

Fine-grained Complexity Meets $\text{IP} = \text{PSPACE}$

Lijie Chen* Shafi Goldwasser† Kaifeng Lyu‡ Guy N. Rothblum§ Aviad Rubinstein¶

Abstract

In this paper we study the fine-grained complexity of finding exact and approximate solutions to problems in P . Our main contribution is showing reductions from an exact to an approximate solution for a host of such problems.

As one (notable) example, we show that the Closest-LCS-Pair problem (Given two sets of strings A and B , compute exactly the maximum $\text{LCS}(a, b)$ with $(a, b) \in A \times B$) is equivalent to its approximation version (under near-linear time reductions, and with a constant approximation factor). More generally, we identify a class of problems, which we call BP-Pair-Class, comprising both exact and approximate solutions, and show that they are all equivalent under near-linear time reductions.

Exploring this class and its properties, we also show:

- Under the NC-SETH assumption (a significantly more relaxed assumption than SETH), solving any of the problems in this class requires essentially quadratic time.
- Modest improvements on the running time of known algorithms (shaving log factors) would imply that NEXP is not in non-uniform NC^1 .
- Finally, we leverage our techniques to show new barriers for deterministic approximation algorithms for LCS.

A very important consequence of our results is that they continue to hold in the *data structure setting*. In particular, it shows that a data structure for *approximate* Nearest Neighbor Search for LCS (NNS_{LCS}) implies a data structure for *exact* NNS_{LCS} and a data structure for answering regular expression queries with essentially the same complexity.

At the heart of these new results is a deep connection between interactive proof systems for bounded-space computations and the fine-grained complexity of exact and approximate solutions to problems in P . In particular, our results build on the proof techniques from the classical $\text{IP} = \text{PSPACE}$ result.

*MIT, lijie@mit.edu. Supported by an Akamai Fellowship.

†MIT and Simons Institute for the Theory of Computing, Berkeley, shafi@theory.csail.mit.edu. Supported by NSF grant CNS-1413920.

‡Tsinghua University, lkf15@mails.tsinghua.edu.cn. Most of the work was done while visiting MIT.

§Weizmann Institute of Science, rothblum@alum.mit.edu. Part of this research was done while the author was visiting MIT, supported by NSF grant CNS-1413920.

¶Stanford University, aviad@cs.stanford.edu. Supported by a Robert N. Noyce Family Faculty Fellowship. Most of the work was done while the author was at Harvard University, supported by a Rabin Postdoctoral Fellowship.

1 Introduction

The study of the fine-grained hardness of problems in P is one of the most interesting developments of the last few years in complexity theory. The study was initially aimed at the complexity of *exact versions* of important problem in P , such as Longest Common Subsequence (LCS), Edit Distance, All Pair Shortest Path (APSP), and 3-SUM. This was the natural starting point. There are several main thrusts of the study: establishing equivalence classes of problems that are “equivalent” to each other in the sense that a substantial improvement in one would imply a similar improvement in the other; showing fine-grained hardness under complexity assumptions, most notably the SETH¹; and showing implications of even slight algorithmic improvements, such as “shaving-logs” off algorithms for P time problems, to circuit lower bounds.

However, for many of these problems, approximate solutions are of interest as well, as they originate in natural problems which arise in pattern matching and bioinformatics [AVW14, BI15, BI16, BGL17, BK18], dynamic data structures [Pat10, AV14, AV14, HKNS15, KPP16, AD16, HLN17, GKLP17], graph algorithms [RW13, GIKW17, AVY15, KT17], computational geometry [Bri14, Wil18a, DKL18, Che18] and machine learning [BIS17]. Thus, studying the hardness of the *approximation version* of the problems, soon became the next frontier.

There are two ways one can imagine to attack the hardness of approximation of problems.

1. **Show approximation hardness under complexity assumptions.** This has been the approach by the recent breakthrough result of Abboud, Rubinstein and Williams [ARW17] who introduced a “Distributed PCP” framework and used it to show tight conditional lower bounds, under the SETH assumption, for several fundamental approximation problems, including approximate Bichromatic Max-Inner Product, Subset Query, Bichromatic LCS Closest Pair, Regular Expression Matching and Diameter in Product Metrics.

We remark that the challenge in showing tight lower bounds for the hardness on approximation problems in P in contrast to exact problems, is that the traditional PCP paradigm can not be applied directly to fine-grained complexity, due to the super-linear size blow up in the constructed PCP instances [AS98, ALM⁺98, Din07], which becomes super-polynomial after reducing to problems in P (when we care about the *exact* exponent of the running time, a super-polynomial blow-up is certainly unacceptable).

2. **Show equivalence between the hardness of exact and approximate problems.** Namely, the latter is not substantially easier than the former. This is in essence the original PCP methodology for showing the hardness of approximation of intractable problems (e.g. NP-hard problems).

This is the **focus** of the current paper for P -time problems.

Interestingly, in terms of proof techniques, the key to establish our equivalence results is the application of the classical $\mathsf{IP} = \mathsf{PSPACE}$ result [LFKN92, Sha92]. In particular, our results are established via an application of the efficient IP communication protocol for low space computation [AW09]. This demonstrates that the techniques in interactive proofs can be “scaled down” to establish better results in fine-grained complexity. Furthermore, it adds yet another example of the connection between communication complexity and fine-grained complexity (arguably, the most involved one).

1.1 Exact to Approximate Reduction for Nearest Neighbor Search for LCS

We begin with one of our most interesting results: an equivalence between *exact* and *approximate* Nearest Neighbor Search for LCS.

¹The Strong Exponential Time Hypothesis (SETH) states that for every $\varepsilon > 0$ there is a k such that k -SAT cannot be solved in $O((2 - \varepsilon)^n)$ time [IP01].

- Nearest Neighbor Search for LCS (NNS_{LCS}): Preprocess a database \mathcal{D} of N strings of length $D \ll N$, and then for each query string x , find $y \in \mathcal{D}$ maximizing $\text{LCS}(x, y)$.

The approximate version only requires to find $y \in \mathcal{D}$ such that $\text{LCS}(x, y)$ is an $f(D)$ -approximation of the maximum value.

Approximate data structures for the above problem that support fast preprocessing and queries would be highly relevant for bioinformatics. For the similar $\text{NNS}_{\text{Edit-Distance}}$ problem, a breakthrough work of [OR07] used a metric embedding technique to obtain an $2^{O(\sqrt{\log D} \log \log D)}$ -approximate data structure with polynomial preprocessing time, $\text{poly}(D, \log n)$ query time.

In contrast, our first result shows that exact NNS_{LCS} and approximate NNS_{LCS} are essentially equivalent. That is, a similar data structure for *approximate* NNS_{LCS} would directly imply a data structure for *exact* NNS_{LCS} with essentially the same complexity!

Theorem 1.1 (Informal). *For $D = 2^{(\log N)^{o(1)}}$, suppose there is a data structure for NNS_{LCS} with approximation ratio $2^{(\log D)^{1-\Omega(1)}}$, then there is another data structure for exact NNS_{LCS} with essentially the same preprocessing time/space and query time.*

In the following, we first discuss Closest-LCS-Pair (a natural *offline* version of NNS_{LCS}) to illustrate our techniques, and then discuss our other results in details.

1.2 Techniques: Hardness of Approximation in \mathbf{P} via Communication Complexity and the Theory of Interactive Proofs

Closest-LCS-Pair is the problem that given two sets of strings A and B , compute the maximum $\text{LCS}(a, b)$ with $(a, b) \in A \times B$. We show how to reduce exact Closest-LCS-Pair to approximate Closest-LCS-Pair as an illustration of our proof techniques.

Theorem 1.2 (Informal). *There is a near-linear time² reduction from Closest-LCS-Pair to $2^{(\log N)^{1-\Omega(1)}}$ factor approximate Closest-LCS-Pair, when A, B are two sets of N strings of length $D = 2^{(\log N)^{o(1)}}$.*

We first introduce the concept of \mathcal{A} -Satisfying-Pair problem. This problem asks whether there is a pair of (a, b) from two given sets A and B such that (a, b) is a yes-instance of \mathcal{A} . By binary search, Closest-LCS-Pair can be easily formulated as an \mathcal{A} -Satisfying-Pair problem: is there a pair $(a, b) \in A \times B$ such that $\text{LCS}(a, b) \geq k$? The key property we are going to use is that the function $A_{\text{LCS}}^{\geq k}(a, b) := [\text{LCS}(a, b) \geq k]$ can be computed in very small space, i.e., it is in NL (see Lemma 6.4). Indeed, we will show in this paper that for all \mathcal{A} -Satisfying-Pair such that \mathcal{A} can be computed in small space, \mathcal{A} -Satisfying-Pair can be reduced to approximate Closest-LCS-Pair.

The Reduction in a Nutshell. First, we consider an IP communication protocol for LCS. In this setting, Alice and Bob hold strings a and b , and they want to figure out whether $\text{LCS}(a, b) \geq k$. To do so, they seek help from an untrusted prover Merlin by engaging in a conversation with him. The protocol should satisfy that when $\text{LCS}(a, b) \geq k$, Merlin has a strategy to convince Alice and Bob w.h.p., and when $\text{LCS}(a, b) < k$, no matter what Merlin does, Alice and Bob will reject w.h.p. The goal is to minimize the total communication bits (between Alice and Bob, or Alice/Bob and Merlin).

Next, by a result of Aaronson and Wigderson [AW09], it is shown that any function $f(a, b)$ which can be computed in NL admits an IP communication protocol with $\text{polylog}(N)$ total communication bits. Finally, using an observation from [AR18], an efficient IP communication protocol can be embedded into approximate LCS, which completes the reduction. In the following we explain each step in details.

²Throughout this paper, we use near-linear time to denote the running time of $N^{1+o(1)}$.

IP Communication Protocols for Low Space Computation. The key technical ingredient of our results is the application of IP communication protocols for low space computation by Aaronson and Wigderson [AW09]. It would be instructive to explain how it works.

Let us use the sum-check IP protocol for the Inner Product problem as an example. Arthur gets access to a function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ and its multilinear extension $\tilde{f} : \mathbb{F}_q^n \rightarrow \mathbb{F}_q$ over a finite field \mathbb{F}_q . Let $f_i(x) = f(i \circ x)$ for $i \in \{0, 1\}$ be the restrictions of f after setting the first bit of the input, Arthur wants to compute the inner product $\sum_{x \in \{0, 1\}^{n-1}} f_0(x) \cdot f_1(x)$. To do so, he engages in a conversation with an untrusted Merlin who tries to convince him. During the conversation, Merlin is doing all the “real work”, while Arthur only has to query \tilde{f} once at the last step, which is the crucial observation in [AW09].

Now, imagine a slightly different setting where f_0 and f_1 are held by Alice and Bob respectively, which means each of them holds a string of length $N = 2^{n-1}$. They still want to compute the inner product with Merlin, while using minimum communication between each other.

In this setting, Alice (pretending she is Arthur) can still run the previous IP protocol with Merlin. When she has to query $f(z)$ for a point z at the last step, she only needs Bob to send her the contribution of his part to $\tilde{f}(z)$, which only requires $O(\log q)$ bits. In terms of the input size of Alice and Bob, this IP communication protocol runs in $\text{poly}(n) = \text{polylog}(N)$ time. The same also extends to any $\text{poly}(n) = \text{polylog}(N)$ space computation on f , if we use the IP protocol for PSPACE [LFKN92, Sha92].

In Section 5, we provide a parameterized IP communication protocol for Branching Program³ (Theorem 5.5). Informally, we have:

Theorem 1.3 (IP Communication Protocol for BP (Informal)). *Let P be a branching program of length T and width W with n input bits, equally distributed among Alice and Bob. For every soundness parameter $\varepsilon > 0$, there is an IP-protocol for P , such that:*

- Merlin and Alice exchange $\tilde{O}(\log^2 W \log^2 T \log \varepsilon^{-1})$ bits, and toss the same amount of public coins;
- Bob sends $O(\log \log(WT) \cdot \log \varepsilon^{-1})$ bits to Alice;
- Alice always accepts if P accepts the input, and otherwise rejects with probability at least $1 - \varepsilon$.

Since $A_{\text{LCS}}^{\geq k}(a, b)$ is in NL, the above in particular implies that there is an IP communication protocol for $A_{\text{LCS}}^{\geq k}$ with $\text{polylog}(D)$ total communication bits (D is length of strings).

IP Communication Protocols and Tropical Tensors. The next technical ingredient is the reduction from an IP communication protocol to a certain Tropical Tensors problem [AR18]. We use a 3-round IP communication protocol as an example to illustrate the reduction. Consider the following 3-round IP communication protocol Π :

- Alice and Bob hold strings x and y , Merlin knows both x and y .
- Merlin sends Alice and Bob a string $z_1 \in \mathcal{Z}_1$.
- Alice sends Merlin a uniform random string $z_2 \in \mathcal{Z}_2$.
- Merlin sends Alice and Bob another string $z_3 \in \mathcal{Z}_3$.
- Bob sends Alice a string $z_4 \in \mathcal{Z}_4$, and Alice decides whether to accept or reject.

³Informally speaking, a branching program with length T and width W formulates a *non-uniform* low-space computation with running time T and space $\log W$, see Definition 2.1 for a formal definition.

The main idea of [AR18] is that the above IP protocol can be reduced into a certain Tropical Similarity function. That is, for x and y , we build two tensors $u = u(x)$ and $v = v(y)$ of size $|\mathcal{Z}_1| \times |\mathcal{Z}_2| \times |\mathcal{Z}_3| \times |\mathcal{Z}_4|$ as follows: we set u_{z_1, z_2, z_3, z_4} to indicate whether Alice accepts, given the transcript (z_1, z_2, z_3, z_4) and input x ; we also set v_{z_1, z_2, z_3, z_4} to indicate whether Bob sends the string z_4 , given the previous transcript (z_1, z_2, z_3) and input y . Then, by the definition of IP protocols, it is not hard to see the acceptance probability when Merlin uses optimal strategy is:

$$\text{acc}(u, v) := \max_{z_1 \in \mathcal{Z}_1} \mathbb{E}_{z_2 \in \mathcal{Z}_2} \max_{(z_3, z_4) \in \mathcal{Z}_3 \times \mathcal{Z}_4} u_{z_1, z_2, z_3, z_4} \cdot v_{z_1, z_2, z_3, z_4}.$$

In the above equality, the max operator corresponds to the actions of Merlin, who wishes to maximize the acceptance probability, while the \mathbb{E} operator corresponds to actions of Alice, who sends a uniform random string. It can be easily generalized to IP protocols of any rounds, by replacing acc with a series of max and \mathbb{E} operators, which is called Tropical Similarity (denoted by $s(u, v)$) in [AR18] (see also Definition 2.8). When the number of total communication bits is d , both u and v are of size 2^d .

Applying the above to the polylog(D) bits IP protocol for $A_{\text{LCS}}^{\geq k}$, it means for two strings a, b , we can compute two tensors u, v of length $2^{\text{polylog}(D)} = 2^{(\log N)^{o(1)}}$, such that when $\text{LCS}(a, b) \geq k$, $s(u, v)$ is large, and otherwise $s(u, v)$ is small.

Simulating Tropical Tensors by Composing max and Σ Gadgets. While the above reduction is interesting in its own right, the Tropical Similarity function seems quite artificial. Another key idea from [AR18] is that $s(u, v)$ can be simulated by LCS. The reduction works by noting that with LCS, one can implement the max and Σ (which is equivalent to \mathbb{E}) gadgets straightforwardly, and composing them recursively leads to gadgets for Tropical Similarity. That is, for tensors u and v , one can construct strings $S(u)$ and $T(v)$ of similar sizes, such that $\text{LCS}(S(u), T(v))$ is proportional to $s(u, v)$.

Putting everything together, for two strings a, b , we can compute two other strings $S(a)$ and $T(b)$ of length $2^{\text{polylog}(D)} = 2^{(\log N)^{o(1)}}$, such that $\text{LCS}(a, b) \geq k$, $\text{LCS}(S(a), T(b))$ is large, and otherwise $\text{LCS}(S(a), T(b))$ is small. This completes our reduction.

Our Results In Detail

1.3 From Exact to Approximate in the Fine-Grained World

More generally, we consider the following four (general flavor) problems.

- The Closest-LCS-Pair problem.
- The Closest-RegExp-String-Pair problem: Given a set A of N regular expressions of length $2^{(\log N)^{o(1)}}$ and a set B of N strings of length $2^{(\log N)^{o(1)}}$, find $(a, b) \in A \times B$ with maximum Hamming Similarity⁴.
- The Max-LCST-Pair problem: Given two sets A, B of N bounded-degree trees with size $2^{(\log N)^{o(1)}}$, find a pair $(a, b) \in A \times B$ such that a and b 's have the largest common subtree.
- The Max-TropSim problem: Given two sets A, B of N binary tensors with size $2^{(\log N)^{o(1)}}$, find the pair with maximum Tropical Similarity⁵.

⁴Hamming Similarity between two strings are defined as the fraction of positions that they are equal, while the hamming similarity between a regular expression a and a string b is the maximum of the hamming similarity between z and b where z is in the language of a .

⁵see Definition 2.8 for a formal definition.

Our main theorem shows equivalence between exact and approximation versions of the above problems. In fact, we show that all these problems, together with the following two closely related decision problems and a generic satisfying pair problem, are *equivalent* under *near-linear* time reductions. See Theorem 3.1 for a formal statement of the equivalence class.

- The RegExp-String-Pair problem: Given a set A of N regular expressions of length $2^{(\log N)^{o(1)}}$ and a set B of N strings of length $2^{(\log N)^{o(1)}}$, is there a pair $(a, b) \in A \times B$ such that b matches a ?
- The Subtree-Isomorphism-Pair problem: Given two sets A, B of N bounded-degree trees with size $2^{(\log N)^{o(1)}}$, is there a pair $(a, b) \in A \times B$ such that a is isomorphic to a subtree of b ?
- The BP-Satisfying-Pair problem: Given a branching program⁶ P of size $2^{(\log N)^{o(1)}}$ and two sets A, B of N strings, is there a pair $(a, b) \in A \times B$ making P accepts the input (a, b) ?

We will refer to this set of problems as BP-Pair-Class.

Remark 1.4. *Subtree-Isomorphism-Pair and Max-LCST-Pair may seem artificial, but they are nice intermediate problems for showing hardness of the closely related problems Subtree Isomorphism and Longest Common Subtree, which are extensively studied natural problems (see Section 1.5 and Section 1.9.4).*

Equivalence in the Data Structure Setting

These pair problems are interesting as they are natural *off-line* versions of closely related data structure problem, which are highly relevant in the practice [Tho68, Mye92, BT09, MM89, WMM95, KM95, MOG98, Nav04, BR13]. Therefore, a lower bound on the time complexity of these pair problems directly implies a lower bound for corresponding data structure problems (see Theorem 9.4). In the next section, we will show under some conjecture which is much more plausible than SETH, these problems requires essentially quadratic time.

Our equivalence continues to hold in the data structure version, in particular, for the following data structure problems, any algorithm for one of them implies an algorithm for all of them with essentially the same preprocessing time/space and query time (up to a factor of $N^{o(1)}$). See Section 9 for the details.

- $\text{NNS}_{\text{LCS}}^7$: Preprocess a database \mathcal{D} of N strings of length $D = 2^{(\log N)^{o(1)}}$, and then for each query string x , find $y \in \mathcal{D}$ maximizing $\text{LCS}(x, y)$.
- Approx. NNS_{LCS} : Find $y \in \mathcal{D}$ s.t. $\text{LCS}(x, y)$ is a $2^{(\log D)^{1-\Omega(1)}}$ approximation to the maximum value.
- Regular Expression Query: Preprocess a database \mathcal{D} of N strings of length $D = 2^{(\log N)^{o(1)}}$, and then for each query regular expression y , find an $x \in \mathcal{D}$ matching y .
- Approximate Regular Expression Query: For a query expression y , distinguish between⁸: (1) there is an $x \in \mathcal{D}$ matching y ; and (2) for all $x \in \mathcal{D}$, the Hamming distance between x and all $z \in L(y)$ is at least $(1 - o(1)) \cdot D$, where $L(y)$ is the set of all strings matched by y .

That is, a non-trivial data structure for finding *approximate* nearest point with LCS metric would imply a non-trivial data structure for answering regular expression query! The latter one is supported by most modern database systems such as MySQL, Oracle Database, Microsoft SQL etc., but all of them implement it by simply using full table scan for the most general case.

⁶see Definition 2.1 for a formal definition

⁷already discussed in Section 1.1

⁸behavior can be arbitrary when neither of the two cases hold

LCS is the Hardest Distance Function for Approximate NNS. In fact, our results also suggest in a formal sense that LCS is the *hardest* distance function for approximate Nearest Neighbor Search. We show that for all distance function dist which is computable in poly-logarithmic space⁹, exact NNS for dist can be reduced to approximate NNS_{LCS} .

Theorem 1.5 (Informal). *For a distance function dist that is computable in poly-logarithmic space, exact NNS for dist can be reduced to $2^{(\log D)^{1-\Omega(1)}}$ -approximate NNS_{LCS} in near-linear time.*

1.4 Weaker Complexity Assumptions for Approximation Hardness

An important goal in the study of fine-grained complexity is to find more plausible conjectures, under which to base hardness. For example, although SETH is based on the historically unsuccessful attempts on finding better algorithms for k -SAT, there is no consensus on its validity (see, e.g. [Wil16, Wil18b]).

This concern has been addressed in various ways. For example, in [AVY15], the authors prove hardness for several problems, basing on *at least one of* the SETH, the APSP conjecture, or the 3-SUM Conjecture being true. In [AHVW16], Abboud et al. introduce a hierarchy of \mathcal{C} -SETH assumptions: the \mathcal{C} -SETH asserts that there is no $2^{(1-\varepsilon)n}$ time satisfiability algorithm for circuits from \mathcal{C} .¹⁰ They show that the quadratic time hardness of Edit-Distance, LCS and other related sequence alignment problems can be based on the much weaker and much more plausible assumption NC-SETH. However, this has not been shown for approximation version of fine-grained problems.

In this work, we show that all problems in BP-Pair-Class require essentially quadratic time under NC-SETH. Indeed, our hardness results are based on a weaker assumption which we call $2^{n^{o(1)}}$ -size BP-SETH:¹¹

Hypothesis 1.6 ($2^{n^{o(1)}}$ -size BP-SETH). *The satisfiability of a given $2^{n^{o(1)}}$ -size non-deterministic branching program cannot be solved in $O(2^{(1-\delta)n})$ time for any $\delta > 0$.*

Theorem 1.7. *All problems in BP-Pair-Class require $N^{2-o(1)}$ time if we assume $2^{n^{o(1)}}$ -size BP-SETH.*

Note that $2^{n^{o(1)}}$ -size BP-SETH is even weaker than $n^{o(1)}$ -depth circuit SETH: satisfiability for $n^{o(1)}$ -depth bounded fan-in circuits cannot be solved in $O(2^{(1-\delta)n})$ time for any $\delta > 0$. This is because by Barrington's Theorem [Bar89], $n^{o(1)}$ -depth bounded fan-in circuits can be simulated by branching programs of size $2^{n^{o(1)}}$.

It is worthwhile to compare with [ARW17]. It is shown in [ARW17] that assuming SETH, a $2^{(\log N)^{1-o(1)}}$ -approximation to Closest-LCS-Pair (Closest-RegExp-String-Pair) requires $N^{2-o(1)}$ time for $D = N^{o(1)}$. (The $2^{(\log N)^{1-o(1)}}$ factor is later improved to $N^{o(1)}$ in [Rub18, Che18].)

Although our results here are quantitatively worse, it is “qualitatively” better in many ways: (1) the results in [ARW17] is based on SETH, while our hardness results are based on the assumptions in Theorem 1.7, which are much more plausible than SETH; (2) we in fact have established an equivalence between Closest-LCS-Pair and its approximation version, which seems not possible with the techniques in [ARW17]; (3) our framework allows us to show that even a tiny improvement on the running time would have important algorithmic and circuit lower bound consequences (see Theorem 1.10), which again seems not possible with the techniques in [ARW17].

⁹Which is true for almost all nature distance functions. For example, edit distance and LCS can be computed in NL, thus in $(\log^2 N)$ space by Savitch's Theorem [Sav70].

¹⁰In this way, the original SETH assert that there is no $2^{(1-\varepsilon)n}$ time algorithm for satisfiability of CNF with arbitrary constant bottom fan-in.

¹¹Indeed, results in [AHVW16] are also based on a conjecture about branching programs, which can be seen as $O(1)$ -width and $2^{o(n)}$ -length BP-SETH or $2^{o(\sqrt{n})}$ -size BP-SETH using the terminology in Hypothesis 1.6.

1.5 BP-Pair-Class Hard Problems

We also identify a set of other problems which are at least as hard as any problem in BP-Pair-Class, but not necessarily in it. We say these problems are BP-Pair-Hard.

Theorem 1.8 (BP-Pair-Hard Problems). *There are near-linear time reductions from all the problems in BP-Pair-Class to any of the following problems:*

1. (Subtree Isomorphism) Given two trees a, b of size at most N , determine whether a is isomorphic to a subtree of b (even if restricted to the case of binary rooted trees);
2. (Largest Common Subtree) Given two trees a, b of size at most N , compute the exact value or a $2^{(\log N)^{o(1)}}$ -approximation of the size of the largest common subtree of a and b (even if restricted to the case of binary rooted trees);
3. (Regular Expression Membership Testing) Given a regular expression a of length M and a string b of length N , determine whether b is in the language of a ;

Corollary 1.9. *All the above BP-Pair-Hard problems require $N^{2-o(1)}$ time (or $(NM)^{1-o(1)}$ time for Regular Expression Membership Testing) under the same assumption as in Theorem 1.7.*

We remark that both Subtree Isomorphism and Largest Common Subtree are studied in [ABH⁺16]. In particular, they showed that Subtree Isomorphism and Largest Common Subtree require quadratic-time under SETH, even for binary rooted trees.

Our results improve theirs in many ways: (1) for Subtree Isomorphism, we establish the same quadratic time hardness, with a much safer conjecture; (2) for Largest Common Subtree, we not only put its hardness under a better conjecture, but also show that even a $2^{(\log N)^{o(1)}}$ -approximation would be hard; (3) for both of these problems, we demonstrate that even a tiny improvement on the running time would have interesting algorithmic and circuit lower bound consequences (see Theorem 1.11).

[BGL17] (which builds on [BI16]) classified the running time of constant-depth regular expression membership testing. In particular, they showed a large class of regular expression testing requires quadratic-time, under SETH. Our results are incomparable with theirs, as our hard instances may have unbounded depth regular expressions. On the bright side, our hardness results rely on a much safer conjecture, and we show interesting consequences even for a tiny improvement of the running time.

1.6 The Consequence of “shaving-logs” for Approximation Algorithms

There has been a large number of works focusing on “shaving logs” of the running time of fundamental problems [ADKF70, BW12, Cha15, Yu15] (see also a talk by Chan [Cha13], named “The Art of Shaving Logs”). In a recent exciting algorithmic work by Williams [Wil14a], the author shaves “all the logs” on the running time of APSP, by getting an $n^3/2^{\Theta(\sqrt{\log n})}$ time algorithm.

However, the best exact algorithms for LCS and Edit distance [MP80, Gra16] remain $O(n^2/\log^2 n)$, which calls for an explanation. An interesting feature of [AHVW16] is that their results show that even shaving logs on LCS or Edit Distance would be very hard. In particular, they prove that an $n^2/\log^{\omega(1)} n$ time algorithm for either of them would imply a $2^n/n^{\omega(1)}$ time algorithm for polynomial-size formula satisfiability, which is much better than the current state of arts [San10, Tal15]. Such an algorithm would also imply that NEXP is not contained in non-uniform NC¹, thereby solving a notorious longstanding open question in complexity theory.

The “shaving logs barrier” only has been studied for a few problems. It was not clear whether we can get the same barriers for some *approximation problems*.

In this work we show that slightly improved algorithms (such as shaving all the logs) for any BP-Pair-Class or BP-Pair-Hard problems, would imply *circuit lower bounds* which are notoriously hard to prove. This extends all the results of [AHVW16] to approximation problems.

Theorem 1.10. *If there is an $O(N^2 \text{poly}(D)/2^{(\log \log N)^3})$ or $O(N^2/(\log N)^{\omega(1)})$ time deterministic algorithm for any decision, exact value or polylog(D)-approximation problems in BP-Pair-Class, where D is the maximum length (or size) of elements in sets, then the following holds:*

- $\text{NTIME}[2^{O(n)}]$ is not contained in non-uniform NC^1 and
- *Formula-SAT* with $n^{\omega(1)}$ size can be solved in $2^n/n^{\omega(1)}$ time.

Theorem 1.11. *If there is a deterministic algorithm for any decision, exact value or polylog(N)-approximation problems among BP-Pair-Hard problems listed in Theorem 1.8 running in $O(N^2/2^{\omega(\log \log N)^3})$ time (or $O(NM/2^{\omega(\log \log NM)^3})$ time for Regular Expression Membership Testing), then the same consequences in Theorem 1.10 follows.*

1.7 Circuit Lower Bound Consequence for Improving Approximation Algorithms for P Time Problems

Finally, we significantly improve the results from [AR18], by showing much stronger circuit lower bound consequences for deterministic approximation algorithms to LCS.

Theorem 1.12. *The following holds for deterministic approximation to LCS:*

1. A $2^{(\log N)^{1-\Omega(1)}}$ -approximation algorithm in $N^{2-\delta}$ time for some constant $\delta > 0$ implies that E^{NP} has no $n^{o(1)}$ -depth bounded fan-in circuits;
2. A $2^{o(\log N/(\log \log N)^2)}$ -approximation algorithm in $N^{2-\delta}$ time for some constant $\delta > 0$ implies that $\text{NTIME}[2^{O(n)}]$ is not contained in non-uniform NC^1 ;
3. An $O(\text{polylog}(N))$ -approximation algorithm in $N^2/2^{\omega(\log \log N)^3}$ time implies that $\text{NTIME}[2^{O(n)}]$ is not contained in non-uniform NC^1 .

In comparison with [AR18], they show that an $O(N^{2-\varepsilon})$ time algorithm for constant factor deterministic approximation algorithm to LCS would imply that E^{NP} does not have non-uniform linear-size NC^1 circuits or VSP circuits. Our results here generalize theirs in all aspects: (1) we show that a much stronger lower bound consequence would follow from even a sub-quadratic time $2^{(\log N)^{1-\Omega(1)}}$ -approximation algorithm; (2) we also show that a modestly stronger lower bound would follow even from a quasi-polylogarithmic improvement over the quadratic time, for approximate LCS.

More generally, following a similar argument to [AHVW16], we can show that truly-subquadratic time algorithms for these BP-Pair-Class or BP-Pair-Hard problems would imply strong circuit lower bounds against E^{NP} .

Corollary 1.13. *If any of the problems listed in Theorem 3.1 and Theorem 1.8 admits an $N^{2-\varepsilon}$ time algorithm (or $(NM)^{1-\varepsilon}$ time algorithm for Regular Expression Membership Testing) for some $\varepsilon > 0$, then E^{NP} does not have:*

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

1.8 Discussion and Open Problems

Here we discuss some open problems arising from our work.

Find More Members for BP-Pair-Class

One immediate question is to find more natural quadratic-time problems belonging to BP-Pair-Class:

Open Question 1. *Find more natural problems which belong to BP-Pair-Class.*

It could be helpful to revisit all SETH-hard problems to see whether they can simulate BP-Satisfying-Pair. In particular, one may ask whether the Orthogonal Vectors problem (OV), the most studied problem in fine-grained complexity, belongs to this equivalence class:

Open Question 2. *Does OV belong to BP-Pair-Class?*

If it does, then it would open up the possibility that perhaps all SETH-hard quadratic-time problems are equivalent. However, some evidence suggests that the answer may be negative, as OV seems to be much easier than problems in BP-Pair-Class:

- **The Inner Function in OV is Much Weaker.** When viewing as a Satisfying-Pair problem, the inner function in OV is just a simple *Set-Disjointness*, which seems incapable of simulating generic low-space computation.
- **There are Non-trivial Algorithms for OV.** We know that for OV with N vectors of length $D = c \log n$, there are algorithms with running time $N^{2-1/O(\log c)}$ [AWY15, CW16]. This type of non-trivial speed up seems quite unlikely (or at least much harder to obtain) for problems in BP-Pair-Class (see Theorem 1.10).

It would be interesting to show that OV and BP-Satisfying-Pair are not equivalent under certain plausible conjectures, perhaps ideas from [CGI⁺16] could help.

Quasi-Polynomial Blow Up of the Dimension

In the reductions between our BP-Pair-Class problems, we get a quasi-polynomial blowup on the dimensions: that is, a problem with element size (vector dimension, string length or tree size) D is transformed into another problem with element size $2^{\text{polylog}(D)}$. This is the main reason that we have to restrict the element size to be small, i.e., $D = 2^{(\log N)^{o(1)}}$.

This technical subtlety arises from the polynomial blow-up in the $\text{IP} = \text{PSPACE}$ proof: given a language in $\text{NSPACE}[S]$, it is first transformed into a TQBF instance of size $O(S^2)$, which is proved by an $\tilde{O}(S^4)$ time IP protocols, using arithmetization.

Applying that into our setting, an NL computation on two strings of length D (like LCS), is transformed into an $\tilde{O}(\log^4 D)$ time IP communication protocol, which is then embedded into an approximate problem with at least $2^{\tilde{O}(\log^4 D)}$ dimensions (say approximate LCS).

However, if we have an $O(\log D)$ time IP communication protocol for NL. The new dimension would be $2^{O(\log D)} = D^{O(1)}$, only a polynomial blow up. Which motivates the following interesting question:

Open Question 3. *Is there an $O(\log D)$ time IP communication protocol for every problems in NL?*

A positive resolution of the above question would also tighten several parameters in many of our results. For example, in Theorem 1.7, $n^{o(1)}$ -depth circuit SETH could be replaced by $o(n)$ -depth circuit SETH, and Theorem 1.10, Theorem 1.11, Theorem 1.8 and Corollary 1.13 would also have improved parameters.

It is worth noting that IP communication lower bounds are extremely hard to prove—proving a non-trivial lower bound for AM communication protocols is already a long-standing open question [Lok01, GPW16, GPW18]. Hence, resolving Open Question 3 negatively could be hard.

1.9 Related Works

1.9.1 Hardness of Approximation in P

In [ARW17] the Distributed PCP framework is introduced, which is utilized and generalized by several follow up works. Using Algebraic Geometry codes, in a recent work, [Rub18] obtains a better MA protocol for Set-Disjointness, improving the efficiency of the distributed PCP construction, and shows quadratic-time hardness for $(1 + o(1))$ -approximation to Bichromatic Closest Pair and several other related problems.

Building on the technique of [Rub18], [Che18] obtains a characterization of what multiplicative/additive approximation ratios to Maximum Inner Product can be computed in sub-quadratic time. He also shows a connection between BQP communication protocol for Set-Disjointness and conditional lower bound for Maximum Inner Product with $\{-1, 1\}$ -valued vectors.

[KLM18] generalizes the distributed PCP framework by considering multi-party communication protocols, and derive inapproximability results for k -Dominating Set under various assumptions. In particular, using the techniques of [Rub18], they prove that under SETH, k -Dominating Set has no $(\log n)^{1/\text{poly}(k, e(\varepsilon))}$ approximation in $n^{k-\varepsilon}$ time¹².

[AB17] takes a different approach, which makes use of the connection between weak derandomization and circuit lower bound [Wil13, BV14]. They show that, under a certain plausible complexity assumption, LCS does not have a *deterministic* $(1 + o(1))$ -approximation in $n^{2-\varepsilon}$ time. They also establish a connection with circuit lower bounds and prove that such a *deterministic* algorithm implies E^{NP} does not have non-uniform linear-size Valiant Series Parallel circuits. In [AR18], the circuit lower bound connection is improved to that any constant factor deterministic approximation for LCS in $n^{2-\varepsilon}$ time implies that E^{NP} does not have non-uniform linear-size NC¹ circuits. See [ARW17] for more related results in hardness of approximation in P.

1.9.2 Equivalence Classes in P

A partial list of the APSP equivalence class [VW10, BDT16, AGV15, LVW18] includes: Negative Triangle, Triangle listing, Shortest Cycle, 2nd Shortest Path, Max Subarray, Graph Median, Graph Radius, Wiener Index (see [Vas18] for more details).

In [GIKW17], it is shown that “medium-dimensional” OV (i.e., OV with $n^{o(1)}$ dimensions) is equivalent to High-dimension Sparse OV, High-dimension 2-Set Cover, and High-dimension Sperner Family. It is also proved that for every $(k + 1)$ -quantifier first-order property, its model-checking problem can be reduced to Sparse k -OV.

In [CMWW17], the authors introduce an equivalence class for $(\min, +)$ -convolution, including some variants of classical knapsack problem and problems related to subadditive sequences.

1.9.3 Hardness for Shaving Logs

In [AHVW16], it is shown that an $n^2 / \log^{\omega(1)}$ time algorithm for LCS would imply that the Formula-SAT have a $2^n / n^{\omega(1)}$ time algorithm. The construction is later tightened in [AB18], which shows that an $n^2 / \log^{7+\varepsilon} n$

¹²where $e : \mathbb{R}^+ \rightarrow \mathbb{N}$ is some function

time algorithm for any of LCS, regular expression pattern matching or Fréchet distance is already enough to imply new algorithm for Formula-SAT.

1.9.4 Related Works for Specific Problems

Longest Common Subsequence. LCS is a very basic problem in computer science and has been studied for decades [CKK72, BHR98, dM99, CIPR01]. In recent years, a series of works [BI15, ABV15, BK15, AHVW16, AR18] have shown different evidences that LCS may not have truly sub-quadratic time algorithm, even for approximation.

Subtree Isomorphism and Largest Common Subtree. Subtree Isomorphism has been studied in [Mat68, Mat78, Lin83, Rey77, Chu87, LK89, GKMS90, ST99]. Largest Common Subtree is an NP-hard problem when the number of trees is not fixed, and has been studied in [KMY95, AH00, ATMT15]. For (rooted or unrooted) bounded-degree trees, both the two problems can be solved in $O(N^2)$ time, and the fastest algorithm for Subtree Isomorphism runs in $O(N^2/\log N)$ time [Lin83, ST99]. In [ABH⁺16], these two problems are shown to be SETH-hard.

Regular Expression. The found of $O(NM)$ algorithm for regular expression matching and membership testing in [Tho68] is a big sucess in 70s, but after that no algorithm has been found to improve it to truly sub-quadratic time [Mye92, BT09]. Related works about the hardness of exact regular expression matching or membership testing include [BI16, BGL17]. There are many works on approximate regular expression matching in different formulations [MM89, WMM95, KM95, MOG98, Nav04, BR13], and its hardness has been analyzed in [ARW17].

Organization of this Paper

In Section 2, we introduce the needed preliminaries, as well as the formal definitions of the problems we studied. In Section 3, we outline the structure of all reductions for our BP-Pair-Class and BP-Pair-Hard problems (Theorem 3.1 and Theorem 1.8). For ease of presentation, these reductions are presented from Section 4 to Section 8. In Section 10, we show that tiny improvements on the running time of BP-Pair-Class or BP-Pair-Hard problems would have important algorithmic and circuit lower bound consequences. In Section 11, we establish the consequences of faster deterministic approximation algorithms to LCS.

2 Preliminaries

In this section, we define the SAT problem for branching program (BP-SAT), and then we introduce the formal definitions for each problem in BP-Pair-Class and BP-Pair-Hard.

Definition 2.1 (Branching Program). A (nondeterministic) *Branching Program* (BP) on n boolean inputs x_1, \dots, x_n is defined as a layered directed graph with T layers L_1, \dots, L_T . Each layer L_i contains W nodes. Except the last layer, every node in L_i ($1 \leq i < T$) is associated with the same variable $x_{f(i)}$ for some $f(i) \in [n]$. T and W are called the *length* and *width* of the BP.

For every two adjacent layers L_i and L_{i+1} , there are edges between nodes in L_i to nodes in L_{i+1} , and each edge is marked with either 0 or 1. The *size* of a BP is defined as the total number of edges $O(W^2T)$.

For $1 \leq i \leq T, 1 \leq j \leq W$, the j -th node in the i -th layer L_i is labeled as (i, j) . $u_{\text{start}} = (1, 1)$ is the starting node, and $u_{\text{acc}} = (T, 1)$ is the accepting node. A BP accepts an input x iff there is a path from the starting node to the accepting node consisting of only the edges marked with the value of the variable associated with its starting endpoint.

Definition 2.2 (BP-SAT). Given a branching program P on n boolean inputs, the BP-SAT problem asks whether there is an input making P accept.

Like the relationship between k -SAT and Orthogonal Vectors (OV), we can define BP-Satisfying-Pair problem as the counterpart of BP-SAT in the P world. BP-Satisfying-Pair can be trivially solved in $O(N^2 \cdot \text{poly}(W, T))$ time, and a faster algorithm for BP-Satisfying-Pair running in $O(N^{2-\varepsilon})$ time implies a faster algorithm for BP-SAT running in $O(2^{(1-\varepsilon/2)n})$ time.

Definition 2.3 (BP-Satisfying-Pair). Given a branching program P on n boolean inputs (assume n is even) and two sets of N strings $A, B \subseteq \{0, 1\}^{n/2}$, the BP-Satisfying-Pair problem asks whether there is a pair $a \in A, b \in B$ such that P accepts the concatenation of a and b .

Throughout this paper, unless otherwise stated, we use BP-Satisfying-Pair to denote the BP-Satisfying-Pair problem on branching program of size $2^{(\log N)^{o(1)}}$ for convenience.

2.1 Satisfying Pair and Best Pair Problems

Satisfying Pair Problems. Note that problems like *Orthogonal Vectors* are in the form of deciding whether there is a “satisfying pair”. In general, we can define the \mathcal{A} -Satisfying-Pair problem, where \mathcal{A} is an arbitrary decision problem on two input strings x, y :

Definition 2.4 (\mathcal{A} -Satisfying-Pair). Given two sets A, B of N strings, the \mathcal{A} -Satisfying-Pair problem asks whether there is a pair of $a \in A, b \in B$ such that (a, b) is an Yes-instance of \mathcal{A} .

In this work, we study a series of \mathcal{A} -Satisfying-Pair problems, including OAPT, RegExp-String-Pair and Subtree-Isomorphism-Pair, which will be formally defined in later subsections.

Best Pair Problems. For an optimization problem \mathcal{A} on two input strings x, y , we can define the Max- \mathcal{A} -Pair and the Min- \mathcal{A} -Pair problems:

Definition 2.5 (Max- \mathcal{A} -Pair / Min- \mathcal{A} -Pair). Given two sets A, B of N strings, the Max- \mathcal{A} -Pair (or Min- \mathcal{A} -Pair) problem asks to compute the maximum value (or minimum value) of the result of problem \mathcal{A} on input (a, b) .

In this work, we study a series of Max- \mathcal{A} -Pair / Min- \mathcal{A} -Pair problems, including Max-TropSim, Min-TropSim, Closest-LCS-Pair, Furthest-LCS-Pair, Closest-RegExp-String-Pair, Max-LCST-Pair and Min-LCST-Pair, which will be formally defined in later subsections.

Note that both satisfying pair problems and best pair problems contain two sets A, B in the input. Without additional explanation, we use N to denote the set size, and D to denote the maximum element size in sets.

2.2 Two Tensor Problems

We introduce two kinds of tensor problems: the *Orthogonal Alternating Product Tensors* problem (OAPT) and the *Max / Min Tropical Similarity* problem (Max-TropSim / Min-TropSim). The former one is an \mathcal{A} -Satisfying-Pair problem, which helps us to prove hardness for decision problems; the latter one is a Max- \mathcal{A} -Pair / Min- \mathcal{A} -Pair problem, which helps us proving hardness of approximation for optimization problems.

First we define OAPT. OAPT implicitly appears in the reduction from BP-SAT to LCS in [AHVW16] as an intermediate problem. In our reductions, OAPT appears naturally, and we show that BP-Satisfying-Pair and OAPT of certain size are equivalent under near-linear time reduction in Section 5.

Definition 2.6 (OAPT). Let t be an even number and $d_1 = d_2 = \dots = d_t = 2$. The *Alternating Product* $p_{\text{alt}}(u, v)$ of two tensors $u, v \in \{0, 1\}^{d_1 \times \dots \times d_t}$ is defined as an alternating sequence of logical operators \wedge and \vee applied to the coordinatewise product of u and v :

$$p_{\text{alt}}(u, v) = \bigwedge_{i_1 \in [d_1]} \left[\bigvee_{i_2 \in [d_2]} \left(\bigwedge_{i_3 \in [d_3]} \left[\dots \bigvee_{i_t \in [d_t]} (u_i \wedge v_i) \dots \right] \right) \right]. \quad (1)$$

Given two sets of N tensors $A, B \subseteq \{0, 1\}^{d_1 \times \dots \times d_t}$, the Orthogonal Alternating Product Tensors (OAPT) problem asks whether there is a pair $a \in A, b \in B$ such that the Alternating Product $p_{\text{alt}}(a, b) = 0$.

Restricted OAPT is a restricted version of OAPT. We mainly use this restricted version in our analysis.

Definition 2.7 (Restricted OAPT). We say that a tensor $x \in \{0, 1\}^{d_1 \times \dots \times d_t}$ is \wedge -invariant if the value of $x_{i_1 \dots i_t}$ does not depend on $i_1, i_3, i_5, \dots, i_{t-1}$. The Restricted OAPT problem is a restricted version of OAPT, where one set of A, B contains only \wedge -invariant tensors.

For proving our hardness of approximation results, we further define the Max-TropSim and Min-TropSim problems. The Max-TropSim problem was firstly proposed by Abboud and Rubinstein in [AR18] under the name *Tropical Tensors* in their study of LCS.

Definition 2.8 (Max-TropSim / Min-TropSim). Let t be an even number and $d_1 = d_2 = \dots = d_t = 2$. The *Tropical Similarity* score $s(u, v)$ of two tensors $u, v \in \{0, 1\}^{d_1 \times \dots \times d_t}$ is defined as an alternating sequence of operators \mathbb{E} and max applied to the coordinatewise product of u and v :

$$s(u, v) = \mathbb{E}_{i_1 \in [d_1]} \left[\max_{i_2 \in [d_2]} \left\{ \mathbb{E}_{i_3 \in [d_3]} \left[\dots \max_{i_t \in [d_t]} \{u_i \cdot v_i\} \dots \right] \right\} \right]. \quad (2)$$

Given two sets of N tensors $A, B \subseteq \{0, 1\}^{d_1 \times \dots \times d_t}$, the Max-TropSim problem asks to compute the maximum Tropical Similarity $s(a, b)$ among all pairs of $(a, b) \in A \times B$, while the Min-TropSim problem asks to compute the minimum Tropical Similarity $s(a, b)$ among all pairs of $(a, b) \in A \times B$.

Restricted OAPT is a restricted version of OAPT. We mainly use this restricted version in our analysis.

Definition 2.9 (Restricted Max-TropSim / Restricted Min-TropSim). We say that a tensor $x \in \{0, 1\}^{d_1 \times \dots \times d_t}$ is max-invariant if the value of $x_{i_1 \dots i_t}$ does not depend on $i_2, i_4, i_6, \dots, i_t$. The Restricted Max-TropSim / Restricted Min-TropSim problem is a restricted version of Max-TropSim/ Min-TropSim, where one set of A, B contains only \wedge -invariant tensors.

Like in [AR18], we also define the following approximation variants of the Max-TropSim and Min-TropSim.

Definition 2.10 (ε -Gap-Max-TropSim). Let t be an even number and $d_1 = d_2 = \dots = d_t = 2$. Given two sets of N tensors $A, B \in \{0, 1\}^{d_1 \times \dots \times d_t}$ of size $D = 2^t$, distinguish between the following:

- **Completeness:** There is a pair of $(a, b) \in A \times B$ with a perfect Tropical Similarity $s(a, b) = 1$;
- **Soundness:** Every pair has low Tropical Similarity score, $s(a, b) < \varepsilon$.

Here ε is a threshold value that may depend on N and D . Restricted ε -Gap-Max-TropSim is defined similarly.

Definition 2.11 (ε -Gap-Min-TropSim). Let t be an even number and $d_1 = d_2 = \dots = d_t = 2$. Given two sets of N tensors $A, B \in \{0, 1\}^{d_1 \times \dots \times d_t}$ of size $D = 2^t$, distinguish between the following:

- **Completeness:** There is a pair of $(a, b) \in A \times B$ with a low Tropical Similarity $s(a, b) < \varepsilon$;
- **Soundness:** Every pair has perfect Tropical Similarity score, $s(a, b) = 1$.

Here ε is a threshold value that may depend on N and D . Restricted ε -Gap-Min-TropSim is defined similarly.

In this paper we use the proof idea for IP = PSPACE to show that BP-Satisfying-Pair can be reduced to ε -Gap-Max-TropSim / ε -Gap-Min-TropSim of certain size in near-linear time. The proof is in Section 5.

2.3 Longest Common Subsequence

We study the hardness of *Longest Common Subsequence* (LCS) and its pair version in this paper.

Definition 2.12 (LCS). Given two strings a, b of length N over alphabet Σ , the LCS problem asks to compute the length of the longest sequence that appears in both a and b as a subsequence.

Definition 2.13 (Closest-LCS-Pair / Furthest-LCS-Pair). Given two sets of N strings A, B , the Closest-LCS-Pair (or Furthest-LCS-Pair) problem asks to compute the maximum (or minimum) length of the longest common subsequence among all pairs of $(a, b) \in A \times B$.

2.4 Subtree Isomorphism and Largest Common Subtrees

We study the hardness for the following two problems on trees:

Definition 2.14. (Subtree Isomorphism) Given two trees G and H , the *Subtree Isomorphism* problem asks whether G is isomorphic to a subtree of H , i.e., can G and H be isomorphic after removing some nodes and edges from H .

Definition 2.15. (Largest Common Subtree) Given two trees G and H , the *Largest Common Subtree* problem asks to compute the size of the largest tree that is isomorphic to both a subtree of G and a subtree of H .

In this paper, we focus on the case of unordered trees with bounded degrees. We are interested in both rooted and unrooted trees. Here “rooted” means that the root of G must be mapped to the root of H in the isomorphism.

The pair versions of these two problems are defined as follows:

Definition 2.16 (Subtree-Isomorphism-Pair). Given two sets of N trees A, B , the Subtree-Isomorphism-Pair problem asks whether there is a pair of trees $(a, b) \in A \times B$ such that the tree a is isomorphic to a subtree of the tree b .

Definition 2.17 (Max-LCST-Pair / Min-LCST-Pair). Given two sets of N trees A, B , the Max-LCST-Pair (or Min-LCST-Pair) problem asks to compute the maximum (or minimum) size of the largest common subtrees among all pairs of $(a, b) \in A \times B$.

2.5 Regular Expression Membership Testing

We study the hardness of testing membership for regular expression. A regular expression over an alphabet set Σ and an operator set $O = \{\circ, |, +, *\}$ is defined in a inductive way: (1) Every $a \in \Sigma$ is a regular expression; (2) All of $[R | S]$, $R \circ S$, R , $[R]^+$ are regular expressions if R and S are regular expressions. A regular expression p determines a language $L(p)$ over alphabet Σ . Specifically, for any regular expressions R, S and any $a \in \Sigma$, we have: $L(a) = \{a\}$; $L([R | S]) = L(R) \cup L(S)$; $L(R \circ S) = \{uv \mid u \in L(R), v \in L(S)\}$; $L([R]^+) = \bigcup_{k \geq 1} \{u_1 u_2 \cdots u_k \mid u_1, \dots, u_k \in L(R)\}$; and $L([R]^*) = L(R^+) \cup \{\varepsilon\}$, where ε denotes the empty string. The concatenation operator \circ and unnecessary parenthesis is often omitted if the meaning is clear from the context.

In this paper, we study the *Regular Expression Membership Testing* problem, which is defined as follows:

Definition 2.18 (Exact Regular Expression Membership Testing). Given a regular expression p of length M and a string t of length N over alphabet Σ , the *Exact Regular Expression Membership Testing* problem asks whether t is in the language $L(p)$ of p .

And its pair version is defined as follows:

Definition 2.19 (RegExp-String-Pair). Given a set A of regular expressions of length $O(\text{poly}(D))$ and a set B of N strings of length D , the RegExp-String-Pair problem asks to determine whether there is a pair (a, b) such that b is in the language $L(a)$ of a .

In [ARW17], Abboud, Rubinstein and Williams studied a problem called RegExp Closest Pair and showed that it is SETH-hard using their distributed PCP framework. In this work, we study a slightly different problem.

Definition 2.20 (Closest-RegExp-String-Pair). For two strings x, y of the same length n , the Hamming Similarity $\text{HamSim}(x, y)$ between x and y is defined as the fraction of positions for which the corresponding symbols are equal, i.e.,

$$\text{HamSim}(x, y) = \frac{1}{n} \sum_{i=1}^n \mathbb{1}_{[x_i=y_i]}.$$

Given a set A of N regular expressions of length $O(\text{poly}(D))$ and a set of N strings of length D , the Closest-RegExp-String-Pair problem asks to compute the maximum Hamming Similarity among all pairs of (x, b) satisfying $x \in L(a)$ is a string of length D for some $a \in A$ and $b \in B$.

3 BP-Pair-Class and an Outline of all Reductions

In this section, we first state the equivalence class formally, and outline how it is proved in this paper.

Theorem 3.1 (BP-Pair-Class). *There are near linear-time¹³ reductions between all pairs of following problems:*

1. *(Exact or Approximate Closest-LCS-Pair / Furthest-LCS-Pair) Given two sets A, B of N strings of length $D = 2^{(\log N)^{o(1)}}$, compute the exact value or a $2^{(\log D)^{1-\Omega(1)}}$ -approximation of the maximum (minimum) LCS among $(a, b) \in A \times B$;*
2. *(Approximate Closest-RegExp-String-Pair) Given a set A of N regular expressions of length $2^{(\log N)^{o(1)}}$ and a set of N strings of length $D = 2^{(\log N)^{o(1)}}$, distinguish between the case that there is a pair $(a, b) \in A \times B$ such that $b \in L(a)$ (the language of a), and the case that every string in B has Hamming Similarity $< 2^{-(\log D)^{1-\Omega(1)}}$ from every string of length D in $\bigcup_{a \in A} L(a)$.*
3. *(Exact or Approximate Max-LCST-Pair / Min-LCST-Pair) Given two sets A, B of N bounded-degree trees with size at most $D = 2^{(\log N)^{o(1)}}$, compute the exact value or a $2^{(\log D)^{1-\Omega(1)}}$ -approximation of the maximum (minimum) size of largest common subtree among $(a, b) \in A \times B$;*
4. *(Exact or Approximate Max-TropSim / Min-TropSim) Given two sets A, B of N binary tensors with size $D = 2^{(\log N)^{o(1)}}$, compute the exact value or a $2^{(\log D)^{1-\Omega(1)}}$ -approximation of the maximum (minimum) Tropical Similarity among $(a, b) \in A \times B$;*
5. *Orthogonal-Alternating-Product-Tensors (OAPT): Given two sets A, B of N binary tensors with size $2^{(\log N)^{o(1)}}$, is there a pair $(a, b) \in A \times B$ with Alternating Product¹⁴ 0?*
6. *BP-Satisfying-Pair;*
7. *RegExp-String-Pair;*

¹³Throughout this paper, we use near-linear time to denote the running time of $N^{1+o(1)}$.

¹⁴see Definition 2.6 for a formal definition

8. Subtree-Isomorphism-Pair;

Remark 3.2. A technical remark is that our reductions actually have a quasi-polynomial blow-up on the string length (tensor / tree size) D . That is, the new string length after the transformation would be at most $D' = 2^{\text{polylog}(D)}$, which is still $2^{(\log N)^{o(1)}}$ assuming $D = 2^{(\log N)^{o(1)}}$. That is the reason we set the size parameter to be $2^{(\log N)^{o(1)}}$ in the equivalence class.

For the ease of exposition, we break the proofs for Theorem 3.1 and Theorem 1.8 into many sections, each one dealing with one kind of problems. Here we give an outline of the reductions for proving Theorem 3.1 and Theorem 1.8 (Figure 1).

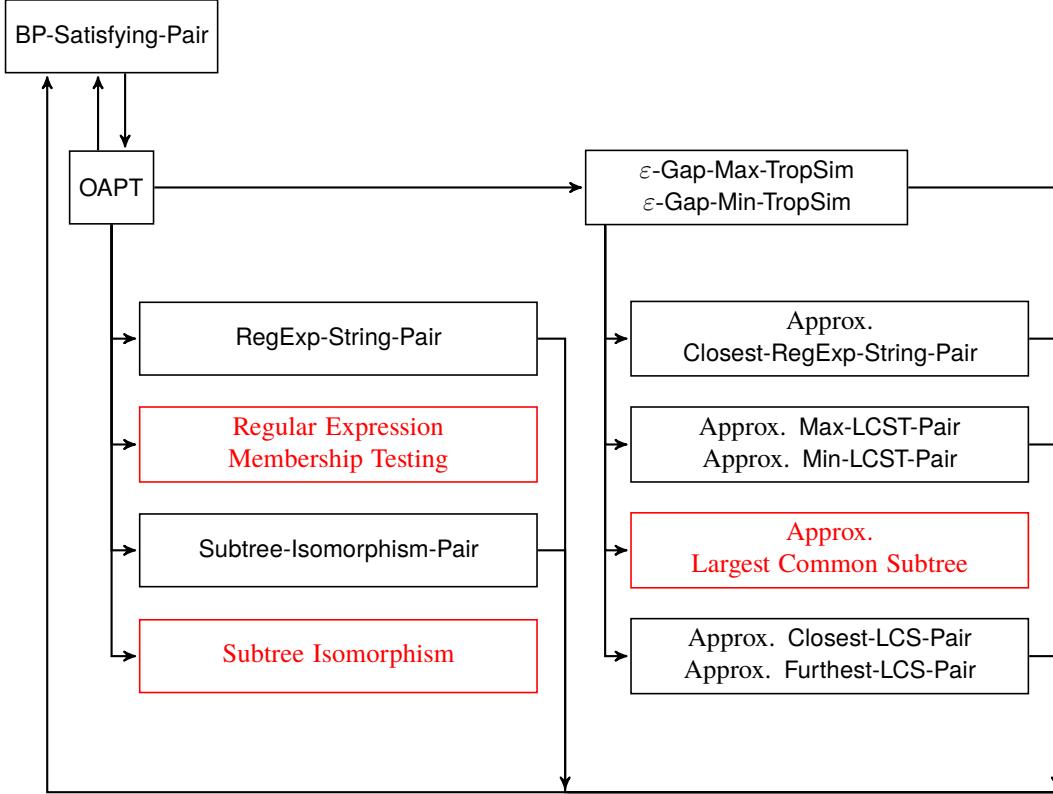


Figure 1: A diagram for all our reductions. (red means it is BP-Pair-Hard.)

Section 4 **BP-Satisfying-Pair.** We present a generic reduction from \mathcal{A} -Satisfying-Pair and Max- \mathcal{A} -Pair / Min- \mathcal{A} -Pair problems to BP-Satisfying-Pair (Theorem 4.1 and Theorem 4.2). This implies all problems in BP-Pair-Class can be reduced to BP-Satisfying-Pair.

Section 5 **Tensors Problems.** We show reductions from BP-Satisfying-Pair to tensors problems OAPT, approximate Max-TropSim and Min-TropSim (Theorem 5.1, Theorem 5.6 and Theorem 5.7), putting these tensor problems into our BP-Pair-Class (Theorem 5.4 and Theorem 5.10).

Section 6 **LCS.** We show reductions from approximate Max-TropSim (Min-TropSim) to approximate Closest-LCS-Pair (Furthest-LCS-Pair) (Theorem 6.2 and Theorem 6.3), putting these LCS-Pair problems into our BP-Pair-Class (Theorem 6.5).

Section 7 Regular Expression. We show reductions from OAPT to RegExp-String-Pair and from approximate Max-TropSim to approximate Closest-RegExp-String-Pair (Theorem 7.3 and Corollary 7.2), putting these regular expression pair problems into our BP-Pair-Class (Theorem 7.5).

We also show a reduction from OAPT to Regular Expression Membership Testing, showing the latter problem is BP-Pair-Hard (Theorem 7.6).

Section 8 Subtree isomorphism. We show reductions from OAPT to Subtree-Isomorphism-Pair and from approximate Max-TropSim (Min-TropSim) to approximate Max-LCST-Pair (Min-LCST-Pair) (Theorem 8.1 and Theorem 8.5), putting these problems related to subtree isomorphism into our BP-Pair-Class (Theorem 8.3 and Theorem 8.10).

We also show reductions from OAPT to Subtree Isomorphism and from approximate Max-TropSim to approximate Largest Common Subtree, showing that the latter problems are BP-Pair-Hard (Theorem 8.4 and Theorem 8.12).

4 Low-Space Algorithm Implies Reduction to BP-Satisfying-Pair

In this section, we present two important theorems for showing reductions from \mathcal{A} -Satisfying-Pair, Max- \mathcal{A} -Pair / Min- \mathcal{A} -Pair problems to BP-Satisfying-Pair.

The key observation is a classic result in space complexity: for $S(n) \geq \log n$, if a decision problem \mathcal{A} is in $\text{NSPACE}[S(n)]$, then there is a BP of length $T = 2^{O(S(n))}$ and width $W = 2^{O(S(n))}$ that decides \mathcal{A} (See, e.g., [AB09] for the proof). This means that if \mathcal{A} can be solved in small space, then we can construct a BP of not too large size to represent this algorithm.

Now we introduce our first theorem, which shows that a low-space algorithm for a decision problem \mathcal{A} implies a reduction from \mathcal{A} -Satisfying-Pair to BP-Satisfying-Pair:

Theorem 4.1. *If the decision problem \mathcal{A} on inputs a, b of length n can be decided in $\text{NSPACE}[\text{polylog}(n)]$, then \mathcal{A} -Satisfying-Pair with two sets of N strings of length $2^{(\log N)^{o(1)}}$ can be reduced to BP-Satisfying-Pair on branching program of size $2^{(\log N)^{o(1)}}$ in near-linear time.*

Proof. Let $n = 2^{(\log N)^{o(1)}}$. Since \mathcal{A} is in $\text{NSPACE}[\text{polylog}(n)]$, we can construct a BP P of size $2^{\text{polylog}(n)} \leq 2^{(\log N)^{o(1)}}$ that decides \mathcal{A} on inputs a, b of length n . Then to check if there is a pair of $(a, b) \in A \times B$ such that (a, b) is an Yes-instance of \mathcal{A} , it is sufficient to check if there is a pair of a, b making P accept. \square

Our second theorem is similar. It shows that a low-space algorithm for the decision problem of an optimization problem \mathcal{A} implies a reduction from Max- \mathcal{A} -Pair / Min- \mathcal{A} -Pair to BP-Satisfying-Pair:

Theorem 4.2. *Let \mathcal{A} be an optimization problem. If the answer to \mathcal{A} on input a, b of length n is bounded in $[-O(\text{poly}(n)), O(\text{poly}(n))]$, and the decision version of \mathcal{A} (deciding whether the answer is greater than a given number k) can be decided in $\text{NSPACE}[\text{polylog}(n)]$, then Max- \mathcal{A} -Pair (or Min- \mathcal{A} -Pair) with two sets of N strings of length $2^{(\log N)^{o(1)}}$ can be reduced to $2^{(\log N)^{o(1)}}$ instances of BP-Satisfying-Pair with branching programs of size $2^{(\log N)^{o(1)}}$ in near-linear time.*

Proof. For Max- \mathcal{A} -Pair, we enumerate each possible answer k , and then we check if there is a pair of $(a, b) \in A \times B$ with answer $> k$ by reducing to BP-Satisfying-Pair via Theorem 4.1. This reduction results in $O(\text{poly}(n)) \leq 2^{(\log N)^{o(1)}}$ instances of BP-Satisfying-Pair, and each branching program is of size $2^{(\log N)^{o(1)}}$. Note that $\text{NSPACE}[\text{polylog}(n)] = \text{coNSPACE}[\text{polylog}(n)]$, thus deciding whether the answer of \mathcal{A} is $\leq k$ is also in $\text{NSPACE}[\text{polylog}(n)]$. Using a similar argument we can reduce Min- \mathcal{A} -Pair to BP-Satisfying-Pair. \square

5 Tensor Problems

In this section, we show that BP-Satisfying-Pair on branching program of size $2^{(\log N)^{o(1)}}$ is equivalent to OAPT and (exact or approximate) Max-TropSim/ Min-TropSim problems on tensors of size $2^{(\log N)^{o(1)}}$ under near-linear time reductions.

5.1 Orthogonal Alternating Product Tensors

First we show the equivalence between BP-Satisfying-Pair and OAPT. To start with, we present the reduction from BP-Satisfying-Pair to OAPT:

Theorem 5.1. *There exists an $O(N \cdot 2^{O(\log W \log T)})$ -time reduction from a BP-Satisfying-Pair instance with a branching program of length T and width W and two sets of N strings to an OAPT problem with two sets of N tensors of size $2^{O(\log W \log T)}$.*

Proof. Let P be a branching program of length T and width W on n boolean inputs $x = (x_1, \dots, x_n)$. First, we follow the proof for the PSPACE-completeness of TQBF [SM73] to construct a quantified boolean formula $\phi(x)$, which holds true iff the branching program P accepts x . Then, we construct two sets A', B' of N tensors such that there is a pair $(a, b) \in A \times B$ satisfying $\phi(a, b)$ is true iff there is a pair $(a', b') \in A' \times B'$ with $p_{\text{alt}}(a', b') = 0$.

Construction of Quantified Boolean Formula. We assume that $n, T - 1, W$ are powers of two without loss of generality. First we construct formulas $\psi_k(x, u, v, i)$ for all $0 \leq k \leq \log(T - 1)$, $u, v \in [W]$, $i \in [T]$ such that $\psi_k(x, u, v, i)$ holds true iff the node $(i + 2^k, v)$ is reachable from the node (i, u) on the input $x = (x_1, \dots, x_n)$.

The construction is by induction on k . For $k = 0$, we split the n input variables $x = (x_1, \dots, x_n)$ into two halves: $x^a = (x_1, \dots, x_{n/2})$ and $x^b = (x_{n/2+1}, \dots, x_n)$. We construct two formulas $\alpha(x^a, u, v, i)$ and $\beta(x^b, u, v, i)$. We construct the formula α to be true iff the variable $x_{f(i)}$ associated with the layer L_i is in x^a , and there is an edge that goes from the node (i, u) to the node $(i + 1, v)$ and is marked with the value of $x_{f(i)}$. We define the formula β similarly for x^b . Then we construct $\psi_0(x, u, v, i) = \alpha(x^a, u, v, i) \vee \beta(x^b, u, v, i)$. It is easy to see that $\psi_0(x, u, v, i)$ holds true iff the node $(i + 1, v)$ is reachable from the node (i, u) on the input $x = (x_1, \dots, x_n)$. For $k \geq 1$, we construct $\psi_k(x, u, v, i)$ as:

$$(\exists m)(\forall u')(\forall v')(\forall j)[((u', v', j) = (u, m, 0) \vee (u', v', j) = (m, v, 1)) \Rightarrow \psi_{k-1}(x, u', v', i + j \cdot 2^{k-1})]$$

where $m, u', v' \in [W]$ and $j \in \{0, 1\}$. It is easy to see that the above formula is equivalent to

$$(\exists m)[\psi_{k-1}(x, u, m, i) \wedge \psi_{k-1}(x, m, v, i + 2^{k-1})],$$

thus it holds true iff the node $(i + 2^k, v)$ is reachable from the node (i, u) .

In the end, we construct the formula $\varphi(x) = \psi_{\log(T-1)}(x, 1, 1, 1)$, so $\varphi(x)$ holds true iff the branching program P accepts the input $x = (x_1, \dots, x_n)$ (Recall that $u_{\text{start}} = (1, 1)$ and $u_{\text{acc}} = (T, 1)$).

We split all the variables m, u', v', j occurred in $\varphi(x)$ into t boolean variables $z_1, \dots, z_t \in \{0, 1\}$ for some $t = O(\log W \log T)$. Without loss of generality we assume t is even. Then we transform $\varphi(x)$ into the following equivalent formula ϕ :

$$\phi(x) = (\exists z_1)(\forall z_2)(\exists z_3) \cdots (\forall z_t)(f(z) \Rightarrow (g_1(x^a, z) \vee g_2(x^b, z)))$$

where $f(z)$ is the logical conjunction of all the predicates $((a, b, j) = (u, m, 0) \vee (a, b, j) = (m, v, 1))$ in $\psi_1, \dots, \psi_{\log(T-1)}$, and $g_1(x^a, z), g_2(x^b, z)$ are the innermost $\alpha(x^a, u, v, i), \beta(x^b, u, v, i)$. The quantifiers \forall and \exists appear alternatively.

Converting Quantified Boolean Formula into Tensors. Let $d_1 = \dots = d_t = 2$. Now we construct two sets of N tensors $A', B' \subseteq \{0, 1\}^{d_1 \times \dots \times d_t}$ to be our OAPT instance. For $1 \leq k \leq t$, we associate the k -th dimension of a tensor with the variable z_k and associate each index $p \in [d_1] \times \dots \times [d_t]$ with an assignment to z_1, \dots, z_t . Note that strings in the set A correspond to assignments to x^a , and strings in the set B correspond to assignments to x^b . Thus every two strings $(a, b) \in A \times B$ along with an index p specify an assignment to x and z .

For each string $a \in A$, we construct a tensor $a' \in \{0, 1\}^{d_1 \times \dots \times d_t}$ where for every index p , a'_p is 0 iff the formula $\neg f(z) \vee g_1(x^a, z)$ is true with corresponding assignments to x^a and z ; for each string $b \in B$, we construct a tensor $b' \in \{0, 1\}^{d_1 \times \dots \times d_t}$ where for every index p , b'_p is 0 iff the formula $g_2(x^b, z)$ is true with corresponding assignments to x^b and z .

Note that $f(z) \Rightarrow (g_1(x^a, z) \vee g_2(x^b, z))$ is equivalent to $\neg f(z) \vee g_1(x^a, z) \vee g_2(x^b, z)$, so we have

$$a'_p \wedge b'_p = 0 \iff [f(z) \Rightarrow (g_1(x^a, z) \vee g_2(x^b, z))].$$

Then it is easy to see that $p_{\text{alt}}(a', b') = 0$ iff $\phi(a, b)$ is true, and thus P accepts a pair of $(a, b) \in A \times B$ iff $p_{\text{alt}}(a', b') = 0$ for their corresponding a', b' . Note that $d_1 d_2 \cdots d_t = 2^t = 2^{O(\log W \log T)}$, so each tensor has size $2^{O(\log W \log T)}$. \square

Note that in the above construction, tensors in B are all \wedge -invariant, so we have the following corollary for Restricted OAPT:

Corollary 5.2. *There exists an $O(N \cdot 2^{O(\log W \log T)})$ -time reduction from a BP-Satisfying-Pair instance with a branching program of length T and width W and two sets of N strings to an Restricted OAPT problem with two sets of N tensors of size $2^{O(\log W \log T)}$.*

For the other direction, by Theorem 4.1, it is sufficient to show that computing Alternating Product can be done in $\text{SPACE}[O(\log n)] \subseteq \text{NSPACE}[\text{polylog}(n)]$.

Lemma 5.3. *Given two tensors a, b of size $n = 2^t$, their Alternating Product $p_{\text{alt}}(a, b)$ can be computed in $\text{SPACE}[O(\log n)]$.*

Proof. We compute the Alternating Product recursively according to the definition. There are t levels of recursion in total. Since $t = \log n$, space $O(\log n)$ is enough for our algorithm. \square

Combining Theorem 5.1 and Lemma 5.3, we can prove the equivalence between BP-Satisfying-Pair and OAPT.

Theorem 5.4. *The OAPT problem on tensors of size $2^{(\log N)^{o(1)}}$ is equivalent to BP-Satisfying-Pair on branching program of size $2^{(\log N)^{o(1)}}$ under near-linear time reductions.*

5.2 A Communication Protocol for Branching Program

Before we turn to show the equivalence between BP-Satisfying-Pair and Max-TropSim / Min-TropSim, we introduce the following IP-protocol for branching program. Our reduction from BP-Satisfying-Pair to Max-TropSim (or Min-TropSim) directly follows by simulating the communication protocol using tropical algebra.

Theorem 5.5. *Let P be a branching program of length T and width W on n boolean inputs x_1, \dots, x_n . Suppose Alice holds the input $x_1, \dots, x_{n/2}$ and Bob holds the input $x_{n/2+1}, \dots, x_n$. For every $\varepsilon > 0$, there exists a computationally efficient IP-protocol for checking whether P accepts on x_1, \dots, x_n , in which:*

1. Merlin and Alice exchange $O(\log^2 W \log^2 T \cdot (\log \log W + \log \log T + \log \varepsilon^{-1}))$ bits;

2. Alice tosses $O(\log^2 W \log^2 T \cdot (\log \log W + \log \log T + \log \varepsilon^{-1}))$ public coins;
3. Bob sends $O(\log \log W + \log \log T + \log \varepsilon^{-1})$ bits to Alice;
4. Alice accepts or rejects in the end.

If P accepts on the input x_1, \dots, x_n , then Alice always accepts; otherwise, Alice rejects with probability at least $1 - \varepsilon$.

Proof. Let \bar{a} be the assignment to the input variables held by Alice, and \bar{b} be the assignment to the input variables held by Bob. Recall the construction of the tensors in the proof for Theorem 5.1. First Alice constructs a tensor $a = G(\bar{a})$, and Bob constructs a tensor $b = H(\bar{b})$. Each tensor here is of shape $d_1 \times d_2 \times \dots \times d_t = 2 \times 2 \times \dots \times 2$ for $t = O(\log T \log W)$. Then the problem reduces to check whether the Alternating Product $p_{\text{alt}}(a, b)$ equals 0. Now we show that there exists a communication protocol for checking $p_{\text{alt}}(a, b) = 0$, using the idea for proving $\text{IP} = \text{PSPACE}$ [LFKN92, Sha92].

Arithmetization. First we arithmetize the computation of Alternating Product. Let $q \geq 1$ be a parameter to be specified. Construct a finite field \mathbb{F}_{2^q} . Then Alice finds a multilinear extension α over \mathbb{F}_{2^q} for her tensor a , i.e., Alice finds a function $\alpha(z_1, \dots, z_t)$ such that α is linear in each of its variables, and $\alpha(z_1, \dots, z_t) = a_i$ for all $i \in [d_1] \times \dots \times [d_t]$ and $i_k = z_k + 1$ ($1 \leq k \leq t$). Bob finds a multilinear extension β for his tensor b similarly. Recall that the definition of Alternating Product. $p_a(a, b)$ can be rewritten as

$$p_{\text{alt}}(a, b) = \bigwedge_{z_1 \in \{0,1\}} \left[\bigvee_{z_2 \in \{0,1\}} \left(\bigwedge_{z_3 \in \{0,1\}} \left[\dots \bigvee_{z_t \in \{0,1\}} (\alpha(z_1, \dots, z_t) \cdot \beta(z_1, \dots, z_t)) \dots \right] \right) \right].$$

To arithmetize $\bigwedge_{z_k \in \{0,1\}}$ and $\bigvee_{z_k \in \{0,1\}}$, we define three kinds of operators acting on polynomials:

1. Π_{z_m} operator, which arithmetizes the formula $\bigwedge_{z_m \in \{0,1\}} F(z_1, \dots, z_{m-1}, z_m)$.

$$\Pi_{z_m} F(z_1, \dots, z_m) = F(z_1, \dots, z_{m-1}, 0) \cdot F(z_1, \dots, z_{m-1}, 1)$$

2. Σ_{z_m} operator, which arithmetizes the formula $\bigvee_{z_m \in \{0,1\}} F(z_1, \dots, z_{m-1}, z_m)$.

$$\Sigma_{z_m} F(z_1, \dots, z_m) = 1 - (1 - F(z_1, \dots, z_{m-1}, 0)) \cdot (1 - F(z_1, \dots, z_{m-1}, 1)).$$

3. \mathcal{R}_{z_i} operator, which is used for the degree reduction. When acting on a polynomial $F(z_1, \dots, z_m)$, it replaces z_i^k for $k \geq 1$ by z_i in all terms. In this way, any polynomial $F(z_1, \dots, z_m)$ can be converted into a multilinear one preserving the values at every $(z_1, \dots, z_m) \in \{0,1\}^m$. \mathcal{R}_{z_i} operator can be written as

$$\mathcal{R}_{z_i} F(z_1, \dots, z_m) = F(z_1, \dots, z_{i-1}, 0, z_{i+1}, \dots, z_m) + z_i \cdot F(z_1, \dots, z_{i-1}, 1, z_{i+1}, \dots, z_m).$$

Then it is easy to see that

$$p_{\text{alt}}(a, b) = \Pi_{z_1} \Sigma_{z_2} \Pi_{z_3} \dots \Sigma_{z_t} (\alpha \cdot \beta).$$

Note that in the computation of Alternating Product, we only use the function value at Boolean inputs, thus we can insert i operators $R_{z_1} R_{z_2} \dots R_{z_i}$ right after each π_{z_i} or Σ_{z_i} without changing the final result:

$$p_{\text{alt}}(a, b) = \Pi_{z_1} R_{z_1} \Sigma_{z_2} R_{z_1} R_{z_2} \Pi_{z_3} R_{z_1} R_{z_2} R_{z_3} \dots \Sigma_{z_t} R_{z_1} \dots R_{z_t} (\alpha \cdot \beta).$$

In total we use only $M = O(t^2) \leq O(\log^2 T \log^2 W)$ operators.

The Protocol. We introduce our IP-protocol in an inductive way. Suppose that we have an IP-protocol for some polynomial $F(z_1, \dots, z_m)$, in which for any given $(v_1, \dots, v_m) \in \mathbb{F}_{2^q}^m$ and $u = F(v_1, \dots, v_m)$, Merlin can convince Alice and Bob that $F(v_1, \dots, v_m) = u$ with perfect completeness and soundness error ε_0 . We show that for $G(z_1, \dots, z_{m'}) = \mathcal{O}_{z_i} F(z_1, \dots, z_m)$ and given $v_1, \dots, v_{m'}$ and u' ($\mathcal{O}_{z_i} \in \{\Sigma_{z_i}, \Pi_{z_i}, \mathcal{R}_{z_i}\}$, $m' = m$ when $\mathcal{O}_{z_i} = \mathcal{R}_{z_i}$ and $m' = m - 1$ otherwise), Merlin can convince Alice and Bob that $G(v_1, \dots, v_{m'}) = u'$ with perfect completeness and soundness error $\varepsilon_0 + O(2^{-q})$:

- First Merlin sends the coefficients of the polynomial $F(v_1, \dots, v_{i-1}, z_i, v_{i+1}, \dots, v_m)$ to Alice (note that it is a univariate polynomial of z_i);
- Alice calculates the value of $G(v_1, \dots, v_{m'})$ using the information sent by Merlin (assuming Merlin is honest), and reject if $G(v_1, \dots, v_{m'}) \neq u'$;
- Alice randomly draws an element $r \in \mathbb{F}_{2^q}$. Let $v_i = r$ (reset $v_i = r$ if v_i already exists);
- Alice checks if $F(v_1, \dots, v_{i-1}, r, v_{i+1}, \dots, v_m) = u$ via the IP-protocol for F .

It is easy to see that the above protocol has perfect completeness. For soundness, notice that F and G are always of $O(1)$ degree because our use of degree reduction operators, thus Alice can find $G(v_1, \dots, v_{m'}) \neq u'$ with probability $O(2^{-q})$ if Merlin lies. By the union bound, the soundness error of the IP-protocol for G is $\varepsilon_0 + O(2^{-q})$.

Our IP-protocol starts by checking $p_{\text{alt}}(a, b) = 0$. Following the inductive process above, there are M rounds of communication between Merlin and Alice. And after the last round, $p_{\text{alt}}(a, b) = 0$ reduces to check if $\alpha(v_1, \dots, v_t) \cdot \beta(v_1, \dots, v_t) = u$ for given $(v_1, \dots, v_t) \in \mathbb{F}_{2^q}^t, u \in \mathbb{F}_{2^q}$. Note that all the values of v_1, \dots, v_t can be inferred by the results of public coins Alice tossed. Thus the IP-protocol for $\alpha \cdot \beta$ is as follows: Bob learns the results of public coins and obtains v_1, \dots, v_t . Then Bob sends the value of $\beta(v_1, \dots, v_t)$ to Alice. Finally, Alice accepts iff $\alpha(v_1, \dots, v_t) \cdot \beta(v_1, \dots, v_t) = u$.

By induction, we can show that the whole IP-protocol has perfect completeness and soundness error $O(M \cdot 2^{-q})$. Setting $2^q = c \cdot M \cdot \varepsilon^{-1}$ for large enough constant c , we can achieve the soundness error ε . And in this case we have

$$q = \log M + \log \varepsilon^{-1} + \log c = O(\log \log W + \log \log T + \log \varepsilon^{-1}).$$

It can be easily seen that Alice tosses

$$O(Mq) = O(\log^2 T \log^2 W (\log \log T + \log \log W + \log \varepsilon^{-1}))$$

public coins and Bob sends

$$O(q) = O(\log \log T + \log \log W + \log \varepsilon^{-1})$$

bits to Alice in our communication protocol.

In each of the M rounds, Merlin sends $O(1)$ elements in \mathbb{F}_{2^q} since F is of at most constant degree. Thus Merlin sends at most

$$O(Mq) = O(\log^2 T \log^2 W (\log \log T + \log \log W + \log \varepsilon^{-1}))$$

bits to Alice. □

5.3 Tropical Tensors

Following from [AR18], we can show a reduction from BP-Satisfying-Pair to ε -Gap-Max-TropSim based on our IP-protocol for branching program.

Theorem 5.6. *There is a reduction from BP-Satisfying-Pair on branching program of length T and width W and two sets of N strings to ε -Gap-Max-TropSim on two sets of N tensors of size*

$$D = 2^{O(\log^2 W \log^2 T (\log \log W + \log \log T + \log \varepsilon^{-1}))},$$

and the reduction runs in $O(N \text{poly}(D))$. Here ε is a threshold value that can depend on N .

Proof. For convenience, let

$$K = \log^2 W \log^2 T (\log \log W + \log \log T + \log \varepsilon^{-1}).$$

By Theorem 5.5, there is an IP-protocol using $O(K)$ bits for determining whether a branching program accepts when Alice knows the first half and Bob knows the second half, with soundness error ε . We can easily modify the communication protocol such that

- Alice and Merlin interact for $m = O(K)$ rounds, in each round Merlin sends one bit to Alice and Alice tosses one public coin;
- After the interaction between Alice and Merlin, Bob sends $\ell = O(K)$ bits to Alice, and after Bob sending each bit Merlin sends a dummy bit to Alice;
- Alice accepts or rejects in the end.

Let $t = 2\ell + 2m$ and $d_1 = \dots = d_t = 2$. Now we construct two sets of $N = 2^{n/2}$ tensors $A, B \in \{0, 1\}^{d_1 \times \dots \times d_t}$ as our ε -Gap-Max-TropSim instance. For every $0 \leq k < m$, we associate the $(t - 2k)$ -th dimension with the result of the public coin Alice tosses at the $(k+1)$ -th round and associate the $(t - 2k - 1)$ -th dimension with the bit Merlin sends to Alice at the $(k+1)$ -th round. For every $0 \leq k < \ell$, we associate the $(2\ell - 2k)$ -th dimension of a tensor with the $(k+1)$ -th bit sent by Bob and associate the $(2\ell - 2k - 1)$ -th dimension with the bit Merlin sent to Alice right after Bob sending the $(k+1)$ -th bit to Alice. In this way, every index $p \in [d_1] \times \dots \times [d_t]$ of a tensor can be seen as a communication transcript.

Let $A, B \subseteq \{0, 1\}^{n/2}$ be the two sets in the BP-Satisfying-Pair instance. For each assignment $a \in A$ to the first half variables $x_1, \dots, x_{n/2}$, we construct a tensor $G(a) \in \{0, 1\}^{d_1 \times \dots \times d_t}$ where for every index p , $G(a)_p$ is 1 iff Alice accepts after seeing the communication transcript p when she holds the assignment a for $x_1, \dots, x_{n/2}$. For each assignment $b \in B$ to the second half variables $x_{n/2+1}, \dots, x_n$, we construct a tensor $H(b) \in \{0, 1\}^{d_1 \times \dots \times d_t}$ where for every index p , $H(b)_p$ is 1 iff the bits sent by Bob in the communication transcript p matches what Bob sends when he holds the assignment b for $x_{n/2+1}, \dots, x_n$ and learning the results of Alice's public coins in p .

Let $D = 2^t = 2^{O(K)}$ be the size of each tensor. It is easy to see that when Alice holds a and Bob holds b , the maximum probability (over all Merlin's actions) that Alice accepts in our communication protocol equals the Tropical Similarity $s(G(a), H(b))$. \square

By negating the branching program P , we can also show a similar reduction from BP-Satisfying-Pair to ε -Gap-Min-TropSim:

Theorem 5.7. *There is a reduction from BP-Satisfying-Pair on branching program of length T and width W and two sets of N strings to ε -Gap-Min-TropSim on two sets of N tensors of size*

$$D = 2^{O(\log^2 W \log^2 T (\log \log W + \log \log T + \log \varepsilon^{-1}))},$$

and the reduction runs in $O(N \text{poly}(D))$. Here ε is a threshold value that can depend on N .

Proof. The IP-protocol in Theorem 5.5 can be easily adapted to check the branching program P does *not* accept, i.e., if P rejects on the input x_1, \dots, x_n , then Alice always accepts; otherwise, Alice rejects with probability $1 - \varepsilon$. To do this, the only thing we need to change is to check whether the Alternating Product is 1 rather than 0. Then, using the same reduction as in Theorem 5.6, we can obtain two sets $A' = \{G(a) \mid a \in A\}, B' = \{H(b) \mid b \in B\}$ of N tensors of size

$$D = 2^{O(\log^2 W \log^2 T (\log \log W + \log \log T + \log \varepsilon^{-1}))}$$

such that for every pair of strings $(a, b) \in A \times B$, the maximum probability (over all Merlin's actions) that Alice accepts in the IP-protocol equals the Tropical Similarity score of the corresponding tensor gadgets $G(a)$ and $H(b)$. Thus, to decide whether there exists a pair of $(a, b) \in A \times B$ that can make P accept, it is sufficient to distinguish from the case that there is a pair of $G(a), H(b)$ such that the Tropical Similarity score $s(G(a), H(b)) \leq \varepsilon$ and the case that every pair of $G(a), H(b)$ has perfect Tropical Similarity score $s(G(a), H(b)) = 1$. \square

Note that in the above constructions in Theorem 5.6 and 5.7, tensors in B are all max-invariant, so we have the following corollary for Restricted OAPT:

Corollary 5.8. *There is a reduction from BP-Satisfying-Pair on branching program of length T and width W and two sets of N strings to a Restricted ε -Gap-Max-TropSim / Restricted ε -Gap-Min-TropSim on two sets of N tensors of size*

$$D = 2^{O(\log^2 W \log^2 T (\log \log W + \log \log T + \log \varepsilon^{-1}))},$$

and the reduction runs in $O(N \text{poly}(D))$. Here ε is a threshold value that can depend on N .

Theorem 5.6 and 5.7 also imply reductions from BP-Satisfying-Pair to exact Max-TropSim or exact Min-TropSim. For the other direction of reduction, we have the following lemma:

Lemma 5.9. *Given two tensors a, b of size $n = 2^t$, their Tropical Similarity $s(a, b)$ can be computed in $\text{SPACE}[O(\log^2 n)]$.*

Proof. We compute the Tropical Similarity recursively according to the definition. Note that there are t levels of recursion in total, and $O(t)$ -bit precision is sufficient in this computation. Thus this algorithm uses only $O(t^2) \leq O(\log^2 n)$ space. \square

Combining Theorem 5.6, Theorem 5.7 and Lemma 5.9, we can establish the equivalence between BP-Satisfying-Pair and the exact or approximate Tropical Similarity problems:

Theorem 5.10. *For the case that the maximum element size $D = 2^{(\log N)^{o(1)}}$, there are near-linear time reductions between all pairs of the following problems:*

- BP-Satisfying-Pair;
- Exact Max-TropSim or Min-TropSim;
- $2^{-(\log D)^{1-\Omega(1)}}$ -Gap-Max-TropSim or $2^{-(\log D)^{1-\Omega(1)}}$ -Gap-Min-TropSim;
- $2^{(\log D)^{1-\Omega(1)}}$ -approximate Max-TropSim or Min-TropSim;

Proof. Let $c > 0$ be a constant. For any instance of BP-Satisfying-Pair on BP of size S , by Theorem 5.6 with parameter $\varepsilon = 2^{-(\log S)^c}$, we can reduce it to an ε -Gap-Max-TropSim instance on tensors of size $D = 2^{\Theta(\log^4 S \log \varepsilon^{-1})} = 2^{\Theta(\log^{4+c} S)}$ (adding dummy dimensions to tensors if necessary).

Thus, we have $\varepsilon = 2^{-\Theta(\log D)^{c/(4+c)}}$. For any $0 < \delta \leq 1$, by choosing an appropriate value for c , we can obtain a reduction from BP-Satisfying-Pair on BP of size $S = 2^{(\log N)^{\omega(1)}}$ to $2^{-(\log D)^{1-\delta}}$ -Gap-Max-TropSim.

$2^{-(\log D)^{1-\delta}}$ -Gap-Max-TropSim can be trivially reduced to $2^{(\log D)^{1-\delta}}$ -approximate Max-TropSim, and $2^{(\log D)^{1-\delta}}$ -approximate Max-TropSim can be trivially reduced to Max-TropSim. By Lemma 5.9 and Theorem 4.2, Max-TropSim can be reduced to BP-Satisfying-Pair.

Therefore under near-linear time reductions BP-Satisfying-Pair, exact Max-TropSim, $2^{-(\log D)^{1-\Omega(1)}}$ -Gap-Max-TropSim and $2^{(\log D)^{1-\Omega(1)}}$ -approximate Max-TropSim are all equivalent. Using a similar argument, we can also prove the same result for Min-TropSim. \square

6 Longest Common Subsequence

In this section, we show near-linear time reductions between BP-Satisfying-Pair and (exact or approximate) Closest-LCS-Pair / Furthest-LCS-Pair.

Our reduction from BP-Satisfying-Pair to Closest-LCS-Pair / Furthest-LCS-Pair relies on the following LCS gadgets in [AR18].

Theorem 6.1 ([AR18]). *Let t be an even number and $d_1 = \dots = d_t = 2$. Given two sets of N tensors A, B in $\{0, 1\}^{d_1 \times \dots \times d_t}$, there is a deterministic algorithm running in $O(N \text{poly}(2^t))$ time which outputs two sets A', B' of N strings of length 2^t over an $O(2^t)$ -size alphabet Σ , such that each $a \in A$ corresponds to a string $a' \in A'$, each $b \in B$ corresponds to a string $b \in B'$, and $\text{LCS}(a', b') = 2^{t/2} \cdot s(a, b)$ for every $a \in A, b \in B$, where $s(a, b)$ stands for the Tropical Similarity score of two tensors a and b .*

Theorem 6.1 directly leads to the following two theorems:

Theorem 6.2. *There exists an $O(N \text{poly}(D))$ -time reduction from an $\varepsilon(D)$ -Gap-Max-TropSim instance with two sets of N tensors of size D to an instance of the following approximation variant of Closest-LCS-Pair: Given two sets of N strings of length D , distinguish between the following:*

- **Completeness:** There exists a pair of a, b such that $\text{LCS}(a, b) = \sqrt{D}$;
- **Soundness:** For every pair a, b , $\text{LCS}(a, b) < \sqrt{D} \cdot \varepsilon(D)$.

Thus $\varepsilon(D)$ -Gap-Max-TropSim can be $O(N \text{poly}(D))$ -time reduced to $\varepsilon(D)^{-1}$ -approximate Closest-LCS-Pair.

Theorem 6.3. *There exists an $O(N \text{poly}(D))$ -time reduction from an $\varepsilon(D)$ -Gap-Min-TropSim instance with two sets of N tensors of size D to an instance of the following approximation variant of Furthest-LCS-Pair: Given two sets of N strings of length D , distinguish between the following:*

- **Completeness:** There exists a pair of a, b such that $\text{LCS}(a, b) < \sqrt{D} \cdot \varepsilon(D)$;
- **Soundness:** For every pair a, b , $\text{LCS}(a, b) = \sqrt{D}$.

Thus $\varepsilon(D)$ -Gap-Min-TropSim can be $O(N \text{poly}(D))$ -time reduced to $\varepsilon(D)^{-1}$ -approximate Furthest-LCS-Pair.

According to Theorem 4.2, in order to reduce Closest-LCS-Pair or Furthest-LCS-Pair problem to BP-Satisfying-Pair, we only need to show that given a number k , deciding $\text{LCS}(a, b) \geq k$ for two strings a, b of length n is in $\text{NSPACE}[\text{polylog}(n)]$:

Lemma 6.4 (Folklore). *Given two strings a, b of length n and a number k , deciding whether $\text{LCS}(a, b) \geq k$ is in NL .*

Proof. The algorithm consists of k stages. Let c be a longest common subsequence of a and b . In the i -th stage, we nondeterministically guess which positions of a and b are matched with the i -th character of c , and then we check if the characters on the two positions of a and b are the same. Also, in each stage we store the positions being matched in the last stage, so that we can check if the matched positions in each string are increasing. Finally, we accept if the nondeterministic guesses pass all the checks. The total space for this nondeterministic algorithm is $O(\log n)$ since we only need to maintain $O(1)$ positions in each stage. \square

Combining Theorem 6.2, 6.3 and Lemma 6.4 together, we can prove the equivalence between BP-Satisfying-Pair and exact or approximate LCS pair problems:

Theorem 6.5. *For the case that the maximum element size $D = 2^{(\log N)^{o(1)}}$, there are near-linear time reductions between all pairs of the following problems:*

- *BP-Satisfying-Pair;*
- *Exact Closest-LCS-Pair or Furthest-LCS-Pair;*
- $2^{(\log D)^{1-\Omega(1)}}$ -approximate *Closest-LCS-Pair or Furthest-LCS-Pair.*

Proof. By Theorem 5.10, the BP-Satisfying-Pair problem on branching program of size $2^{(\log N)^{o(1)}}$ is equivalent to $2^{(\log D)^{1-\delta}}$ -Gap-Max-TropSim under near-linear time reductions. By Theorem 6.2, $2^{(\log D)^{1-\delta}}$ -Gap-Max-TropSim can be reduced to $2^{(\log D)^{1-\delta}}$ -approximate Closest-LCS-Pair. $2^{(\log D)^{1-\delta}}$ -approximate Closest-LCS-Pair can be trivially reduced to Closest-LCS-Pair. By Lemma 6.4 and Theorem 4.2, Closest-LCS-Pair can further be reduced to BP-Satisfying-Pair. Thus BP-Satisfying-Pair, exact Closest-LCS-Pair and $2^{(\log D)^{1-\delta}}$ -approximate Closest-LCS-Pair are equivalent under near-linear time reductions for all $\delta > 0$.

Using a similar argument, we can also prove the same result for exact and approximate Furthest-LCS-Pair. \square

7 Regular Expression Membership Testing

In this section, we study the hardness of regular expression problems. First we prove that BP-Satisfying-Pair, RegExp-String-Pair and Closest-RegExp-String-Pair are equivalent under near-linear time reductions, then we show the hardness for the Regular Expression Membership Testing problem.

For simplicity, we denote $\max_{x \in L(a), |x|=|b|} \text{HamSim}(x, b)$ by $\text{MaxSim}(a, b)$ for any regular expression a and string b . The following theorem gives a construction to implement the Tropical Similarity using MaxSim .

Theorem 7.1. *Let t be an even number and $d_1 = \dots = d_t = 2$. Given two sets of N tensors $A, B \subseteq \{0, 1\}^{d_1 \times \dots \times d_t}$ satisfying that all the tensors in B are max-invariant, there is a deterministic algorithm running in $O(N \text{poly}(2^t))$ time which outputs a set A' of N regular expressions and a set B' of N strings. Here strings are of length 2^t , regular expressions are of length $\text{poly}(2^t)$, and both of them are over alphabet $\Sigma = \{0, 1, \perp\}$. Each $a \in A$ corresponds to a regular expression $a' \in A'$, each $b \in B$ corresponds to a string $b' \in B'$, and*

$$\text{MaxSim}(a', b') = s(a, b).$$

Proof. For each k and each prefix of index $i_{(k)} \in [d_1] \times \dots \times [d_k]$, we construct corresponding gadget $a' = G_{i_{(k)}}(a)$ and $b' = H_{i_{(k)}}(b)$ for each $a \in A$ and $b \in B$ (when $k = 0$, $i_{(k)}$ can only be the empty prefix, and we simply use $G(a)$ and $H(b)$ for convenience) inductively, mimicking the evaluation of the Tropical Similarity. For this purpose, we need to construct the following three types of gadgets.

Bit Gadgets. First we need bit gadgets to simulate the innermost coordinatewise product in the evaluation of Tropical Similarity. For each coordinate $i \in [d_1] \times \cdots \times [d_t]$, for every $a \in A$ and $b \in B$, we construct

$$G_i(a) = \begin{cases} \perp & \text{if } a_i = 0, \\ 1 & \text{if } a_i = 1, \end{cases} \quad \text{and} \quad H_i(b) = \begin{cases} 0 & \text{if } b_i = 0, \\ 1 & \text{if } b_i = 1. \end{cases}$$

It is easy to see that $a_i \cdot b_i = \text{MaxSim}(G_i(a), H_i(b))$.

Now we combine bit gadgets recursively according to the max and \mathbb{E} operators in the evaluation for Tropical Similarity. Starting from $k = t - 1$, there are two cases to consider.

Expectation Gadgets. The first case is when \mathbb{E} operator is applied to $(k + 1)$ -th dimension. We construct the corresponding gadgets $G_{i(k)}(a)$ and $H_{i(k)}(b)$ for any $i_{(k)} \in [d_1] \times \cdots \times [d_k]$, $a \in A$ and $b \in B$ as follows:

$$G_{i(k)}(a) = G_{i(k),0}(a) \circ G_{i(k),1}(a) \quad \text{and} \quad H_{i(k)}(b) = H_{i(k),0}(b) \circ H_{i(k),1}(b).$$

where \circ stands for concatenation as usual. It is easy to see that

$$\text{MaxSim}(G_{i(k)}(a), H_{i(k)}(b)) = \mathbb{E}_{j \in \{0,1\}} [\text{MaxSim}(G_{i(k),j}(a), H_{i(k),j}(b))].$$

Max Gadgets. The second case is when max operator is applied to $(k + 1)$ -th dimension. For $i_{(k)} \in [d_1] \times \cdots \times [d_k]$ and $a \in A$, $G_{i(k)}(a)$ is constructed as follows:

$$G_{i(k)}(a) = [G_{i(k),0}(a) \mid G_{i(k),1}(a)],$$

and we construct $H_{i(k)}(b)$ for $b \in B$ to be

$$H_{i(k)}(b) = H_{i(k),j}(b)$$

for all $j \in \{0, 1\}$, which is well-defined since b is max-invariant. It is easy to see that

$$\text{MaxSim}(G_{i(k)}(a), H_{i(k)}(b)) = \max_{j \in \{0,1\}} [\text{MaxSim}(G_{i(k),j}(a), H_{i(k),j}(b))].$$

Finally, we can obtain tensor gadgets $G(a), H(b)$ for each $a \in A$ and $b \in B$. \square

From Theorem 7.1, we have the following reduction:

Corollary 7.2. Let ε -Gap-Closest-RegExp-String-Pair be the approximation variant of Closest-RegExp-String-Pair: Given a set of N regular expressions of length $O(\text{poly}(D))$ and a set of N strings of length D , distinguish between the following:

- **Completeness:** There exists a pair of a, b such that $\text{MaxSim}(a, b) = 1$ (i.e., $b \in L(a)$);
- **Soundness:** For every pair a, b , $\text{MaxSim}(a, b) < \varepsilon$.

There exists an $O(N \text{poly}(D))$ -time reduction from a Restricted $\varepsilon(D)$ -Gap-Max-TropSim instance on two sets A, B of N tensors of size D to an instance of $\varepsilon(D)$ -Gap-Closest-RegExp-String-Pair on a set of N regular expressions of length $O(\text{poly}(D))$ and a set of N strings of length D .

This corollary also follows that there is a reduction from ε -Gap-Max-TropSim to RegExp-String-Pair, which is enough to show that RegExp-String-Pair is no easier than BP-Satisfying-Pair. But actually it is possible to show a direct reduction from Restricted OAPT to RegExp-String-Pair, without using the reduction from Restricted OAPT to Restricted ε -Gap-Max-TropSim:

Theorem 7.3. *There exists an $O(N \text{poly}(D))$ -time reduction from a Restricted OAPT instance with two sets A, B of N tensors of size D to an RegExp-String-Pair instance with a set of N regular expressions of length $O(\text{poly}(D))$ and a set of N strings of length D .*

Proof. We use nearly the same reduction as in Theorem 7.1. The bit gadgets are constructed as follows:

$$G_i(a) = \begin{cases} [0 \mid 1] & \text{if } a_i = 0, \\ 0 & \text{if } a_i = 1, \end{cases} \quad \text{and} \quad H_i(b) = \begin{cases} 0 & \text{if } b_i = 0, \\ 1 & \text{if } b_i = 1. \end{cases}$$

for any $a \in A, b \in B$. And we construct \wedge gadgets in the same way as the max gadgets, \vee gadgets in the same way as the \mathbb{E} gadgets. By De Morgan's laws, we can show that $H(b) \in L(G(a))$ iff $p_{\text{alt}}(a, b) = 0$. \square

For the other direction, we note that the following theorem gives a low-space algorithm for exact and approximate regular expression membership testing, then we can obtain a reduction by Theorem 4.1. The following theorem is noted in [JR91]:

Theorem 7.4 ([JR91]). *Given a regular expression a and a string b , deciding whether $b \in L(a)$ is in NL.*

Combining all the above reductions together, we can show the equivalence between all pairs of BP-Satisfying-Pair, RegExp-String-Pair and ε -Gap-Closest-RegExp-String-Pair.

Theorem 7.5. *For the case that the maximum element size $D = 2^{(\log N)^{o(1)}}$, there are near-linear time reductions between all pairs of the following problems:*

- BP-Satisfying-Pair;
- RegExp-String-Pair;
- $2^{(\log D)^{1-\Omega(1)}}$ -Gap-Closest-RegExp-String-Pair.

Proof. By Theorem 7.1 and 7.4, we can show cyclic reductions between these three problems in a similar way as in the proof of Theorem 6.5. \square

We can also show a reduction from Restricted OAPT to Regular Expression Membership Testing on two strings using the same gadgets in Theorem 7.3.

Theorem 7.6. *There exists an $O(N \text{poly}(D))$ -time reduction from a Restricted OAPT instance with two sets A, B of N tensors of size D to an instance of Regular Expression Membership Testing on a regular expression R of length $O(N \text{poly}(D))$ and a string S of length $O(N \text{poly}(D))$.*

Proof. We construct the two sets A', B' as in Theorem 7.1. For the construction for the regular expression, let w be the concatenation of all $a' \in A$ separated by “ $|$ ”. Then we construct the regular expression R to be

$$R = [\bigcirc_{i=1}^D [0 \mid 1]]^* \circ [w] \circ [\bigcirc_{i=1}^D [0 \mid 1]]^*$$

and we construct the string S by concatenating all $b' \in B$ directly. It is easy to see that there exists a pair $(a, b) \in A \times B$ with $p_{\text{alt}}(a, b) = 0$ iff $S \in L(R)$, by noticing that all the strings $x \in L(w), b' \in B$ are of the same length D . \square

8 Subtree Isomorphism and Largest Common Subtree

In this section, we study the hardness of Subtree Isomorphism and Largest Common Subtree. Our reductions here are inspired by [ABH⁺16, AR18]. We begin with some notations to ease our construction of trees. Recall that all trees considered in this paper are bounded-degree and unordered. We are interested in both rooted and unrooted trees. Here “rooted” means that the root of G must be mapped to the root of H in the isomorphism.

We use \mathcal{T}_2 to denote the tree with exactly two nodes. Let \mathcal{T}_3^0 be the 3-node tree with root degree 1, and let \mathcal{T}_3^1 be the 3-node tree with root degree 2. For a tree T , let $\mathcal{P}^k(T)$ be the tree constructed by joining a path of k nodes and the tree T : one end of the path is regarded as the root, the other end of the path is linked to the root of T by an edge. For two trees \mathcal{T}_a and \mathcal{T}_b , we use $(\mathcal{T}_a \circ \mathcal{T}_b)$ to denote the tree whose root has two children \mathcal{T}_a and \mathcal{T}_b .

8.1 Subtree Isomorphism

In this subsection, first we prove that BP-Satisfying-Pair and Subtree-Isomorphism-Pair are equivalent under near-linear time reductions, then we show the hardness for Subtree Isomorphism on two trees.

For two trees T_a, T_b , we use $\text{STI}(T_a, T_b)$ to indicate whether T_a is isomorphic to a subtree of T_b when T_a, T_b are seen as unrooted trees. Also, we use $\text{RSTI}(a, b)$ to indicate whether T_a is isomorphic to a subtree of T_b when T_a, T_b are seen as rooted trees.

Theorem 8.1. *Let t be an even number and $d_1 = \dots = d_t = 2$. Given two sets of N tensors A, B in $\{0, 1\}^{d_1 \times \dots \times d_t}$ satisfying that all the tensors in A are \wedge -invariant, there is a deterministic algorithm running in $O(N \text{poly}(2^t))$ time which outputs two sets A', B' of N binary trees of size $O(2^t)$ and depth $O(t)$, such that each $a \in A$ corresponds to a tree $a' \in A'$, each $b \in B$ corresponds to a tree $b' \in B'$, and*

$$\overline{p_{\text{alt}}(a, b)} = \text{RSTI}(a', b') = \text{STI}(a', b'),$$

where $\overline{p_{\text{alt}}(a, b)}$ is the negation of the Alternating Product of a and b .

Proof. For each k and each prefix of index $i_{(k)} \in [d_1] \times \dots \times [d_k]$, we construct corresponding tree gadgets $G_{i_{(k)}}(a)$ and $H_{i_{(k)}}(b)$ for each $a \in A$ and $b \in B$ (when $k = 0$, $i_{(k)}$ can only be the empty prefix, and we simply use $G(a)$ and $H(b)$ for convenience) inductively, mimicking the evaluation of the alternating product. Our gadgets satisfy that

$$\text{RSTI}(G_{i_{(k)}}(a), H_{i_{(k)}}(b)) = \overline{p_{\text{alt}}(a_{i_{(k)}}, b_{i_{(k)}})}$$

for any subtensors $a_{i_{(k)}}, b_{i_{(k)}}$. For this purpose, we need to construct the following three types of tree gadgets.

Bit Gadgets. First we need bit gadgets to simulate the innermost coordinatewise product in the alternating product. For each coordinate $i \in [d_1] \times \dots \times [d_t]$, for every $a \in A$ and $b \in B$, we construct

$$G_i(a) = \begin{cases} \mathcal{T}_2 & \text{if } a_i = 0, \\ \mathcal{T}_3^1 & \text{if } a_i = 1, \end{cases} \quad \text{and} \quad H_i(b) = \begin{cases} \mathcal{T}_3^1 & \text{if } b_i = 0, \\ \mathcal{T}_2 & \text{if } b_i = 1. \end{cases}$$

It is easy to see that $\text{RSTI}(G_i(a), H_i(b))$ iff $a_i \wedge b_i = 0$.

Now we combine bit gadgets recursively according to the \wedge and \vee operators in the Alternating Product. Starting from $k = t - 1$, there are two cases to consider.

AND Gadgets. The first case is when \wedge operator is applied to $(k + 1)$ -th dimension, then by De Morgan's laws, we need to construct our gadgets such that for all $i_{(k)} \in [d_1] \times \cdots \times [d_k]$

$$\text{RSTI}(G_{i_{(k)}}(a), H_{i_{(k)}}(b)) = \text{RSTI}(G_{i_{(k)},0}(a), H_{i_{(k)},0}(b)) \vee \text{RSTI}(G_{i_{(k)},1}(a), H_{i_{(k)},1}(b)).$$

To do so, for $a \in A$, we construct $G_{i_{(k)}}(a)$ to be

$$G_{i_{(k)}}(a) = \mathcal{P}^1(G_{i_{(k)},0}(a)) = \mathcal{P}^1(G_{i_{(k)},1}(a)),$$

which is well-defined since a is \wedge -invariant. And we construct $H_{i_{(k)}}(b)$ to be

$$H_{i_{(k)}}(b) = (H_{i_{(k)},0}(b) \circ H_{i_{(k)},1}(b)).$$

In any subtree isomorphism, it is easy to see that $G_{i_{(k)},0}(a)$ (or $G_{i_{(k)},1}(a)$) can only be mapped to either $H_{i_{(k)},0}(b)$ or $H_{i_{(k)},1}(b)$, so $G_{i_{(k)}}(a), H_{i_{(k)}}(b)$ implement an \wedge operator.

OR Gadgets. The second case is when \vee operator is applied to $(k + 1)$ -th dimension, then by De Morgan's laws, we need to construct our gadgets such that for all $i_{(k)} \in [d_1] \times \cdots \times [d_k]$,

$$\text{RSTI}(G_{i_{(k)}}(a), H_{i_{(k)}}(b)) = \text{RSTI}(G_{i_{(k)},0}(a), H_{i_{(k)},0}(b)) \wedge \text{RSTI}(G_{i_{(k)},1}(a), H_{i_{(k)},1}(b)).$$

First for any tree T , we define two auxiliary trees $\mathcal{U}_0(T), \mathcal{U}_1(T)$ to ease our construction:

$$\mathcal{U}_0(T) = (\mathcal{P}^3(T) \circ \mathcal{T}_3^0) \quad \text{and} \quad \mathcal{U}_1(T) = (\mathcal{P}^3(T) \circ \mathcal{T}_3^1).$$

It is easy to verify that for any two trees T_1, T_2 , $\text{RSTI}(\mathcal{U}_0(T_1), \mathcal{U}_1(T_2)) = \text{RSTI}(\mathcal{U}_1(T_1), \mathcal{U}_0(T_2)) = 0$ and $\text{RSTI}(\mathcal{U}_0(T_1), \mathcal{U}_0(T_2)) = \text{RSTI}(\mathcal{U}_1(T_1), \mathcal{U}_1(T_2)) = \text{RSTI}(T_1, T_2)$.

We construct the corresponding tensor gadgets $G_{i_{(k)}}(a)$ and $H_{i_{(k)}}(b)$ for $a \in A$ and $b \in B$ as follows:

$$G_{i_{(k)}}(a) = (\mathcal{U}_0(G_{i_{(k)},0}(a)) \circ \mathcal{U}_1(G_{i_{(k)},1}(a))),$$

and

$$H_{i_{(k)}}(b) = (\mathcal{U}_0(H_{i_{(k)},0}(b)) \circ \mathcal{U}_1(H_{i_{(k)},1}(b))).$$

In any subtree isomorphism, it is easy to see that $\mathcal{U}_0(G_{i_{(k)},0}(a))$ can only be mapped to $\mathcal{U}_0(H_{i_{(k)},0}(b))$, and $\mathcal{U}_1(G_{i_{(k)},1}(a))$ can only be mapped to $\mathcal{U}_1(H_{i_{(k)},1}(b))$, so $G_{i_{(k)}}(a), H_{i_{(k)}}(b)$ implement an \vee operator.

Correctness. It is not hard to verify that $\text{RSTI}(G(a), H(b)) = \overline{p_{\text{alt}}(a, b)}$ by De Morgan's laws. To show $\text{RSTI}(G(a), H(b)) = \text{STI}(G(a), H(b))$, we focus on the case that $t > 0$ since the case that $t = 0$ is obvious. Let the root of G_A be r and the height of G_A be h . The outermost operator in an Alternating Product is \wedge , so r has only one child which has two subtrees of equal height $h - 2$. It is easy to see that the height of H_B is also h . Suppose that r is mapped to a node r' in H_B and c is mapped to c' . If we regard c' as the root of H_B , then after deleting c' , H_B should be split into two subtrees of height $\geq h - 2$ and a single node r' . The only possible case is that c' is of depth 1 w.r.t. the original root of H_B (the depth of a root is 0) and r' is the original root of H_B . \square

Theorem 8.2. *Given two bounded-degree unrooted trees T_A and T_B , it can be decided in $\text{NSPACE}[(\log n)^2]$ that whether T_A is isomorphic to a subtree of T_B .*

Proof. This algorithm works by divide and conquer on trees. At each recursion, we have two trees S_A and S_B (implicit representation) as well as a set of node pairs $M = \{(a_1, b_1), \dots, (a_k, b_k)\}$ (initially, $S_A = T_A, S_B = T_B$ and $M = \emptyset$). We need to decide whether there is an isomorphism from S_A to some subtree of S_B satisfying a_i in S_A is mapped to b_i in S_B for all $1 \leq i \leq k$.

First we find a centroid c of S_A , i.e., a node of S_A that decomposes S_A into subtrees of size at most $\lceil |S_A|/2 \rceil$ when the node is deleted. Then we nondeterministically guess a node c' in S_B to be the node that is mapped by c in the isomorphism. If $c = a_i$ for some i but $c' \neq b_i$, then we reject; otherwise, we guess an injective mapping from the neighbors of c in S_A to the neighbors of c' in S_B .

For each neighbor v of c , let v' be the neighbor of c' mapped by v , S_A^v be the subtree of S_A containing v when the edge between v and c is deleted, $S_B^{v'}$ be the subtree of S_B containing v' when the edge between v' and c' is deleted. We create a new set of node pairs $M' = \{(a_i, b_i) \in M \mid a_i \in S_A^v\}$. If $b_i \notin S_B^{v'}$ for some pair $(a_i, b_i) \in M'$, then we reject; otherwise, we recursively checking if there is an isomorphism from S_A^v to some subtree of $S_B^{v'}$ satisfying a_i is mapped to b_i for all $(a_i, b_i) \in M'$ and v is mapped to v' .

This algorithm terminates when S_A is a single node. There are at most $O(\log n)$ levels of recursion by the property of centroid. At each level, we use only $O(\log n)$ space for c, c' and their neighbors (note that S_A, S_B are bounded-degree trees), and S_A, S_B can always be accessed according to the information stored at the upper levels of recursion. Thus this algorithm runs in $\text{NSPACE}[(\log n)^2]$. \square

Combining the above reductions together, we can show the equivalence between BP-Satisfying-Pair and Subtree-Isomorphism-Pair.

Theorem 8.3. *BP-Satisfying-Pair on branching program of size $2^{(\log N)^{o(1)}}$ and Subtree-Isomorphism-Pair on (rooted or unrooted) trees of size $2^{(\log N)^{o(1)}}$ are equivalent under near-linear time reductions.*

We can also show a reduction from OAPT to Subtree Isomorphism on two trees using the same gadgets in Theorem 8.1.

Theorem 8.4. *There exists an $O(N \text{poly}(D))$ -time reduction from a Restricted OAPT instance with two sets A, B of N tensors of size D to an instance of Subtree Isomorphism on two (rooted or unrooted) binary trees of size $O(N \text{poly}(D))$ and depth $2 \log N + O(\log D)$.*

Proof. Using the recursive construction in Theorem 8.1 we can obtain tensor gadgets $G(a), H(b)$ for each $a \in A$ and $b \in B$, such that $p_{\text{alt}}(a, b) = \text{RSTI}(G(a), H(b)) = \text{STI}(G(a), H(b))$.

We can assume the set size N is a power of 2 by adding dummy vectors into each set. Now we combine the tensor gadgets in each set respectively to construct two trees G_A, H_B as our instance for Subtree Isomorphism:

a) To construct G_A for set A :

- Initialize G_A by a complete binary tree of N leaves;
- Associate each leaf with a tensor $a \in A$;
- For all $a \in A$, construct $\mathcal{P}^{\log N}(G(a))$ and link an edge from its root to the corresponding leaf of a .

b) To construct H_B for set B :

- Initialize H_B by a complete binary tree of N leaves;
- Select one leaf node v_ℓ ;
- For every unselected leaf, construct $\mathcal{P}^{\log N}(H(\mathbf{0}))$ and link an edge from its root to the leaf.
- Construct a complete binary tree of N leaves rooted at v_ℓ ;
- Associate each leaf of the tree rooted at v_ℓ with a tensor $b \in B$;
- For all $b \in B$, construct $\mathcal{P}^{\log N}(H(b))$ and link an edge from its root to the corresponding leaf of b .

Correctness For any subtree isomorphism, one $G(a)$ can be mapped to any $H(b)$ or $H(\mathbf{0})$. Since there are only $N - 1$ gadgets of $H(\mathbf{0})$, there must be some $G(a)$ mapped to some $H(b)$. Thus $\text{RSTI}(G_A, H_B)$ iff there exists a pair of $(a, b) \in A \times B$ with $p_{\text{alt}}(a, b) = 0$. It is not hard to see that the root of G_A can only be mapped to the root of H_B by arguing about the tree height (similar as Theorem 8.1), so $\text{RSTI}(G_A, H_B) = \text{STI}(G_A, H_B)$. \square

8.2 Largest Common Subtree

In this subsection, first we prove that under near-linear time reductions between BP-Satisfying-Pair and (exact or approximate) Max-LCST-Pair / Min-LCST-Pair are equivalent, then we show the hardness for Largest Common Subtree on two trees.

For two trees a, b , define $\text{LCST}(a, b)$ to be the size of the largest common subtree of a and b when a, b are seen as unrooted trees. Also, we define $\text{RLCST}(a, b)$ to be the size of the largest common subtree of a and b when a, b are seen as rooted trees.

Now we establish a connection between Restricted Max-TropSim and Max-LCST-Pair:

Theorem 8.5. *Let t be an even number and $d_1 = \dots = d_t = 2$. Given two sets of N tensors A, B in $\{0, 1\}^{d_1 \times \dots \times d_t}$ satisfying that all the tensors in A are max-invariant, for any $L \geq 2^t$, there is a deterministic algorithm running in $O(N \cdot \text{poly}(2^t) \cdot L)$ time which outputs two sets A', B' of N binary trees of size $O(\text{poly}(2^t) \cdot L)$ and depth $O((2^{t/2} + \log L) \cdot t)$, such that each $a \in A$ corresponds to a tree $a' \in A'$, each $b \in B$ corresponds to a tree $b' \in B'$, and*

$$\begin{aligned}\text{RLCST}(a', b') &= (2^{t/2}s(a, b) + O(1))L \\ \text{LCST}(a', b') &= (2^{t/2}s(a, b) + O(1))L\end{aligned}$$

where $s(a, b)$ is the Tropical Similarity score of a and b . In particular, if $s(a, b) = 1$, then a', b' satisfy $\text{RLCST}(a', b') = \text{LCST}(a', b') = |a'|$.

Proof. For each k and each prefix of index $i_{(k)} \in [d_1] \times \dots \times [d_k]$, we construct corresponding gadget $a' = G_{i_{(k)}}(a)$ and $b' = H_{i_{(k)}}(b)$ for each $a \in A$ and $b \in B$ (when $k = 0$, $i_{(k)}$ can only be the empty prefix, and we simply use $G(a)$ and $H(b)$ for convenience) inductively, mimicking the evaluation of the Tropical Similarity. For this purpose, we need to construct the following three types of gadgets.

Bit Gadgets. For each coordinate $i \in [d_1] \times \dots \times [d_t]$, let \mathcal{C}_i be the tree constructed by join a path of length $2^{t/2}$ and a complete binary tree of L nodes: one end of the path is regarded as the root, and we link an edge between the node of depth $\text{bin}_{\text{odd}}(i)$ and the root of the complete binary tree, where $\text{bin}_{\text{odd}}(i) \in [0, 2^{t/2})$ is the number whose binary representation is $i_1 i_3 \dots i_{t-1}$.

For every $a \in A$ and for each coordinate $i \in [d_1] \times \dots \times [d_t]$, we construct $G_i(a) = \mathcal{C}_i$ if $a_i = 1$, or simply a path of length $2^{t/2}$ if $a_i = 0$. Similarly, for every $b \in B$ and for each coordinate $i \in [d_1] \times \dots \times [d_t]$, we construct $H_i(b) = \mathcal{C}_i$ if $b_i = 1$, or simply a path of length $2^{t/2}$ if $b_i = 0$.

If $a_i \cdot b_i = 0$, then $\text{RLCST}(G_i(a), H_i(b)) = 2^{t/2}$; otherwise $\text{RLCST}(G_i(a), H_i(b)) = 2^{t/2} + L$. Furthermore, for any two coordinates i, j with $\text{bin}_{\text{odd}}(i) \neq \text{bin}_{\text{odd}}(j)$, $\text{RLCST}(G_i(a), H_j(b)) = 2^{t/2}$.

Let $K = 2^{t/2} + \lceil \log L \rceil + 1$ be the maximum possible height of a bit gadget. Now we combine bit gadgets recursively according to the \mathbb{E} and max operators in the evaluation of Tropical Similarity score. Starting from $k = t - 1$, there are two cases to consider.

Expectation Gadgets. The first case is when \mathbb{E} operator is applied to $(k + 1)$ -th dimension. For $i_{(k)} \in [d_1] \times \dots \times [d_k]$ and $a \in A$, we construct $G_{i_{(k)}}(a), H_{i_{(k)}}(b)$ as follows:

$$G_{i_{(k)}}(a) = \left(\mathcal{P}^{K-1}(G_{i_{(k)},0}(a)) \circ \mathcal{P}^{K-1}(G_{i_{(k)},1}(a)) \right)$$

and

$$H_{i(k)}(a) = \left(\mathcal{P}^{K-1}(H_{i(k),0}(b)) \circ \mathcal{P}^{K-1}(H_{i(k),1}(b)) \right)$$

Max Gadgets. The second case is when max operator is applied to $(k+1)$ -th dimension. For $i_{(k)} \in [d_1] \times \cdots \times [d_k]$ and $a \in A$, we construct $G_{i(k)}(a), H_{i(k)}(b)$ as follows:

$$G_{i(k)}(a) = \mathcal{P}^K(G_{i(k),0}(a)) = \mathcal{P}^K(G_{i(k),1}(a))$$

and

$$H_{i(k)}(a) = \left(\mathcal{P}^{K-1}(H_{i(k),0}(b)) \circ \mathcal{P}^{K-1}(H_{i(k),1}(b)) \right).$$

Note that $G_{i(k)}(a)$ is well-defined since a is max-invariant.

Finally we obtain tensor gadgets $G(a), H(b)$ for every $a \in A, b \in B$. It is easy to see that the depth of trees is $O(tK) \leq O((2^{t/2} + \log L) \cdot t)$, and the size of trees is $O(L \cdot 2^t + K \cdot 2^t) = O(\text{poly}(2^t) \cdot L)$.

Correctness for RLCST. First we show that $\text{RLCST}(G(a), H(b)) = (2^{t/2}s(a, b) + O(1))L$. We fix two tensors a, b . For every dimension k , let U_k be a set of gadget pairs:

$$U_k = \{(G_{i(k)}(a), H_{j(k)}(b)) \mid i_{(k)}, j_{(k)} \in [d_1] \times \cdots \times [d_k], i_p \neq j_p \text{ for some odd } p\}.$$

Let ε_k be the maximum RLCST among the pairs in U_k .

For every $i_{(k)} \in [d_1] \times \cdots \times [d_t]$, let $f(i_{(k)}) = \text{RLCST}(G_{i(k)}(a), H_{i(k)}(b))$. For the last dimension t , it is easy to see that $f(i) = 2^{t/2} + (a_i \cdot b_i) \cdot L$ and $\varepsilon_t = 2^{t/2}$. Now we prove by induction that $f(i_{(k)}) \geq \varepsilon_k$ holds for every $0 \leq k \leq t$ and $i_{(k)} \in [d_1] \times \cdots \times [d_k]$.

On the one hand, if an \mathbb{E} operator is applied to the $(k+1)$ -th dimension, by induction hypothesis we have $f(i_{(k)}, 0) + f(i_{(k)}, 1) \geq 2\varepsilon_{k+1}$, so $f(i_{(k)}) = f(i_{(k)}, 0) + f(i_{(k)}, 1) + 2(K-1) + 1$. Note that $\varepsilon_k \leq 2\varepsilon_{k+1} + 2(K-1) + 1$, so $f(i_{(k)}) \geq \varepsilon_k$ holds. On the other hand, If the max operator is applied to the $(k+1)$ -th dimension, then we have $f(i_{(k)}) = \max\{f(i_{(k)}, 0), f(i_{(k)}, 1)\} + K$ and $\varepsilon_k \leq \varepsilon_{k-1} + K$, so $f(i_{(k)}) \geq \varepsilon_k$ holds as well.

Expanding the above recurrence relation of $f(i_{(k)})$, we have

$$\begin{aligned} \text{RLCST}(G(a), H(b)) &= 2^{t/2}s(a, b)L + O(K) \cdot 2^{t/2} \\ &= 2^{t/2}s(a, b)L + O(2^{t/2} + \log L) \cdot 2^{t/2} \\ &= (2^{t/2}s(a, b) + O(1))L. \end{aligned}$$

Correctness for LCST. Now we show that $\text{LCST}(G(a), H(b)) = \text{RLCST}(G(a), H(b)) + O(L)$. Fix a pair of $(a, b) \in A \times B$. If a node of $G(a)$ or $H(b)$ is in a bit gadget, then we call it *bit node*. If a node of $G(a)$ or $H(b)$ is not in any bit gadget, then we call it *operator node*.

Let $I_G(a)$ and $I_H(b)$ be the largest isomorphic subtrees in $G(a)$ and $H(b)$. Let r_a be the root of $I_G(a)$, i.e., the lowest node when the tree is directed with respect to the root of $G(a)$, and let r_b be the root of $I_H(b)$. Let r'_b be the node in $G(a)$ that is mapped to r_b , and r'_a be the node in $H(b)$ that is mapped from r_a .

If $r_a = r'_b$, then let $q = 1$ and $u_1 = r_a, u'_1 = r_b$. Otherwise, let u_1, \dots, u_q be the list of nodes that are in $I_G(a)$ and are adjacent to some node on the path from r_a to r'_b . Assume u_1, \dots, u_q is in depth-increasing order (it is easy to see that no two such nodes are of same depth). Let u'_1, \dots, u'_q be the nodes in $I_H(b)$ that are mapped by u_1, \dots, u_q , respectively. Each node in u'_1, \dots, u'_q should be adjacent to some node on the path from r'_a to r_b in $I_H(b)$.

For a node u_i , we denote the whole subtree of u_i in $G(a)$ as T_{u_i} , and we define $T'_{u'_i}$ similarly. We can decompose the subtree $I_G(a)$ into two parts: the first part is the path from r_a to r'_b , and the second part is

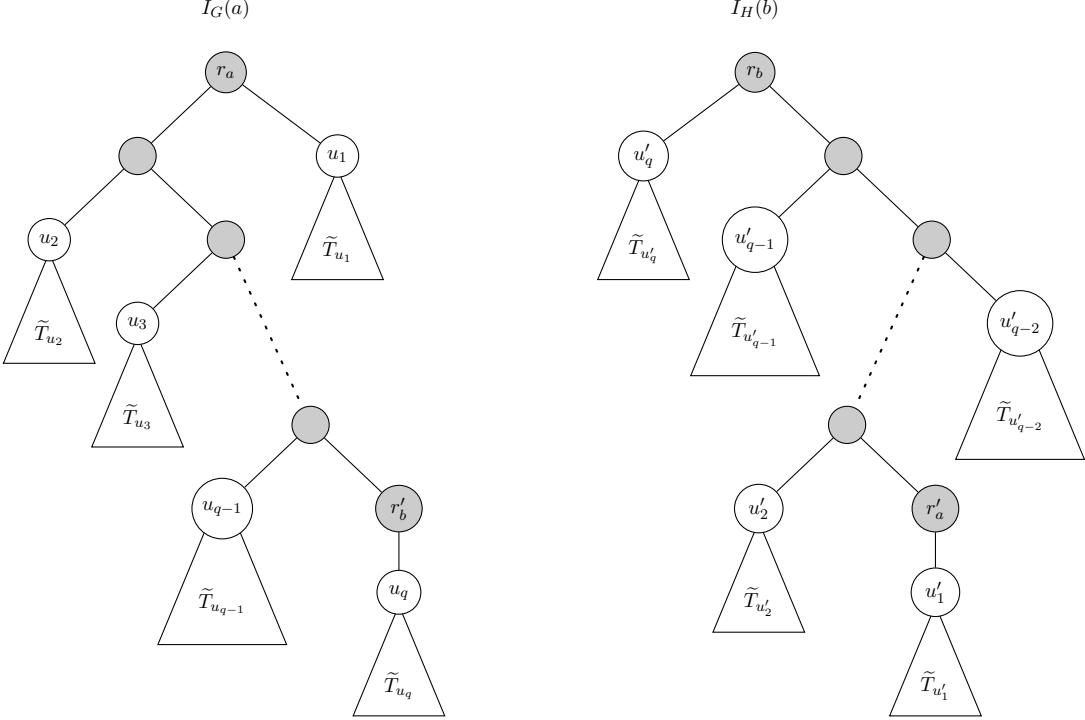


Figure 2: An illustration of $I_G(a)$ and $I_H(b)$.

the q rooted subtrees $\tilde{T}_{u_1} = T_{u_1} \cap I_G(a), \dots, \tilde{T}_{u_q} = T_{u_q} \cap I_G(a)$. Similarly, we can decompose $I_H(b)$ into the path from r_b to r'_a and q rooted subtrees $\tilde{T}_{u'_1} = T_{u'_1} \cap I_H(b), \dots, \tilde{T}_{u'_q} = T_{u'_q} \cap I_H(b)$. Thus we have

$$\text{LCST}(G(a), H(b)) = q + \sum_{i=1}^q |\tilde{T}_{u_i}| \leq O(tK) + \sum_{i=1}^q \text{RLCST}(T_{u_i}, T'_{u'_i})$$

It is sufficient to obtain a bound for the sum of RLCST of $T_{u_i}, T'_{u'_i}$. If some u_j is a bit node, then u_i is also a bit node for all $i > j$, and all of them are in the same bit gadget, so $\sum_{i=j}^q \text{RLCST}(T_{u_i}, T'_{u'_i}) \leq O(2^{t/2} + L)$. Similarly, if some u'_j is a bit node, then u'_i is also a bit node for all $i < j$, and all of them are in the same bit gadget, so $\sum_{i=1}^j \text{RLCST}(T_{u_i}, T'_{u'_i}) \leq O(2^{t/2} + L)$.

Now we consider the following three cases when both u_i and u'_i are operator nodes: ($\text{depth}(u_i)$ stands for the depth of u_i in $G(a)$, $\text{depth}(u'_i)$ stands for the depth of u'_i in $H(b)$)

Case 1. $\text{depth}(u_i) \not\equiv \text{depth}(u'_i) \pmod{K}$, then it is impossible to map some operator node with two children in T_{u_i} to an operator node with two children in $T'_{u'_i}$. If $\text{depth}(u_i) > \text{depth}(u'_i)$, then at most one bit gadget in T_{u_i} can have nodes being mapped to $T'_{u'_i}$, and the case that $\text{depth}(u_i) < \text{depth}(u'_i)$ is similar. Thus $\text{RLCST}(T_{u_i}, T'_{u'_i}) \leq O(tK) + (2^{t/2} + L) = O(tK + L)$.

Note that the depth of parent nodes of u_i and u'_i should also be different modulo K , so either u_i has no parent in $I_G(a)$ (this is the case when $r_a = r'_b$) or the parent of u_i has only one child in $I_G(a)$, and either case implies $i = q$.

Case 2. If $\text{depth}(u_i) \equiv \text{depth}(u'_i) \pmod{K}$ but $\text{depth}(u_i) > \text{depth}(u'_i)$, then since K is an upper bound of the height of any bit gadget, all the nodes in T_{u_i} can only be mapped to the operator nodes in $T'_{u'_i}$,

which implies $\text{RLCST}(T_{u_i}, T'_{u'_i})$ is no more than the number of operator nodes in $T'_{u'_i}$. The case that $\text{depth}(u_i) \equiv \text{depth}(u'_i) \pmod{K}$ but $\text{depth}(u_i) < \text{depth}(u'_i)$ is similar.

All the trees $T'_{u'_i}$ are disjoint. Thus the sum $\sum_i \text{RLCST}(T_{u_i}, T'_{u'_i})$ over all i in this case can be upper-bounded by the total number of operator nodes in $G(a)$, which is $O(K \cdot 2^{t/2})$.

Case 3. If $\text{depth}(u_i) = \text{depth}(u'_i)$, then there exists two prefixes of index $i_{(k)}, j_{(k)}$, such that

$$\text{RLCST}(T_{u_i}, T_{u'_i}) = \text{RLCST}(G_{i_{(k)}}(a), H_{j_{(k)}}(b)) + O(K)$$

Note that u_1, \dots, u_q are in depth-increasing order, and u'_q, \dots, u'_1 are in depth-decreasing order, so this case can only happen for at most one pair of nodes.

Summing up all the above cases, we have

$$\text{LCST}(G(a), H(b)) \leq O(tK) + O(2^{t/2} + L) + O(tK + L) + O(K \cdot 2^{t/2}) + \text{RLCST}(G_{i_{(k)}}(a), H_{j_{(k)}}(b))$$

for any $i_{(k)}, j_{(k)}$. Thus $\text{LCST}(G(a), H(b)) \leq \text{RLCST}(G(a), H(b)) + O(L)$. \square

By Theorem 8.5 with $L = 2^t$, we can easily have the following reductions from $\varepsilon(D)$ -Gap-Max-TropSim and $\varepsilon(D)$ -Gap-Min-TropSim to approximation variants of Max-LCST-Pair and Min-LCST-Pair. Here we focus on the case that $\varepsilon(D) = \Omega(D^{-1/2})$. It is reasonable since $\sqrt{D} \cdot s(a, b)$ is always an integer, and $o(D^{-1/2})$ -Gap-Max-TropSim is essentially equivalent to Max-TropSim. This argument also holds for $o(D^{-1/2})$ -Gap-Min-TropSim.

Theorem 8.6. *For any function $\varepsilon(D) = \Omega(D^{-1/2})$, there exists an $O(N \text{poly}(D))$ -time reduction from a Restricted $\varepsilon(D)$ -Gap-Max-TropSim instance with two sets of N tensors of size D to an instance of the following approximation variant of Max-LCST-Pair: Given two sets A, B of N trees of size $\text{poly}(D)$ and a set B of N strings of length D over a constant-size alphabet, distinguish between the following:*

- **Completeness:** There exists a pair of a, b such that $\text{LCST}(a, b) = |a| = (1 + o(1))D^{3/2}$;
- **Soundness:** For every pair a, b , $\text{LCST}(a, b) \leq O(\varepsilon(D)D^{3/2})$.

And this conclusion also holds for RLCST.

Remark 8.7. Note that Theorem 8.6 also implies a reduction from BP-Satisfying-Pair to Subtree-Isomorphism-Pair, but the trees constructed by the reduction in Theorem 8.1 have a smaller size and a lower depth.

Theorem 8.8. *For any function $\varepsilon(D) = \Omega(D^{-1/2})$, there exists an $O(N \text{poly}(D))$ -time reduction from a Restricted $\varepsilon(D)$ -Gap-Min-TropSim instance with two sets of N tensors of size D to an instance of the following approximation variant of Min-LCST-Pair: Given two sets A, B of N trees of size $\text{poly}(D)$ and a set B of N strings of length D over a constant-size alphabet, distinguish between the following:*

- **Completeness:** There exists a pair of a, b such that $\text{LCST}(a, b) \leq O(\varepsilon(D)D^{3/2})$;
- **Soundness:** For every pair a, b , $\text{LCST}(a, b) = |a| = (1 + o(1))D^{3/2}$.

And this conclusion also holds for RLCST.

By Theorem 4.2, we show reductions from Max-LCST-Pair (or Min-LCST-Pair) to BP-Satisfying-Pair via the following theorem:

Theorem 8.9. *Given two bounded-degree unrooted trees T_A and T_B and a number q , it can be decided in $\text{NSPACE}[(\log n)^2]$ that whether there is an isomorphism between a subtree of T_A and a subtree of T_B of size q .*

Proof. The algorithm in Theorem 8.2 suffices to fulfill the requirement if modified slightly. At each level of recursion, we have two trees S_A and S_B , a number q , and a set of node pairs $M = \{(a_1, b_1), \dots, (a_k, b_k)\}$. We need to decide whether there is an isomorphism from a subtree of S_A to a subtree of S_B satisfying it is of size q and a_i in S_A is mapped to b_i in S_B for all $1 \leq i \leq k$.

First we find a centroid c of S_A , then we guess if there is a subtree of size q that contains c and is isomorphic to a subtree of S_B . If not, then we delete c to decompose S_A into subtrees, guess which subtree contains a subtree that is isomorphic to a subtree of S_B of size q , and runs our algorithm to check recursively; If it is, then follow the same routine as in Theorem 8.2: we guess a node c' in S_B to be the node that mapped by c in the isomorphism and a bijective mapping from some of the neighbors of c in S_A to some of the neighbors of c' in S_B . Additionally, we guess a number q_v for each neighbor v of c and ensure the sum of q_v over all neighbors equals to $q - 1$. We then check recursively if there is an isomorphism from a subtree of S_A^v to a subtree of $S_B^{v'}$ of size q subject to the constraint that some set of node pairs are matched. It is clear that this algorithm runs in $\text{NSPACE}[(\log n)^2]$. \square

Theorem 8.10. *For the case that the maximum element size $D = 2^{(\log N)^{o(1)}}$, there are near-linear time reductions between all pairs of the following problems:*

- *BP-Satisfying-Pair;*
- *Max-LCST-Pair or Min-LCST-Pair on (rooted or unrooted) trees;*
- $2^{(\log D)^{1-\Omega(1)}}$ -approximate *Max-LCST-Pair or Min-LCST-Pair on (rooted or unrooted) trees;*

Proof. By Theorem 8.6, 8.8 and 8.9, we can show cyclic reductions between these three problems in a similar way as in the proof of Theorem 6.5. \square

We can also show a reduction from $\varepsilon(N)$ -Gap-Max-TropSim to Largest Common Subtree on two large trees using the same gadgets in Theorem 8.5.

Theorem 8.11. *Let t be an even number and $d_1 = \dots = d_t = 2$. Given two sets of N tensors A, B in $\{0, 1\}^{d_1 \times \dots \times d_t}$, for $L = \Theta(2^t \log^2 N)$, there is a deterministic algorithm running in $O(N \log^2 N 2^{O(t)})$ time which outputs two binary trees A', B' of size $O(N \log^2 N 2^{O(t)})$ and depth $O(\log^2 N \text{poly}(t) 2^{t/2})$, such that*

$$\begin{aligned} RLCST(A', B') &= (2^{t/2} s_{\max} + O(1))L \\ LCST(A', B') &= (2^{t/2} s_{\max} + O(1))L \end{aligned}$$

where s_{\max} is the maximum Tropical Similarity among all pairs of $(a, b) \in A \times B$.

Proof. Using the recursive construction in Theorem 8.5, we can obtain tensor gadgets $G(a), H(b)$ for each $a \in A$ and $b \in B$. Let m be the smallest number such that $N \leq 2^m$. Now we combine the tensor gadgets in each set respectively to construct two trees G_A, H_B as our instance for Largest Common Subtree.

For any number K , we define K -zoomed complete binary tree \mathcal{Z}_K of 2^m as follows: first we construct a complete binary tree of 2^m leaves, then we insert $K - 1$ internal nodes between every pair of adjacent nodes (so \mathcal{Z}_K is of height $mK + 1$).

Let $K_D = O((2^{t/2} + \log L)t)$ be the maximum diameter of any tensor gadget ($G(a)$ or $H(b)$). Let $K_G = 2(m + 1)K_D$.

We construct $G_A = \mathcal{P}^{mK_G+1}(T_A)$, where T_A is the following auxiliary tree:

- Initialize $T_A = \mathcal{Z}_{K_D}$, and arbitrarily select N leaves;
- For each selected leaf, associate it with a tensor $a \in A$;
- Construct $G(a)$ for every $a \in A$, and link an edge from its root to the corresponding leaf of a .

And the tree H_B for set B is constructed as follows:

- Initialize $H_B = \mathcal{Z}_{K_G}$ and arbitrarily select N leaves;
- For each selected leaf, associate it with a tensor $b \in B$;
- Construct $\mathcal{P}^{mK_D+1}(b)$ for every $b \in B$, and link an edge from its root to the corresponding leaf of b .

Proof for RLCST. Note that all the tensor gadgets are of same depth, and only one gadget $G(a)$ in G_A can be mapped to a gadget $H(b)$ in H_B .

For $L = \Theta(2^t \log^2 N)$, using the fact that $\text{RLCST}(G(a), H(b)) = (2^{t/2} s(a, b) + O(1))L$, one can easily show that

$$\text{RLCST}(G_A, H_B) = (2^{t/2} s_{\max} + O(1))L + O(mK_G + mK_D) = (2^{t/2} s_{\max} + O(1))L.$$

Proof for LCST. Now we show that $\text{LCST}(G_A, H_B) \leq \text{LCST}(G(a), H(b)) + O(L)$ for some $(a, b) \in A \times B$.

If a node of G_A or H_B is in a tensor gadget, then we call it *tensor node*. If a node of $G(a)$ or $H(b)$ is not in any tensor gadget, then we call it *assembly node*. Let G'_A be the tree G_A with all tensor nodes removed, and we define H'_B respectively.

We consider the following three cases:

Case 1. If none of tensor node of G_A is in the LCST, then

$$\text{LCST}(G_A, H_B) = \text{LCST}(G'_A, H_B) \leq \text{LCST}(P_A, H_B) + \text{LCST}(\mathcal{Z}_{K_D}, H_B),$$

where P_A is the path of length $mK_G + 1$ linked with the root of T_A . It is easy to see that $\text{LCST}(P_A, H_B) \leq mK_G + 1$. Note that every pair of two tensor node from different tensor gadgets has distance at least $2K_G$, which is greater than the diameter of T_A , so the isomorphic subtree of T_A in H_B cannot contain nodes from more than one tensor gadgets. By noticing that $\text{LCST}(\mathcal{Z}_{K_D}, H(b)) = O(K_D)$ for all $b \in B$ and $\text{LCST}(\mathcal{Z}_{K_D}, \mathcal{Z}_{K_G}) = O(K_G)$, we have $\text{LCST}(\mathcal{Z}_{K_D}, H_B) = O(K_G)$. Thus $\text{LCST}(G_A, H_B) = mK_G + O(K_G) = O(mK_G)$.

Case 2. If LCST contains some tensor nodes of G_A , and all such tensor nodes are mapped to assembly nodes of H_B , then $\text{LCST}(G_A, H_B)$ is no more than $\text{LCST}(G'_A, H_B)$ plus the number of tensor nodes in the LCST. By Case 1, $\text{LCST}(G'_A, H_B) = O(mK_G)$. Note that every two tensor nodes in G_A has distance at most K_G and any set of nodes of diameter $O(K_G)$ in H_B can only have size $O(K_G)$, so the number of tensor nodes in LCST is at most $O(K_G)$. Thus $\text{LCST}(G_A, H_B) = O(mK_G) + O(K_G) = O(mK_G)$.

Case 3. If some tensor node in G_A is mapped to a tensor node in H_B , then all the tensor nodes of G_A in the LCST are in the same tensor gadget, and it also holds for G_B . This can be shown as follows: Let u_1, u_2 be two tensor node in G_A that are mapped to tensor nodes u'_1, u'_2 in H_B , then

- u_1, u_2 are in the same tensor gadget. This is because that the minimum distance between two tensor nodes from different tensor gadgets in G_A is at least $2K_D$ and at most K_G , but the distance between any two tensor nodes in H_B is either $\leq K_D$ or $\geq 2K_G$.

- u'_1, u'_2 are in the same tensor gadget. This is because that the minimum distance between two tensor nodes from different tensor gadgets in H_B is at least $2K_G$, but the distance between any two tensor nodes in G_A is at most K_G .

Let $G(a)$ be the unique tensor gadget in G_A that has nodes in the LCST, and let $H(b)$ be the unique tensor gadget in H_B that has nodes in the LCST. By Case 1, $\text{LCST}(G'_A, H_B) = O(mK_G)$. Thus we have

$$\begin{aligned}\text{LCST}(G_A, H_B) &\leq \text{LCST}(G_A \setminus G(a), H_B) + \text{LCST}(G(a), H(b)) \\ &= O(mK_G) + \text{LCST}(G(a), H(b)).\end{aligned}$$

In any case, we can show that $\text{LCST}(G_A, H_B) \leq \text{LCST}(G(a), H(b)) + O(L)$ for some $(a, b) \in A \times B$, which completes the proof. \square

Theorem 8.12. *There exists an $O(N \text{poly}(D))$ -time reduction from an $\varepsilon(N)$ -Gap-Max-TropSim instance with two sets of N tensors of size D to an instance of $o(\varepsilon(N)^{-1})$ -approximate Largest Common Subtrees on two (rooted or unrooted) binary trees of size $O(N \text{poly}(D, \varepsilon(N)^{-1}))$ and depth $O(\log^2 N \text{poly}(D, \varepsilon(N)^{-1}))$.*

Proof. First we add dummy dimensions to each tensor such that the new size C of every tensor is at least $\Omega(\varepsilon^{-2}(N))$. We construct the two trees A', B' as in Theorem 8.11. By setting $L = C \log^2 N$, we have

$$\text{LCST}(A', B') = (\sqrt{C} \cdot s_{\max} + O(1)) \cdot C \log^2 N.$$

Thus it reduces to distinguish $\text{LCST}(A', B')$ from being $\geq C^{3/2} \log^2 N$ and $\leq O(\varepsilon(N)C^{3/2} \log^2 N)$, which can be solved by an $o(\varepsilon(N)^{-1})$ -approximation algorithm for LCST. This conclusion also holds for RLCST. \square

9 Equivalence in the Data Structure Setting

In this section, we establish the equivalence between BP-Pair-Class problems in the data structure setting.

Theorem 9.1. *For the following data structure problems, if any of the following problems admits an algorithm with preprocessing time $T(N)$, space $S(N)$ and query time $Q(N)$, then all other problems admits a similar algorithm with preprocessing time $T(N) \cdot N^{o(1)}$, space $S(N) \cdot N^{o(1)}$ and query time $Q(N) \cdot N^{o(1)}$.*

- NNS_{LCS} : Preprocess a database \mathcal{D} of N strings of length $D = 2^{(\log N)^{o(1)}}$, and then for each query string x , find $y \in \mathcal{D}$ maximizing $\text{LCS}(x, y)$.
- Approx. NNS_{LCS} : Find $y \in \mathcal{D}$ s.t. $\text{LCS}(x, y)$ is a $2^{(\log D)^{1-\Omega(1)}}$ approximation to the maximum value.
- Regular Expression Query: Preprocess a database \mathcal{D} of N strings of length $D = 2^{(\log N)^{o(1)}}$, and then for each query regular expression y , find an $x \in \mathcal{D}$ matching y .
- Approximate Regular Expression Query: for a query expression y , distinguish between: (1) there is an $x \in \mathcal{D}$ matching y ; and (2) for all $x \in \mathcal{D}$, the hamming distance between x and all $z \in L(y)$ is at least $(1 - o(1)) \cdot D$.

Proof. We show a reduction from NNS_{LCS} to Approx. NNS_{LCS} for illustration, and the proofs for the other pairs of problems are essentially the same.

By Lemma 6.4, there is a BP of poly-logarithmic size that accepts (a, b, k) iff $\text{LCS}(a, b) \geq k$. Then using a similar argument as in Theorem 6.5, we can show a reduction from an instance $\phi = (A, B)$ of the

following problem to an Approx. Closest-LCS-Pair instance $\phi' = (A', B')$: given a set of strings A and a set of string-integer pairs, determine whether there are $a \in A$ and $(b, k) \in B$ such that $\text{LCS}(a, b) \geq k$. Moreover, this reduction maps each element separately, i.e., there exist two maps f, g such that $A' = \{f(a) \mid a \in A\}$, $B' = \{g(b, k) \mid (b, k) \in B\}$, and both f and g are computable in $O(2^{\text{polylog}(D)}) = 2^{(\log N)^{o(1)}}$ time and space for each string of length D .

Now suppose there is a data structure for Approx. NNS_{LCS} with preprocessing time $T(N)$, space $S(N)$ and query time $Q(N)$. For a set of string A , we construct a data structure for NNS_{LCS} as follows. In the preprocessing stage, we map all the strings in A via f and store them in a data structure \mathcal{D} for Approx. NNS_{LCS} . For each query string, we do a binary search for the maximum LCS. For every length k encountered, we first map the query string and the length k via g and then query it in the data structure \mathcal{D} . The time cost and space usage of the new data structure can be easily analyzed. \square

A direct generalization of the above proof is that NNS_{LCS} is actually the hardest NNS problem among all distance that can be computed in small space.

Corollary 9.2. *For every distance function dist that can be computed in poly-logarithmic space, the exact NNS problem with respect to dist (NNS_{dist}) can be reduced to $2^{(\log D)^{1-\Omega(1)}}$ -approximate NNS_{LCS} in near-linear time.*

Remark 9.3. *Here we assume that when the size parameter for NNS_{dist} is N , the inputs to dist takes $2^{(\log N)^{o(1)}}$ bits to describe, and the output of dist takes $\text{polylog}(N)$ bits to describe.*

Furthermore, basing on the hardness of solving BP-SAT, we can show that there may not be an efficient data structure for NNS_{LCS} in the following sense:

Theorem 9.4. *Assuming the satisfiability for branching programs of size $2^{n^{o(1)}}$ cannot be decided in $O(2^{(1-\delta)n})$ for some $\delta > 0$, there is no data structure for Approx. NNS_{LCS} (or other data structure problems listed in Theorem 9.1) with preprocessing time $O(N^c)$ and query time $N^{1-\varepsilon}$ for some $c, \varepsilon > 0$.*

Proof. Our proof closely follows the proof for Corollary 1.3 in [Rub18]. We only prove that it is true for Approx. NNS_{LCS} , the case for other data structures are similar. Assume such a data structure for Approx. NNS_{LCS} does exist. Now we show that there is an algorithm for approximate Closest-LCS-Pair that runs in $O(N^{2-\delta'})$ time for some $\delta' > 0$.

Let (A, B) be an instance of approximate Closest-LCS-Pair. Let $\gamma = 1/(2c)$. We partition the set A into $M = O(N^{1-\gamma})$ subsets A_1, \dots, A_M , each of size $O(N^\gamma)$. Now for each subset A_i , we build a data structure for Approx. NNS_{LCS} , and query each $b \in B$ in the data structure. Finally, we take the maximum among all the query results. The total time for preprocessing is $O(M \cdot (N^\gamma)^c) = O(N^{3/2})$ and that for query is $O(N \cdot M \cdot (N^\gamma)^{1-\varepsilon}) = O(N^{2-\varepsilon\gamma})$. \square

10 Faster BP-SAT Implies Circuit Lower Bounds

In [AHVW16], Abboud et al. showed that faster exact algorithms for Edit Distance or LCS imply faster BP-SAT, and it leads to circuit lower bound consequences that are far stronger than any state of art. Using a similar argument, strong circuit lower bounds can also be shown if any of BP-Pair-Class or BP-Pair-Hard problems has faster algorithms, even for shaving a quasipolylog factor.

We apply the following results from [AHVW16] to show the circuit lower bound consequences, which are direct corollaries from [Wil13, Wil14b]:

Theorem 10.1 ([Wil13, Wil14b]). *Let $n \leq S(n) \leq 2^{o(n)}$ be a time constructible and monotone non-decreasing function. Let \mathcal{C} be a class of circuits. If the satisfiability of a function of the form*

- AND of fan-in in $O(S(n))$ of
- arbitrary functions of fan-in 3 of
- $O(S(n))$ -size circuits from \mathcal{C}

can be decided in $\text{DTIME}[O(2^n/n^{10})]$ time, then E^{NP} does not have $S(n)$ -size \mathcal{C} -circuits.

Theorem 10.2 ([Wil13]). *For the complexity class $\text{NTIME}[2^{O(n)}]$, we have*

1. *If for every constant $k > 0$, there is a satisfiability algorithm for bounded fan-in formulas of size n^k running in $\text{DTIME}[O(2^n/n^k)]$ time, then $\text{NTIME}[2^{O(n)}]$ is not contained in non-uniform NC^1 ;*
2. *If for every constant $k > 0$, there is a satisfiability algorithm for NC -circuits of size n^k running in $\text{DTIME}[O(2^n/n^k)]$ time, then $\text{NTIME}[2^{O(n)}]$ is not contained in non-uniform NC .*

First we show the circuit lower bound consequences if truly-subquadratic algorithm exists:

Reminder of Corollary 1.13 *If any of the BP-Pair-Class or BP-Pair-Hard problems admits an $N^{2-\varepsilon}$ time deterministic algorithm (or $(NM)^{1-\varepsilon}$ time algorithm for regular expression membership testing) for some $\varepsilon > 0$, then E^{NP} does not have:*

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

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

Proof. A truly-subquadratic time algorithm for BP-Pair-Class or BP-Pair-Hard problems implies a $2^{(1-\Omega(1))n}$ -time algorithm for BP-SAT on branching program of size $2^{n^{o(1)}}$. Let $S(n) = 2^{n^{o(1)}}$. $O(S(n))$ -size boolean formulas, $O(\log S(n))$ -depth circuits, $2^{n^{o(1)}}$ -size nondeterministic branching programs are all closed under AND, OR and NOT gates proscribed in Theorem 10.1. Note that any formula of size $2^{n^{o(1)}}$ can be transformed into an equivalent $n^{o(1)}$ -depth circuit [Spi71], and any $n^{o(1)}$ -depth circuit can be transformed into $2^{n^{o(1)}}$ -size branching program by Barrington's Theorem [Bar89]. Then all the consequences in Item 1, 2, 3 follow from Theorem 10.1. Combining Item 2 and Theorem 10.2, we can obtain the consequence that $\text{NTIME}[2^{O(n)}]$ is not in non-uniform NC . \square

We can also obtain results showing that even shaving a quasipolylog factor $2^{(\log \log N)^3}$ for problems in BP-Pair-Class and BP-Pair-Hard can imply new circuit lower bound. First, it is easy to see that shaving a $(\log N)^{\omega(1)}$ factor can lead to new circuit lower bound by Theorem 10.2.

Theorem 10.3. *If there is a deterministic algorithm for BP-Satisfying-Pair on BP of size $S = 2^{(\log N)^{o(1)}}$ running in $O(N^2 \text{poly}(S)/(\log N)^{\omega(1)})$ time, then the following holds:*

1. *For any constant $k > 0$, SAT on bounded fan-in formula of size n^k can be solved in $O(2^n/n^{\omega(1)})$ deterministic time;*
2. *$\text{NTIME}[2^{O(n)}]$ is not contained in non-uniform NC^1 .*

Proof. By Theorem 10.2, Item 1 implies Item 2, so we only need to show the conclusion in Item 1.

If BP-Satisfying-Pair can be solved in $O(N^2 \text{poly}(S)/(\log N)^{\omega(1)})$ time, then BP-SAT on BP of size $O(\text{poly}(n))$ can be solved in

$$O(2^n \text{poly}(n)/n^{\omega(1)}) = O(2^n/n^{\omega(1)}).$$

Note that any formula of size n^k can be transformed into an equivalent BP of width $W = 5$ and length $T = O(n^{8k})$ (by rebalancing into a formula of depth $4k \log n$ [Spi71] and using Barrington's Theorem [Bar89]). Thus SAT on bounded fan-in formulas of size n^k can also be solved in $O(2^n/n^{\omega(1)})$. \square

A part of our reductions from BP-Satisfying-Pair to problems in BP-Pair-Class can be summarized below. In the rest of this section, for each problem in BP-Pair-Class (but except BP-Satisfying-Pair), we use the variable N to denote the number of elements in each set, and D to denote the maximum length (or size) of each element. We exclude BP-Satisfying-Pair here because the size S of BP is more important than D in BP-Satisfying-Pair.

Corollary 10.4. *For every problem \mathcal{P} in BP-Pair-Class except BP-Satisfying-Pair:*

- If \mathcal{P} is a decision problem, then there is an $O(N \text{poly}(D))$ -time reduction from Restricted OAPT to \mathcal{P} ;
- If \mathcal{P} is an approximate problem, then for every $\varepsilon(D) = \Omega(D^{-1/2})$, there is an $O(N \text{poly}(D))$ -time reduction from Restricted $\varepsilon(D)$ -Gap-Max-TropSim or Restricted $\varepsilon(D)$ -Gap-Max-TropSim to \mathcal{P} with approximation ratio $o(\varepsilon(D)^{-1})$, and each element has size $O(\text{poly}(D))$.

And any reduction here preserves the value of N .

Reminder of Theorem 1.10 For $D = 2^{(\log N)^{\omega(1)}}$, if there is an

$$O\left(N^2 \text{poly}(D)/2^{(\log \log N)^3}\right) \text{ or } O\left(N^2/(\log N)^{\omega(1)}\right)$$

time deterministic algorithm for the decision, exact value, or $O(\text{polylog}(D))$ -approximation problems in BP-Pair-Class, then the same consequences in Theorem 10.3 follows.

Proof. By Corollary 10.4 and the fact that exact value problem can be trivially reduced to its approximation version, we only need to show that this statement is true for OAPT and $(\log D)^c$ -Gap-Max-TropSim for every $c > 0$ (the proof for $(\log D)^c$ -Gap-Min-TropSim should be similar).

Note that all our reductions here preserve the value of N . If there is an $O(N^2/(\log N)^{\omega(1)})$ -time algorithm, then BP-Satisfying-Pair can also be solved in $O(N^2/(\log N)^{\omega(1)})$ -time and the consequences in Theorem 10.3 follows.

Now consider the case that a $O(N^2 \text{poly}(D)/2^{(\log \log N)^3})$ -time algorithm exists. Recall that the hard instances of BP-Satisfying-Pair we constructed in the proof of Theorem 10.3 is on BP of width $W = O(1)$ and length $T = O(\text{poly}(n)) = O(\text{polylog}(N))$. By Theorem 5.1, we know that this instance can be near-linear time reduced to an OAPT instance with

$$D = 2^{O(\log W \log T)} = 2^{O(\log \log N)} = \text{polylog}(N).$$

Thus shaving an $O(2^{(\log \log N)^3})$ factor to OAPT implies an $O(N^2/(\log N)^{\omega(1)})$ -time algorithm for BP-Satisfying-Pair.

By Theorem 5.6, for $\varepsilon = \log^{-3c}(T)$, we know that a hard instance of BP-Satisfying-Pair can also be near-linear time reduced to an ε -Gap-Max-TropSim instance with (adding dummy dimensions if necessary)

$$D = 2^{\Theta(\log^2 W \log^2 T (\log \log W + \log \log T + \log \varepsilon^{-1}))} = 2^{\Theta(\log^2 T \log \log T)}.$$

Then we have $(\log D)^c = o(\varepsilon^{-1})$, and thus shaving an $O(2^{(\log \log N)^3})$ factor to $(\log D)^c$ -Gap-Max-TropSim implies an algorithm for the hard instances of BP-Satisfying-Pair running in the following time:

$$\begin{aligned} O(N^2 \text{poly}(D)/2^{(\log \log N)^3}) &= O(N^2 \cdot 2^{\Theta(\log^2 T \log \log T)}/2^{(\log \log N)^3}) \\ &= O(N^2/(\log N)^{\omega(1)}). \end{aligned}$$

□

For BP-Pair-Hard problems, recall that part of our reduction can be summarized below:

Corollary 10.5. *For every problem \mathcal{P} in BP-Pair-Hard:*

- If \mathcal{P} is a decision problem, then there is an $O(N \text{poly}(D))$ -time reduction from Restricted OAPT to \mathcal{P} on input of length $O(N \text{poly}(D))$;
- If \mathcal{P} is an approximate problem, then for every $\varepsilon(N)$, there is an $O(N \text{poly}(D, \varepsilon(N)^{-1}))$ -time reduction from Restricted $\varepsilon(N)$ -Gap-Max-TropSim or Restricted $\varepsilon(N)$ -Gap-Max-TropSim to \mathcal{P} with approximation ratio $o(\varepsilon(N)^{-1})$ on input of length $O(N \text{poly}(D, \varepsilon(N)^{-1}))$.

Then we can obtain the following result:

Reminder of Theorem 1.11 *If there is an deterministic algorithm for any decision, exact value or $\text{polylog}(N)$ -approximation problems among BP-Pair-Hard problems listed in Theorem 1.8 running in running in*

$$O\left(N^2/2^{\omega(\log \log N)^3}\right)$$

time (or $O\left(NM/2^{\omega(\log \log (NM))^3}\right)$ time for Regular Expression Membership Testing), then the same consequences in Theorem 1.10 follows.

Proof. By Corollary 10.5 and the fact that exact value problem can be trivially reduced to its approximation version, we only need to show that this statement is true for OAPT and $(\log N)^c$ -Gap-Max-TropSim for every $c > 0$.

The proof for OAPT is similar as in Theorem 1.10. For $(\log N)^c$ -Gap-Max-TropSim, we know that the hard instances of BP-Satisfying-Pair in Theorem 10.3 can be reduced to a ε -Gap-Max-TropSim instance with

$$D = 2^{O(\log^2 W \log^2 T (\log \log W + \log \log T + \log \varepsilon^{-1}))} = 2^{O(\log \log N)^3}$$

for $\varepsilon = (\log N)^c$. Thus shaving an $O(2^{\omega(\log \log N)^3})$ factor to $(\log N)^c$ -Gap-Max-TropSim implies an algorithm for the hard instances of BP-Satisfying-Pair running in $O(N^2/(\log N)^{\omega(1)})$ time. □

11 Derandomization Implies Circuit Lower Bounds

For the some problems \mathcal{A} like *Longest Common Subsequence*, despite its approximating for the pair version of \mathcal{A} (Approximate Max- \mathcal{A} -Pair) is subquadratically equivalent to Max-TropSim, it is still hard to find a reduction from approximating \mathcal{A} . The main barrier is when trying to construct gadgets to reduce Approximate Max- \mathcal{A} -Pair to Approximate \mathcal{A} , the contribution to the final result for just one pair is too small to make a large approximating gap.

To overcome this barrier, we follows from [AR18] to define $\varepsilon(N)$ -Super-Gap-Max-TS, which is a variant of $\varepsilon(N)$ -Gap-Max-TropSim with a large fraction of pairs having perfect Tropical Similarities:

Definition 11.1 (ε -Super-Gap-Max-TS). Let t be an even number and $d_1 = d_2 = \dots = d_t = 2$. Given two sets of tensors $A, B \in \{0, 1\}^{d_1 \times \dots \times d_t}$ of size $D = 2^t$, distinguish between the following:

- **Completeness:** A $(1 - 1/\log^{10} N)$ -fraction of the pairs of $a \in A, b \in B$ have a perfect Tropical Similarity, $s(a, b) = 1$;
- **Soundness:** Every pair has low Tropical Similarity score, $s(a, b) < \varepsilon$.

where ε is a threshold that can depend on N and D .

In [AR18], Abboud and Rubinstein has shown that $o(1)$ -Super-Gap-Max-TS can be reduced to $O(1)$ -approximate LCS. Using the same reduction, we have the following corollary for arbitrary approximation ratio:

Theorem 11.2 ([AR18]). *Given an $\varepsilon(N)$ -Super-Gap-Max-TS instance on N tensors of size D , we can construct two strings x, y of length ND in $O(N \text{poly}(D))$ deterministic time such that:*

- If $(1 - 1/\log^{10} N)$ -fraction of the pairs have a perfect Tropical Similarity, then $\text{LCS}(x, y) > (1/3)ND$;
- If every pair has low Tropical Similarity score, then $\text{LCS}(x, y) < 2\varepsilon(N)ND$

Thus, if there is an $\varepsilon(N)^{-1}$ -approximation algorithm for such kind of (x, y) pairs, then there is a faster algorithm for $(\varepsilon(N)/6)$ -Super-Gap-Max-TS.

Proof. We construct strings $G(a), H(b)$ as tensor gadgets for each tensor $a \in A, b \in B$ as in the reduction in [AR18] (stated in Theorem 6.1). Then we construct the final strings x, y by concatenating all the tensor gadgets. Using a similar argument as in [AR18], we can show that if there are at least $(1 - 1/\log^{10} N)N^2$ pairs of tensors with perfect Tropical Similarities, then $\text{LCS}(x, y) > (1 - 1/\log^{10} N) \cdot ND/2$; if every pair has low Tropical Similarity score, then $\text{LCS}(x, y) < \varepsilon(N) \cdot 2ND$. \square

There is no obvious reduction from BP-Satisfying-Pair to $\varepsilon(N)$ -Super-Gap-Max-TS, and a randomized algorithm can even solve $\varepsilon(N)$ -Super-Gap-Max-TS in nearly linear time. But finding a *deterministic* algorithm for $\varepsilon(N)$ -Super-Gap-Max-TS is still hard: as noted by Abboud and Rubinstein in [AR18], a truly-subquadratic time deterministic algorithm for $\varepsilon(N)$ -Super-Gap-Max-TS can imply some circuit lower bound for E^{NP} . Combining their ideas with the connection between Tropical Tensors and BP-SAT we established, we can show even stronger circuit lower bounds if such algorithm exists.

We base our proof on the following results in the literature:

Theorem 11.3 ([BV14]). *Let F_n be a set of function from $\{0, 1\}^n$ to $\{0, 1\}$ that are efficiently closed under projections. If the acceptance probability of a function of the form*

- AND of fan-in in $n^{O(1)}$ of
- OR's of fan-in 3 of
- functions from $F_{n+O(\log n)}$

can be distinguished from being = 1 or $\leq 1/n^{10}$ in $\text{DTIME}[2^n / n^{\omega(1)}]$, then there is a function $f \in E^{NP}$ on n variables and $f \notin F_n$.

Theorem 11.4 ([Wil13, BV14]). *If the acceptance probability of a function from NC^1 can be distinguished from being = 1 or $\leq 1/n^{10}$ in $\text{DTIME}[2^n / n^{\omega(1)}]$, then $NTIME[2^{O(n)}]$ is not contained in NC^1 .*

Theorem 11.5. Let AC-BP-SAT be the following problem: given a branching program P of length T and width W on n inputs, distinguish the acceptance probability of P from being $= 1$ or $\leq 1/n^{10}$.

There is a reduction from AC-BP-SAT to ε -Super-Gap-Max-TS on two sets of $N = 2^{n/2}$ tensors of size $D = 2^{O(\log^2 W \log^2 T (\log \log W + \log \log T + \log \varepsilon^{-1}))}$, and the reduction runs in $O(N \text{poly}(D))$. Here ε is a threshold value that can depend on N (but cannot depend on D).

Reminder of Theorem 1.12 The following holds for deterministic approximation to LCS:

1. A $2^{(\log N)^{1-\Omega(1)}}$ -approximation algorithm in $N^{2-\delta}$ time for some constant $\delta > 0$ implies that E^{NP} has no $n^{o(1)}$ -depth bounded fan-in circuits;
2. A $2^{o(\log N / (\log \log N)^2)}$ -approximation algorithm in $N^{2-\delta}$ time for some constant $\delta > 0$ implies that $\text{NTIME}[2^{O(n)}]$ is not contained in non-uniform NC^1 ;
3. An $O(\text{polylog}(N))$ -approximation algorithm in $N^2 / 2^{\omega(\log \log N)^3}$ time implies that $\text{NTIME}[2^{O(n)}]$ is not contained in non-uniform NC^1 .

Proof. By Theorem 11.3 and Theorem 11.4, for Item 1, it is sufficient to show AC-BP-SAT on BP of length $2^{n^{o(1)}}$ and width $O(1)$ on n inputs can be solved in $2^{(1-\Omega(1))n}$ time; for Item 2 and 3, it is sufficient to show AC-BP-SAT on BP of length $O(\text{poly}(n))$ and width $O(1)$ on n inputs can be solved in $2^n / n^{\omega(1)}$ time (the former scale of BP is able to simulate $n^{o(1)}$ -depth circuit, while the later one is able to simulate NC^1 by Barrington's Theorem [Bar89]).

Item 1. Assume there exists a $2^{(\log N)^{1-c}}$ -approximation algorithm for LCS in $N^{2-\delta}$ time for some $c > 0$ and $\delta > 0$. By Theorem 11.5, AC-BP-SAT on BP of length $T = 2^{n^{o(1)}}$ and width $W = O(1)$ on n inputs can be reduced to $(2^{-(\log K)^{1-c}}/6)$ -Super-Gap-Max-TS on $K = 2^{n/2}$ tensors of size

$$D = 2^{O(n^{o(1)} \cdot (o(\log n) + (\log K)^{1-c}))} = 2^{n^{1-c+o(1)}}.$$

Then by Theorem 11.2, $(2^{-(\log K)^{1-c}}/6)$ -Super-Gap-Max-TS can be reduced to $2^{(\log K)^{1-c}}$ -approximate LCS for strings of length

$$N = KD = 2^{n/2+n^{1-c+o(1)}} = 2^{(1/2+o(1))n}.$$

By our assumption, the last problem can be solved in $N^{2-\delta}$ time, so AC-BP-SAT on branching program of length $2^{n^{o(1)}}$ and width $O(1)$ on n inputs can be solved in $2^{(1-\delta/2+o(1))n}$ time. Applying Theorem 11.3 completes the proof.

Item 2. Assume there exists a $2^{f(\log N)}$ -approximation algorithm for LCS in $N^{2-\delta}$ time for some constant $\delta > 0$ and some function $f(k) = o(k/\log^2 k)$. Let $g(k) = 2f(k) + \log k$. Then we have $g(\log N) = o(\log N / (\log \log N)^2)$ and

$$2^{f((1+o(1)) \log K)} \leq 2^{(1+o(1))f(\log K)} \leq 2^{2f(\log K)} = o(2^{g(\log K)}).$$

By Theorem 11.5, AC-BP-SAT on BP of length $T = O(\text{poly}(n))$ and width $W = O(1)$ on n inputs can be reduced to $2^{-g(\log K)}$ -Super-Gap-Max-TS on $K = 2^{n/2}$ tensors of size

$$D = 2^{O(\log^2 n \cdot (\log \log n + g(\log K)))} = 2^{O(\log^2 n \cdot o(\log K / (\log \log K)^2))} = 2^{o(n)}.$$

Then by Theorem 11.2, $2^{-g(\log K)}$ -Super-Gap-Max-TS can be reduced to $o(2^{g(\log K)})$ -approximate LCS for strings of length $N = KD = 2^{(1/2+o(1))n}$. Note that $2^{f(\log(K^{1+o(1)}))} = o(2^{g(\log K)})$. Thus by our assumption, the last problem can be solved in $N^{2-\delta}$ time, which means AC-BP-SAT on branching program of length $2^{n^{o(1)}}$ and width $O(1)$ on n inputs can be solved in $2^{(1-\delta/2+o(1))n}$ time. Applying Theorem 11.4 completes the proof.

Item 3. Assume there exists a $\log^c(N)$ -approximation algorithm for LCS in $N^2/2^{\omega(\log \log N)^3}$ time for some $c > 0$. Using a similar calculation as in Theorem 1.11, we know that AC-BP-SAT on branching program of length $O(\text{poly}(n))$ and width $O(1)$ on n inputs can be solved in

$$N^2/2^{\omega(\log \log N)^3} \leq 2^{n+O(\log^3 n)}/2^{\omega(\log^3 n)} \leq 2^n/n^{\omega(1)}$$

time. Applying Theorem 11.4 completes the proof. \square

Acknowledgement

The authors would like to thank Ryan Williams for helpful discussion.

References

- [AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity - A Modern Approach*. Cambridge University Press, 2009.
- [AB17] Amir Abboud and Arturs Backurs. Towards hardness of approximation for polynomial time problems. In *8th Innovations in Theoretical Computer Science Conference, ITCS 2017, January 9-11, 2017, Berkeley, CA, USA*, pages 11:1–11:26, 2017.
- [AB18] Amir Abboud and Karl Bringmann. Tighter connections between formula-sat and shaving logs. In *45th International Colloquium on Automata, Languages, and Programming, ICALP 2018, July 9-13, 2018, Prague, Czech Republic*, pages 8:1–8:18, 2018.
- [ABH⁺16] Amir Abboud, Arturs Backurs, Thomas Dueholm Hansen, Virginia Vassilevska Williams, and Or Zamir. Subtree isomorphism revisited. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1256–1271, 2016.
- [ABV15] Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. Tight hardness results for LCS and other sequence similarity measures. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 59–78, 2015.
- [AD16] Amir Abboud and Søren Dahlgaard. Popular conjectures as a barrier for dynamic planar graph algorithms. In *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 477–486, 2016.
- [ADKF70] V. L. Arlazarov, E. A. Dinic, M. A. Kronrod, and I. A. Faradžev. On economical construction of the transitive closure of a directed graph. *Soviet Mathematics—Doklady*, 11(5):1209–1210, 1970.
- [AGV15] Amir Abboud, Fabrizio Grandoni, and Virginia Vassilevska Williams. Subcubic equivalences between graph centrality problems, APSP and diameter. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 1681–1697, 2015.
- [AH00] Tatsuya Akutsu and Magnús M Halldórsson. On the approximation of largest common subtrees and largest common point sets. *Theoretical Computer Science*, 233(1-2):33–50, 2000.

- [AHVW16] 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 *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 375–388, 2016.
- [ALM⁺98] Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and the hardness of approximation problems. *J. ACM*, 45(3):501–555, 1998.
- [AR18] Amir Abboud and Aviad Rubinstein. Fast and deterministic constant factor approximation algorithms for LCS imply new circuit lower bounds. In *9th Innovations in Theoretical Computer Science Conference, ITCS 2018, January 11-14, 2018, Cambridge, MA, USA*, pages 35:1–35:14, 2018.
- [ARW17] Amir Abboud, Aviad Rubinstein, and Ryan Williams. Distributed PCP theorems for hardness of approximation in P. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 25–36, 2017.
- [AS98] Sanjeev Arora and Shmuel Safra. Probabilistic checking of proofs: A new characterization of NP. *J. ACM*, 45(1):70–122, 1998.
- [ATMT15] Tatsuya Akutsu, Takeyuki Tamura, Avraham A Melkman, and Atsuhiro Takasu. On the complexity of finding a largest common subtree of bounded degree. *Theoretical Computer Science*, 590:2–16, 2015.
- [AV14] Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 434–443, 2014.
- [AVW14] Amir Abboud, Virginia Vassilevska Williams, and Oren Weimann. Consequences of faster alignment of sequences. In *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, pages 39–51, 2014.
- [AVY15] Amir Abboud, Virginia Vassilevska Williams, and Huacheng Yu. Matching triangles and basing hardness on an extremely popular conjecture. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 41–50, 2015.
- [AW09] Scott Aaronson and Avi Wigderson. Algebrization: A new barrier in complexity theory. *TOCT*, 1(1):2:1–2:54, 2009.
- [AWY15] Amir Abboud, Ryan Williams, and Huacheng Yu. More applications of the polynomial method to algorithm design. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 218–230, 2015.
- [Bar89] David A. Mix Barrington. Bounded-width polynomial-size branching programs recognize exactly those languages in NC¹. *J. Comput. Syst. Sci.*, 38(1):150–164, 1989.
- [BDT16] Arturs Backurs, Nishanth Dikkala, and Christos Tzamos. Tight hardness results for maximum weight rectangles. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, pages 81:1–81:13, 2016.

- [BGL17] Karl Bringmann, Allan Grønlund, and Kasper Green Larsen. A dichotomy for regular expression membership testing. In *58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, Berkeley, CA, USA, October 15-17, 2017*, pages 307–318, 2017.
- [BHR98] Lasse Bergroth, Harri Hakonen, and Timo Raita. New approximation algorithms for longest common subsequences. In *String Processing and Information Retrieval: A South American Symposium, SPIRE 1998, Santa Cruz de la Sierra Bolivia, September 9-11, 1998*, pages 32–40, 1998.
- [BI15] Arturs Backurs and Piotr Indyk. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 51–58, 2015.
- [BI16] Arturs Backurs and Piotr Indyk. Which regular expression patterns are hard to match? In *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 457–466, 2016.
- [BIS17] Arturs Backurs, Piotr Indyk, and Ludwig Schmidt. On the fine-grained complexity of empirical risk minimization: Kernel methods and neural networks. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 4311–4321, 2017.
- [BK15] Karl Bringmann and Marvin Künnemann. Quadratic conditional lower bounds for string problems and dynamic time warping. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 79–97, 2015.
- [BK18] Karl Bringmann and Marvin Künnemann. Multivariate fine-grained complexity of longest common subsequence. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1216–1235, 2018.
- [BR13] Djamal Belazzougui and Mathieu Raffinot. Approximate regular expression matching with multi-strings. *Journal of Discrete Algorithms*, 18:14–21, 2013.
- [Bri14] Karl Bringmann. Why walking the dog takes time: Frechet distance has no strongly subquadratic algorithms unless SETH fails. In *55th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2014, Philadelphia, PA, USA, October 18-21, 2014*, pages 661–670, 2014.
- [BT09] Philip Bille and Mikkel Thorup. Faster regular expression matching. In *Automata, Languages and Programming, 36th International Colloquium, ICALP 2009, Rhodes, Greece, July 5-12, 2009, Proceedings, Part I*, pages 171–182, 2009.
- [BV14] Eli Ben-Sasson and Emanuele Viola. Short pcps with projection queries. In *Automata, Languages, and Programming - 41st International Colloquium, ICALP 2014, Copenhagen, Denmark, July 8-11, 2014, Proceedings, Part I*, pages 163–173, 2014.
- [BW12] Nikhil Bansal and Ryan Williams. Regularity lemmas and combinatorial algorithms. *Theory of Computing*, 8(1):69–94, 2012.

- [CGI⁺16] Marco L. Carmosino, Jiawei Gao, Russell Impagliazzo, Ivan Mihajlin, Ramamohan Paturi, and Stefan Schneider. Nondeterministic extensions of the strong exponential time hypothesis and consequences for non-reducibility. In *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science, Cambridge, MA, USA, January 14-16, 2016*, pages 261–270, 2016.
- [Cha13] Timothy M. Chan. The art of shaving logs. In *Algorithms and Data Structures - 13th International Symposium, WADS 2013, London, ON, Canada, August 12-14, 2013. Proceedings*, page 231, 2013.
- [Cha15] Timothy M. Chan. Speeding up the four russians algorithm by about one more logarithmic factor. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2015, San Diego, CA, USA, January 4-6, 2015*, pages 212–217, 2015.
- [Che18] Lijie Chen. On the hardness of approximate and exact (bichromatic) maximum inner product. In *33rd Computational Complexity Conference, CCC 2018, June 22-24, 2018, San Diego, CA, USA*, pages 14:1–14:45, 2018.
- [Chu87] Moon-Jung Chung. $O(n^{(2.55)})$ time algorithms for the subgraph homeomorphism problem on trees. *J. Algorithms*, 8(1):106–112, 1987.
- [CIPR01] 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.
- [CKK72] Vašek Chvátal, David A Klarner, and Donald Ervin Knuth. *Selected combinatorial research problems*. Computer Science Department, Stanford University, 1972.
- [CMWW17] Marek Cygan, Marcin Mucha, Karol Węgrzycki, and Michał Włodarczyk. On problems equivalent to $(\min, +)$ -convolution. In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, pages 22:1–22:15, 2017.
- [CW16] Timothy M. Chan and Ryan Williams. Deterministic apsp, orthogonal vectors, and more: Quickly derandomizing razborov-smolensky. In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 1246–1255, 2016.
- [Din07] Irit Dinur. The PCP theorem by gap amplification. *J. ACM*, 54(3):12, 2007.
- [DKL18] Roee David, Karthik C. S., and Bundit Laekhanukit. On the complexity of closest pair via polar-pair of point-sets. In *34th International Symposium on Computational Geometry, SoCG 2018, June 11-14, 2018, Budapest, Hungary*, pages 28:1–28:15, 2018.
- [dM99] J Boutet de Monvel. Extensive simulations for longest common subsequences. *The European Physical Journal B-Condensed Matter and Complex Systems*, 7(2):293–308, 1999.
- [GIKW17] Jiawei Gao, Russell Impagliazzo, Antonina Kolokolova, and R. Ryan Williams. Completeness for first-order properties on sparse structures with algorithmic applications. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 2162–2181, 2017.

- [GKLP17] Isaac Goldstein, Tsvi Kopelowitz, Moshe Lewenstein, and Ely Porat. Conditional lower bounds for space/time tradeoffs. In *Algorithms and Data Structures - 15th International Symposium, WADS 2017, St. John's, NL, Canada, July 31 - August 2, 2017, Proceedings*, pages 421–436, 2017.
- [GKMS90] Phillip B Gibbons, Richard M Karp, Gary L Miller, and Danny Soroker. Subtree isomorphism is in random NC. *Discrete Applied Mathematics*, 29(1):35–62, 1990.
- [GPW16] Mika Göös, Toniann Pitassi, and Thomas Watson. Zero-information protocols and unambiguity in arthur-merlin communication. *Algorithmica*, 76(3):684–719, 2016.
- [GPW18] Mika Göös, Toniann Pitassi, and Thomas Watson. The landscape of communication complexity classes. *Computational Complexity*, 27(2):245–304, 2018.
- [Gra16] Szymon Grabowski. New tabulation and sparse dynamic programming based techniques for sequence similarity problems. *Discrete Applied Mathematics*, 212:96–103, 2016.
- [HKNS15] Monika Henzinger, Sebastian Krinninger, Danupon Nanongkai, and Thatchaphol Saranurak. Unifying and strengthening hardness for dynamic problems via the online matrix-vector multiplication conjecture. In *Proceedings of the 47th Annual ACM Symposium on Theory of Computing*, pages 21–30, 2015.
- [HLNV17] Monika Henzinger, Andrea Lincoln, Stefan Neumann, and Virginia Vassilevska Williams. Conditional hardness for sensitivity problems. In *8th Innovations in Theoretical Computer Science Conference, ITCS 2017, January 9-11, 2017, Berkeley, CA, USA*, pages 26:1–26:31, 2017.
- [IP01] Russell Impagliazzo and Ramamohan Paturi. On the complexity of k-SAT. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001.
- [JR91] Tao Jiang and Bala Ravikumar. A note on the space complexity of some decision problems for finite automata. *Information Processing Letters*, 40(1):25–31, 1991.
- [KLM18] Karthik C. S., Bundit Laekhanukit, and Pasin Manurangsi. On the parameterized complexity of approximating dominating set. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 1283–1296, 2018.
- [KM95] James R. Knight and Eugene W. Myers. Approximate regular expression pattern matching with concave gap penalties. *Algorithmica*, 14(1):85–121, 1995.
- [KMY95] Sanjeev Khanna, Rajeev Motwani, and F Frances Yao. *Approximation algorithms for the largest common subtree problem*. Citeseer, 1995.
- [KPP16] Tsvi Kopelowitz, Seth Pettie, and Ely Porat. Higher lower bounds from the 3sum conjecture. In *Proceedings of the 27th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1272–1287, 2016.
- [KT17] Robert Krauthgamer and Ohad Trabelsi. Conditional lower bounds for all-pairs max-flow. In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, pages 20:1–20:13, 2017.

- [LFKN92] Carsten Lund, Lance Fortnow, Howard J. Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. *J. ACM*, 39(4):859–868, 1992.
- [Lin83] Andrzej Lingas. An application of maximum bipartite c-matching to subtree isomorphism. In *CAAP'83, Trees in Algebra and Programming, 8th Colloquium, L'Aquila, Italy, March 9-11, 1983, Proceedings*, pages 284–299, 1983.
- [LK89] Andrzej Lingas and Marek Karpinski. Subtree isomorphism is NC reducible to bipartite perfect matching. *Information Processing Letters*, 30(1):27–32, 1989.
- [Lok01] Satyanarayana V. Lokam. Spectral methods for matrix rigidity with applications to size-depth trade-offs and communication complexity. *J. Comput. Syst. Sci.*, 63(3):449–473, 2001.
- [LVW18] Andrea Lincoln, Virginia Vassilevska Williams, and Ryan Williams. Tight hardness for shortest cycles and paths in sparse graphs. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 1236–1252, 2018.
- [Mat68] David W. Matula. An algorithm for subtree identification. *SIAM Review*, 10(2):273–274, 1968.
- [Mat78] David W. Matula. Subtree isomorphism in $O(n^{5/2})$. In B. Alspach, P. Hell, and D.J. Miller, editors, *Algorithmic Aspects of Combinatorics*, volume 2 of *Annals of Discrete Mathematics*, pages 91 – 106. Elsevier, 1978.
- [MM89] Eugene W. Myers and Webb Miller. Approximate matching of regular expressions. *Bulletin of mathematical biology*, 51(1):5–37, 1989.
- [MOG98] Eugene W. Myers, Paulo Oliva, and Katia S. Guimarães. Reporting exact and approximate regular expression matches. In *Combinatorial Pattern Matching, 9th Annual Symposium, CPM 98, Piscataway, New Jersey, USA, July 20-22, 1998, Proceedings*, pages 91–103, 1998.
- [MP80] William J. Masek and Mike Paterson. A faster algorithm computing string edit distances. *J. Comput. Syst. Sci.*, 20(1):18–31, 1980.
- [Mye92] Eugene W. Myers. A four russians algorithm for regular expression pattern matching. *J. ACM*, 39(2):430–448, 1992.
- [Nav04] Gonzalo Navarro. Approximate regular expression searching with arbitrary integer weights. *Nord. J. Comput.*, 11(4):356–373, 2004.
- [OR07] Rafail Ostrovsky and Yuval Rabani. Low distortion embeddings for edit distance. *J. ACM*, 54(5):23, 2007.
- [Pat10] Mihai Patrascu. Towards polynomial lower bounds for dynamic problems. In *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, pages 603–610, 2010.
- [Rey77] Steven W Reyner. An analysis of a good algorithm for the subtree problem. *SIAM Journal on Computing*, 6(4):730–732, 1977.
- [Rub18] Aviad Rubinstein. Hardness of approximate nearest neighbor search. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018, Los Angeles, CA, USA, June 25-29, 2018*, pages 1260–1268, 2018.

- [RW13] Liam Roditty and Virginia Vassilevska Williams. Fast approximation algorithms for the diameter and radius of sparse graphs. In *Symposium on Theory of Computing Conference, STOC’13, Palo Alto, CA, USA, June 1-4, 2013*, pages 515–524, 2013.
- [San10] Rahul Santhanam. Fighting perebor: New and improved algorithms for formula and QBF satisfiability. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pages 183–192, 2010.
- [Sav70] Walter J. Savitch. Relationships between nondeterministic and deterministic tape complexities. *J. Comput. Syst. Sci.*, 4(2):177–192, 1970.
- [Sha92] Adi Shamir. IP = PSPACE. *J. ACM*, 39(4):869–877, 1992.
- [SM73] Larry J Stockmeyer and Albert R Meyer. Word problems requiring exponential time (preliminary report). In *Proceedings of the fifth annual ACM symposium on Theory of computing*, pages 1–9. ACM, 1973.
- [Spi71] Philip Spira. On time-hardware complexity tradeoffs for boolean functions. In *Proceedings of the 4th Hawaii Symposium on System Sciences, 1971*, pages 525–527, 1971.
- [ST99] Ron Shamir and Dekel Tsur. Faster subtree isomorphism. *Journal of Algorithms*, 33(2):267–280, 1999.
- [Tal15] Avishay Tal. #SAT algorithms from shrinkage. *Electronic Colloquium on Computational Complexity (ECCC)*, 22:114, 2015.
- [Tho68] Ken Thompson. Programming techniques: Regular expression search algorithm. *Communications of the ACM*, 11(6):419–422, 1968.
- [Vas18] Virginia Vassilevska Williams. On some fine-grained questions in algorithms and complexity. In *Proceedings of the ICM*, 2018. To appear.
- [VW10] Virginia Vassilevska Williams and Ryan Williams. Subcubic equivalences between path, matrix and triangle problems. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pages 645–654, 2010.
- [Wil13] Ryan Williams. Improving exhaustive search implies superpolynomial lower bounds. *SIAM Journal on Computing*, 42(3):1218–1244, 2013.
- [Wil14a] Ryan Williams. Faster all-pairs shortest paths via circuit complexity. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*, pages 664–673. ACM, 2014.
- [Wil14b] Ryan Williams. Nonuniform ACC circuit lower bounds. *J. ACM*, 61(1):2:1–2:32, 2014.
- [Wil16] Ryan Williams. Strong ETH breaks with merlin and arthur: Short non-interactive proofs of batch evaluation. In *31st Conference on Computational Complexity, CCC*, pages 2:1–2:17, 2016.
- [Wil18a] Ryan Williams. On the difference between closest, furthest, and orthogonal pairs: Nearly-linear vs barely-subquadratic complexity. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1207–1215, 2018.
- [Wil18b] Ryan Williams. Some estimated likelihoods for computational complexity. 2018. Invited paper in the Springer LNCS 10,000 volume.

- [WMM95] Sun Wu, Udi Manber, and Eugene Myers. A subquadratic algorithm for approximate regular expression matching. *Journal of algorithms*, 19(3):346–360, 1995.
- [Yu15] Huacheng Yu. An improved combinatorial algorithm for boolean matrix multiplication. In *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part I*, pages 1094–1105, 2015.