Simulating Branching Programs with Edit Distance and Friends

Or: A Polylog Shaved Is a Lower Bound Made*

Amir Abboud
Stanford University, USA
abboud@cs.stanford.edu

Thomas Dueholm Hansen
Aarhus University, Denmark
tdh@cs.au.dk

Virginia Vassilevska Williams
Stanford University, USA
virgi@cs.stanford.edu

Ryan Williams
Stanford University, USA
rrw@cs.stanford.edu

ABSTRACT

A recent, active line of work achieves tight lower bounds for fundamental problems under the Strong Exponential Time Hypothesis (SETH). A celebrated result of Backurs and Indyk (STOC'15) proves that computing the Edit Distance of two sequences of length n in truly subquadratic $O(n^{2-\varepsilon})$ time, for some $\varepsilon > 0$, is impossible under SETH. The result was extended by follow-up works to simpler looking problems like finding the Longest Common Subsequence (LCS).

SETH is a very strong assumption, asserting that even *linear* size CNF formulas cannot be analyzed for satisfiability with an exponential speedup over exhaustive search. We consider much safer assumptions, e.g. that such a speedup is impossible for SAT on more expressive representations, like subexponential-size NC circuits. Intuitively, this assumption is much more plausible: NC circuits can implement linear algebra and complex cryptographic primitives, while CNFs cannot even approximately compute an XOR of bits.

Our main result is a surprising reduction from SAT on Branching Programs to fundamental problems in P like Edit Distance, LCS, and many others. Truly subquadratic algorithms for these problems therefore have far more remarkable consequences than merely faster CNF-SAT algorithms. For example, SAT on arbitrary $o(n)$-depth bounded fan-in circuits (and therefore also NC-Circuit-SAT) can be solved in $(2-\epsilon)^n$ time.

An interesting feature of our work is that we get major consequences even from mildly subquadratic algorithms for Edit Distance or LCS. For example, we show that if an arbitrarily large polylog factor is shaved from n^2 for Edit Distance then NEXP does not have non-uniform NC^1 circuits.

Categories and Subject Descriptors

F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems

General Terms

Algorithms, Theory

Keywords

Edit distance, longest common subsequence, lower bounds, SETH, circuit complexity

1. INTRODUCTION

A central goal of complexity theory is to understand and prove lower bounds for the time complexity of fundamental problems. One of the most important computational problems is Edit-Distance, the problem of computing the minimum number of edit operations (insertions, deletions, and substitutions) required to transform one sequence into another. A classical dynamic programming algorithm that is taught in basic algorithms courses solves Edit-Distance on sequences of length n in $O(n^2)$ time [25]. This quadratic runtime is prohibitive in many applications, like computational biology and genomics where n is typically a few billions. A faster, e.g. linear time, algorithm would therefore have far-reaching consequences. Despite decades of attempts, no upper bound below $O(n^2/\log^2 n)$ is known for Edit-Distance [31]. All the above applies to the simpler looking Longest Common Subsequence problem (LCS), for which the existence of a faster algorithm was already posed as an important open question in combinatorics by Knuth decades ago [24]. This is a situation where lower bounds are highly desirable, but unfortunately, the current state of the art in complexity is far from providing a lower bound

*This work was done in part while the authors were visiting the Simons Institute for the Theory of Computing, Berkeley, CA. A.A. and V.V.W. were supported by NSF Grants CCF-1417238 and CCF-1514339, and BSF Grant BSF:2012338. T.D.H. was supported by the Carlsberg Foundation, grant no. CF14-0617. R.W. was supported by an Alfred P. Sloan Fellowship and NSF CCF-1212372. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the NSF.

STOC '16 Cambridge, Massachusetts USA

that is close to quadratic for any natural problem in NP, let alone Edit-Distance. Therefore, researchers have turned their attention to conditional lower bounds, and a recent breakthrough by Backurs and Indyk [12] showed that Edit-Distance cannot be solved in truly subquadratic $O(n^{2-\varepsilon})$ time, for some $\varepsilon > 0$, unless the *strong exponential time hypothesis* (SETH), a well-known hypothesis on the complexity of k-SAT, is false.

HYPOTHESIS 1 (SETH). *There does not exists an $\varepsilon > 0$ such that for all $k \geq 3$, k-SAT on n variables and m clauses can be solved in $O(2^{(1-\varepsilon)n} \cdot m)$ time.*

Other interesting recent results show that under SETH, the current algorithms for many central problems in diverse areas of computer science are optimal, up to $n^{o(1)}$ factors. These areas include pattern matching [7, 2, 18, 1], graph algorithms [35, 3, 5, 8, 6], parameterized complexity [32, 2], computational geometry [17, 19], and the list is growing by the day. Bringmann and Künnemann [18] generalize many of the previous SETH lower bounds [7, 17, 12, 2] into one framework; they prove that the problem of computing any similarity measure δ over two sequences (of bits, symbols, points, etc) will require quadratic time, as long as the similarity measure has a certain property (namely, if δ admits *alignment gadgets*). Such similarity measures include Edit-Distance and LCS (even on *binary* sequences), and the Dynamic Time Warping Distance, which is an extensively studied metric in time-series analysis.

These SETH lower bounds are a part of a more general line of work in which one bases the hardness of important problems in P on well-known conjectures about the exact complexity of other famous problems. Other conjectures are 3-SUM and All-Pairs-Shortest-Paths, but in recent years, SETH has been most "successful" at explaining barriers.

Evidence for SETH.

SETH was introduced [27, 21] as a plausible explanation for the lack of $(2-\varepsilon)^n$ algorithms for CNF-SAT, despite of it being one of the most extensively studied problems in computer science. The fastest known algorithms for k-SAT run in time $2^{n-n/O(k)}$ (e.g. [33]), and for CNF-SAT the bound is $2^{n-n/O(\log \Delta)}$ where $\Delta = m/n$ is the clause-to-variable ratio [20, 26, 9]. That is, these algorithms are *almost* enough to refute SETH. Evidence in favor of SETH is circumstantial. For example, natural algorithmic approaches like resolution were shown to require exponentially many steps [15].

There is evidence that SETH will be hard to *refute*, in the form of a "*circuit lower bounds barrier*": refuting SETH is as hard as proving longstanding open lower bound results. Williams showed that faster-than-trivial Circuit-SAT algorithms for many circuit classes $\mathcal{C}$ would imply interesting new lower bounds against that class [40, 42]. Via this connection, and known reductions from certain circuit families to CNF formulas, it is possible to show that refuting SETH implies a new circuit lower bound [29]: E^{NP} cannot be solved by *linear-size series-parallel circuits*[1]. However, this is a *very* weak lower bound consequence.

[1]The class E^{NP} or $\mathsf{TIME}[2^{O(n)}]^{\mathsf{NP}}$ is the class of problems solvable in exponential time with access to an NP oracle. Series-parallel circuits are a special kind of log-depth circuits, also known as Valiant-Series-Parallel circuits [38].

A hierarchy of SAT problems.

A weakness of SETH as a hardness hypothesis is that it is an assumption about CNF SAT, as opposed to a more general SAT problem. Consider a problem in which you are given some representation of a function $\mathcal{B}$ on n input bits and are asked whether $\mathcal{B}$ is satisfiable. If $\mathcal{B}$ is treated as a black-box that we only have input/output access to, then any algorithm will need to spend $\Omega(2^n)$ time in the worst case. Of course, a clever algorithm should attempt to analyze $\mathcal{B}$ in order to decide satisfiability in $o(2^n)$ time. Whether this is possible, depends on how complex and obfuscating the representation is. There is a whole spectrum of increasing complexity of representations, starting from simple objects like DNFs, which are very bad at hiding their satisfiability, up until large circuits or nondeterministic turing machines that we have no idea how to analyze.

For each class of representations $\mathcal{C}$ we can define the corresponding $\mathcal{C}$-SETH, stating that this abstract SAT problem cannot be solved in $(2-\varepsilon)^n$ time. For example, NC-SETH would be the assumption that Circuit-SAT on polynomial size polylog depth circuits (NC circuits) cannot be solved in $(2-\varepsilon)^n$ time. It is well known that NC circuits are capable of complex computations, including most linear-algebraic operations. Moreover, they are believed to be capable of implementing cryptographic primitives like One Way Functions and Pseudorandom Generators, for which the ability to hide satisfiability is essential. In sharp contrast, the original SETH is equivalent (due to the sparsification lemma [28, 20]) to the assumption that even representations that are very low on this spectrum, namely *linear size CNF formulas*, are enough to obfuscate satisfiability. While from the viewpoint of polynomial time solvability, CNF-SAT and NC-SAT are equivalent, this is not the case from a more fine-grained perspective: an algorithm that can decide satisfiability of arbitrary *polynomial* size circuits faster than exhaustive search is far more remarkable than a similar algorithm that can handle only linear size CNF formulas.

As our class $\mathcal{C}$ gets more complex and rich, the $\mathcal{C}$-SETH becomes more credible and appealing as a basis for conditional lower bounds than SETH. However, all previous SETH lower bound proofs relied heavily on the simple nature of CNFs. In this work, we prove the first lower bounds under the much more reliable $\mathcal{C}$-SETH, for classes $\mathcal{C}$ that are far more expressive than CNFs.

Our results.

Our main result is a new efficient reduction from SAT on super-polynomial size *nondeterministic Branching Programs* (BPs) to Edit-Distance, LCS and many other important problems in P. As we discuss below, BPs are vastly more expressive than CNFs. For example, our reduction allows us to *tightly* reduce SAT on arbitrary polynomial size NC circuits to problems in P. Thus, we are able to replace SETH with NC-SETH, and derive far more remarkable consequences from truly subquadratic algorithms for Edit Distance and LCS. Moreover, we show that *any* problem for which the general framework of Bringmann and Künnemann is capable of showing an $n^{2-o(1)}$ SETH lower bound, will suffer from the much stronger NC-SETH hardness barrier. In fact, we are able to show reductions even to problems that fall outside their framework, like LCS on k sequences, a classical problem in parameterized complexity with an $O(n^k)$ algorithm [16, 34, 2].

BPs are a popular non-uniform model of computation [11]. Roughly speaking, a nondeterministic Branching Program on n input variables of width W and length T is a layered directed graph on T layers, each layer having W nodes, such that every edge is labelled with a constraint of the form $(x_i = b)$ where x_i is an input variable, and $b \in \{0,1\}$. Note that typically $T \gg n$ and each node appears on many edges. One of the nodes in the first layer is marked as the *start node*, and one of the nodes in the last layer is marked as the *accept node*. For an input $x \in \{0,1\}^n$ let G_x be the subgraph of edges whose constraints are satisfied by x. We say that the BP accepts the input x iff the accept node is reachable from the start node in G_x. The *size* of a BP is the total number of edges, i.e. $O(W^2T)$. We refer to Section 2 for additional details about branching programs. Even when the width is constant, BPs are surprisingly expressive: Barrington's Theorem states that any fan-in 2, depth d circuit can be converted into an equivalent BP of width 5 and size 4^d, over the same set of inputs [14]. Therefore, any circuit with fan-in 2 of polylog depth of any size (in particular, NC circuits) can be expressed as a BP of length $2^{\mathrm{polylog}\, n}$ and constant width. Our reduction shows that truly subquadratic Edit Distance would imply a $(2-\delta)^n$ algorithm for SAT of constant-width $2^{o(n)}$-length BPs.

THEOREM 2. *There is a reduction from SAT on nondeterministic branching programs on n variables, length T, and width W, to an instance of Edit-Distance or LCS on two binary sequences of length $N = 2^{n/2} \cdot T^{O(\log W)}$, and the reduction runs in $O(N)$ time.*

Besides the constant width case, another interesting setting is where W and T are both $2^{o(\sqrt{n})}$, which corresponds to BPs that can represent *any nondeterministic Turing machine* that uses $o(\sqrt{n})$ space [11]. Thus, truly subquadratic Edit Distance or LCS would allow us to get an exponential improvement over exhaustive search for checking SAT of complex objects that can easily implement cryptographic primitives, and many of our favorite algorithms. This would be much more surprising than a faster SAT algorithm on linear size CNFs (as in SETH). To support this, we point at a few strong circuit lower bounds that would follow from such an algorithm.

If we assume that Edit-Distance or LCS can be solved in truly subquadratic time, then (among other things) Theorem 2 implies $O(2^{n-\varepsilon n/2})$ time algorithms for SAT on arbitrary formulas of size $2^{o(n)}$ and for SAT on nondeterministic branching programs of size $2^{o(\sqrt{n})}$. Combining this with connections between faster SAT algorithms and circuit lower bounds from prior work (see the full version [4] for formal statements and a sketch of the proof), we obtain the following circuit lower bound consequences.

COROLLARY 1. *If Edit Distance or LCS on two binary sequences of length N is in $O(N^{2-\varepsilon})$ time for some $\varepsilon > 0$, then the complexity class E^{NP} does not have:*

1. *non-uniform $2^{o(n)}$-size Boolean formulas,*
2. *non-uniform $o(n)$-depth circuits of bounded fan-in, and*
3. *non-uniform $2^{o(n^{1/2})}$-size nondeterministic branching programs.*

Furthermore, $\mathsf{NTIME}[2^{O(n)}]$ is not in non-uniform NC.

The above lower bound consequences are far stronger than any state of the art. The first consequence is interesting due to the rarity of $2^{\Omega(n)}$ circuit lower bounds: it is still open whether the humongous class $\Sigma_2\mathsf{EXP}$ has $2^{o(n)}$ size *depth-three* circuits. The third consequence is interesting because it yields an exponential lower bound for arbitrary nondeterministic BPs; this model is vastly bigger than $\mathsf{NL}/\mathrm{poly}$. The fourth is interesting because the lower bound holds for the smaller class $\mathsf{NTIME}[2^{O(n)}]$. These consequences are on a different scale compared to the ones obtained from refuting SETH, and therefore the "circuit lower bounds barrier" for faster Edit Distance is much stronger.

Our first corollary was a strict improvement over the previously known SETH lower bounds, in terms of the significance of the consequences. Next, we show that our reduction allows us to derive consequences even from *mildly* subquadratic algorithms, a feature that did not exist in the previous conditional lower bounds in P.

Given the status of Edit-Distance and LCS as core computer science problems, any asymptotic improvement over the longstanding $O(n^2/\log^2 n)$ upper bound is highly desirable. Recent algorithmic techniques were able to beat similar longstanding bounds for other core problems like All Pairs Shortest Path (APSP) [41, 23], 3-SUM [30], and Boolean Matrix Multiplication[13, 22, 43]. For example, the *polynomial method*[41] has allowed for superpolylogarithmic shavings for APSP, and more recently to two other problems that are more closely related to ours, namely Longest Common *Substring* [9], and Hamming Nearest Neighbors [10]. A natural open question [41, 9, 10] is whether these techniques can lead to an $n^2/\log^{\omega(1)} n$ algorithm for Edit-Distance as well. The lower bound of Backurs and Indyk is not sufficient to address this question, and only a much faster $n^2/2^{\omega(\log n/\log\log n)}$ would have been required to improve the current CNF-SAT algorithms. Our approach of considering $\mathcal{C}$-SETH for more expressive classes $\mathcal{C}$ allows us to prove strong "circuit lower bounds barriers" *even for shaving log factors.*

Any formula of size $O(n^f)$ can be transformed into an equivalent BP of width 5 and size $O(n^{8f})$ (first rebalance into a formula of depth $4f\log n$ [37] and then use Barrington's Theorem [14]). Applying Theorem 1 to the resulting BPs, we get LCS instances of size $N = O(2^{n/2} \cdot n^{8fd})$, for some constant $d \leq 25$ (the constant depends on the problem and the alphabet size). Shaving an $\Omega((\log N)^{8fd+f+10})$ factor would translate into an $O(2^n/(n^{10} \cdot n^f))$ algorithm for SAT of formulas of size $O(n^f)$. Thus, we get that if LCS can be solved in $O(n^2/\log^{1000} n)$ time, then SAT on formulas of size $O(n^5)$ can be solved in $O(2^n/n^{15})$ time, which would imply that E^{NP} does not have such formulas. We obtain that solving Edit-Distance or LCS in $n^2/\log^{\omega(1)} n$ time still implies a major circuit lower bound, namely that $\mathsf{NTIME}[2^{O(n)}]$ is not in non-uniform NC^1.

COROLLARY 2. *If Edit Distance or LCS on two binary sequences of length N can be solved in $O(n^2/\log^c n)$ time for every $c > 0$, then $\mathsf{NTIME}[2^{O(n)}]$ does not have non-uniform polynomial-size log-depth circuits.*

It is likely that these connections could be sharpened even further and that similar consequences can be derived even from shaving fewer log factors. Some inefficiencies in these connections are due to constructions of certain gadgets in

our proofs, while others come from the framework for obtaining circuit lower bounds from faster SAT algorithms, and the reductions from circuits to BPs.

One striking interpretation of these corollaries is that when an undergraduate student learns the simple dynamic programming algorithms for Edit-Distance or Longest Common Subsequence and wonders whether there is a faster algorithm, he or she is implicitly trying to resolve very difficult open questions in complexity theory.

Technical remarks.

All known SETH lower bound proofs for problems in P have relied as a first step on a reduction [39] to the following *Orthogonal Vectors* (OV) problem: given a set of n boolean vectors $S \subseteq \{0,1\}^d$ of dimension $d = \omega(\log n)$, does there exist a pair $a, b \in S$ such that for all $j \in [d]$ we have $(a[j] = 0)$ or $(b[j] = 0)$, i.e. the vectors are orthogonal or "disjoint". If OV can be solved in $O(n^{2-\varepsilon})$ time, then CNF-SAT can be solved in $O(2^{(1-\varepsilon/2)n})$ time. It is important to notice that reductions in the direction are not known, i.e. refuting SETH is not known to imply subquadratic algorithms for some hard quadratic-time problems. Therefore, lower bounds under the assumption that OV requires $n^{2-o(1)}$ time are more reliable than SETH lower bounds. However, the above weaknesses of SETH apply to OV as well: a much harder problem is the $\mathcal{C}$-Satisfying-Pair problem, where instead of searching for an orthogonal pair of vectors, we ask for a pair of vectors that (together) satisfy a certain function that can be represented in more complex ways than an orthogonality check. Again, there is a spectrum of increasing expressiveness, and OV is quite low on it. Indeed, we have no idea how to solve the NC-Satisfying-Pair problem in $O(n^2/\log^3 n)$ time (it would readily imply faster NC-SAT algorithms), while for OV the current upper bound $n^{2-1/O(\log(d/\log n))}$ is barely not truly subquadratic. All the reductions in this paper (except for the k-LCS proof, which has been deferred to the full version [4]) are via a certain Branching-Program-Satisfying-Pair problem, which can be solved in quadratic time, while faster algorithms would be very surprising and imply all the aforementioned consequences.

Previous SETH lower bound proofs, when stripped of all the gadgetry, are rather simple, due to the simplicity of the OV problem (which, in turn, is due to the simplicity of CNFs). Each vector is represented by some *vector gadget*, so that two gadgets "align well" if and only if the vectors are *good* (in this case, orthogonal), and then all the gadgets are combined so that the "total score" reflects the existence of a good pair. Vector gadgets that are capable of checking orthogonality can be constructed in natural ways by concatenating *coordinate gadgets* that have straightforward functionality (checking that not both coordinates are 1), which in turn can be constructed via certain atomic sequences of constant size. We observe that these reductions do not exhaust the expressive capabilities of problems like Edit Distance and LCS.

Our new reductions follow this same scheme, except that the functionality of the vector gadgets is no longer so simple. Our main technical contribution is the construction of certain *reachability gadgets*, from which our vector gadgets are constructed. These gadgets are capable of checking reachability between two nodes in a subgraph (e.g. $u_{\mathtt{start}}$ and $u_{\mathtt{acc}}$) of a graph (the branching program) that is obtained from two given vectors. These gadgets exhibit the ability of sequence similarity measures to execute nontrivial algorithmic tasks. Our reduction can be viewed as encoding of graphs into two sequences such that the optimal LCS must implicitly execute the classical small-space algorithm for directed reachability of Savitch's Theorem [36].

Previous work on better hypotheses.

Finding more reliable hypotheses (than SETH, 3-SUM, APSP, etc) that can serve as an alternative basis for the "hardness in P" is an important goal. Previous progress towards this end was achieved by Abboud, Vassilevska Williams, and Yu [8] where the authors prove tight lower bounds for various graph problems under the hypothesis that *at least one* of the SETH, APSP, and 3-SUM conjectures is true. The $\mathcal{C}$-SETH hypothesis (say, for $\mathcal{C} = \mathsf{NC}$) that we consider in this work is incomparable in strength to theirs, yet it has certain advantages. First, the known connections between faster SAT algorithms and circuit lower bounds allow us to point at remarkable consequences of refuting our hypothesis, which is not known for any of the previous conjectures. Second, it allows us to show barriers even for *mildly* subquadratic algorithms. And third, it allows us to explain the barriers for many problems like Edit Distance and LCS for which a lower bound under 3-SUM or APSP is not known (unless the alphabet size is near-linear [7]).

Organization of the paper.

The rest of the paper is organized as follows. In Section 2 we define the SAT problem on Branching Programs (BP-SAT), and briefly describe how it is used as the source of our reductions. In Section 3 we give a direct and simplified reduction from BP-SAT to LCS. We present the framework of Bringmann and Künnemann [18] in Section 4, along with a sketch of our full reduction. We then present the details of the full reduction in Section 5. The full reduction also applies to LCS, and is more efficient than the simplified reduction from Section 3. Due to lack of space, many of the details are deferred to the full version of the paper [4].

2. SATISFIABILITY OF BRANCHING PROGRAMS

In this section we define the SAT problem on Branching Programs (BP-SAT), which we later reduce to various sequence-problems such as Edit Distance and LCS.

A nondeterministic *Branching Program* (BP) of length T and width W on n boolean inputs $x_1, \ldots, x_n$ is a layered directed graph P with T layers $L_1, \ldots, L_T$. The nodes of P have the form (i, j) where $i \in [T]$ is the layer number and $j \in [W]$ is the index of the node inside the layer. The node $u_{\mathtt{start}} = (1, 1)$ is called the starting node of the program, and the node $u_{\mathtt{acc}} = (T, 1)$ is the accepting node of the program. For all layers $i < T$ except the last one: all nodes in L_i are marked with the same variable $x(i) = x_{f(i)}$, and each node has an arbitrary number of outgoing edges, each edge marked with 0 or 1. Note that typically $T \gg n$ and each variable appears in many layers.

An *evaluation* of a branching program P on an input $x_1, \ldots, x_n \in \{0,1\}$ is a path that starts at $u_{\mathtt{start}}$ and then (nondeterministically) follows an edge out of the current node: if the node is in layer L_i we check the value of the corresponding variable $x_{f(i)}$, denote it by $\eta \in \{0,1\}$, and then

we follow one of the outgoing edges marked with η. The BP *accepts* the input iff the evaluation path ends in $u_{\texttt{acc}}$. That is, each input restricts the set of edges that can be taken, and the BP accepts an input iff there is a path from $u_{\texttt{start}}$ to $u_{\texttt{acc}}$ in the subgraph induced by the input.

DEFINITION 1 (BP-SAT). *Given a Branching Program P on n boolean inputs, decide if there is an assignment to the variables that makes P accept.*

To reduce BP-SAT to a sequence-problem we go through the following problem: Let $X_1 = \{x_1, \ldots, x_{n/2}\}$ and $X_2 = \{x_{n/2+1}, \ldots, x_n\}$ be the first and last half of the inputs to the branching program, respectively. Do there exist $a \in \{0,1\}^{n/2}$ and $b \in \{0,1\}^{n/2}$, such that when viewed as partial assignments to X_1 and X_2, respectively, together they form an accepting input to the branching program? This problem is clearly just a reformulation of BP-SAT. Our reductions also work, however, when a and b are restricted to two given sets of vectors $S_1, S_2 \subseteq \{0,1\}^{n/2}$, i.e., we ask whether there exists an accepting pair $(a,b) \in S_1 \times S_2$. We refer to this problem as the *satisfying pair problem* on branching programs (or BP-SAT-Pair). Proving a reduction from this more general problem corresponds to proving a reduction from the *orthogonal vectors problem (OV)* to get a SETH-based lower bound (see [39]). To simplify the presentation we assume, however, that $S_1 = S_2 = \{0,1\}^{n/2}$.

Our reductions construct for each partial assignment $a \in S_1$ and $b \in S_2$, respectively, a sequence whose length depends on the branching program. When the branching program is not too large these sequences have length at most $2^{o(n)}$. The sequences from S_1 are combined into one long sequence of length $N = 2^{(1/2+o(1))n}$, and the same is done for S_2. We then show that solving, e.g., the longest common subsequence problem on these two strings of length N decides whether the original branching program is satisfiable, and gives us a satisfying assignment if possible. This establishes a connection between BP-SAT, which is solvable in exponential time, and sequence-problems that are solvable in quadratic time.

3. A SIMPLIFIED REDUCTION TO LONGEST COMMON SUBSEQUENCE

Given two strings of N symbols over some alphabet Σ, the longest common subsequence (LCS) problem asks for the length of the longest sequence that appears as a subsequence in both strings. For two strings a, b, we let $LCS(a,b)$ denote the length of the longest common subsequence of a and b. In this section we present a reduction from BP-SAT to LCS, proving the following theorem. To simplify the presentation we give a less efficient reduction that uses $|\Sigma| = O(W \log T)$ symbols for branching programs of width W and length T. We refer to sections 4 and 5 for a more efficient reduction with $|\Sigma| = 2$, that is based on the framework of Bringmann and Künnemann [18].

THEOREM 3. *There is a constant c such that if LCS can be solved in time $S(N)$, then BP-SAT on n variables and programs of length T and width W can be solved in time $S(2^{n/2} \cdot T^{c \log W})$.*

Abboud *et al.* [2] gave a similar reduction from CNF-SAT to LCS, or more precisely from orthogonal vectors (OV) to LCS. As a first step they reduced OV to the following problem of finding a pair of strings with a long common subsequence, which we refer to as the LCS-Pair problem.

DEFINITION 2 (LCS-PAIR). *Let Y be a given integer, and let $S_1, S_2 \subseteq \Sigma^L$ be two given sets of strings of length L such that $LCS(a,b) \leq Y$ for every $a \in S_1$ and $b \in S_2$. The LCS-Pair problem asks if there is a pair $(a,b) \in S_1 \times S_2$ with $LCS(a,b) = Y$?*

Abboud *et al.* then reduced the LCS-Pair problem to LCS by proving the following lemma. The proof of the lemma uses *normalized vector gadgets* similar to those used in the reduction by Backurs and Indyk [12] from OV to Edit Distance. We sketch the proof in Section 4 in the context of Bringmann and Künnemann's framework [18], and give a formal proof of a corresponding lemma in Section 5.

LEMMA 1 ([2]). *Let (S_1, S_2, Y) be an LCS-Pair problem with $|S_1| = |S_2| = M$. Then one can construct two strings A, B of length $M \cdot poly(L)$, where L is the length of the strings from S_1 and S_2, such that for an integer E,*

- *$LCS(A,B) = E$ if there exists a pair $(a,b) \in S_1 \times S_2$ with $LCS(a,b) = Y$, and*
- *$LCS(A,B) \leq E - 1$ otherwise.*

To reduce BP-SAT to LCS we first apply the trivial reduction from Section 2 from BP-SAT to the satisfying pair problem on branching programs (BP-SAT-Pair). We then reduce BP-SAT-Pair to LCS-Pair, and use the reduction from Lemma 1 to complete the proof of Theorem 3.

It should be noted that the main challenge for Abboud *et al.* [2] was to prove Lemma 1, whereas the reduction from OV to LCS-Pair was nearly trivial. Moreover, this is a common phenomenon for SETH-based lower bounds. In our case the main challenge was, however, to reduce BP-SAT-Pair to LCS-Pair. This indicates that previous SETH-based lower bounds, unlike our new reduction, do not fully exploit the strength of the problems in question.

We next describe our reduction from BP-SAT-Pair to LCS-Pair. Let P be a given branching program on n boolean inputs, and let F be the corresponding function. Furthermore, let $X_1 = \{x_1, \ldots, x_{n/2}\}$ and $X_2 = \{x_{n/2+1}, \ldots, x_n\}$ be the first and last half of the inputs to F, respectively. For two partial assignments a and b in $\{0,1\}^{n/2}$, we use the notation $a \odot b$ to denote concatenation, forming a complete assignment. We must decide whether there exist $a, b \in \{0,1\}^{n/2}$ such that $F(a \odot b) = 1$.

The reduction maps each partial assignment $a \in \{0,1\}^{n/2}$ for the first half of the input to a string $G(a)$, and each partial assignment $b \in \{0,1\}^{n/2}$ for the second half of the input to another string $\overline{G}(b)$. We refer to G and $\overline{G}$ as sequence gadgets. The strings $G(a)$ and $\overline{G}(b)$ are constructed such that

- $LCS(G(a), \overline{G}(b)) = Y$ if $F(a \odot b) = 1$, and
- $LCS(G(a), \overline{G}(b)) \leq Y - 1$ otherwise,

where Y is a known integer that depends on the width W and length T of the branching program, but not on a and b. Solving LCS for $G(a)$ and $\overline{G}(b)$ can therefore be viewed as

evaluating $F(a \odot b)$. To complete the reduction we simply construct the two sets:

$$S_1 = \{G(a) \mid a \in \{0,1\}^{n/2}\}$$
$$S_2 = \{\overline{G}(b) \mid b \in \{0,1\}^{n/2}\}$$

It follows that the branching program has a satisfying assignment if and only if the LCS-Pair problem (S_1, S_2, Y) has a pair of strings $(a, b) \in S_1 \times S_2$ with $LCS(a, b) = Y$.

Armed with Lemma 1, we see that in order to prove Theorem 3, it suffices to create sequence gadgets G and $\overline{G}$ of length $T^{O(\log W)}$ such that for some Y, $LCS(G(a), \overline{G}(b)) = Y$ if on input $a \odot b$, the starting state of the branching program reaches the accepting state, and $LCS(G(a), \overline{G}(b)) \leq Y - 1$ otherwise. For the remainder of the section we therefore let $a, b \in \{0,1\}^{n/2}$ be fixed. Note, however, that we must construct $G(a)$ and $\overline{G}(b)$ independently.

To construct $G(a)$ and $\overline{G}(b)$ we follow an inductive approach, mimicking Savitch's theorem [36]. Since a and b are fixed, $G(a)$ and $\overline{G}(b)$ represent the corresponding subsets of edges of the given branching program P, and the goal is to decide if there is a directed path from $u_{\texttt{start}}$ to $u_{\texttt{acc}}$ in the resulting graph. Such a path must go through some node in the middle layer, and if we can guess which node then we can split P into two branching programs of half the length and evaluate each part recursively. We use *reachability gadgets* to implement this decomposition, and an LCS algorithm must then make the correct guess to find the longest common subsequence.

Before describing our reachability gadgets we first introduce the use of weighted symbols.

Weighted LCS.

To simplify the proof we will work with the following generalized version of LCS in which each letter in the alphabet can have a different weight. For two sequences P_1 and P_2 of length N over an alphabet Σ and a weight function $w : \Sigma \to [K]$, let X be the sequence that appears in both P_1, P_2 as a subsequence and maximizes the expression $w(X) = \sum_{i=1}^{|X|} w(X[i])$. We say that X is the *Weighted Longest Common Subsequence* (WLCS) of P_1, P_2 and write $\text{WLCS}(P_1, P_2) = w(X)$. The WLCS problem asks us to output $\text{WLCS}(P_1, P_2)$.

Note that a common subsequence X of two sequences P_1, P_2 can be thought of as an alignment or a matching $A = \{(a_i, b_i)\}_{i=1}^{|X|}$, where $a_i, b_i \in \mathbb{N}$ are indices, between the two sequences, so that for all $i \in [|X|] : P_1[a_i] = P_2[b_i]$, and $a_1 < \cdots < a_{|X|}$ and $b_1 < \cdots < b_{|X|}$. Clearly, the weight $\sum_{i=1}^{|X|} w(P_1[a_i]) = \sum_{i=1}^{|X|} w(P_2[b_i])$ of the matching A corresponds to the weighted length $w(X)$ of the common subsequence X.

Abboud *et al.* [2] showed a simple reduction from WLCS on length N sequences over an alphabet Σ with largest weight K to LCS on (unweighted) sequences of length $N \cdot K$ over the same alphabet. The reduction simply copies each symbol $\ell \in \Sigma$ in each of the sequences $w(\ell)$ times and then treats the sequences as unweighted. Abboud *et al.* showed that the optimal solution is preserved under this reduction.

For a sequence over a weighted alphabet Σ we define the *total length* of the sequence to be the sum of the weights of all symbols in this sequence. Note that this is the real length of the unweighted sequence that we obtain after applying the reduction from WLCS to LCS.

Reachability gadgets.

Let P be the given branching program of length T and width W, and assume for simplicity that $T = 2^t + 1$ for some $t \geq 0$. Recall that the input $a \odot b$ is fixed, and that our goal is to decide whether there is a directed path from $u_{\texttt{start}}$ to $u_{\texttt{acc}}$, which we do recursively by finding paths to and from the middle layer. Let therefore $u \in L_i$ and $v \in L_j$ be two nodes in layers that are at distance $j - i = 2^k$ from each other. Furthermore, let Y_k be some integer that will be specified later as a function of W and k. We refer to Y_k as a threshold.

The main component in our reduction are recursive constructions of two kinds of reachability gadgets $\mathsf{RG}_k^{u \to v}(a)$ and $\overline{\mathsf{RG}}_k^{u \to v}(b)$, with the following very useful property.

- $LCS(\mathsf{RG}_k^{u \to v}(a), \overline{\mathsf{RG}}_k^{u \to v}(b)) = Y_k$ if on input $a \odot b$, one can reach v from u in 2^k steps, and
- $LCS(\mathsf{RG}_k^{u \to v}(a), \overline{\mathsf{RG}}_k^{u \to v}(b)) \leq Y_k - 1$ otherwise.

Since $F(a \odot b) = 1$ if and only if $u_{\texttt{acc}}$ is reachable from $u_{\texttt{start}}$, we define our sequence gadgets as:

$$G(a) = \mathsf{RG}_t^{u_{\texttt{start}} \to u_{\texttt{acc}}}(a)$$
$$\overline{G}(b) = \overline{\mathsf{RG}}_t^{u_{\texttt{start}} \to u_{\texttt{acc}}}(b)$$

To complete the proof we show that the total length Z_k of $\mathsf{RG}_k^{u \to v}(a)$ and $\overline{\mathsf{RG}}_k^{u \to v}(b)$, respectively, is upper bounded by W^{ck} for some constant c. $G(a)$ and $\overline{G}(b)$ therefore have length $T^{O(\log W)}$, and when we apply Lemma 1 we obtain two strings of length $2^{n/2} \cdot T^{O(\log W)}$ as desired.

We next show how to inductively construct such gadgets and upper bound their lengths.

Base Case: $k = 0$.

Let $u \in L_i$ and $v \in L_{i+1}$ be two nodes in adjacent layers. Then u can reach v if and only if the edge (u, v) is present, which requires (u, v) to be part of the branching program, and the input variable $x(i) = x_{f(i)}$ for layer i to match the boolean value associated with (u, v).

Recall that a is an assignment to the variables $X_1 = \{x_1, \ldots, x_{n/2}\}$, and that b is an assignment to the variables $X_2 = \{x_{n/2+1}, \ldots, x_n\}$. Let e, \$1, and \$2 be letters in the alphabet Σ with weight $w(e) = w(\$1) = w(\$2) = 1$. For the base case we define:

$$RG_0^{u \to v}(a) = \begin{cases} e & \text{if } (u, v) \text{ is present or } x(i) \in X_2 \\ \$1 & \text{otherwise} \end{cases}$$

$$\overline{RG_0}^{u \to v}(b) = \begin{cases} e & \text{if } (u, v) \text{ is present or } x(i) \in X_1 \\ \$2 & \text{otherwise} \end{cases}$$

Note that $WLCS(RG_0^{u \to v}(a), \overline{RG_0}^{u \to v}(b))$ is 1 if (u, v) is present and 0 otherwise. Indeed, if (u, v) is present then both strings consist of the single letter e, and otherwise one of the strings will be \$1 or \$2. The gadgets therefore have threshold $Y_0 = 1$ and length $Z_0 = 1$. (The letters \$1 and \$2 will in fact only appear in their respective sequences and can therefore alternatively be viewed as empty strings.)

Inductive step: $k > 0$.

Let now $u \in L_i$ and $v \in L_j$ be two nodes at distance $j - i = 2^k$ from each other, and let $L_h = \{y_1, y_2, \ldots, y_W\}$ be the nodes in layer $h = \frac{i+j}{2}$, halfway between i and j. Any

path from u to v must pass through some $y \in L_h$. Let $u \rightsquigarrow v$ denote that v is reachable from u. We therefore have:

$$u \rightsquigarrow v \quad \Longleftrightarrow \quad OR \begin{cases} u \rightsquigarrow y_1 & AND \quad y_1 \rightsquigarrow v \\ u \rightsquigarrow y_2 & AND \quad y_2 \rightsquigarrow v \\ \quad \dots & \\ u \rightsquigarrow y_W & AND \quad y_W \rightsquigarrow v \end{cases}$$

The reachability gadgets will be similarly composed of AND and OR gadgets.

AND gadget.

By recursion, each reachability expression, e.g., $u \rightsquigarrow y_1$ is mapped to a pair of strings $(RG_{k-1}^{u \to y_1}(a), \overline{RG}_{k-1}^{u \to y_1}(b))$, such that

- $WLCS(RG_{k-1}^{u \to y_1}(a), \overline{RG}_{k-1}^{u \to y_1}(b)) = Y_{k-1}$ if y_1 is reachable from u, and
- $WLCS(RG_{k-1}^{u \to y_1}(a), \overline{RG}_{k-1}^{u \to y_1}(b)) < Y_{k-1}$ otherwise.

The purpose of the AND gadget is to combine two pairs of strings corresponding to $u \rightsquigarrow y_1$ and $y_1 \rightsquigarrow v$ into a single pair that corresponds to $u \rightsquigarrow y_1 \rightsquigarrow v$, i.e., the new pair should share a long subsequence iff there is a path from u to v through y_1. As the following lemma shows, this is obtained with a simple concatenation operation and the use of a new, heavy letter.

LEMMA 2. *Let (a_1, b_1) and (a_2, b_2) be two pairs of strings where each string has weight Z. Let & be a letter with weight $w(\&) = 2Z$ that does not appear in a_1, b_1, a_2, b_2. Assume that $s_1 := WLCS(a_1, b_1) \leq Y$ and $s_2 := WLCS(a_2, b_2) \leq Y$. Then:*

$$\begin{aligned} WLCS(a_1\&a_2, b_1\&b_2) &= 2Y + 2Z \quad \text{if } s_1 = s_2 = Y \\ WLCS(a_1\&a_2, b_1\&b_2) &< 2Y + 2Z \quad \text{otherwise} \end{aligned}$$

PROOF. The letter & must be matched in a longest common subsequence of $a_1\&a_2$ and $b_1\&b_2$ since it makes up half the weight of the strings. The lemma then easily follows from the fact that $WLCS(a_1\&a_2, b_1\&b_2) = s_1 + w(\&) + s_2$. □

OR gadget.

The OR gadget is significantly more involved. In this case we wish to combine W pairs of strings corresponding to $u \rightsquigarrow y_1 \rightsquigarrow v, u \rightsquigarrow y_2 \rightsquigarrow v, \dots, u \rightsquigarrow y_W \rightsquigarrow v$ into a single pair of strings corresponding to $u \rightsquigarrow v$. The new pair of strings should share a long subsequence iff one of the original W pairs shared a long subsequence.

Let therefore $(a_1, b_1), (a_2, b_2), \dots, (a_W, b_W)$ be W pairs of strings where each string has weight Z, and assume that $s_\ell := WLCS(a_\ell, b_\ell) \leq Y$ for all $\ell \in [W] = \{1, \dots, W\}$. Furthermore, let % and # be two new letters with weight $w(\%) = w(\#) = 2Z$, and let c_ℓ, for $\ell \in [W]$, be ℓ new letters with weight $w(c_\ell) = Z$. We then define two new strings as follows, where $\%^W$ means that % is copied W times:

$$\begin{aligned} OR(a_1, \dots, a_W) &= \%^W \left(\bigcup_{\ell \in [W]} a_\ell c_\ell \# \right) \%^W \\ \overline{OR}(b_1, \dots, b_W) &= \bigcup_{\ell \in [W]} \#^W b_\ell c_\ell \#^W \% \end{aligned}$$

The following is an example of $A = OR(a_1, a_2, a_3)$ and $B = \overline{OR}(b_1, b_2, b_3)$. Note that A is shorter than B. To make the length of A match the length of B we add an appropriate number of $\$_1$ letters (from the base case) to A.

$$A = \quad \%\%\%a_1c_1\#a_2c_2\#a_3c_3\#\%\%\%$$

$$B = \quad \#\#\#b_1c_1\#\#\#\%\#\#\#b_2c_2\#\#\#\%\#\#\#b_3c_3\#\#\#\%$$

LEMMA 3. *Let the two strings $A = OR(a_1, \dots, a_W)$ and $B = \overline{OR}(b_1, \dots, b_W)$ be defined as above. Then:*

$$\begin{aligned} WLCS(A, B) &= 4WZ + Z + Y \quad \text{if } s_\ell = Y \text{ for some } \ell \\ WLCS(A, B) &< 4WZ + Z + Y \quad \text{otherwise} \end{aligned}$$

PROOF. Observe first that A contains W of the letters #, and that B contains W of the letters %. It is always possible to match these letters, which gives a total weight of $4WZ$. The remaining # and % letters can never be matched. We start by showing that this many # and % letters must be matched in some longest common subsequence.

CLAIM 1. *In some optimal WLCS, all % letters in B are matched to % letters in A.*

Suppose for the sake of contradiction that there is no longest common subsequence where all % in B are matched. Each % can be matched either to the left or the right part of A, and these matchings cannot cross. In particular there is some ℓ such that % letters to the left of b_ℓ are only matched to the left part of A, and % letters to the right of b_ℓ are only matched to the right part of A. Suppose some % to the left of b_ℓ is unmatched. (The argument is analogous when the unmatched % is to the right of b_ℓ.) Consider in particular the first unmatched % in B. Matching this letter to the left part of A adds a weight of $2Z$ to the matching, but removes all matched letters until the previous % letter in B. Any lost # letters can however be recovered by being matched to the W copies of # that appear immediately after the newly matched % in B. Hence, the total loss is at most $2Z$ and comes from the a string and the c letter. This process can be repeated until all % letters in B are matched, which gives a contradiction.

CLAIM 2. *In some optimal WLCS, all # letters in A are matched to # letters in B.*

The proof of Claim 2 is essentially the same as the proof of Claim 1. Since we may assume that all % letters in B are matched, the only remaining part of B that can be matched to A is some substring of the form $B' = \#^W b_\ell c_\ell \#^W$. Here the # letters again only appear to the left and to the right, whereas in A they separate the string into substrings of weight $2Z$. We therefore again observe that there is some index m such that # letters to the left of a_m are only matched to the left part of B', and # letters to the right of a_m are only matched to the right part of B'. The rest of the argument is the same as for Claim 1.

To complete the proof of Lemma 3 we observe that we may assume that the indices ℓ and m from the proofs of Claim 1 and Claim 2 are the same. Indeed, when the maximum number of % and # letters are matched, the only remaining contribution to the WLCS must come from some pair of substrings $a_m c_m$ and $b_\ell c_\ell$. Moreover, the c letters can only be matched when $\ell = m$, and they make up half of the

weight of the substrings. We may therefore assume that $\ell = m$, which means that $WLCS(A, B) = 4WZ + w(c_\ell) + WLCS(a_\ell, b_\ell)$. Since $w(c_\ell) = Z$ and the index ℓ can be chosen freely from $[W]$, this completes the proof. □

Combining AND and OR gadgets.

To complete the inductive step it remains to combine the AND and OR gadgets and thereby define the reachability gadgets. We do so now:

$$\begin{aligned} RG_k^{u\to v}(a) \;=\; OR(&RG_{k-1}^{u\to y_1}(a)\&RG_{k-1}^{y_1\to v}(a),\\ &\ldots,\\ &RG_{k-1}^{u\to y_W}(a)\&RG_{k-1}^{y_W\to v}(a)) \end{aligned}$$

$$\begin{aligned} \overline{RG}_k^{u\to v}(b) \;=\; \overline{OR}(&\overline{RG}_{k-1}^{u\to y_1}(b)\&\overline{RG}_{k-1}^{y_1\to v}(b),\\ &\ldots,\\ &\overline{RG}_{k-1}^{u\to y_W}(b)\&\overline{RG}_{k-1}^{y_W\to v}(b)) \end{aligned}$$

Recall that Z_{k-1} is the length of, e.g., $RG_{k-1}^{u\to y_1}(a)$ and $\overline{RG}_{k-1}^{u\to y_1}(b)$. Applying the AND gadget increases the length by a factor of 4, and applying the OR gadget increases the length by a factor of $4W(W+1)$. It follows that the length of $RG_k^{u\to v}(a)$ and $\overline{RG}_k^{u\to v}(b)$ is $Z_k = 16W(W+1)Z_{k-1}$. Since $Z_0 = 1$ we therefore get that $Z_k = (16W(W+1))^k$, and that $Z_t = (16W(W+1))^t = T^{O(\log W)}$ as desired.

Recall also that Y_{k-1} is the threshold for, e.g., $RG_{k-1}^{u\to y_1}(a)$ and $\overline{RG}_{k-1}^{u\to y_1}(b)$. Applying the AND gadget increases the threshold to $Y' = 2Y_{k-1} + 2Z_{k-1}$, and applying the OR gadget further increases the threshold to $Y_k = 16WZ_{k-1} + 4Z_{k-1} + Y' = 16WZ_{k-1} + 6Z_{k-1} + 2Y_{k-1}$. (Note that the constants are larger than those appearing in Lemma 3 because applying the AND gadget multiplied the length by 4.) We therefore see that Y_k is expressed only as a function of W and k as desired.

The correctness of the inductive step follows from Lemma 2 and Lemma 3. Indeed, the two strings generated by the OR gadget in Lemma 3 have a common subsequence with length that matches the threshold Y_k iff one of the pairs of strings generated by AND gadgets in Lemma 2 can match the threshold $Y' = 2Y_{k-1} + 2Z_{k-1}$, which by the induction hypothesis happens iff there is a directed path from u to v.

Finally, we note that each level of the recursion adds another $W + 3$ new letters to the alphabet. (1 for the AND gadget and $W + 2$ for the OR gadget.) The total number of letters used is thus $O(W \log T)$.

4. SEQUENCE PROBLEMS WITH ALIGNMENT GADGETS

Bringmann and Künnemann [18] introduced a framework for showing SETH lower bounds for problems that compute a similarity measure of two sequences of, e.g., bits, symbols, or points. They showed that any sequence-problem that implements so-called *alignment gadgets* cannot be solved in truly subquadratic time under SETH. We show that alignment gadgets are actually much stronger. So strong, in fact, that they can simulate nondeterministic branching programs. It follows that Edit-Distance, Longest Common Subsequence (LCS), Dynamic Time Warping Distance, and every other problem that implements alignment gadgets can solve SAT on nondeterministic branching programs. This proves our main result, Theorem 2. We start here with a sketch of the proof, and then provide the details in Section 5.

A similarity measure δ is a function that measures the similarity, e.g., Edit-Distance or LCS, of two given sequences A and B. Suppose $|A| \geq |B|$. A *structured alignment* maps B to a consecutive subsequence A' of A, and the cost of the alignment is $\delta(A', B)$. Note that the alignment does not correspond to, e.g., a common subsequence. Instead the alignment restricts the similarity measure to only work with A' and B. An alignment gadget is a gadget that takes two collections of sequences and combines each collection into a single sequence such that the minimum cost of a structured alignment closely approximates (within an additive constant) the similarity measure of the two constructed sequences. An alignment gadget can thus be interpreted as a way of forcing the similarity measure to respect the structure of the two collections of sequences. Alignment gadgets are then combined recursively in order to obtain a reduction. We next sketch how this is done for branching programs.

Let P be a given branching program on n boolean inputs, and let F be the corresponding function. To prove a reduction from BP-SAT to a sequence-problem with alignment gadgets we again go through the satisfying pair problem on branching programs (see Section 2). Let therefore $X_1 = \{x_1, \ldots, x_{n/2}\}$ and $X_2 = \{x_{n/2+1}, \ldots, x_n\}$ be the first and last half of the inputs to the branching program, respectively. We must decide whether there exist $a, b \in \{0,1\}^{n/2}$ such that $F(a \odot b) = 1$, where $a \odot b$ is the concatenation of a and b.

Our reduction uses alignment gadgets to construct for each $a \in \{0,1\}^{n/2}$ a sequence $G(a)$, and for each $b \in \{0,1\}^{n/2}$ another sequence $\overline{G}(b)$. These sequences are constructed such that their similarity measure is $\delta(G(a), \overline{G}(b)) = Y$, for some integer Y, if $F(a \odot b) = 1$; and such that $\delta(G(a), \overline{G}(b)) \geq Y + \rho$ otherwise, where $\rho > 0$ is the same for all a and b. In previous reductions from OV the construction of $G(a)$ and $\overline{G}(b)$ was nearly trivial, but as in Section 3 it is now the main challenge when proving our reduction.

Once we have constructed $G(a)$ and $\overline{G}(b)$ for all $a, b \in \{0,1\}^{n/2}$, we combine them into two sequences that will be the output of the reduction. This step is almost identical to a corresponding step in the reduction by Backurs and Indyk [12] from orthogonal vectors to Edit-Distance, and later by Abboud *et al.* [2] to LCS, and by Bringmann and Künnemann [18] to sequence-problems with alignment gadgets. If there exist $a_i, b_j \in \{0,1\}^{n/2}$ with $F(a_i \odot b_j) = 1$, then the goal is to make the structured alignment match $G(a_i)$ and $\overline{G}(b_j)$. It is therefore tempting to apply the alignment gadget to $A = (G(a_1), \ldots, G(a_N), G(a_1), \ldots, G(a_N))$ and $B = (\overline{G}(b_1), \ldots, \overline{G}(b_N))$, where $N = 2^{n/2}$, since we can then freely map B to a consecutive subsequence of A such that $\overline{G}(b_j)$ maps to $G(a_i)$. The contribution to the similarity measure from this pair would then be Y, but unfortunately we have no control over the rest of the sequence. To finish the proof we therefore need one more idea: the *normalized vector gadget*. We put a dummy sequence next to every subsequence in A, such that sequences in B always have the option of mapping to dummy sequences, thereby contributing $Y + \rho$ to the similarity measure. (The alignment gadgets framework allows gadgets of different *types* that are implicitly padded to some specified length. This is used to handle the technicality that, e.g., the length of subsequences in B no longer correctly match those in A.) We

finally get that if F evaluates to 1 for a pair of inputs then $\delta(A,B) \leq N(Y+\rho)-\rho$, and otherwise $\delta(A,B) = N(Y+\rho)$. This completes the reduction.

How to simulate branching programs.

The construction of $G(a)$ and $\overline{G}(b)$ is essentially the same as the one given in Section 3. We again mimic the proof of Savitch's theorem by implementing the reachability gadgets recursively with alignment gadgets. At the k-th level of the recursion we are given two nodes $u \in L_i$ and $v \in L_j$ with $j - i = 2^k$, and we want to decide if there is a directed path from u to v. We denote the sequence constructed in this case for a by $\mathsf{RG}_X^{k,u\to v}(a)$ and for b by $\mathsf{RG}_Y^{k,u\to v}(b)$. In particular $G(a) = \mathsf{RG}_X^{t,u_{\text{start}}\to u_{\text{acc}}}(a)$ and $\overline{G}(b) = \mathsf{RG}_Y^{t,u_{\text{start}}\to u_{\text{acc}}}(b)$, where we assume that the branching program has length $T = 2^t + 1$. (Note that the notation differs slightly from the notation used in Section 3. The parameters X and Y are used to determine the types of the sequences, and thereby their implicit length, but we ignore that aspect here.)

The base case, $k = 0$, is analogous to the base case in Section 3, such that we produce sequences consisting of single symbols. Such unit sequences are called *coordinate values* in the framework of alignment gadgets, and it requires a proof to show that the problem in question supports them. Bringmann and Künnemann [18] provided the relevant proofs for our purposes.

For $k > 0$, we use an alignment gadget to pick the node that the path from u to v passes through. This is again done as in Section 3. Recall that in Section 3 we inserted special symbols of large weight between the recursively defined sequences, such that these symbols had to be matched correctly. We now introduce *index gadgets* that serve a similar function. We also use an *OR gadget* that is composed of two alignment gadgets to put things together. In the end we prove the following technical theorem, which implies Theorem 2. The conditions for the theorem were proved by Bringmann and Künnemann [18] for, e.g., Edit-Distance, LCS, and Dynamic Time Warping.

THEOREM 4. *Let δ be a similarity measure over sequences in $\mathcal{I}$ admitting an alignment gadget of size $f(n) = cn$ and coordinate values, and consider the problem of computing $\delta(x,y)$ for sequences $x, y \in \mathcal{I}$ of length N. SAT of nondeterministic Branching Programs over n variables of width W and length T can be reduced to an instance of this problem on sequences where $N = O(n \cdot T^{\log_2(12W^2c^3)} \cdot c^2 \cdot \log_2 W) = T^{O(\log W)} \cdot n$.*

4.1 Definitions

We start by defining the building blocks from which we will implement our reduction. We will prove a generic reduction from BP-SAT to a generic problem of computing a similarity measure δ of two sequences, for any δ that has a certain property. This property will be the ability to implement *alignment gadgets* which we define below. The advantage of this generic proof over a direct proof is that we can reduce the amount of case-analysis that is required in the proofs: many steps in the reduction will require similar functionalities, and this framework allows us to only prove this functionality once. We will borrow the notation of [18] and reintroduce their *alignment gadgets*.

Let $\delta : \mathcal{I} \times \mathcal{I} \to \mathbb{N}_0$ be a similarity measure for a pair of inputs from $\mathcal{I}$. As a running example it is convenient to think of $\mathcal{I}$ as being all binary sequences of length n and let δ be the Edit Distance between two sequences. For a sequence $x \in \mathcal{I}$ we define its *type* to be a tuple encoding its length and the sum of its entries, i.e. $\text{type}(x) := (|x|, \sum_i x[i])$. For example, if x is a binary sequence, its type encodes its length $|x|$ and the number of ones in x. For a type $t \in \mathbb{N} \times \mathbb{N}_0$ we let $\mathcal{I}_t := \{x \in \mathcal{I} \mid \text{type}(x) = t\}$ be the set of all inputs of type t. We remark that the exact definition of *type* will not be crucial to the framework and for different measures it might be convenient to use a different definition.

Alignments.

Let $0 \leq k \leq m \leq n$. A set $A = \{(i_1, j_1), \ldots, (i_k, j_k)\}$ is called a (partial) alignment if $1 \leq i_1 < \ldots < i_k \leq n$ and $1 \leq j_1 < \ldots < j_k \leq m$. We say that a pair $(i,j) \in A$ are *aligned*. Let $\mathcal{A}_{n,m}$ be the set of all partial alignments with respect to n, m. A partial alignment of the form $\{(\Delta+1, 1), \ldots, (\Delta+m, m)\}$ for some $0 \leq \Delta \leq n-m$ will be called a *structured alignment*, and we will denote the set of all such alignments by $\mathcal{S}_{n,m} \subseteq \mathcal{A}_{n,m}$.

Consider any sequences $x_1, \ldots, x_n \in \mathcal{I}$ and $y_1, \ldots, y_m \in \mathcal{I}$. Let $Q = \max_{i,j} \delta(x_i, y_j)$ be the maximum distance between a pair of sequences under our measure δ. We define the *cost* of an alignment $A \in \mathcal{A}_{n,m}$ (with respect to our sequences) by

$$\mathtt{cost}(A) := \sum_{(i,j)\in A} \delta(x_i, y_j) + (m - |A|) \cdot Q$$

so that any $j \in [m]$ that is not aligned in A will incur the maximal cost Q.

Alignment Gadget.

Intuitively, an alignment gadget combines two collections of sequences $x_1, \ldots, x_n \in \mathcal{I}$ and $y_1, \ldots, y_m \in \mathcal{I}$ into two single sequences $X = \mathsf{GA}(x_1, \ldots, x_n)$ and $Y = \mathsf{GA}(y_1, \ldots, y_m)$, while preserving that $\delta(X,Y)$ is essentially equal to $\delta(A)$, where A is the optimal *structured* alignment $A \in \mathcal{S}_{n,m}$ with respect to our sequences. We will be interested in showing and deriving consequences of the existence of efficient implementations of such gadgets for a similarity measure δ. The formal definition introduces additional technicalities which simplify the proofs of existence of these gadgets considerably.

DEFINITION 3 (ALIGNMENT GADGETS). *The similarity measure δ admits an* alignment gadget *of size $f(n)$, if the following conditions hold: Given instances $x_1, \ldots, x_n \in \mathcal{I}_{\tau_X}$ and $y_1, \ldots, y_m \in \mathcal{I}_{\tau_Y}$ with $m \leq n$ and types $\tau_X = (\ell_X, s_X)$ and $\tau_Y = (\ell_Y, s_Y)$, we can construct new instances $x = \mathsf{GA}_X^{m,\tau_Y}(x_1, \ldots, x_n)$ and $y = \mathsf{GA}_Y^{n,\tau_X}(y_1, \ldots, y_m)$ and $C \in \mathbb{Z}$ such that:*

- $\min_{A\in\mathcal{A}_{n,m}} \mathtt{cost}(A) \leq \delta(x,y) - C \leq \min_{A\in\mathcal{S}_{n,m}} \mathtt{cost}(A)$.
- *type(x) and type(y) only depend on n, m, τ_X, and τ_Y.*
- $|x|, |y| \leq f(n) \cdot (\ell_X + \ell_Y)$.
- *This construction runs in $O(n(\ell_X + \ell_Y))$ time.*

The second ingredient that is required in order to implement a reduction from SAT problems to computing similarity measures are *coordinate values* which are atomic sequences of constant length that we will combine in various ways with the alignment gadgets.

DEFINITION 4 (COORDINATE VALUES). *We say that the similarity measure δ admits* coordinate values *if there are instances $\mathbf{1}_X, \mathbf{0}_X, \mathbf{1}_Y, \mathbf{0}_Y \in \mathcal{I}$ such that:*

$$\delta(\mathbf{1}_X, \mathbf{1}_Y) > \delta(\mathbf{0}_X, \mathbf{1}_Y) = \delta(\mathbf{1}_X, \mathbf{0}_Y) = \delta(\mathbf{0}_X, \mathbf{0}_Y)$$

and type$(\mathbf{1}_X)$ = type$(\mathbf{0}_X)$ and type$(\mathbf{1}_Y)$ = type$(\mathbf{0}_Y)$.

4.2 OR Gadgets

In our reduction from BP SAT, it will be convenient to work with an OR gadget, besides an alignment gadget. Next, we define this gadget and then prove that any similarity measure δ that can implement alignment gadgets will also be able to implement OR gadgets (with some loss of efficiency).

DEFINITION 5 (OR GADGETS). *The similarity measure δ admits* OR gadgets *of size $f(n)$, if the following conditions hold: Given instances $x_1, \ldots, x_n \in \mathcal{I}_{\tau_X}, y_1, \ldots, y_n \in \mathcal{I}_{\tau_Y}$ and types $\tau_X = (\ell_X, s_X), \tau_Y = (\ell_Y, s_Y)$, we can construct new instances*

$$x = \mathsf{OR}_X^{\tau_Y}(x_1, \ldots, x_n)$$

$$y = \mathsf{OR}_Y^{\tau_X}(y_1, \ldots, y_n)$$

and $C \in \mathbb{Z}$ such that:

- $\delta(x, y) = C + \min_{i,j \in [n]} \delta(x_i, y_j)$
- *type(x) and type(y) only depend on n, τ_X, and τ_Y.*
- $|x|, |y| \leq f(n) \cdot (\ell_X + \ell_Y)$.
- *This construction runs in $O(n(\ell_X + \ell_Y))$ time.*

By combining two alignment gadgets in a careful way, we obtain an OR gadget. Note that there is a quadratic blow up in the size of the gadgets in the following lemma. For this reason, we will only use OR gadgets when working with a small number of sequences.

LEMMA 4. *Any similarity measure δ that admits alignment gadgets of size $f(n) = c \cdot n$, also admits OR gadgets of size $f'(n) = c^2 n^2$.*

PROOF. Given instances $x_1, \ldots, x_n \in \mathcal{I}_{\tau_X}$ and $y_1, \ldots, y_n \in \mathcal{I}_{\tau_Y}$, and types $\tau_X = (\ell_X, s_X)$ and $\tau_Y = (\ell_Y, s_Y)$, we will construct OR gadgets as follows. First, construct $x' := \mathsf{GA}_X^{\tau_Y,1}(x_1, \ldots, x_n)$, and construct $y'_j := \mathsf{GA}_Y^{\tau_X,n}(y_j)$ for all $j \in [n]$. Let $t'_X = \text{type}(x')$ and $t'_Y = \text{type}(y'_j)$ (and note that it is independent of j). Then, our final gadgets are $x := \mathsf{GA}_X^{t'_Y,n}(x')$ and $y := \mathsf{GA}_Y^{t'_X,1}(y'_1, \ldots, y'_n)$.

First, note that type(x) and type(y) only depend on n, τ_X and τ_Y. Then, let us bound the lengths of x, y. We know that $|x'|, |y'_j| \leq c \cdot n \cdot (\ell_X + \ell_Y)$, and therefore $|x|, |y| \leq c \cdot n \cdot (|x'| + |y'_j|) \leq c^2 \cdot n^2 \cdot (\ell_X + \ell_Y)$.

Finally, we prove the correctness. By definition of alignment gadgets, we have $\delta(x', y'_j) = C + \min_{i \in [n]} \delta(x_i, y_j)$ for all $j \in [n]$ and some fixed integer C. Moreover, for some integer C', we have that $\delta(x, y) = C' + \min_{j \in [n]} \delta(x', y'_j)$ which is equal to $C' + C + \min_{j \in [n]} \min_{i \in [n]} \delta(x_i, y_j)$. □

4.3 Similarity Measures with Alignment Gadgets

Bringmann and Künnemann [18] construct alignment gadgets for a few fundamental similarity measures. Combining these gadgets with our main theorem implies significantly stronger lower bounds for computing these measures. We list these measures and the corresponding sizes of alignments gadgets below.

Edit Distance.

The Edit Distance between two sequences x, y is the minimum number of insertions, deletions, and substitutions that is required to transform one sequence to the other. The most basic case is when the sequences are binary, that is the set of instances $\mathcal{I}$ is the set of binary sequences $\{0,1\}^n$.

LEMMA 5 ([18]). *There is a constant $c \leq 10^3$ such that Edit Distance similarity measure over binary sequences admits an alignment gadget of size $f(n) \leq c \cdot n$.*

It is likely that smaller alignment gadgets can be obtain if the alphabet size is larger.

Longest Common Subsequence.

In Section 3 we showed a direct reduction from BP-SAT to LCS. We will use the alignment gadgets framework in order to obtain the same reduction but to sequences over *binary* inputs. That is, the surprising expressibility of LCS that is exhibited by our proofs is already present when we only have two distinct letters to match.

LEMMA 6 ([18]). *There is a constant $c \leq 10^3$ such that Longest Common Subsequence similarity measure over binary sequences admits an alignment gadget of size $f(n) \leq c \cdot n$.*

Dynamic Time Warping Distance.

The DTWD over curves in various metric spaces is of great practical interest. We are able to show lower bounds even in the special case of one-dimensional curves. Let $x, y \in \mathbb{Z}^n$ be two sequences of n integers. The DTWD $\delta_{DTWD}(x, y)$ of the two curves is the minimum cost of a joint traversal of both curves. A *traversal* of two curves is a process that places a marker at the beginning of each curve and during each step one or both markers are moved forward one point, until the end of both curves is reached. Each step aligns two points, one from each curve. The cost of a traversal is the sum of distances between all aligned points. In our case, the distance is simply the absolute value of the difference between the aligned integers.

LEMMA 7 ([18]). *There is a constant $c \leq 10$ such that Dynamic Time Warping Distance similarity measure over one dimensional curves admits an alignment gadget of size $f(n) \leq c \cdot n$.*

5. THE FULL REDUCTION

We are now ready to prove our main theorem, from which Theorem 2 follows, by the Lemmas in the previous section. Due to lack of space, the proofs of many of the claims in this section have been deferred to the full version [4].

THEOREM 5 (MAIN). *Let δ be a similarity measure over sequences in $\mathcal{I}$ admitting an alignment gadget of size $f(n) = cn$ and coordinate values, and consider the problem of computing $\delta(x, y)$ for sequences $x, y \in \mathcal{I}$ of length N. SAT of nondeterministic Branching Programs over n variables of width W and length T can be reduced to an instance of this problem on sequences where $N = O(n \cdot T^{\log_2(12W^2c^3)} \cdot c^2 \cdot \log_2 W) = T^{O(\log W)} \cdot n$.*

PROOF. Let δ be a similarity measure that admits alignment gadgets and coordinate values. We will construct several other gadgets with certain properties from these primitives. Given an instance of SAT on BPs of on n variables width W and length T we will construct and combine our gadgets in a certain way into two sequences x, y such that $\delta(x, y)$ will determine whether our instance is "yes" or "no".

Let A and B both be the set of all binary vectors of length $n/2$. The goal of the reduction is to find a pair $a \in A, b \in B$ such that our given branching program is satisfied by the assignment in which the first half of the variables are assigned according to a while the second half are assigned according to b.

We will use the parameters $w := \log_2 W$, and $t = \log_2 T$, and assume the these are integers.

Reachability gadgets.

The main component in our reduction are recursive constructions of two kinds of *reachability gadgets* $\mathsf{RG}_X(a)$ and $\mathsf{RG}_Y(b)$, with the following very useful property. For every pair of vectors $a \in A, b \in B$ and pair of nodes in the branching program $u \in L_i$ from layer i and $v \in L_j$ from layer j, such that $j - i = 2^{k-1}$ is a power of two, we have that:

- the δ-distance between the two sequences $\mathsf{RG}_X^{k,u\to v}(a)$ and $\mathsf{RG}_Y^{k,u\to v}(b)$ is equal to a certain fixed value ρ_k that depends only on k if the path of the branching starting at u and induced by the assignment (a, b) reaches v, and the δ-distance is greater than ρ_k otherwise.

- the total length of these gadgets can be upper bounded by $\ell_k \le O(W^2c^3)^k$.

We will now show how to construct such gadgets and prove that they satisfy the above properties. Then, we will use these gadgets in order to check whether a pair (a, b) makes the accept node $u_{\mathtt{acc}}$ reachable from the start node $u_{\mathtt{start}}$.

Base Case: $k = 1$.

We start by defining the gadgets $\mathsf{RG}^{1,u\to v}$ for the case that $k = 1$ and our two nodes u, v are in consecutive layers i and $i+1$ for some $i \in [s]$. To do this, we consider the variable x_j that the layer i in our branching program is labelled with, and check whether that variable appears in the vectors in A or in B – that is, we check whether $j \le d/2$ or $j > d/2$ where d is the number of variables in our branching program.

In the first case, the vector a is "responsible" for verifying the consistency of the edge $u \to v$, and we define the gadgets as follows: we set

$$\mathsf{RG}_X^{1,u\to v}(a) := EG_X(0) := \mathsf{GA}_X^{2w+2,\tau_Y}(\mathbf{0}_X, \ldots, \mathbf{0}_X, \mathbf{1}_X, \mathbf{0}_X)$$

if the edge $u \to v$ is labelled with the boolean value $\eta \in \{0, 1\}$ which is the same as the boolean value that a assigns to the variable x_j, and otherwise we set

$$\mathsf{RG}_X^{1,u\to v}(a) = EG_X(1) := \mathsf{GA}_X^{2w+2,\tau_Y}(\mathbf{1}_X, \ldots, \mathbf{1}_X, \mathbf{1}_X, \mathbf{0}_X).$$

Note that inconsistency can either be caused by the edge not existing in the branching program, or by the vector assigning a different value to the corresponding variable. On the other hand, since the vector b is not "responsible" for the edge $u \to v$, we unconditionally set

$$\mathsf{RG}_Y^{1,u\to v}(b) := EG_Y(1) := \mathsf{GA}_Y^{2w+2,\tau_X}(\mathbf{1}_Y, \ldots, \mathbf{1}_Y, \mathbf{0}_Y, \mathbf{1}_Y).$$

The reason for defining these gadgets as a concatenation of $2w + 2$ value gadgets instead of one is purely technical: we want to ensure that these gadgets have the same type as our "index gadgets" which we will define shortly. Also, the last two coordinates are supposed to distinguish between our "reachability gadgets" and our "index gadgets", so that matching an index gadget with a reachability gadget will incur a loss due to these coordinates.

In the second case, the vector b is "responsible" for verifying the consistency of the edge $u \to v$, and we define the gadgets in a symmetric way: The gadget $\mathsf{RG}_X^{1,u\to v}(a)$ will be unconditionally set to $EG_X(1)$. While we set

$$\mathsf{RG}_Y^{1,u\to v}(b) := EG_Y(0) := \mathsf{GA}_Y^{2w+2,\tau_X}(\mathbf{0}_Y, \ldots, \mathbf{0}_Y, \mathbf{0}_Y, \mathbf{1}_Y)$$

if the edge $u \to v$ is labelled with the boolean value $\eta \in \{0, 1\}$ which is the same as the boolean value that b assigns to the variable x_j, and otherwise we set it to $EG_X(1)$.

Let $\tau_X^1 := \mathrm{type}(EG_X(0)) = \mathrm{type}(EG_X(1)) = (\ell_X^1, s_X^1)$ and $\tau_Y^1 := \mathrm{type}(EG_Y(0)) = \mathrm{type}(EG_Y(1)) = (\ell_Y^1, s_Y^1)$. Let $L_0 = (\ell_X^1 + \ell_Y^1)$ and note that $L_0 \le c \cdot (2w + 2) \cdot D$ where D is some constant that upper bounds the lengths of our coordinate values, and therefore $L_0 = O(cw)$.

Let $\rho_T := \delta(\mathbf{0}_X, \mathbf{0}_Y) = \delta(\mathbf{0}_X, \mathbf{1}_Y) = \delta(\mathbf{1}_X, \mathbf{0}_Y)$ and $\rho_F := \delta(\mathbf{1}_X, \mathbf{1}_Y)$, and by definition we have $\rho_F \ge \rho_T + 1$. By definition of alignment gadgets there is a constant $C_1 \in \mathbb{Z}$ such that

$$\begin{aligned}\delta(EG_X(0), EG_Y(0)) &= \delta(EG_X(1), EG_Y(0)) = \\ \delta(EG_X(0), EG_Y(1)) &= C_1 + (2w + 2)\rho_T =: \rho_1\end{aligned}$$

while $\delta(EG_X(1), EG_Y(1)) = C_1 + 2w\rho_F + 2\rho_T$ is larger than ρ_1. Combining these formulas with the definitions of our gadgets proves the following claim.

CLAIM 3. *For any two vectors $a \in A, b \in B$ and two nodes u, v in the branching program, the δ-distance between the two gadgets $\mathsf{RG}_X^{1,u\to v}(a)$ and $\mathsf{RG}_Y^{1,u\to v}(b)$ is equal to ρ_1 if there is an edge from u to v in the branching program induced by the assignment (a, b), and the δ-distance is larger otherwise.*

Level 1 *index gadgets.*

For a boolean value $b \in \{0, 1\}$ we let $CV_X(b), CV_Y(b)$ be $\mathbf{0}_X, \mathbf{0}_Y$ respectively if $b = 0$ and $\mathbf{1}_X, \mathbf{1}_Y$ otherwise. Recall that $w = \log_2 W$, and for each number $z \in [W]$ we let $\bar{z} = (z_1, \ldots, z_w) \in \{0, 1\}^w$ be the binary representation of z, and define the level 1 index gadgets as follows.

$$\begin{aligned}\mathsf{IG}_X^1(z) := \mathsf{GA}_X^{2w+2,\tau_Y}\big(&CV_X(z_1), \ldots, CV_X(z_w),\\ &CV_X(\neg z_1), \ldots, CV_X(\neg z_w), \mathbf{0}_X, \mathbf{1}_X\big)\end{aligned}$$

$$\begin{aligned}\mathsf{IG}_Y^1(z) := \mathsf{GA}_Y^{2w+2,\tau_X}\big(&CV_Y(\neg z_1), \ldots, CV_Y(\neg z_w),\\ &CV_Y(z_1), \ldots, CV_Y(z_w), \mathbf{1}_Y, \mathbf{0}_Y\big)\end{aligned}$$

Observe that by definition of our alignment gadgets, the sequence $\mathsf{IG}_X^1(z)$ will have the same type $\mathrm{type}(\mathsf{IG}_X^1(z)) = \tau_X^1$ for all $z \in [W]$, and similarly $\mathrm{type}(\mathsf{IG}_Y^1(z)) = \tau_Y^1$ for all $z \in [W]$, which are the same types we had in the level-1 reachability gadgets. This definition allows us to prove the following property.

CLAIM 4. *The distance $\delta(\mathsf{IG}_X^1(z), \mathsf{IG}_Y^1(z'))$, for any $z, z' \in [W]$, is equal to ρ_1 if $z = z'$ and it is larger otherwise.*

Finally, we show that due to the last two coordinates, the distance between an index gadget and a reachability gadget is large.

Claim 5. *For any $z \in [W]$, and any two vectors $a \in A, b \in B$ and two nodes u, v in the branching program, we have that $\delta(\mathsf{IG}^1_X(z), \mathsf{RG}^{1,u\to v}_Y(b))$ and $\delta(\mathsf{RG}^{1,u\to v}_X(a), \mathsf{IG}^1_Y(z))$ are both larger than ρ_1.*

This proves the base case for the following lemma, which is our main construction.

Lemma 8 (Reachability gadgets). *For all integers $k \geq 1$, the following statement is true: For any vectors $a \in A, b \in B$, integer $z, z' \in [W]$, nodes $u, v \in V$, we can construct gadgets:*

$$\mathsf{RG}^{k-1,u\to v}_X(a) \in \mathcal{I}_{\tau^k_X} \quad \text{and} \quad \mathsf{RG}^{k-1,u\to v}_Y(b) \in \mathcal{I}_{\tau^k_Y}$$

$$\mathsf{IG}^{k-1}_X(z) \in \mathcal{I}_{\tau^k_X} \quad \text{and} \quad \mathsf{IG}^{k-1}_Y(z') \in \mathcal{I}_{\tau^k_Y}$$

such that for some value ρ_k:

- *$\delta(\mathsf{RG}^{k,u\to v}_X(a), \mathsf{RG}^{k,u\to v}_Y(b))$ is equal to ρ_k if there is a path of length 2^{k-1} from u to v in the branching program induced by the assignment (a, b), and is larger otherwise.*
- *$\delta(\mathsf{IG}^k_X(z), \mathsf{IG}^k_Y(z'))$ is equal to ρ_k if $z = z'$ and is larger otherwise.*
- *$\delta(\mathsf{IG}^k_X(z), \mathsf{RG}^{k,u\to v}_Y(b))$, and $\delta(\mathsf{RG}^{k,u\to v}_X(a), \mathsf{IG}^k_Y(z'))$ are larger than ρ_k.*
- *$\tau^k_X = (\ell^k_X, s^k_X)$ and $\tau^k_Y = (\ell^k_Y, s^k_Y)$ only depend on k.*
- *The construction can be computed in $O(\ell_X + \ell_Y)$ time.*
- *The length of these gadgets can be upper bounded by $\ell^k_X, \ell^k_Y \leq (12W^2c^3)^k \cdot L_0$.*

Proof. To prove the lemma, it remains to show the inductive step. Fix any $k > 1$ and from now on, assume that the statement of the lemma is true for $k-1$, and we will show that it is also true for k.

The $k > 1$ Case.

Let $u = (i, i_z) \in [T] \times [W]$ be a node on layer i and let $v = (j, j_z) \in [T] \times [W]$ be a node on layer j. Assume that $j - i = 2^{k-1}$ and we are at level $k \in [\log_2 T]$ of the recursive construction. We define $h = \frac{i+j}{2}$ to be the layer in the middle between i and j and note that $h - i = j - h = 2^{k-2}$. For each node $w = (h, z)$ for $z \in [W]$ in layer h we will add a gadget that enables the path from u to v to pass through this node w. We do this by recursively adding the two gadgets $\mathsf{RG}^{k-1,u\to w}$ and $\mathsf{RG}^{k-1,w\to v}$.

To do this formally, we will combine an alignment gadget with an OR gadget.

We start by defining *path gadgets*, which will be used to determine whether there is a path from u to v through a specific node $w = (h, z)$. For all vectors $a \in A, b \in B$ and all numbers $z \in [W]$, we define:

$$\mathsf{PG}^{k,u\to v}_X(a, z) := \mathsf{GA}^{3,\tau^{k-1}_Y}_X\left(\mathsf{RG}^{k-1,u\to(h,z)}_X(a), \mathsf{RG}^{k-1,(h,z)\to v}_X(a), \mathsf{IG}^{k-1}_X(z)\right)$$

$$\mathsf{PG}^{k,u\to v}_Y(b, z) := \mathsf{GA}^{3,\tau^{k-1}_X}_Y\left(\mathsf{RG}^{k-1,u\to(h,z)}_Y(b), \mathsf{RG}^{k-1,(h,z)\to v}_Y(b), \mathsf{IG}^{k-1}_Y(z)\right)$$

and note that the types

$$\tau^{(1),k}_X := \text{type}(\mathsf{PG}^{k,u\to v}_X(a, z)) = (\ell^{(1),k}_X, s^{(1),k}_X) \text{ and}$$
$$\tau^{(1),k}_Y := \text{type}(\mathsf{PG}^{k,u\to v}_Y(b, z)) = (\ell^{(1),k}_Y, s^{(1),k}_Y)$$

depend only k, and that $\ell^{(1),k}_X, \ell^{(1),k}_Y \leq c \cdot 3 \cdot (\ell^{k-1}_X + \ell^{k-1}_Y) \leq 3c \cdot 2 \cdot (12W^2c^3)^{k-1} \cdot L_0$.

Claim 6. *There is a constant $C^{(1)}_k \in \mathbb{Z}$ such that for all vectors $a \in A, b \in B$ and integers $z, z' \in [W]$ we have that*

$$\delta(\mathsf{PG}^{k,u\to v}_X(a, z), \mathsf{PG}^{k,u\to v}_Y(b, z')) = C^{(1)}_k + 3 \cdot \rho_{k-1}$$

if $z = z'$ and there is a path from u to v through (h, z) in the branching program induced by (a, b), and the δ-distance is larger otherwise.

We are now ready to define our reachability gadgets, using OR gadgets. Recall that by Lemma 4, any measure that admits alignment gadgets of size cn also admits OR gadgets of size c^2n^2.

$$\mathsf{RG}^{k,u\to v}_X(a) := \mathsf{OR}^{W,\tau^{(1),k}_Y}_X\left(\mathsf{PG}^{k,u\to v}_X(a, 1), \ldots, \mathsf{PG}^{k,u\to v}_X(a, W)\right)$$

$$\mathsf{RG}^{k,u\to v}_Y(b) := \mathsf{OR}^{W,\tau^{(1),k}_X}_Y\left(\mathsf{PG}^{k,u\to v}_Y(b, 1), \ldots, \mathsf{PG}^{k,u\to v}_Y(b, W)\right)$$

Note that the types $\tau^k_X := \text{type}(\mathsf{RG}^{k,u\to v}_X(a)) = (\ell^k_X, s^k_X)$ and $\tau^k_Y := \text{type}(\mathsf{RG}^{k,u\to v}_Y(b)) = (\ell^k_Y, s^k_Y)$ depend only k, and that

$$\ell^k_X, \ell^k_Y \;\leq\; c^2 \cdot W^2 \cdot (\ell^{(1),k}_X + \ell^{(1),k}_Y) \;\leq$$
$$c^2W^2 \cdot (2 \cdot 6c \cdot (12W^2c^3)^{k-1} \cdot L_0) \;=\; (12W^2c^3)^k \cdot L_0 \;.$$

Claim 7. *There is a constant $\rho_k \in \mathbb{Z}$ such that for all vectors $a \in A, b \in B$ we have that*

$$\delta(\mathsf{RG}^{k,u\to v}_X(a), \mathsf{RG}^{k,u\to v}_Y(b)) = \rho_k$$

there is a path from u to v in the branching program induced by (a, b), and the δ-distance is larger otherwise.

To complete the proof of Lemma 8 we need to construct index gadgets. These gadgets have straightforward functionality, but their definitions are a bit complicated because we must enforce that the type of these gadgets is exactly the same as the types of the reachability gadgets.

For all $z \in [W]$ we define

$$\mathsf{IG}^{(1),k}_X(z) := \mathsf{GA}^{3,\tau^{k-1}_Y}_X(\mathsf{IG}^{k-1}_X(z), \mathsf{IG}^{k-1}_X(z), \mathsf{IG}^{k-1}_X(z))$$

$$\mathsf{IG}^{(1),k}_Y(z) := \mathsf{GA}^{3,\tau^{k-1}_X}_Y(\mathsf{IG}^{k-1}_Y(z), \mathsf{IG}^{k-1}_Y(z), \mathsf{IG}^{k-1}_Y(z))$$

so that the types are $\tau^{(1),k}_X$ and $\tau^{(1),k}_Y$. By an argument similar (but simpler) to the one in Claim 6, we get that for all integers $z, z' \in [W]$ we have $\delta(\mathsf{IG}^{(1),k}_X(z), \mathsf{IG}^{(1),k}_X(z')) = C^{(1)}_k + 3 \cdot \rho_{k-1}$ if $z = z'$ and the δ-distance is larger otherwise. Another straightforward consequence of these definitions is that for any vectors $a \in A, b \in B$ and integers

$z, z' \in [W]$ we have that $\delta(\mathsf{IG}_X^{(1),k}(z), \mathsf{PG}_Y^{k,u\to v}(b,z'))$ and $\delta(\mathsf{PG}_X^{k,u\to v}(a,z'), \mathsf{IG}_Y^{(1),k}(z))$ are larger than $C_k^{(1)} + 3\cdot\rho_{k-1}$.

Finally, we define

$$\mathsf{IG}_X^k(z) := \mathsf{OR}_X^{W,\tau_Y^{(1),k}}\left(\mathsf{IG}_X^{(1),k}(z),\ldots,\mathsf{IG}_X^{(1),k}(z)\right)$$

$$\mathsf{IG}_Y^k(z) := \mathsf{OR}_Y^{W,\tau_X^{(1),k}}\left(\mathsf{IG}_Y^{(1),k}(z),\ldots,\mathsf{IG}_Y^{(1),k}(z)\right)$$

so that the types are τ_X^k and τ_Y^k. Again, by an argument similar (but simpler) to the one in Claim 7, we have that $\delta(\mathsf{IG}_X^k(z), \mathsf{IG}_Y^k(z')) = \rho_k$ if $z = z'$ and is larger otherwise. And, moreover, that $\delta(\mathsf{IG}_X^k(z), \mathsf{RG}_Y^k(b)), \delta(\mathsf{RG}_X^k(a), \mathsf{IG}_Y^k(z))$ are larger than ρ_k, for all a, b, z.

To complete the proof of Lemma 8, we remark that this construction takes linear time in its output, since the runtime is dominated by the constructions of alignment and OR gadgets. □

We now continue with the reduction from BP-SAT to the problem of computing the δ-similarity of two sequences.

Recall that $t = \log_2 T$, and let the start node of the branching program be $u_{\mathtt{start}} = (1,1)$ and the only accept node be $u_{\mathtt{acc}} = (2^t, 1)$. Intuitively, we would like to define vector gadgets of the form $VG(a) = \mathsf{RG}^{t,u_{\mathtt{start}}\to u_{\mathtt{acc}}}(a)$ and $VG(b) = \mathsf{RG}^{t,u_{\mathtt{start}}\to u_{\mathtt{acc}}}(b)$ so that $\delta(VG(a), VG(b))$ will tell us whether (a,b) is a satisfying pair or not (whether it induces a path from the start node to the accepting node). However, this does not quite work for the following technical reason: When we combine all these $2n$ vector gadgets into two sequences x, y, the score of $\delta(VG(a'), VG(b'))$ of other pairs a', b' will affect the overall score, and could potentially hide the contribution of the satisfying pair.

To fix this, we "normalize" the vector gadgets so that the distance $\delta(VG(a'), VG(b'))$ of unsatisfying pairs is fixed (and is slightly worse than the distance of satisfying pairs). This "normalization" trick was introduced by Backurs and Indyk in their reduction from OV to Edit-Distance.

We will need the following simple constructions of instances S^k such that the distance between S^k and any reachability gadget is fixed, and is slightly worse than the score of a "good pair".

Claim 8. *For all $k \geq 1$, there are sequences $S^k, T^k \in \mathcal{I}_{\tau_X^k}$ such that for all vectors $b \in B$ and nodes u, v we have that $\delta(S^k, \mathsf{RG}_Y^{k,u\to v}) = \rho_k + (\rho_F - \rho_T)$ and $\delta(T^k, \mathsf{RG}_Y^{k,u\to v}) = \rho_k$.*

We can now define our *normalized vector gadgets*. For all vectors $a \in A, b \in B$ we define:

$$\mathsf{NVG}_X(a) := \mathsf{GA}_X^{1,\tau_Y^t}(S^t, \mathsf{RG}_X^{t,u_{\mathtt{start}}\to u_{\mathtt{acc}}}(a))$$

$$\mathsf{NVG}_Y(b) := \mathsf{GA}_Y^{2,\tau_X^t}(\mathsf{RG}_Y^{t,u_{\mathtt{start}}\to u_{\mathtt{acc}}}(b))$$

We denote the types of these gadgets by $\mathrm{type}(\mathsf{NVG}_X(a)) =: \tau_X'$ and $\mathrm{type}(\mathsf{NVG}_Y(b)) =: \tau_Y'$ and remark that they are independent of a, b. Also, note that the length of these gadgets can be upper bounded by $c\cdot 2\cdot(\ell_X^t + \ell_Y^t) \leq 2c\cdot(12W^2c^3)^t \cdot L_0 = O((12W^2c^3)^{\log_2 T}\cdot c\log_2 W)$.

Lemma 9 (Vector Gadgets). *There is a constant $C \in \mathbb{Z}$ such that for any two vectors $a \in A, b \in B$ we have that:*

$$\delta(\mathsf{NVG}_X(a), \mathsf{NVG}_Y(b)) = C + \rho_t$$

if the pair (a,b) satisfies the branching program, and otherwise

$$\delta(\mathsf{NVG}_X(a), \mathsf{NVG}_Y(b)) = C + \rho_t + (\rho_F - \rho_T).$$

Proof. The proof follows from the definition of alignment gadgets and from Lemma 8 and Claim 8. □

Let $A = \{a_1, \ldots, a_{2^{n/2}}\}$ and $B = \{b_1, \ldots, b_{2^{n/2}}\}$ be our sets of vectors. Our final sequences are defined as follows:

$$x := \mathsf{GA}_X^{2^{n/2},\tau_Y'}\big(\mathsf{NVG}_X(a_1),\ldots,\mathsf{NVG}_X(a_{2^{n/2}}), \mathsf{NVG}_X(a_1),\ldots,\mathsf{NVG}_X(a_{2^{n/2}})\big)$$

$$y := \mathsf{GA}_Y^{2\cdot 2^{n/2},\tau_X'}\left(\mathsf{NVG}_Y(b_1),\ldots,\mathsf{NVG}_Y(b_{2^{n/2}})\right)$$

First, we upper bound the length of these sequences:

$$\begin{aligned}|x|, |y| &\leq c\cdot 2\cdot 2^{n/2}\cdot O((12W^2c^3)^{\log_2 T}\cdot c\log_2 W)\\ &= O((12W^2c^3)^{\log_2 T}\cdot 2^{n/2}\cdot c^2\log_2 W)\\ &= T^{O(\log W)}\cdot 2^{n/2}\end{aligned}$$

Finally, the theorem follows from this claim which shows that the answer to our BP-SAT can be deduced from $\delta(x,y)$.

Claim 9. *There is a constant $C^* \in \mathbb{Z}$ such that*

$$\delta(x,y) \leq C^* + (2^{n/2}-1)\cdot(C + \rho_t + (\rho_F - \rho_T)) + (C + \rho_t)$$

if and only if there is a pair $a \in A, b \in B$ that satisfies the branching program.

6. ACKNOWLEDGEMENT

We thank Arturs Backurs for comments on an earlier version.

7. REFERENCES

[1] A. Abboud, A. Backurs, T. D. Hansen, V. Vassilevska Williams, and O. Zamir. Subtree isomorphism revisited. In *Proc. of 27th SODA*, pages 1256–1271, 2016.

[2] A. Abboud, A. Backurs, and V. Vassilevska Williams. Tight Hardness Results for LCS and other Sequence Similarity Measures. In *Proc. of 56th FOCS*, pages 59–78, 2015.

[3] A. Abboud, F. Grandoni, and V. Vassilevska Williams. Subcubic equivalences between graph centrality problems, APSP and diameter. In *Proc. of 26th SODA*, pages 1681–1697, 2015.

[4] 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. *CoRR*, abs/1511.06022, 2015.

[5] A. Abboud and V. Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *Proc. of 55th FOCS*, pages 434–443, 2014.

[6] A. Abboud, V. Vassilevska Williams, and J. R. Wang. Approximation and fixed parameter subquadratic algorithms for radius and diameter in sparse graphs. In *Proc. of 27th SODA*, pages 377–391, 2016.

[7] A. Abboud, V. Vassilevska Williams, and O. Weimann. Consequences of faster sequence alignment. In *Proc. of 41st ICALP*, pages 39–51, 2014.

[8] A. Abboud, V. Vassilevska Williams, and H. Yu. Matching triangles and basing hardness on an extremely popular conjecture. In *Proc. of 47th STOC*, pages 41–50, 2015.

[9] A. Abboud, R. Williams, and H. Yu. More applications of the polynomial method to algorithm design. In *Proc. of 26th SODA*, pages 218–230, 2015.

[10] J. Alman and R. Williams. Probabilistic polynomials and hamming nearest neighbors. In *Proc. of 56th FOCS*, pages 136–150, 2015.

[11] S. Arora and B. Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.

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

[13] N. Bansal and R. Williams. Regularity lemmas and combinatorial algorithms. In *Proc. of 50th FOCS*, pages 745–754, 2009.

[14] D. A. Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC1. *Journal of Computer and System Sciences*, 38(1):150–164, 1989.

[15] C. Beck and R. Impagliazzo. Strong ETH holds for regular resolution. In *Proc. of 45th STOC*, pages 487–494, 2013.

[16] H. Bodlaender, R. G. Downey, M. Fellows, and H. T. Wareham. The parameterized complexity of sequence alignment and consensus. In *Combinatorial Pattern Matching*, pages 15–30, 1994.

[17] K. 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.

[18] K. Bringmann and M. Kunnemann. Quadratic Conditional Lower Bounds for String Problems and Dynamic Time Warping. In *Proc. of 56th FOCS*, pages 79–97, 2015.

[19] K. Bringmann and W. Mulzer. Approximability of the Discrete Fréchet Distance. In *Proc. of 31st SoCG*, pages 739–753, 2015.

[20] C. Calabro, R. Impagliazzo, and R. Paturi. A duality between clause width and clause density for SAT. In *Proc. of 21st CCC*, pages 252–260, 2006.

[21] C. Calabro, R. Impagliazzo, and R. Paturi. The complexity of satisfiability of small depth circuits. In *Proc. of 4th IWPEC*, pages 75–85, 2009.

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

[23] T. M. Chan and R. Williams. Deterministic apsp, orthogonal vectors, and more: Quickly derandomizing razborov-smolensky. In *Proc. of 27th SODA*, pages 1246–1255, 2016.

[24] V. Chvatal, D. Klarner, and D. E. Knuth. Selected combinatorial research problems. Technical Report STAN-CS-72-292, Computer Science Department, Stanford University.

[25] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.

[26] E. Dantsin and E. A. Hirsch. Worst-case upper bounds. In *Handbook of Satisfiability*, pages 403–424. 2009.

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

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

[29] H. Jahanjou, E. Miles, and E. Viola. Local reductions. In *Proc. of 42nd ICALP*, pages 749–760, 2015.

[30] A. G. Jørgensen and S. Pettie. Threesomes, degenerates, and love triangles. In *Proc. of 55th FOCS*, pages 621–630, 2014.

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

[32] M. Patrascu and R. Williams. On the possibility of faster SAT algorithms. In *Proc. of 21st SODA*, pages 1065–1075, 2010.

[33] R. Paturi, P. Pudlák, M. E. Saks, and F. Zane. An improved exponential-time algorithm for k-sat. *J. ACM*, 52(3):337–364, 2005.

[34] K. Pietrzak. On the parameterized complexity of the fixed alphabet shortest common supersequence and longest common subsequence problems. *Journal of Computer and System Sciences*, 67(4):757–771, 2003.

[35] L. Roditty and V. Vassilevska Williams. Fast approximation algorithms for the diameter and radius of sparse graphs. In *Proc. of 45th STOC*, pages 515–524, 2013.

[36] W. J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *Journal of computer and system sciences*, 4(2):177–192, 1970.

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

[38] L. G. Valiant. *Graph-theoretic arguments in low-level complexity*. Springer, 1977.

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

[40] R. Williams. Improving exhaustive search implies superpolynomial lower bounds. *SIAM J. Comput.*, 42(3):1218–1244, 2013.

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

[42] R. Williams. Nonuniform ACC circuit lower bounds. *J. ACM*, 61(1):2:1–2:32, 2014.

[43] H. Yu. An improved combinatorial algorithm for boolean matrix multiplication. In *Proc. of 42nd ICALP*, pages 1094–1105, 2015.

[44] S. Žák. A turing machine time hierarchy. *Theoretical Computer Science*, 26(3):327–333, 1983.