Tighter Connections Between Formula-SAT and Shaving Logs

Amir Abboud[1]
IBM Almaden Research Center, San Jose, USA
amir.abboud@ibm.com

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

Abstract

A noticeable fraction of Algorithms papers in the last few decades improve the running time of well-known algorithms for fundamental problems by logarithmic factors. For example, the $O(n^2)$ dynamic programming solution to the Longest Common Subsequence problem (LCS) was improved to $O(n^2/\log^2 n)$ in several ways and using a variety of ingenious tricks. This line of research, also known as *the art of shaving log factors*, lacks a tool for proving negative results. Specifically, how can we show that it is unlikely that LCS can be solved in time $O(n^2/\log^3 n)$?

Perhaps the only approach for such results was suggested in a recent paper of Abboud, Hansen, Vassilevska W. and Williams (STOC'16). The authors blame the hardness of shaving logs on the hardness of solving satisfiability on boolean formulas (Formula-SAT) faster than exhaustive search. They show that an $O(n^2/\log^{1000} n)$ algorithm for LCS would imply a major advance in circuit lower bounds. Whether this approach can lead to tighter barriers was unclear.

In this paper, we push this approach to its limit and, in particular, prove that a well-known barrier from complexity theory stands in the way for shaving five additional log factors for fundamental combinatorial problems. For LCS, regular expression pattern matching, as well as the Fréchet distance problem from Computational Geometry, we show that an $O(n^2/\log^{7+\varepsilon} n)$ runtime would imply new Formula-SAT algorithms.

Our main result is a reduction from SAT on formulas of size s over n variables to LCS on sequences of length $N = 2^{n/2} \cdot s^{1+o(1)}$. Our reduction is essentially as efficient as possible, and it greatly improves the previously known reduction for LCS with $N = 2^{n/2} \cdot s^c$, for some $c \geq 100$.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness

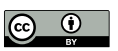
Keywords and phrases Fine-Grained Complexity, Hardness in P, Formula-SAT, Longest Common Subsequence, Frechet Distance

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.8

Related Version A full version can be found at `https://arxiv.org/abs/1804.08978`.

Acknowledgements Part of the work was performed while visiting the Simons Institute for the Theory of Computing, Berkeley, CA. We are grateful to Avishay Tal for telling us about his algorithm for SAT on bipartite formulas. We also thank Mohan Paturi, Rahul Santhanam, Srikanth Srinivasan, and Ryan Williams for answering our questions about the state of the art of Formula-SAT algorithms, and Arturs Backurs, Piotr Indyk, Mikkel Thorup, and Virginia

[1] The work was completed when A.A. was at Stanford University and was supported by Virginia Vassilevska Williams' NSF Grants CCF-1417238 and CCF-1514339, and BSF Grant BSF:2012338.



Vassilevska Williams for helpful discussions regarding regular expressions. We also thank an anonymous reviewer for ideas leading to shaving off a second log-factor for Formula-Pair.

1 Introduction

Since the early days of Algorithms research, a noticeable fraction of papers each year *shave log factors* for fundamental problems: they reduce the best known upper bound on the time complexity from $T(n)$ to $T(n)/\log^c n$, for some $c > 0$. While in some cases a cynic would call such results "hacks" and "bit tricks", there is no doubt that they often involve ingenious algorithmic ideas and suggest fundamental new ways to look at the problem at hand. In his survey, Timothy Chan calls this kind of research "The Art of Shaving Logs" [37]. In many cases, we witness a race of shaving logs for some problem, in which a new upper bound is found every few months, without giving any hints on when this race is going to halt. For example, in the last few years, the upper bound for combinatorial Boolean Matrix Multiplication dropped from $O(n^3/\log^2 n)$ [16], to $O(n^3/\log^{2.25} n)$ [20], to $O(n^3/\log^3 n)$ [38], and most recently to $O(n^3/\log^4 n)$ [99]. Perhaps the single most important missing technology for this kind of research is a tool for proving *lower bounds*.

Consider the problem of computing the Longest Common Subsequence (LCS) of two strings of length n. LCS has a simple $O(n^2)$ time dynamic programming algorithm [93, 46]. Several approaches have been utilized in order to shave log factors such as the "Four Russians" technique [16, 61, 74, 23, 58], utilizing bit-parallelism [10, 47, 62], and working with compressed strings [48, 54]. The best known upper bounds are $O(n^2/\log^2 n)$ for constant size alphabets [74], and $O(n^2 \log\log n/\log^2 n)$ for large alphabets [58]. But can we do better? Can we solve LCS in $O(n^2/\log^3 n)$ time? While the mathematical intrigue is obvious, we remark that even such mild speedups for LCS could be significant in practice. Besides its use as the *diff* operation in unix, LCS is at the core of highly impactful similarity measures between biological data. A heuristic algorithm called BLAST for a generalized version of LCS (namely, the Local Alignment problem [87]) has been cited more than sixty thousand times [14]. While such heurisitics are much faster than the near-quadratic time algorithms above, they are not guaranteed to return an optimal solution and are thus useless in many applications, and biologists often fall back to (highly optimized implementations of) the quadratic solutions, see, e.g. [71, 72].

How would one show that it is hard to shave logs for some problem? A successful line of work, inspired by NP-hardness, utilizes "fine-grained reductions" to prove statements of the form: a small improvement over the known runtime for problem A implies a breakthrough algorithm for problem B, refuting a plausible hypothesis about the complexity of B. For example, it has been shown that if LCS can be solved in $O(n^{2-\varepsilon})$ time, where $\varepsilon > 0$, then there is a breakthrough $(2-\delta)^n$ algorithm for CNF-SAT, and the Strong Exponential Time Hypothesis (SETH, defined below) is refuted [2, 29]. Another conjecture that has been used to derive interesting lower bounds states that the 3-SUM problem[2] cannot be solved in $O(n^{2-\varepsilon})$ time. It is natural to ask: can we use these conjectures to rule out log-factor improvements for problems like LCS? And even more optimistically, one might hope to base the hardness of LCS on a more standard assumption like $\mathsf{P} \neq \mathsf{NP}$. Unfortunately, we can formally prove that these assumptions are not sharp enough to lead to any consequences for log-factor improvements, if only Turing reductions are used. In Section 3 we prove the

[2] 3-SUM asks, given a list of n numbers, to find three that sum to zero. The best known upper bound is $O(n^2(\log\log n)^2/\log n)$ for real numbers [59, 53, 56] and $O(n^2(\log\log n/\log n)^2)$ for integers [21].

following theorem which also shows that an $O(f(n)/\log^c(f(n)))$ time algorithm for problem A cannot imply, via a fine-grained reduction, an $O(g(n)^{1-\varepsilon})$ algorithm for problem B, unless B is (unconditionally) solvable in $O(g(n)^{1-\delta})$ time.

▶ **Theorem 1.1** (Informally). *If for some $c > 0$ there is a fine-grained reduction proving that LCS is not in $O(n^2/\log^c n)$ time unless SETH fails, then SETH is false.*

Note that it also does not suffice to simply make SETH stronger by postulating a higher running time lower bound for CNF-SAT, since *superpolynomial improvements are known* for this problem [81, 34, 49, 8]. Similarly, we cannot base a study of log-factor improvements on the APSP conjecture, since *superlogarithmic improvements are known* for APSP [97]. (However, 3SUM could be a candidate to base higher lower bounds on, since only log-factor improvements are known [59, 53, 56, 21], see Section A of the full version for a discussion.)

Thus, in a time when super-linear lower bounds for problems like LCS are far out of reach, and our only viable approach to obtaining such negative results is reductions-based, we are left with two options. We could either leave the study of log-factor improvements in limbo, without a technology for proving negative results, or we could search for natural and convincing assumptions that are more fine-grained than SETH that could serve as the basis for the negative results we desire. Such assumptions were recently proposed by Abboud, Hansen, Vassilevska Williams and Williams [3]. The authors blame the hardness of shaving logs on the hardness of solving satisfiability on boolean formulas (Formula-SAT) faster than exhaustive search[3], by polynomial factors (which are log-factors in the runtime), a task for which there are well known "circuit lower bound" barriers in complexity theory. They show that an $O(n^2/\log^{1000} n)$ algorithm for LCS would imply a major advance in circuit lower bounds. In the final section of this paper, we give a more detailed argument in favor of this approach. Whether one should expect it to lead to *tight* barriers, i.e. explaining the lack of $O(n^2/\log^3 n)$ algorithms for LCS or any other natural problem, was completely unclear.

The Machine Model

We use the *Word-RAM* model on words of size $\Theta(\log n)$, where there is a set of operations on words that can be performed in time $O(1)$. Most papers do not fix the concrete set of allowed operations, and instead refer to "typical Boolean and arithmetic operations". In this paper, we choose a set of operations $\mathcal{P}$ that is *robust* with respect to changing the word size: For any operation $\circ \in \mathcal{P}$, given two words a, b (of size $\Theta(\log n)$) we can compute $a \circ b$ in time $(\log n)^{1+o(1)}$ on a Word RAM with word size $\Theta(\log \log n)$ and operation set $\mathcal{P}$. In other words, if we split a, b into $\Theta(\log n/\log \log n)$ words of size $\Theta(\log \log n)$ then $a \circ b$ can still be computed very efficiently.

This robustness in particular holds for the following standard set of operations: initializing a cell with a constant, bitwise AND, OR, NOT, shift, addition, subtraction, multiplication, and division with remainder (since multiplication and division have near-linear time algorithms).

The results in this paper will get gradually weaker as we relax the restriction on near-linear time per operation to higher runtimes, however, even with this restriction, to the best of our knowledge this model captures all log shaving results in the literature (on the "standard" Word RAM model without fancy word operations).

[3] In [3] the authors focus on SAT on Branching Programs (BPs) rather than formulas, but due to standard transformations between BPs and formulas, the two problems are equivalent up to polynomial factors. Focusing on Formula-SAT will be crucial to the progress we make in this paper.

Formula-SAT

A boolean formula over n input variables can be viewed as a tree in which every leaf is marked by an input variable or its negation and every internal node or *gate* represents some basic boolean operation. Throughout this introduction we will only talk about *deMorgan* formulas, in which every gate is from the set $\{\wedge, \vee\}$. The *size* of the formula is defined to be the number of leaves in the tree.

In the Formula-SAT problem we are given a formula F of size s over n inputs, and we have to decide whether there is an input $\{0,1\}^n$ that makes it output 1. A naive algorithm takes $O(2^n \cdot s)$ time, since evaluating the formula on some input takes $O(s)$ time. Can we do better? We will call a SAT algorithm *non-trivial*[4] if it has a runtime at most $O(\frac{2^n}{n^\varepsilon})$, for some $\varepsilon > 0$.

It seems like a clever algorithm must look at the given formula F and try to gain a speedup by analyzing it. The more complicated F can be, the harder the problem becomes. Indeed, Dantsin and Hirsch [49] survey dozens of algorithms for SAT on CNF formulas which exploit their structure. For k-CNF formulas of size s there are $2^n s/2^{\Omega(n/k)}$ time algorithms (e.g. [81]), and for general CNF formulas the bound is $2^n s/2^{\Omega(n/\log \Delta)}$ where $\Delta = s/n$ is the clause-to-variable ratio [34, 49, 8]. The popular SETH [66, 35] essentially says that this is close to optimal, and that there is no $2^n s/2^{\Omega(n)}$ algorithm for CNF-SAT. For arbitrary deMorgan formulas, the upper bounds are much worse. A FOCS'10 paper by Santhanam [84] and several recent improvements [41, 43, 42, 70, 91] solve Formula-SAT on formulas of size $s = n^{3-16\varepsilon}$ in time $2^n s^{O(1)}/2^{n^\varepsilon}$, which is non-trivial only for $s = o(n^3)$, and going beyond cubic seems extremely difficult. This leads us to the first barrier which we will transform into a barrier for shaving logs.

▶ **Hypothesis 1.2.** *There is no algorithm that can solve SAT on deMorgan formulas of size $s = n^{3+\Omega(1)}$ in $O(\frac{2^n}{n^\varepsilon})$ time, for some $\varepsilon > 0$, in the Word-RAM model.*

Perhaps the main reason to believe this hypothesis is that despite extensive algorithmic attacks on variants of SAT (perhaps the most extensively studied problem in computer science) over decades, none of the ideas that anyone has ever come up with seem sufficient to refute it. Recent years have been particularly productive in non-trivial algorithms designed for special cases of Circuit-SAT [84, 86, 64, 35, 98, 22, 40, 67, 63, 44, 83, 57] (in addition to the algorithms for deMorgan formulas above) and this hypothesis still stands.

A well-known "circuit lower bounds" barrier seems to be in the way for refuting Hypothesis 1.2: can we find an explicit boolean function that cannot be computed by deMorgan formulas of cubic size? Functions that require formulas of size $\Omega(n^{1.5})$ [89] and $\Omega(n^2)$ [69] have been known since the 60's and 70's, respectively. In the late 80's, Andreev [15] proved an $\Omega(n^{2.5})$ which was later gradually improved to $\Omega(n^{2.55})$ by Nisan and Wigderson [65] and to $\Omega(n^{2.63})$ by Paterson and Zwick [79] until Håstad proved his $n^{3-o(1)}$ lower bound in FOCS'93 [60] (a recent result by Tal improves the $n^{o(1)}$ term [90]). All these lower bound results use the "random restrictions" technique, first introduced in this context by Subbotovskaya in 1961 [89], and it is known that a substantially different approach must be taken in order to go beyond the cubic barrier. What does this have to do with Formula-SAT algorithms? Interestingly, this same "random restrictions" technique was crucial to *all* the non-trivial Formula-SAT algorithms mentioned above. This is not a coincidence, but only one out of

[4] Some works on SAT algorithms used this term for runtimes of the form $2^n \mathrm{poly}(s)/n^{\omega(1)}$. In our context, we need to be a bit more fine-grained.

the many examples of the intimate connection between the task of designing non-trivial algorithms for SAT on a certain class $\mathcal{F}$ of formulas or circuits and the task of proving lower bounds against $\mathcal{F}$. This connection is highlighted in many recent works and in several surveys [85, 78, 96]. The intuition is that both of these tasks seem to require identifying a strong structural property of functions in $\mathcal{F}$. There is even a formal connection shown by Williams [95], which in our context implies that solving Formula-SAT on formulas of size $O(n^{3.1})$ in $O(2^n/n^{10})$ time (which is only slightly stronger than refuting Hypothesis 1.2) is sufficient in order to prove that there is a function in the class E^{NP} that cannot be computed by formulas of size $O(n^{3.1})$ (see [3] for more details). This consequence would be the first polynomial progress on the fundamental question of worst case formula lower bounds since Håstad's result.

1.1 Our Results: New Reductions

Many recent papers have reduced CNF-SAT to fundamental problems in P to prove SETH-based lower bounds (e.g. [80, 82, 6, 4, 27, 18, 7, 1, 33, 5, 19, 75, 39]). Abboud et al. [3] show that even SAT on formulas, circuits, and more, can be efficiently reduced to combinatorial problems in P. In particular, they show that Formula-SAT on formulas of size s over n inputs can be reduced to an instance of LCS on sequences of length $N = O(2^{n/2} \cdot s^{1000})$. This acts as a barrier for shaving logs as follows. A hypothetical $O(N^2/\log^c N)$ time algorithm for LCS can be turned into an algorithm for Formula-SAT in time

$$n^{1+o(1)} \cdot (2^{n/2} \cdot s^{1000})^2/(\log 2^{\Omega(n)})^c = O(2^n \cdot s^{2000}/n^{c-1}),$$

which for a large enough $c \geq 2001$ would refute Hypothesis 1.2. The first $n^{1+o(1)}$ factor in the runtime comes from the jump from n to $N = 2^n$ and our Word-RAM machine model: whenever the LCS algorithm wants to perform a unit-cost operation on words of size $\Theta(\log N)$ (this is much more than the word size of our SAT algorithm which is only $\Theta(\log n) = \Theta(\log \log N)$), the SAT algorithm can simulate it in $(\log N)^{1+o(1)} = n^{1+o(1)}$ time in the Word-RAM model with words of size $\Theta(\log n)$.

Our main result is a *much* more efficient reduction to LCS. For large but constant size alphabets, we get a near-linear dependence on the formula size, reducing the s^{1000} factor to just $s^{1+o(1)}$.

▶ **Theorem 1.3.** *Formula-SAT on formulas of size s on n inputs can be reduced to an instance of LCS on two sequences over an alphabet of size σ of length $N = 2^{n/2} \cdot s^{1+O(1/\log\log\sigma)}$, in $O(N)$ time.*

Thus, if LCS on sequences of length N and alphabet of size $\omega(1)$ can be solved in $O(N^2/\log^c N)$ time, then Formula-SAT can be solved in $2^n \cdot \frac{s^{2+o(1)}}{n^c} \cdot n^{1+o(1)}$ time. Recall that the known upper bound for LCS is $O(n^2/\log^c n)$ for any constant alphabet size, with $c = 2$, and we can now report that the barrier of cubic formulas stands in the way of improving it to $c > 7$ (see Corollary 1.6 below).

The novelty in the proof of Theorem 1.3 over [3] is discussed in Section 2. As an alternative to Theorem 1.3, in Section D of the full version we present another reduction to LCS which is much simpler than all previously known reductions, but uses a larger alphabet.

Fréchet Distance

An important primitive in computational geometry is to judge how similar are two basic geometric objects, such as polygonal curves, represented as sequences of points in d-dimensional Euclidean space. Such curves are ubiquitous, since they arise naturally as trajectory data of

moving objects, or as time-series data of stock prices and other measures. The most popular similarity measure for curves in computational geometry is the Fréchet distance, also known as dog-leash-distance. For formal definitions see Section F of the full version. The Fréchet distance has found many applications (see, e.g., [76, 26, 30]) and developed to a rich field of research with many generalizations and variants (see, e.g., [11, 17, 13, 51, 36, 45, 32, 50, 73, 68]).

This distance measure comes in two variants: the continuous and the discrete. A classic algorithm by Alt and Godau [12, 55] computes the continuous Fréchet distance in time $O(n^2 \log n)$ for two given curves with n vertices. The fastest known algorithm runs in time $O(n^2(\log\log n)^2)$ (on the Word RAM) [31]. If we only want to decide whether the Fréchet distance is at most a given value δ, this algorithm runs in time $O(n^2(\log\log n)^2/\log n)$. For the discrete Fréchet distance, the original algorithm has running time $O(n^2)$ [52], which was improved to $O(n^2 \log\log n/\log n)$ by Agarwal et al. [9]. Their algorithm runs in time $O(n^2 \log\log n/\log^2 n)$ for the decision version. It is known that both versions of the Fréchet distance are SETH-hard [27]. However, this does not rule out log factor improvements. In particular, no reduction from versions of SETH on formulas or branching programs is known.

In this paper we focus on the *decision version of the discrete Fréchet distance* (which we simply call "Fréchet distance" from now on). We show that Fréchet distance suffers from the same barriers for shaving logs like LCS. In particular, this reduction allows us to base the usual $\Omega(n^{2-\varepsilon})$ lower bound on a weaker assumption than SETH, such as NC-SETH (see the discussion in [3]). This is the first NC-SETH hardness for a problem that does not admit alignment gadgets (as in [29]).

▶ **Theorem 1.4.** *Formula-SAT on formulas of size s on n inputs can be reduced to an instance of the Fréchet distance on two curves of length $N = O(2^{n/2} \cdot s)$, in $O(N)$ time.*

Regular Expression Pattern Matching

Our final example is the fundamental *Regular Expression Pattern Matching* problem: Decide whether a given regular expression of length m matches a substring of a text of length n. Again, there is a classical $O(nm)$ algorithm [92], and the applicability and interest in this problem resulted in algorithms shaving log factors; the first one by Myers [77] was improved by Bille and Thorup [24] to time $O(mn/\log^{1.5} n)$. Recently, Backurs and Indyk proved an $n^{2-o(1)}$ SETH lower bound [19], and performed an impressive study of the exact time complexity of the problem with respect to the complexity of the regular expression. This study was essentially completed by Bringmann, Grønlund, and Larsen [28], up to $n^{o(1)}$ factors. In Section E of the full version we show that this problem is also capable of efficiently simulating formulas and thus has the same barriers as LCS and Fréchet distance.

▶ **Theorem 1.5.** *Formula-SAT on formulas of size s on n inputs can be reduced to an instance of Regular Expression Pattern Matching on text and pattern of length $N = O(2^{n/2} \cdot s \log s)$ over a constant size alphabet, in $O(N)$ time.*

Consequences of the Cubic Formula Barrier

We believe that SAT on formulas can be tightly connected to many other natural problems in P. As we discuss in the next section, such reductions seem to require problem-specific engineering and are left for future work. The main point of this paper is to demonstrate the possibility of basing such ultra fine-grained lower bounds on one common barrier. Our conditional lower bounds are summarized in the following corollary, which shows that current log-shaving algorithms are very close to the well-known barrier from complexity theory of cubic formula lower bounds.

▶ **Corollary 1.6.** *For all $\varepsilon > 0$, solving any of the following problems in $O(n^2/\log^{7+\varepsilon} n)$ time refutes Hypothesis 1.2, and solving them in $O(n^2/\log^{17+\varepsilon} n)$ time implies that E^{NP} cannot be computed by non-uniform formulas of cubic size:*

- *LCS over alphabets of size $\omega(1)$*
- *The Fréchet distance on two curves in the plane*
- *Regular Expression Pattern Matching over constant size alphabets.*

The main reason that our lower bounds above are not tight (the gap between 2 and 7) is that we need to start from SAT on *cubic* size formulas rather than linear size ones, due to the fact that clever algorithms do exist for smaller formulas. We remark that throughout the paper we will work with a class of formulas we call $\mathcal{F}_1$ (see Section 2 below), also known as bipartite formulas, that are more powerful than deMorgan formulas yet our reduction to LCS can support them as well. This makes our results stronger, since $\mathcal{F}_1$-Formula-SAT could be a harder problem than SAT on deMorgan formulas. In fact, in an earlier version of the paper we had suggested the hypothesis that $\mathcal{F}_1$-Formula-SAT does not have non-trivial algorithms even on *linear size* formulas. This stronger hypothesis would give higher lower bounds. However, Avishay Tal (personal communication) told us about such a non-trivial algorithm for formulas of size up to $n^{2-\Omega(1)}$ using tools from quantum query complexity. We are optimistic that one could borrow such ideas or the "random restrictions" technique from SAT algorithms in order to shave more logs for combinatorial problems such as LCS. This is an intriguing direction for future work.

2 Technical Overview and the Reduction to LCS

We first define the class of frmulas $\mathcal{F}_1$. A formula F of size s over n variables $x_1, \ldots, x_n$ is in the class $\mathcal{F}_1$ iff it has the following properties. The gates in the first layer (nodes in the tree whose children are all leaves) compute arbitrary functions $C : \{0,1\}^{n/2} \to \{0,1\}$, as long as C can be computed in $2^{o(n)}$ time and all children of a gate are marked with variables in $\{x_1, \ldots, x_{n/2}\}$ or with variables in $\{x_{n/2+1}, \ldots, x_n\}$ but not with both. W.l.o.g. we can assume that the inputs are only connected to nodes in the first layer. The gates in the other layers compute deMorgan gates, i.e., OR and AND gates. The size of F is considered to be the *number of gates in the first layer*. Since F is a formula and thus has fanout 1, our size measure is up to constant factors equal to the total number of all gates except the inputs. Note that the complexity of the functions in the first layer and their number of incoming wires, i.e. the number of leaves in the tree, do not count towards the size of F.

All the reductions from SAT to problems in P mentioned in the introduction start with a split-and-list reduction to some "pair finding" problem. In the SETH lower bounds, CNF-SAT is reduced to the Orthogonal-Vectors problem of finding a pair $a \in A, b \in B, A, B \subseteq \{0,1\}^d$ that are orthogonal [94]. When starting from Formula-SAT, we get a more complex pair-finding problem. In Section B of the full version we show a simple reduction from SAT on formulas from the class $\mathcal{F}_1$ (which contains deMorgan formulas) to the following problem.

▶ **Definition 2.1** (Formula-Pair Problem)**.** Given a deMorgan formula over $2m$ variables $F = F(x_1, \ldots, x_m, y_1, \ldots, y_m)$ (each appearing once in F), and two sets of vectors $A, B \subseteq \{0,1\}^m$ of size n, decide if there is a pair $a \in A, b \in B$ such that $F(a_1, \ldots, a_m, b_1, \ldots, b_m) =$ true.

In Section B of the full version we show a Four-Russians type algorithm that solves Formula-Pair in $O(n^2 m/\log^2 n)$ time, and even when $m = |F| = (\log n)^{1+o(1)}$ no $O(n^2/\log^{1+\varepsilon} n)$ upper bound is known. By our reduction, such an upper bound would

imply a non-trivial algorithm for SAT on formulas from $\mathcal{F}_1$. Moreover, Hypothesis 1.2 implies that we cannot solve Formula-Pair in $O(n^2/\log^{\varepsilon} n)$ time, for $m = (\log n)^{3+\Omega(1)}$. In the next sections, we reduce Formula-Pair to LCS, from which Theorem 1.3 follows. A simpler reduction using much larger alphabet size can be found in Section D of the full version.

▶ **Theorem 2.2.** Formula-Pair *on formulas of size s and lists of size n can be reduced to an instance of LCS on two strings over alphabet of size $\sigma \geq 2$ of length $O(n \cdot s^{1+O(1/\log\log\sigma)})$, in linear time.*

The reduction constructs strings x, y and a number ρ such that $\mathrm{LCS}(x, y) \geq \rho$ holds if and only if the given Formula-Pair instance (F, A, B) is satisfiable. The approach is similar to the reductions from Orthogonal-Vectors to sequence alignment problems (e.g. [6, 27, 18, 2, 29]). The big difference is that our formula F can be much more complicated than a CNF, and so we will need more powerful gadgets. Sequence gadgets that are able to simulate the evaluation of deMorgan formulas were (implicitly) constructed in [3] with a recursive approach. Our main contribution is an *extremely efficient* implementation of such gadgets with LCS.

The main part of the reduction is to construct *gate gadgets*: for any vectors $a, b \in \{0,1\}^m$ and any gate g of F, we construct strings $x(g, a)$ and $y(g, b)$ whose LCS determines whether gate g evaluates to true for input (a, b) to F (see Section 2.1). Once we have this, to find a pair of vectors $a \in A, b \in B$ satisfying F, we combine the strings $x(r, a), y(r, b)$, constructed for the root r of F, using a known construction of so-called alignment gadgets [2, 29] from previous work (see Section C.1 of the full version).

Let us quickly explain how [3] constructed gate gadgets and the main ideas that go into our new construction. There are two kinds of gadgets, corresponding to the two types of gates in F: AND and OR gates. Since the AND gadgets will be relatively simple, let us consider the OR gadgets. Fix two inputs a, b, and let $g = (g_1 \vee g_2)$ be an OR gate, and assume that we already constructed gate gadgets for g_1, g_2, namely $x_1 = x(g_1, a), y_1 = y(g_1, b), x_2 = x(g_2, a), y_2 = y(g_2, b)$ so that for $i \in \{1, 2\}$ we have that $\mathrm{LCS}(x_i, y_i)$ is large if the gate g_i outputs true on input (a, b), and it is smaller otherwise. In [3], these gadgets were combined as follows. Let β be an upper bound on the total length of the gadgets x_i, y_i. We add a carefully chosen padding of 0's and 1's, so that any optimal matching of the two strings will have to match *either* x_1, y_1 or x_2, y_2 but not both.

$$\begin{aligned} x &:= \quad 0^{4\beta}\ x_1\ 1^{\beta}\ x_2\ 0^{4\beta} \\ y &:= \quad y_1\ 1^{\beta} 0^{4\beta} 1^{\beta}\ y_2 \end{aligned}$$

One then argues that, in any optimal LCS matching of x, y, the $0^{4\beta}$ block of y must be matched either left or right. If it's matched left, then the total score will be equal to $4\beta + \beta + LCS(x_2, y_2)$ while if it's matched right, we will get $4\beta + \beta + LCS(x_1, y_1)$. Thus, $LCS(x, y)$ is determined by the OR of g_1, g_2. The blowup of this construction is a multiplicative factor of 11 with every level of the formula, and the length of the gadget of the root will end up roughly $11^{\mathrm{depth}(F)}$. To obtain our tight lower bounds, we will need to decrease this blowup to $1 + \varepsilon_\sigma$ at every level, where ε_σ goes to 0 when the alphabet size σ tends to infinity. With the above construction, decreasing the length of the padding will allow the optimal LCS matching to cheat, e.g. by matching y_1 to both x_1 and x_2, and no longer corresponding to the OR of g_1, g_2.

Our first trick is an ultra-efficient OR gadget in case we are allowed unbounded alphabet size. We take x_1, y_1 and transform all their letters into a new alphabet Σ^{g_1}, and we take x_2, y_2 and transform their letters into a disjoint alphabet Σ^{g_2}. Then our OR gadget does not

require any padding at all:

$$x := x_1\ x_2$$
$$y := y_2\ y_1$$

The crossing structure of this construction means that any LCS matching that matches letters from x_1, y_1 cannot also match letters from x_2, y_2, and vice versa, while the disjoint alphabets make sure that there can be no matches between x_1, y_2 or x_2, y_1. With such gadgets we can encode a formula of size s with $O(s)$ letters, for details see Section D of the full version.

But how would such an idea work for *constant size alphabets*? Once we allow x_1 and y_2 to share even a single letter, this argument breaks. Natural attempts to simulate this construction with smaller alphabets, e.g. by replacing each letter with a random sequence, do not seem to work, and we do not know how to construct such an OR gadget with a smaller alphabet *in a black box way*. The major part of our proof will be a careful examination of the formula and the sub-gadgets g_1, g_2 in order to reduce the alphabet size to a large enough constant, while using padding that is only $1 + \varepsilon_\sigma$ times the length of the sub-gadgets. We achieve this by combining this crossing gadget with a small padding that will reuse letters from alphabets that were used much deeper in the formula, and we will argue that the noise we get from recycling letters is dominated by our paddings, in any optimal matching.

We remark that the reduction of [3] can be implemented in a generic way with any problem that admits *alignment gadgets* as defined in [29], giving formula-gadgets of size $s^{O(1)}$. The list of such problems includes LCS and Edit-Distance on binary strings. However, to get gadgets of length $s^{1+o(1)}$ it seems that problem-specific reductions are necessary. A big open question left by our work is to find the most efficient reduction from Formula-SAT to Edit-Distance. A very efficient OR gadget, even if the alphabet is unbounded, might be (provably) impossible. Can we use this intuition to shave more log factors for Edit-Distance?

Fréchet Distance falls outside the alignment gadgets framework of [29] and no reduction from Formula-SAT was known before. In Section F of the full version we prove such a reduction by a significant boosting of the SETH-lower bound construction of [27]. In order to implement recursive AND/OR gadgets, our new proof utilizes the geometry of the curves, in contrast to [27] which only used ten different points in the plane.

In the remainder of this section we present the details of the reduction to LCS. Some missing proofs can be found in Section C of the full version.

2.1 Implementing Gates

Fix vectors $a, b \in \{0,1\}^m$ (where $2m$ is the number of inputs to F). In this section we prove the following lemma which demonstrates our main construction.

▶ **Lemma 2.3.** *For any sufficiently large $\sigma > 0$ let $\tau = (\log \sigma)^{1/4}$. We can inductively construct, for each gate g of F, strings $x(g) = x(g,a)$ and $y(g) = y(g,b)$ over alphabet size $5\sigma^2$ and a number $\rho(g)$ such that for $L(g) := \mathrm{LCS}(x(g), y(g))$ we have (1) $L(g) \leq \rho(g)$ and (2) $L(g) = \rho(g)$ if and only if gate g evaluates to true on input (a,b) to F. Moreover, we have $|x(g)| = |y(g)| = n(g) \leq 6\tau \cdot |F_g|(1 + 7/\tau)^{\mathrm{depth}(F_g)}$, where F_g is the subformula of F below g.*

In this construction, we use disjoint size-5 alphabets $\Sigma_1, \ldots, \Sigma_{\sigma^2}$, determining the total alphabet size as $5\sigma^2$. Each gate g is assigned an alphabet $\Sigma_{f(g)}$. We fix the function f later.

In the following, consider any gate g of F, and write the gate alphabet as $\Sigma_{f(g)} = \{0,1,2,3,4\}$. For readability, we write $x = x(g)$ and similarly define y, n, L, ρ. If g has fanin 2, write g_1, g_2 for the children of g. Moreover, let $x_1 = x(g_1)$ and similarly define y_1, n_1, L_1, ρ_1 and $x_2, y_2, n_2, L_2, \rho_2$.

Input Gate

The base case is an input bit a_i to F (input bits b_j are symmetric). Interpreting a_i as a string of length 1 over alphabet $\{0,1\}$, note that $\mathrm{LCS}(a_i, 1) = a_i$. Hence, the strings $x = a_i$ and $y = 1$, with $n = \rho = 1$, trivially simulate the input bit a_i.

AND Gates

Consider an AND gate g and let $\beta := \lceil (n_1 + n_2)/\tau^2 \rceil$. We construct strings x, y as

$$\begin{aligned} x &:= x_1\, 0^\beta\, 1^\beta\, x_2 \\ y &:= y_1\, 0^\beta\, 1^\beta\, y_2 \end{aligned}$$

▶ **Lemma 2.4.** *If* $\mathrm{LCS}(x_2, y_1), \mathrm{LCS}(x_1, y_2) \leq \beta/4$ *and the symbols* $0, 1$ *appear at most* $\beta/16$ *times in each of* x_1, x_2, y_1, *and* y_2, *then we have* $L = \mathrm{LCS}(x, y) = 2\beta + L_1 + L_2$.

Later we will choose the gate alphabets $\Sigma_{f(g)}$ such that the precondition of the above lemma is satisfied. Setting $\rho := 2\beta + \rho_1 + \rho_2$ we thus inductively obtain (1) $L \leq \rho$ and (2) $L = \rho$ if and only if g_1 and g_2 both evaluate to true. Thus, we correctly simulated the AND gate g. It remains to prove the lemma.

Proof. Clearly, we have $L \geq \mathrm{LCS}(x_1, y_1) + \mathrm{LCS}(0^\beta, 0^\beta) + \mathrm{LCS}(1^\beta, 1^\beta) + \mathrm{LCS}(x_2, y_2) = 2\beta + L_1 + L_2$. For the other direction, consider any LCS z of x, y. If z does not match any symbol of the left half of x, $x_1 0^\beta$, with any symbol of the right half of y, $1^\beta y_2$, and it does not match any symbol of the right half of x, $1^\beta x_2$, with any symbol of the left half of y, $y_1 0^\beta$, then we can split both strings in the middle and obtain $L = |z| \leq \mathrm{LCS}(x_1 0^\beta, y_1 0^\beta) + \mathrm{LCS}(1^\beta x_2, 1^\beta y_2)$. Greedy suffix/prefix matching now yields $L \leq \big(\mathrm{LCS}(x_1, y_1) + \beta\big) + \big(\beta + \mathrm{LCS}(x_2, y_2)\big) = 2\beta + L_1 + L_2$.

In the remaining case, there is a matching from some left half to some right half. By symmetry, we can assume that there is a matching from the left half of x to the right half of y. We can moreover assume that z matches a symbol of x_1 with a symbol of $1^\beta y_2$, since the case that z matches a symbol of y_2 with a symbol of $x_1 0^\beta$ is symmetric. Now no symbol in 0^β in x can be matched with a symbol in 0^β in y. We obtain a rough upper bound on $L = |z|$ by summing up the LCS length of all remaining $4 \cdot 4 - 1 = 15$ pairs of a part $x' \in \{x_1, 0^\beta, 1^\beta, x_2\}$ in x and a part $y' \in \{y_1, 0^\beta, 1^\beta, y_2\}$ in y. This yields $L \leq L_1 + L_2 + \beta + 2 \cdot \beta/4 + 8 \cdot \beta/16 = 2\beta + L_1 + L_2$, finishing the proof. ◀

OR Gates

Consider an OR gate g and again let $\beta := \lceil (n_1 + n_2)/\tau^2 \rceil$. We first make the LCS target values equal by adding $4^{|\rho_1 - \rho_2|}$ to the shorter of x_2/y_2 and x_1/y_1, i.e., we set $x_1' := 4^{\max\{0, \rho_2 - \rho_1\}} x_1$ and similarly $y_1' := 4^{\max\{0, \rho_2 - \rho_1\}} y_1$, $x_2' := 4^{\max\{0, \rho_1 - \rho_2\}} x_2$, $y_2' := 4^{\max\{0, \rho_1 - \rho_2\}} y_2$. Note that the resulting strings satisfy $L_1' := \mathrm{LCS}(x_1', y_1') \leq \rho' := \max\{\rho_1, \rho_2\}$ and $L_1' = \rho'$ if and only if g_1 evaluates to true, and similarly $L_2' := \mathrm{LCS}(x_2', y_2') \leq \rho'$ and $L_2' = \rho'$ if and only if g_2 evaluates to true. We construct the strings x, y as

$$\begin{aligned} x &:= 0^\beta 1^\beta\, x_1'\, 2^\beta 3^\beta\, x_2'\, 0^\beta 1^\beta \\ y &:= 2^\beta 3^\beta\, y_2'\, 0^\beta 1^\beta\, y_1'\, 2^\beta 3^\beta \end{aligned}$$

▶ **Lemma 2.5.** *If* $\mathrm{LCS}(x_2, y_1), \mathrm{LCS}(x_1, y_2) \leq \beta/8$ *and the symbols* $0, 1, 2, 3$ *appear at most* $\beta/48$ *times in each of* x_1, x_2, y_1, *and* y_2, *then* $L = \mathrm{LCS}(x, y) = 4\beta + \max\{L_1', L_2'\}$.

Later we will choose the gate alphabets $\Sigma_{f(g)}$ such that the precondition of the above lemma is satisfied. Setting $\rho := 4\beta + \rho' = 4\beta + \max\{\rho_1, \rho_2\}$ we thus inductively obtain (1) $L \leq \rho$ and (2) $L = \rho$ if and only if at least one of g_1 and g_2 evaluates to true, so we correctly simulated the OR gate g. The proof of the Lemma is in Section C of the full version.

Analyzing the Length

Note that the above constructions inductively yields strings $x(g), y(g)$ simulating each gate g. We inductively prove bounds for $n(g)$ and $\rho(g)$. See Section C of the full version.

▶ **Lemma 2.6.** *We have* $n(g) \leq 6\tau \cdot |F_g|(1+7/\tau)^{\mathrm{depth}(F_g)}$ *and* $\rho(g) \leq 6|F_g|(1+7/\tau)^{\mathrm{depth}(F_g)}$ *for any gate* g*, where* F_g *is the subformula of* F *below* g*.*

Fixing the Gate Alphabets

Now we fix the gate alphabet $\Sigma_{f(g)}$ for any gate g. Again let $\Sigma_{(i,j)}$, $i, j \in [\sigma]$, be disjoint alphabets of size 5, and let $\Sigma := \bigcup_{i,j} \Sigma_{(i,j)}$. For any gate g of F, we call its distance to the root the *height* $h(g)$. For any h, order the gates with height h from left to right, and let $\iota(g)$ be the index of gate g in this order, for any gate g with height h. Note that $(h(g), \iota(g))$ is a unique identifier of gate g. We define $f(g) := (h(g) \bmod \sigma, \iota(g) \bmod \sigma)$, i.e., we set the gate alphabet of g to $\Sigma_{f(g)} = \Sigma_{(h(g) \bmod \sigma, \iota(g) \bmod \sigma)}$. Note that the overall alphabet Σ has size $5\sigma^2$. Recall that we set $\tau := (\log \sigma)^{1/4}$.

It remains to show that the preconditions of Lemmas 2.4 and 2.5 are satisfied. Specifically, consider a gate g with children g_1, g_2. As before, let x, y, n be the strings and string length constructed for gate g, and let x_i, y_i, n_i be the corresponding objects for g_i, $i \in \{1, 2\}$. We need to show:

(1) $\mathrm{LCS}(x_2, y_1), \mathrm{LCS}(x_1, y_2) \leq (n_1 + n_2)/(8\tau^2)$, and
(2) each $c \in \Sigma_{f(g)}$ appears at most $(n_1 + n_2)/(48\tau^2)$ times in each of x_1, x_2, y_1, and y_2.

We call a gate g' in the subformula F_g *d-deep* if $h(g') \geq h(g) + d$, and *d-shallow* otherwise. For each symbol c in x or y we can trace our construction to find the gate g' in F_g at which we introduced c to x or y. In other words, each symbol in x, y *stems* from some gate g' below g.

First consider (2). Observe that all symbols in x, y stemming from σ-shallow gates do not belong to the gate alphabet $\Sigma_{f(g)}$, since the function $f(g')$ has $(h(g') \bmod \sigma)$ as the first component, which repeats only every σ levels. Thus, if a symbol $c \in \Sigma_{f(g)}$ occurs in x_i or y_i, then this occurence stems from a σ-deep gate. We now argue that only few symbols in x, y stem from deep gates. For any $d > 0$, let N_d be the number of symbols in x (or, equivalently, y) steming from d-deep gates. Note that N_d is equal to the total string length $\sum n(g')$, summed over all gates g' in F_g with height $h(g') = h(g) + d$. Observe that our construction increases the string lengths in each step by at least a factor $1 + 1/\tau^2$, i.e., $N_d \geq (1 + 1/\tau^2)N_{d+1}$ holds for any d. It follows that $N_\sigma \leq N_1/(1 + 1/\tau^2)^{\sigma-1} = (n_1 + n_2)/(1 + 1/\tau^2)^{\sigma-1}$. Hence, each symbol in $\Sigma_{f(g)}$ appears at most $(n_1+n_2)/(1+1/\tau^2)^{\sigma-1}$ times in each of x_1, x_2, y_1, y_2. Since $\tau = (\log \sigma)^{1/4}$, we have $(1 + 1/\tau^2)^{\sigma-1} = 2^{\Omega(\sigma/\sqrt{\log \sigma})} \geq 48\sqrt{\log \sigma} = 48\tau^2$ for sufficiently large σ. This proves (2).

For (1), remove all $\log(\sigma)$-deep symbols from x_1 and y_2 to obtain strings x_1', y_2'. Note that we removed exactly $N_{\log \sigma}$ symbols from each of x_1, y_2. This yields $\mathrm{LCS}(x_1, y_2) \leq 2N_{\log \sigma} + \mathrm{LCS}(x_1', y_2')$. For x_1', y_2', we claim that any $\log(\sigma)$-shallow gates $g_1' \neq g_2'$ in F_g have disjoint alphabets $\Sigma_{f(g_1')}, \Sigma_{f(g_2')}$. Indeed, if $h(g_1') \neq h(g_2')$ then since the first component $(h(g') \bmod \sigma)$ of $f(g')$ repeats only every σ levels we have $f(g_1') \neq f(g_2')$. If $h(g_1') = h(g_2') =: h$,

then note that each gate g' in height h has a unique label $\iota(g') \bmod \sigma$, since there are σ such labels and there are at most $2^{h-h(g)} < \sigma$ gates with height h in F_g. Hence, x'_1 and y'_2 use disjoint alphabets, and we obtain $\mathrm{LCS}(x'_1, y'_2) = 0$. Thus, $\mathrm{LCS}(x_1, y_2) \leq 2N_{\log\sigma}$. As above, we bound $N_{\log\sigma} \leq (n_1+n_2)/(1+1/\tau^2)^{\log\sigma-1}$, so that $\mathrm{LCS}(x_1, y_2) \leq 2(n_1+n_2)/(1+1/\tau^2)^{\log\sigma-1}$. Since $\tau = (\log\sigma)^{1/4}$, we have $(1+1/\tau^2)^{\log\sigma-1}/2 = 2^{\Omega(\sqrt{\log\sigma})} \geq 8\sqrt{\log\sigma} = 8\tau^2$ for sufficiently large σ. This yields (1), since the strings x_2, y_1 are symmetric, finishing the proof of Lemma 2.3.

Finalizing the Proof

Let us sketch how we complete the proof of Theorem 2.2. The full details are in Section C.1 of the full version. First, for all vectors $a \in A, b \in B$ we construct gate gadgets for the output gate of the formula, i.e. *formula gadgets*, by invoking Lemma 2.3. Then we combine all these gadgets by applying a standard alignment gadget [2, 29] to get our final sequences of length $O\big(n\tau|F|(1+7/\tau)^{\mathrm{depth}(F)}\big)$ and with alphabet of size $O(\sigma^2)$. The LCS of the final sequence will be determined by the existence of a satisfying pair. Since a priori the depth of F could be as large as $|F|$, the factor $(1+7/\tau)^{\mathrm{depth}(F)}$ in our length bound is not yet satisfactory. Thus, as a preprocessing before the above construction, we decrease the depth of F using a depth-reduction result of Bonet and Buss [88, 25]: for all $k \geq 2$ there is an equivalent formula F' with depth at most $(3k\ln 2)\log|F|$ and size $|F'| \leq |F|^{1+1/(1+\log(k-1))}$. Choosing the parameters correctly, we get final sequences of length $O\big(n|F|^{1+O(1/\log\log\sigma)}\big)$.

3 On the Limitations of Fine-Grained Reductions

With the increasingly complex web of reductions and conjectures used in the "Hardness in P" research, one might oppose to our use of nonstandard assumptions. Why can't we base the hardness of shaving logs on one of the more established assumptions such as SETH, or even better, on $\mathsf{P} \neq \mathsf{NP}$? We conclude the paper with a proof that such results are not possible if one is restricted to fine-grained reductions, which is essentially the only tool we have in this line of research.

Let A be a problem with best known upper bound of $T_A(n)$ on inputs of size n, and let B be a problem with best known upper bound of $T_B(n)$ on inputs of size n. Throughout this section we assume that these runtime are non-decreasing functions, such as 2^n or n^2. A fine-grained reduction from "solving A in time $T_A(n)/g(n)$" to "solving B in time $T_B(n)/f(n)$" proves that improving $T_B(n)$ to $T_B(n)/f(n)$ improves T_A to $T_A(n)/g(n)$. Formally, it is an algorithm X that solves A and it is allowed to call an oracle for problem B, as long as the following bound holds. Let n_i be the size of the instance in the i^{th} call to problem B that our algorithm performs, where $i \leq t$ for some value t, and let $T_X(n)$ be the runtime of X excluding the time it takes to answer all the instances of problem B. It must be that $T_X(n) + \sum_{i=1}^{t} T_B(n_i)/f(n_i) \leq T_A(n)/g(n)$. This is a natural adaptation of the definition of fine-grained reductions from previous works, where the improvements were restricted to be by polynomial factors. We can now give a formal version of Theorem 1.1 from the introduction.

▶ **Theorem 3.1.** *If for some $c, \varepsilon > 0$ and all $k \geq 2$ there is a fine-grained reduction from solving k-SAT in time $\mathrm{poly}(n,m)2^n/2^{\varepsilon n}$ to solving LCS in time $O(n^2/\log^c n)$, then SETH is false.*

Proof. Assume there was a fine-grained reduction from k-SAT to LCS as above. This means that there is an algorithm X for k-SAT that makes t calls to LCS with instances of size $n_1, \ldots, n_t$ such that $T_X(n) + \sum_{i=1}^{t} n_i^2/\log^c n_i = O(\mathrm{poly}(n,m)2^n/2^{\varepsilon n})$. But then consider

algorithm X' which simulates X and whenever X makes a call to the LCS oracle with an instance of size n_i, our algorithm will execute the known quadratic time solution for LCS. Let n_{max} be the size of the largest instance we call, and note that $n_{max} < 2^n$. Simple calculations show that X' solves k-SAT and has a running time of $T_X(n) + \sum_{i=1}^{t} n_i^2 = O\big(\text{poly}(n,m)2^n/2^{\varepsilon n}\big) \cdot \log^c n_{max} = O(\text{poly}(n,m)2^n/2^{\varepsilon n})$ for all k, refuting SETH. ◀

References

1. Amir Abboud, Arturs Backurs, Thomas Dueholm Hansen, Virginia Vassilevska Williams, and Or Zamir. Subtree isomorphism revisited. In *Proc. of 27th SODA*, pages 1256–1271, 2016.
2. Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. Tight Hardness Results for LCS and other Sequence Similarity Measures. In *Proc. of 56th FOCS*, pages 59–78, 2015.
3. Amir Abboud, Thomas Dueholm Hansen, Virginia Vassilevska Williams, and Ryan Williams. Simulating branching programs with edit distance and friends: or: a polylog shaved is a lower bound made. In *Proc. of the 48th STOC*, pages 375–388, 2016.
4. Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *Proc. of 55th FOCS*, pages 434–443, 2014.
5. Amir Abboud, Virginia Vassilevska Williams, and Joshua R. Wang. Approximation and fixed parameter subquadratic algorithms for radius and diameter in sparse graphs. In *Proc. of 27th SODA*, pages 377–391, 2016.
6. Amir Abboud, Virginia Vassilevska Williams, and Oren Weimann. Consequences of faster sequence alignment. In *Proc. of 41st ICALP*, pages 39–51, 2014.
7. Amir Abboud, Virginia Vassilevska Williams, and Huacheng Yu. Matching triangles and basing hardness on an extremely popular conjecture. In *Proc. of 47th STOC*, pages 41–50, 2015.
8. Amir Abboud, Ryan Williams, and Huacheng Yu. More applications of the polynomial method to algorithm design. In *Proc. of 26th SODA*, pages 218–230, 2015.
9. P. Agarwal, R. B. Avraham, H. Kaplan, and M. Sharir. Computing the discrete Fréchet distance in subquadratic time. In *Proc. 24th ACM-SIAM Symposium on Discrete Algorithms (SODA'13)*, pages 156–167, 2013.
10. Lloyd Allison and Trevor I Dix. A bit-string longest-common-subsequence algorithm. *Information Processing Letters*, 23(5):305–310, 1986.
11. Helmut Alt and Maike Buchin. Can we compute the similarity between surfaces? *Discrete & Computational Geometry*, 43(1):78–99, 2010.
12. Helmut Alt and Michael Godau. Computing the Fréchet distance between two polygonal curves. *International Journal of Computational Geometry & Applications*, 5(1-2):78–99, 1995.
13. Helmut Alt, Christian Knauer, and Carola Wenk. Comparison of distance measures for planar curves. *Algorithmica*, 38(1):45–58, 2004.
14. Stephen F Altschul, Thomas L Madden, Alejandro A Schäffer, Jinghui Zhang, Zheng Zhang, Webb Miller, and David J Lipman. Gapped blast and psi-blast: a new generation of protein database search programs. *Nucleic acids research*, 25(17):3389–3402, 1997.
15. Alexander E Andreev. About one method of obtaining more than quadratic effective lower bounds of complexity of pi-schemes, 1987.
16. V Arlazarov, E Dinic, I Faradzev, and M Kronrod. On economic construction of the transitive closure of a direct graph. In *Sov. Math (Doklady)*, volume 11, pages 1209–1210, 1970.

17 Boris Aronov, Sariel Har-Peled, Christian Knauer, Yusu Wang, and Carola Wenk. Fréchet distance for curves, revisited. In *Proc. 14th Annual European Symposium on Algorithms (ESA'06)*, volume 4168 of *LNCS*, pages 52–63. Springer, 2006.

18 Arturs Backurs and Piotr Indyk. Edit Distance Cannot Be Computed in Strongly Subquadratic Time (unless SETH is false). In *Proc. of 47th STOC*, pages 51–58, 2015.

19 Arturs Backurs and Piotr Indyk. Which regular expression patterns are hard to match? In *FOCS*, 2016.

20 Nikhil Bansal and Ryan Williams. Regularity lemmas and combinatorial algorithms. In *Proc. of 50th FOCS*, pages 745–754, 2009.

21 Ilya Baran, Erik D. Demaine, and Mihai Patrascu. Subquadratic algorithms for 3sum. *Algorithmica*, 50(4):584–596, 2008. doi:10.1007/s00453-007-9036-3.

22 Paul Beame, Russell Impagliazzo, and Srikanth Srinivasan. Approximating acˆ 0 by small height decision trees and a deterministic algorithm for# acˆ 0sat. In *Computational Complexity (CCC), 2012 IEEE 27th Annual Conference on*, pages 117–125. IEEE, 2012.

23 Philip Bille and Martin Farach-Colton. Fast and compact regular expression matching. *Theoretical Computer Science*, 409(3):486–496, 2008. doi:10.1016/j.tcs.2008.08.042.

24 Philip Bille and Mikkel Thorup. Faster regular expression matching. In *International Colloquium on Automata, Languages, and Programming*, pages 171–182. Springer, 2009.

25 Maria Luisa Bonet and Samuel R. Buss. Size-depth tradeoffs for boolean fomulae. *Inf. Process. Lett.*, 49(3):151–155, 1994. doi:10.1016/0020-0190(94)90093-0.

26 Sotiris Brakatsoulas, Dieter Pfoser, Randall Salas, and Carola Wenk. On map-matching vehicle tracking data. In *Proc. 31st International Conference on Very Large Data Bases (VLDB'05)*, pages 853–864, 2005.

27 Karl Bringmann. Why walking the dog takes time: Frechet distance has no strongly subquadratic algorithms unless seth fails. In *Proc. of 55th FOCS*, pages 661–670, 2014.

28 Karl Bringmann, Allan Grønlund, and Kasper Green Larsen. A dichotomy for regular expression membership testing. *CoRR*, abs/1611.00918, 2016. URL: http://arxiv.org/abs/1611.00918, arXiv:1611.00918.

29 Karl Bringmann and Marvin Künnemann. Quadratic Conditional Lower Bounds for String Problems and Dynamic Time Warping. In *Proc. of 56th FOCS*, pages 79–97, 2015.

30 Kevin Buchin, Maike Buchin, Joachim Gudmundsson, Maarten Löffler, and Jun Luo. Detecting commuting patterns by clustering subtrajectories. *International Journal of Computational Geometry & Applications*, 21(3):253–282, 2011.

31 Kevin Buchin, Maike Buchin, Wouter Meulemans, and Wolfgang Mulzer. Four soviets walk the dog - with an application to Alt's conjecture. In *Proc. 25th ACM-SIAM Symposium on Discrete Algorithms (SODA'14)*, pages 1399–1413, 2014.

32 Kevin Buchin, Maike Buchin, and Yusu Wang. Exact algorithms for partial curve matching via the Fréchet distance. In *Proc. 20th ACM-SIAM Symposium on Discrete Algorithms (SODA'09)*, pages 645–654, 2009.

33 Massimo Cairo, Roberto Grossi, and Romeo Rizzi. New bounds for approximating extremal distances in undirected graphs. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 363–376, 2016.

34 Chris Calabro, Russell Impagliazzo, and Ramamohan Paturi. A duality between clause width and clause density for SAT. In *Proc. of 21st CCC*, pages 252–260, 2006.

35 Chris Calabro, Russell Impagliazzo, and Ramamohan Paturi. The complexity of satisfiability of small depth circuits. In *Proc. of 4th IWPEC*, pages 75–85, 2009.

36 Erin Wolf Chambers, Éric Colin de Verdière, Jeff Erickson, Sylvain Lazard, Francis Lazarus, and Shripad Thite. Homotopic Fréchet distance between curves or, walking your dog in the woods in polynomial time. *Computational Geometry*, 43(3):295–311, 2010.

37 Timothy M. Chan. The art of shaving logs. In *Proc. of the 13th WADS*, page 231, 2013.

38 Timothy M. Chan. Speeding up the four russians algorithm by about one more logarithmic factor. In *Proc. of 26th SODA*, pages 212–217, 2015.

39 Krishnendu Chatterjee, Wolfgang Dvorák, Monika Henzinger, and Veronika Loitzenbauer. Model and objective separation with conditional lower bounds: Disjunction is harder than conjunction. *CoRR*, abs/1602.02670, 2016. URL: `http://arxiv.org/abs/1602.02670`, `arXiv:1602.02670`.

40 Ruiwen Chen. Satisfiability algorithms and lower bounds for boolean formulas over finite bases. In *International Symposium on Mathematical Foundations of Computer Science*, pages 223–234. Springer, 2015.

41 Ruiwen Chen and Valentine Kabanets. Correlation bounds and #sat algorithms for small linear-size circuits. In *Computing and Combinatorics - 21st International Conference, COCOON 2015, Beijing, China, August 4-6, 2015, Proceedings*, pages 211–222, 2015.

42 Ruiwen Chen, Valentine Kabanets, Antonina Kolokolova, Ronen Shaltiel, and David Zuckerman. Mining circuit lower bound proofs for meta-algorithms. *computational complexity*, 24(2):333–392, 2015.

43 Ruiwen Chen, Valentine Kabanets, and Nitin Saurabh. An improved deterministic# sat algorithm for small de morgan formulas. In *International Symposium on Mathematical Foundations of Computer Science*, pages 165–176. Springer, 2014.

44 Ruiwen Chen and Rahul Santhanam. Satisfiability on mixed instances. In *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science, Cambridge, MA, USA, January 14-16, 2016*, pages 393–402, 2016.

45 A. F. Cook and Carola Wenk. Geodesic Fréchet distance inside a simple polygon. *ACM Transactions on Algorithms*, 7(1):193–204, 2010.

46 Thomas H.. Cormen, Charles Eric Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*, volume 6. MIT press Cambridge, 2001.

47 Maxime Crochemore, Costas S Iliopoulos, Yoan J Pinzon, and James F Reid. A fast and practical bit-vector algorithm for the longest common subsequence problem. *Information Processing Letters*, 80(6):279–285, 2001.

48 Maxime Crochemore, Gad M Landau, and Michal Ziv-Ukelson. A subquadratic sequence alignment algorithm for unrestricted scoring matrices. *SIAM journal on computing*, 32(6):1654–1673, 2003.

49 Evgeny Dantsin and Edward A. Hirsch. Worst-case upper bounds. In *Handbook of Satisfiability*, pages 403–424. IOS Press, 2009. `doi:10.3233/978-1-58603-929-5-403`.

50 Anne Driemel and Sariel Har-Peled. Jaywalking your dog: computing the Fréchet distance with shortcuts. *SIAM Journal on Computing*, 42(5):1830–1866, 2013.

51 Anne Driemel, Sariel Har-Peled, and Carola Wenk. Approximating the Fréchet distance for realistic curves in near linear time. *Discrete & Computational Geometry*, 48(1):94–127, 2012.

52 Thomas Eiter and Heikki Mannila. Computing discrete Fréchet distance. Technical Report CD-TR 94/64, Christian Doppler Laboratory for Expert Systems, TU Vienna, Austria, 1994.

53 Ari Freund. Improved subquadratic 3sum. *Algorithmica*, pages 1–19, 2015.

54 Paweł 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.

55 Michael Godau. A natural metric for curves - computing the distance for polygonal chains and approximation algorithms. In *Proc. 8th Symposium on Theoretical Aspects of Computer Science (STACS'91)*, volume 480 of *LNCS*, pages 127–136. Springer, 1991.

56 Omer Gold and Micha Sharir. Improved bounds for 3sum, k-sum, and linear degeneracy. *CoRR*, abs/1512.05279, 2015. URL: `http://arxiv.org/abs/1512.05279`, `arXiv:1512.05279`.

57 Alexander Golovnev, Alexander S Kulikov, Alexander Smal, and Suguru Tamaki. Circuit size lower bounds and# sat upper bounds through a general framework. In *Electronic Colloquium on Computational Complexity (ECCC)*, volume 23, page 22, 2016.

58 Szymon Grabowski. New tabulation and sparse dynamic programming based techniques for sequence similarity problems. In *Stringology*, pages 202–211, 2014. URL: `http://www.stringology.org/event/2014/p19.html`.

59 Allan Grønlund and Seth Pettie. Threesomes, degenerates, and love triangles. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 621–630, 2014.

60 Johan Håstad. The shrinkage exponent of de morgan formulas is 2. *SIAM J. Comput.*, 27(1):48–64, 1998.

61 John Hopcroft, Wolfgang Paul, and Leslie Valiant. On time versus space. *Journal of the ACM (JACM)*, 24(2):332–337, 1977.

62 Heikki Hyyrö. Bit-parallel lcs-length computation revisited. In *Proc. 15th Australasian Workshop on Combinatorial Algorithms (AWOCA 2004)*, pages 16–27. Citeseer, 2004.

63 Russell Impagliazzo, Shachar Lovett, Ramamohan Paturi, and Stefan Schneider. 0-1 integer linear programming with a linear number of constraints. *Electronic Colloquium on Computational Complexity (ECCC)*, 21:24, 2014. URL: `http://eccc.hpi-web.de/report/2014/024`.

64 Russell Impagliazzo, William Matthews, and Ramamohan Paturi. A satisfiability algorithm for ac 0. In *Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms*, pages 961–972. SIAM, 2012.

65 Russell Impagliazzo and Noam Nisan. The effect of random restrictions on formula size. *Random Structures & Algorithms*, 4(2):121–133, 1993.

66 Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-sat. *Journal of Computer and System Sciences*, 62(2):367–375, 2001.

67 Russell Impagliazzo, Ramamohan Paturi, and Stefan Schneider. A satisfiability algorithm for sparse depth two threshold circuits. In *54th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2013, 26-29 October, 2013, Berkeley, CA, USA*, pages 479–488, 2013.

68 Piotr Indyk. Approximate nearest neighbor algorithms for Fréchet distance via product metrics. In *Proc. 18th Annual Symposium on Computational Geometry (SoCG'02)*, pages 102–106, 2002.

69 Valeriy M Khrapchenko. Method of determining lower bounds for the complexity of p-schemes. *Mathematical Notes*, 10(1):474–479, 1971.

70 Ilan Komargodski, Ran Raz, and Avishay Tal. Improved average-case lower bounds for demorgan formula size. In *Foundations of Computer Science (FOCS), 2013 IEEE 54th Annual Symposium on*, pages 588–597. IEEE, 2013.

71 Isaac TS Li, Warren Shum, and Kevin Truong. 160-fold acceleration of the smith-waterman algorithm using a field programmable gate array (fpga). *BMC bioinformatics*, 8(1):1, 2007.

72 Yongchao Liu, Adrianto Wirawan, and Bertil Schmidt. Cudasw++ 3.0: accelerating smith-waterman protein database search by coupling cpu and gpu simd instructions. *BMC bioinformatics*, 14(1):1, 2013.

73 Anil Maheshwari, Jörg-Rüdiger Sack, Kaveh Shahbaz, and Hamid Zarrabi-Zadeh. Fréchet distance with speed limits. *Computational Geometry*, 44(2):110–120, 2011.

74 William J Masek and Michael S Paterson. A faster algorithm computing string edit distances. *Journal of Computer and System sciences*, 20(1):18–31, 1980.

75 Daniel Moeller, Ramamohan Paturi, and Stefan Schneider. Subquadratic algorithms for succinct stable matching. In *Computer Science - Theory and Applications - 11th International Computer Science Symposium in Russia, CSR 2016, St. Petersburg, Russia, June 9-13, 2016, Proceedings*, pages 294–308, 2016.

76 Mario E. Munich and Pietro Perona. Continuous dynamic time warping for translation-invariant curve alignment with applications to signature verification. In *Proc. 7th IEEE International Conference on Computer Vision*, volume 1, pages 108–115, 1999.

77 Gene Myers. A four russians algorithm for regular expression pattern matching. *Journal of the ACM (JACM)*, 39(2):432–448, 1992.

78 Igor C Oliveira. Algorithms versus circuit lower bounds. *arXiv preprint arXiv:1309.0249*, 2013.

79 Michael S Paterson and Uri Zwick. Shrinkage of de morgan formulae under restriction. *Random Structures & Algorithms*, 4(2):135–150, 1993.

80 Mihai Patrascu and Ryan Williams. On the possibility of faster SAT algorithms. In *Proc. of 21st SODA*, pages 1065–1075, 2010.

81 Ramamohan Paturi, Pavel Pudlák, Michael E. Saks, and Francis Zane. An improved exponential-time algorithm for k-sat. *J. ACM*, 52(3):337–364, 2005.

82 Liam Roditty and Virginia Vassilevska Williams. Fast approximation algorithms for the diameter and radius of sparse graphs. In *Proc. of 45th STOC*, pages 515–524, 2013.

83 Takayuki Sakai, Kazuhisa Seto, Suguru Tamaki, and Junichi Teruyama. A satisfiability algorithm for depth-2 circuits with a symmetric gate at the top and and gates at the bottom. In *Electronic Colloquium on Computational Complexity (ECCC)*, 2015.

84 Rahul Santhanam. Fighting perebor: New and improved algorithms for formula and QBF satisfiability. In *Proc. of the 51th FOCS*, pages 183–192, 2010.

85 Rahul Santhanam et al. Ironic complicity: Satisfiability algorithms and circuit lower bounds. *Bulletin of EATCS*, 1(106), 2013.

86 Kazuhisa Seto and Suguru Tamaki. A satisfiability algorithm and average-case hardness for formulas over the full binary basis. *computational complexity*, 22(2):245–274, 2013.

87 Temple F Smith and Michael S Waterman. Identification of common molecular subsequences. *Journal of molecular biology*, 147(1):195–197, 1981.

88 Philip M Spira. On time-hardware complexity tradeoffs for boolean functions. In *Proceedings of the 4th Hawaii Symposium on System Sciences*, pages 525–527, 1971.

89 Bella Abramovna Subbotovskaya. Realizations of linear functions by formulas using+. *Doklady Akademii Nauk SSSR*, 136(3):553–555, 1961.

90 Avishay Tal. Shrinkage of de morgan formulae by spectral techniques. In *Foundations of Computer Science (FOCS), 2014 IEEE 55th Annual Symposium on*, pages 551–560. IEEE, 2014.

91 Avishay Tal. #sat algorithms from shrinkage. *Electronic Colloquium on Computational Complexity (ECCC)*, 22:114, 2015. URL: `http://eccc.hpi-web.de/report/2015/114`.

92 Ken Thompson. Programming techniques: Regular expression search algorithm. *Communications of the ACM*, 11(6):419–422, 1968.

93 Robert A Wagner and Michael J Fischer. The string-to-string correction problem. *Journal of the ACM (JACM)*, 21(1):168–173, 1974.

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

95 Ryan Williams. Improving exhaustive search implies superpolynomial lower bounds. *SIAM Journal on Computing*, 42(3):1218–1244, 2013.

96 Ryan Williams. Algorithms for Circuits and Circuits for Algorithms: Connecting the Tractable and Intractable. In *Proc. International Congress of Mathematicians*, 2014.

97 Ryan Williams. Faster all-pairs shortest paths via circuit complexity. In *Proc. of 46th STOC*, pages 664–673, 2014.

98 Ryan Williams. Nonuniform ACC circuit lower bounds. *J. ACM*, 61(1):2:1–2:32, 2014.

99 Huacheng Yu. An improved combinatorial algorithm for boolean matrix multiplication. In *Proc. of 42nd ICALP*, pages 1094–1105, 2015.