(1+\varepsilon/2)(1-\varepsilon) \leq 1-\varepsilon/2$. This contradicts the k-OV conjecture, finishing the proof. ■

We obtain the analogous result for Substring Hamming Distance by a linear-time reduction from Pattern Matching with Wildcards which is provided in the full version.

Theorem III.5. *Assuming the k-OV conjecture, Substring Hamming Distance on constant-size alphabet takes time $\min\{N, nM\}^{1-o(1)}$. This holds even restricted to instances with $n = \Theta(N^{\alpha_n})$, $M = \Theta(N^{\alpha_M})$ and $m = \Theta(N^{\alpha_m})$ for any $0 < \alpha_n < 1$ and $0 < \alpha_m \leq \alpha_M \leq 1$.*

IV. Conclusion

With this paper we started the fine-grained complexity of analyzing compressed data, thus providing lower bound tools for a practically highly relevant area. We focused on the most basic problems on strings, leaving many other stringology problems for future work. Besides strings, there is a large literature on grammar-compressed other forms of data, e.g. graphs. It would be interesting to apply our framework and classify the important problems in these contexts as well.

Specifically, we leave the following open problems.

- Determine the optimal running time for the Disjointness, Hamming Distance, and Subsequence problems.
- Generalize our lower bound for LCS to Edit Distance.
- For NFA Acceptance we obtained tight bounds in case of a potentially dense automaton with q states and up to $O(q^2)$ transitions. Prove tight bounds for the case of sparse automata with $O(q)$ transitions.
- For large (i.e. superconstant) alphabet size, some bounds given in this paper are not tight, most prominently for Generalized Pattern Matching, Substring Hamming Distance, and Pattern Matching with Wildcards. Determine the optimal running time in this case.
- For all lower bounds presented in this paper, check whether they can be improved to work for binary strings.

[6]Add a prefix of wildcards to the pattern and a prefix of 1's to the text, and partially decompress the SLPs.

Acknowledgements

This paper would not have been possible without Oren Weimann and *Schloss Dagstuhl*. Inspired by a Dagstuhl seminar on Compressed Pattern Matching in October, and while attending a Dagstuhl seminar on Fine-Grained Complexity in November, Oren asked in the open problems session whether SETH can explain the lack of $O((nN)^{1-\varepsilon})$ algorithms for problems like LCS on compressed strings. Later, in January, three of the authors of this paper attended a Dagstuhl seminar on Parameterized Complexity and made key progress towards the results of this work. Part of the work was also performed while visiting the Simons Institute for the Theory of Computing, Berkeley, CA. We thank Paweł Gawrychowski for helpful comments.

A.A. was supported by Virginia Vassilevska Williams' NSF Grants CCF-1417238 and CCF-1514339, and BSF Grant BSF:2012338. Arturs Backurs was supported by an IBM PhD Fellowship, the NSF and the Simons Foundation. While performing part of this work, M. Künnemann was affiliated with University of California, San Diego.

References

[1] A. Abboud, A. Backurs, and V. Vassilevska Williams. If the current clique algorithms are optimal, so is Valiant's parser. In *Proc. 56th IEEE Annual Symposium on Foundations of Computer Science (FOCS'15)*, pages 98–117. IEEE, 2015.

[2] A. Abboud, A. Backurs, and V. Vassilevska Williams. Tight Hardness Results for LCS and other Sequence Similarity Measures. In *Proc. 56th IEEE Annual Symposium on Foundations of Computer Science (FOCS'15)*, pages 59–78, 2015.

[3] A. Abboud, T. D. Hansen, V. Vassilevska Williams, and R. Williams. Simulating branching programs with edit distance and friends: or: a polylog shaved is a lower bound made. In *Proc. 48th Annual ACM Symposium on Theory of Computing (STOC'16)*, pages 375–388, 2016.

[4] A. Abboud, V. Vassilevska Williams, and O. Weimann. Consequences of faster sequence alignment. In *Proc. 41st International Colloquium on Automata, Languages, and Programming (ICALP'14)*, pages 39–51, 2014.

[5] A. Abboud, R. Williams, and H. Yu. More applications of the polynomial method to algorithm design. In *Proc. 26th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'15)*, pages 218–230, 2015.

[6] A. Apostolico, G. M. Landau, and S. Skiena. Matching for run-length encoded strings. In *Proc. 1997 International Conference on Compression and Complexity of Sequences (SEQUENCES'97)*, pages 348–356. IEEE, 1997.

[7] O. Arbell, G. M. Landau, and J. S. Mitchell. Edit distance of run-length encoded strings. *Information Processing Letters*, 83(6):307–314, 2002.

[8] A. Backurs and P. Indyk. Edit Distance Cannot Be Computed in Strongly Subquadratic Time (unless SETH is false). In *Proc. 47th Annual ACM Symposium on Theory of Computing (STOC'15)*, pages 51–58, 2015.

[9] A. Backurs and P. Indyk. Which regular expression patterns are hard to match? In *Proc. 57th IEEE Annual Symposium on Foundations of Computer Science (FOCS'16)*, 2016.

[10] P. Bille, P. H. Cording, and I. L. Gørtz. Compressed subsequence matching and packed tree coloring. In *Proc. Annual Symposium on Combinatorial Pattern Matching (CPM'14)*, pages 40–49, 2014.

[11] P. Bille, G. M. Landau, R. Raman, K. Sadakane, S. R. Satti, and O. Weimann. Random access to grammar-compressed strings and trees. *SIAM Journal on Computing*, 44(3):513–539, 2015.

[12] K. Bringmann. Why walking the dog takes time: Frechet distance has no strongly subquadratic algorithms unless seth fails. In *Proc. of 55th IEEE Annual Symposium on Foundations of Computer Science (FOCS'14)*, pages 661–670, 2014.

[13] K. Bringmann, F. Grandoni, B. Saha, and V. Vassilevska Williams. Truly sub-cubic algorithms for language edit distance and rna-folding via fast bounded-difference min-plus product. In *Proc. 57th IEEE Annual Symposium on Foundations of Computer Science (FOCS'16)*, pages 375–384. IEEE, 2016.

[14] K. Bringmann, A. Grønlund, and K. G. Larsen. A dichotomy for regular expression membership testing. In *Proc. 58th IEEE Annual Symposium on Foundations of Computer Science (FOCS'17)*, 2017.

[15] K. Bringmann and M. Künnemann. Quadratic Conditional Lower Bounds for String Problems and Dynamic Time Warping. In *Proc. 56th IEEE Annual Symposium on Foundations of Computer Science (FOCS'15)*, pages 79–97, 2015.

[16] H. Bunke and J. Csirik. An improved algorithm for computing the edit distance of run-length coded strings. *Information Processing Letters*, 54(2):93–96, 1995.

[17] C. Calabro, R. Impagliazzo, and R. Paturi. A duality between clause width and clause density for SAT. In *Proc. 21st IEEE Conference on Computational Complexity (CCC'06)*, pages 252–260, 2006.

[18] P. Cégielski, I. Guessarian, Y. Lifshits, and Y. Matiyasevich. Window subsequence problems for compressed texts. In *Proc. 1st International Computer Science Symposium in Russia (CSR'06)*, pages 127–136. Springer, 2006.

[19] Y. Chang. Conditional lower bound for RNA folding problem. *CoRR*, abs/1511.04731, 2015.

[20] M. Charikar, E. Lehman, D. Liu, R. Panigrahy, M. Prabhakaran, A. Sahai, and A. Shelat. The smallest grammar problem. *STOC'02 and IEEE Transactions on Information Theory*, 51(7):2554–2576, 2005.

[21] R. Clifford, A. Fontaine, E. Porat, B. Sach, and T. Starikovskaya. The k-mismatch problem revisited. In *Proc. 27th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'17)*, pages 2039–2052, 2016.

[22] J. Cocke. Programming languages and their compilers. 1970.

[23] M. Crochemore, G. M. Landau, and M. Ziv-Ukelson. A subquadratic sequence alignment algorithm for unrestricted scoring matrices. *SIAM Journal on Computing*, 32(6):1654–1673, 2003.

[24] G. Das, R. Fleischer, L. Gasieniec, D. Gunopulos, and J. Kärkkäinen. Episode matching. In *Proc. Annual Symposium on Combinatorial Pattern Matching (CPM'97)*, pages 12–27. Springer, 1997.

[25] T. Gagie, P. Gawrychowski, and S. J. Puglisi. Faster approximate pattern matching in compressed repetitive texts. In *International Symposium on Algorithms and Computation*, pages 653–662. Springer, 2011.

[26] L. Gasieniec, M. Karpinski, W. Plandowski, and W. Rytter. Efficient algorithms for Lempel-Ziv encoding. *Proc. 5th Scandinavian Workshop on Algorithm Theory (SWAT'96)*, pages 392–403, 1996.

[27] P. Gawrychowski. Faster algorithm for computing the edit distance between slp-compressed strings. In *International Symposium on String Processing and Information Retrieval*, pages 229–236. Springer, 2012.

[28] R. Giancarlo, D. Scaturro, and F. Utro. Textual data compression in computational biology: a synopsis. *Bioinformatics*, 25(13):1575–1586, 2009.

[29] S. Grumbach and F. Tahi. Compression of DNA sequences. In *Proc. Data Compression Conference (DCC'93)*, pages 340–350, 1993.

[30] S. Grumbach and F. Tahi. A new challenge for compression algorithms: genetic sequences. *Information Processing & Management*, 30(6):875–886, 1994.

[31] D. Hermelin, G. M. Landau, S. Landau, and O. Weimann. Unified compression-based acceleration of edit-distance computation. *Algorithmica*, 65(2):339–353, 2013.

[32] J. E. Hopcroft, R. Motwani, and J. D. Ullman. Automata theory, languages, and computation. *International Edition*, 24, 2006.

[33] R. Impagliazzo and R. Paturi. On the complexity of k-sat. *Journal of Computer and System Sciences*, 62(2):367–375, 2001.

[34] R. Impagliazzo, R. Paturi, and F. Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63:512–530, 2001.

[35] A. Jeż. A really simple approximation of smallest grammar. *Theoretical Computer Science*, 616:141–150, 2016.

[36] A. Jeż. Recompression: a simple and powerful technique for word equations. *Journal of the ACM (JACM)*, 63(1):4, 2016.

[37] T. Kasami. An efficient recognition and syntax algorithm for context-free algorithms. In *Technical Report AFCRL-65-758 Air Force Cambridge Research Lab Bedford, Mass*. 1965.

[38] N. J. Larsson. *Structures of string matching and data compression*. Department of Computer Science, Lund University, 1999.

[39] A. Lempel and J. Ziv. On the complexity of finite sequences. *IEEE Transactions on Information Theory*, 22(1):75–81, 1976.

[40] Y. Lifshits. Processing compressed texts: A tractability border. In *Proc. Annual Symposium on Combinatorial Pattern Matching (CPM'07)*, pages 228–240. Springer, 2007.

[41] Q. Liu, Y. Yang, C. Chen, J. Bu, Y. Zhang, and X. Ye. RNACompress: Grammar-based compression and informational complexity measurement of RNA secondary structure. *BMC bioinformatics*, 9(1):176, 2008.

[42] M. Lohrey. Algorithmics on SLP-compressed strings: A survey. 2012.

[43] U. Manber. A text compression scheme that allows fast searching directly in the compressed file. *ACM Transactions on Information Systems (TOIS)*, 15(2):124–136, 1997.

[44] N. Markey and P. Schnoebelen. A ptime-complete matching problem for slp-compressed words. *Information Processing Letters*, 90(1):3–6, 2004.

[45] C. G. Nevill-Manning and I. H. Witten. Compression and explanation using hierarchical grammars. *The Computer Journal*, 40(2 and 3):103–116, 1997.

[46] W. Plandowski and W. Rytter. Application of lempel-ziv encodings to the solution of word equations. *Automata, Languages and Programming*, pages 731–742, 1998.

[47] W. Plandowski and W. Rytter. Complexity of language recognition problems for compressed words. In *Jewels are forever*, pages 262–272. Springer, 1999.

[48] A. Polak. Why is it hard to beat n^2 for longest common weakly increasing subsequence? *arXiv preprint arXiv:1703.01143*, 2017.

[49] R. Radicioni and A. Bertoni. Grammatical compression: compressed equivalence and other problems. *Discrete Mathematics and Theoretical Computer Science*, 12(4):109, 2010.

[50] W. Rytter. Application of Lempel–Ziv factorization to the approximation of grammar-based compression. *Theoretical Computer Science*, 302(1-3):211–222, 2003.

[51] W. Rytter. Grammar compression, lz-encodings, and string algorithms with implicit input. In *Proc. 31st International Colloquium on Automata, Languages, and Programming (ICALP'04)*, pages 15–27. Springer, 2004.

[52] H. Sakamoto. Grammar compression: Grammatical inference by compression and its application to real data. In *ICGI*, pages 3–20, 2014.

[53] D. Sculley and C. E. Brodley. Compression and machine learning: A new perspective on feature space vectors. In *Proc. Data Compression Conference (DCC'06)*, pages 332–341, 2006.

[54] Y. Shibata, T. Kida, S. Fukamachi, M. Takeda, A. Shinohara, T. Shinohara, and S. Arikawa. Byte pair encoding: A text compression scheme that accelerates pattern matching. Technical report, Technical Report DOI-TR-161, Department of Informatics, Kyushu University, 1999.

[55] A. Tiskin. Faster subsequence recognition in compressed strings. *Journal of Mathematical Sciences*, 158(5):759–769, 2009.

[56] A. Tiskin. Fast distance multiplication of unit-Monge matrices. In *Proc. 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'10)*, pages 1287–1296. SIAM, 2010.

[57] A. Tiskin. Towards approximate matching in compressed strings: Local subsequence recognition. In *Proc. International Computer Science Symposium in Russia (CSR'11)*, pages 401–414. Springer, 2011.

[58] L. G. Valiant. General context-free recognition in less than cubic time. *Journal of Computer and System Sciences*, 10(2):308–315, 1975.

[59] T. A. Welch. A technique for high-performance data compression. *Computer*, 6(17):8–19, 1984.

[60] R. Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theoretical Computer Science*, 348(2):357–365, 2005.

[61] I. H. Witten, A. Moffat, and T. C. Bell. *Managing gigabytes: compressing and indexing documents and images*. Morgan Kaufmann, 1999.

[62] T. Yamamoto, H. Bannai, S. Inenaga, and M. Takeda. Faster subsequence and don't-care pattern matching on compressed texts. In *Proc. Annual Symposium on Combinatorial Pattern Matching (CPM'11)*, pages 309–322. Springer, 2011.

[63] D. H. Younger. Recognition and parsing of context-free languages in time n^3. *Information and Control*, 10(2):189–208, 1967.

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