Consequences of Faster Alignment of Sequences

Amir Abboud[1], Virginia Vassilevska Williams[1], and Oren Weimann[2]

[1] Stanford University, USA. {abboud,virgi}@cs.stanford.edu
[2] University of Haifa, Israel. oren@cs.haifa.ac.il

Abstract. The Local Alignment problem is a classical problem with applications in biology. Given two input strings and a scoring function on pairs of letters, one is asked to find the substrings of the two input strings that are most similar under the scoring function. The best algorithms for Local Alignment run in time that is roughly quadratic in the string length. It is a big open problem whether substantially subquadratic algorithms exist. In this paper we show that for all $\varepsilon > 0$, an $O(n^{2-\varepsilon})$ time algorithm for local alignment on strings of length n would imply breakthroughs on three longstanding open problems: it would imply that for some $\delta > 0$, 3SUM on n numbers is in $O(n^{2-\delta})$ time, CNF-SAT on n variables is in $O((2-\delta)^n)$ time, and Maximum Weight 4-Clique is in $O(n^{4-\delta})$ time. Our result for CNF-SAT also applies to the easier problem of finding the longest common substring of binary strings with don't cares. We also give strong conditional lower bounds for the more general Multiple Local Alignment problem on k strings, under both k-wise and SP scoring, and for other string similarity problems such as Global Alignment with gap penalties and normalized Longest Common Subsequence.

1 Introduction

Many basic string and pattern matching problems have overwhelming importance in current bioinformatics research. A well known such problem is the Local Alignment problem which asks to find the two substrings of two given strings that are most similar, under some given similarity measure. The fastest theoretical algorithm for this problem runs in time $O(n^2/\log n)$ [17,41,9] on n-length strings, and is not much faster than the classical dynamic programming algorithm of Smith and Waterman [53] which runs in $O(n^2)$ time. A faster algorithm for this problem, one that runs in, say $O(n^{1.5})$ time, would have tremendous impact, as witnessed by the 49,000 citations to the paper introducing the practical BLAST (Basic Local Alignment Search Tool) algorithm [4]. However, there seems to be very little optimism in the computational biology community that Local Alignment and other important string problems admit such "truly subquadratic" algorithms, i.e. running in time $O(n^{2-\varepsilon})$ for $\varepsilon > 0$. Yet, the theoretical computer science community has not provided any evidence for this impossibility, and in particular, it is yet to give an answer to this pressing question: can Local Alignment be solved in truly subquadratic time?

Perhaps the main reason for this lack of an answer, is that we do not have a clear technique for providing negative answers to such questions. The state of the art on unconditional lower bounds seems far from proving any significant superlinear lower bounds in the near future. The theory of NP-completeness cannot distinguish between quadratic upper bounds and $n^{1.5}$ ones. The $W[t]$-hardness approach of parameterized complexity requires parameterization, and like NP-hardness, does not distinguish between differing polynomial runtimes. Another approach is to prove lower bounds for a restricted family of algorithms. However, it is unclear what an appropriate candidate family would be, and either way, a restricted model lower bound only gives a partial answer to the question.

Our approach. We follow an approach that can be viewed as a refinement of NP-hardness. The importance of showing NP-hardness for a certain problem lies in the consequence that a polynomial

time algorithm for this problem would also imply a polynomial time algorithm for many other problems that are widely believed to require superpolynomial solutions. The goal of this work and previous works that are mentioned below is to develop such theory that is able to prove that improving the exact running times of certain problems would also imply surprising algorithms for many other problems and is therefore unlikely.

Using this approach we are able to provide the following answer to our pressing question, which is stated more formally in our theorems: A truly subquadratic algorithm for Local Alignment is unlikely because it would also give truly faster algorithms for other famous problems like CNF-SAT, 3-SUM and Max-Weight-4-Clique, implying breakthroughs in three different areas of computer science: the satisfiability algorithms and circuit lower bounds, the computational geometry and the graph algorithms communities!

To provide such answers, to this and other important questions about the optimality of current upper bounds for string problems, we devise careful reductions to the string problems from famous problems that are widely believed to require certain running times, not necessarily quadratic.

1.1 3-SUM Hardness

The most prominent example of this approach is the theory of 3-SUM hardness which was introduced by Gajentaan and Overmars [27] and has been used to show that subquadratic upper bounds for many problems in Computational Geometry are unlikely.

In the 3-SUM problem we are given three lists of n numbers and are asked whether we can pick a number from each list so that the sum is 0. A simple algorithm solves the problem in $\tilde{O}(n^2)$ time, and Baran, Demaine and Pătraşcu [7] were able to get a $O(n^2/\log^2 n)$ solution, yet any improvement beyond this seems unlikely and it is a widely believed conjecture that a polynomial improvement on the upper bound is impossible. Support for this belief comes from the $\Omega(n^2)$ lower bound for the depth of an algebraic decision tree for the problem [25].

Conjecture 1 (3-SUM Conjecture) *In the Word RAM model with words of $O(\log n)$ bits, any algorithm requires $n^{2-o(1)}$ time in expectation to determine whether three sets $A, B, C \subset \{-n^3, \ldots, n^3\}$ with $|A| = |B| = |C| = n$ integers contain three elements $a \in A, b \in B, c \in C$ with $a + b + c = 0$.*

Since [27], there have been many papers proving the hardness of computational geometry problems, based on 3SUM, e.g. [21,40,24,14,8]. More recently, the 3-SUM Conjecture has been used in surprising ways to show polynomial lower bounds for purely combinatorial problems in dynamic algorithms [45,2] and Graph algorithms [45,34,54]. The only previous work relating 3-SUM to a Stringology problem, to our knowledge, is the result of Chen et al. [13] showing that under the 3-SUM Conjecture, when the input strings are encodings of much longer strings, using Run-Length-Encoding, then the string matching with don't cares problem requires time that is quadratic in the lengths of the *compressions*. This string problem, however, is strongly related to geometric problems and is less "combinatorial" than the problems we consider here (e.g. it is solvable by a sweep-line algorithm). Thus our reductions require different techniques.

We expand the list of 3-SUM hard problems, showing a reduction from 3-SUM to the Local Alignment problem, proving that a truly subquadratic algorithm for Local Alignment is impossible under the 3-SUM Conjecture, provided the alphabet is large enough.

Theorem 1. *If for some $\varepsilon > 0$, $\delta \in (0, 1)$, one can solve the Local Alignment problem on two strings of length n over an alphabet of size $n^{1-\delta}$ in time $O(n^{2-\delta-\varepsilon})$, then the 3-SUM Conjecture is false.*

To prove Theorem 1 we combine a recent reduction from k-SUM to k-Vector-SUM [1] with an efficient self reduction for 3-SUM using hashing [7,45], then we carefully construct a scoring scheme. The details are given in Section 3.

We note that it is not hard to argue that there is an unconditional lower bound of $\min\{|\Sigma|^2, n^2\}$ for Local Alignment where Σ is the alphabet. When $|\Sigma| = n^{1-\delta}$, this lower bound is $\Omega(n^{2-2\delta})$. Our lower bound for such alphabets is essentially $n^{2-\delta}$ which is a *polynomial* improvement over $n^{2-2\delta}$. Nevertheless, our reduction requires alphabet size at least n^{ε} for some arbitrarily small but constant $\varepsilon > 0$, and the really interesting case of Local Alignment is when the alphabet size is constant, e.g. 4 for DNA and RNA sequences and 20 for protein sequences. We are not able to use the 3-SUM Conjecture to conclude a lower bound for this case. To handle the constant alphabet case, we turn to the presumed hardness of CNF-SAT to prove a lower bound even for *binary* strings.

1.2 Strong ETH Hardness

Despite hundreds of papers on faster exponential algorithms for NP-Hard problems in recent years (see the surveys by Woeginger for an exposition [59]), and despite the remarkable effort put into obtaining faster satisfiability algorithms, the best upper bounds for CNF-SAT on n variables and m clauses remain of the form $2^{n-o(n)}\text{poly}\,(m)$ (e.g. [30,47,52]). The Strong Exponential Time Hypothesis (Strong ETH) of Impagliazzo, Paturi and Zane, which has received a lot of attention recently, states that better algorithms do not exist.

Conjecture 2 (Strong ETH) *For every $\varepsilon > 0$, there exists a k, such that SAT on k-CNF formulas on n variables cannot be solved in $O^*(2^{(1-\varepsilon)n})$ time.*

Strong ETH is an extremely popular conjecture in the exact exponential time algorithms community [12,20,38,19], and Cygan et al. [18] even showed it to be equivalent to assuming that several other NP-hard problems essentially require exhaustive search. Recently, many surprising lower bounds in several different areas were shown to hold under the SETH, including lower bounds for approximating the diameter of a sparse graph [51], for maintaining the number of strongly connected components in a dynamic graph [2], and for the 3-party communication complexity of Set-Disjointness [49].

We show a reduction from CNF-SAT to the *longest common substring with don't cares* problem, which is one of the simplest string problems for which truly subquadratic algorithms are not known and is a very restricted version of the Local Alignment problem.

Definition 1 (The longest common substring with don't cares problem). *Given a string S over alphabet $\Sigma = \{0, 1\}$ and a string T over $\Sigma \cup \{\star\}$, find the length of the longest string that is a substring of both S and T, where a $\star$ in T can be treated as either 0 or 1.*

If don't care letters are not allowed, the problem can be solved using a generalized suffix tree in $O(n)$ time [29]. If instead of looking for the longest common substring, one wants to check whether a binary string with don't cares appears as a substring in a length n binary string, then there are $O(n \log n)$ time algorithms based on the Fast Fourier Transform [26,33,35]. Thus only slight variations of the longest common substring with don't cares problem admit almost *linear* time solutions, yet our reduction implies that a truly *subquadratic* algorithm for it refutes Strong ETH!

Theorem 2. *If for some $\varepsilon > 0$ one can solve either the Local Alignment problem on two binary strings of length n, or the longest common substring with don't cares problem in time $O(n^{2-\varepsilon})$, then Strong ETH is false.*

To prove Theorem 2 we represent a partial assignment to the variables of our formula with a substring, and we make sure that two substrings will match if and only if the two partial assignments satisfy all the clauses of our formula. The details are given in Section 2.

1.3 Multiple Sequence Alignment

In Gusfield's book [29], algorithms for comparing multiple strings are called "the holy grail" of current research in computational biology. One of the important tasks is the Multiple Local Alignment (MLA) problem [36] defined as follows. Given k strings $T_1, \ldots, T_k$ over some alphabet, find substrings $S_1, \ldots, S_k$ (where S_i is a substring of T_i) whose alignment has maximum score. There are multiple ways to define the alignment score between k substrings but we focus on two options, the most general k-wise scoring scheme and the most popular Sum of Pairs (SP) scoring scheme [6].

The k-wise scoring function $s(\cdot, \ldots, \cdot)$ which is given as a k dimensional matrix with $(|\Sigma|)^k$ entries. This case is called k-wise scoring, and the score of an alignment of k strings $s_1, \ldots, s_k$ is $\sum_i s(s_1[i], \ldots, s_k[i])$[3]. The second option, which is called sum of pairs (SP) scoring, is to use a pairwise scoring function $s(\cdot, \cdot)$ and to define the score of an alignment to be $\sum_i \sum_{k<\ell} s(s_k[i], s_\ell[i])$.

The MLA problem on $k \geq 3$ strings can be defined using either k-wise scoring or SP scoring. Local Alignment is the $k = 2$ case and both scoring rules coincide.

For both scoring schemes, unsurprisingly, the best upper bound for the problem is $O(n^k)$ using dynamic programming. The reduction of Wang and Jiang [56] from the shortest common superstring problem on k strings to a polynomial number of instances of the (global or local) alignment problem on $k+2$ strings with SP scoring implies that our problem is NP-hard when the number of strings is unbounded. Moreover, the W[1]-hardness results of Bodlaender et al. [11] for unbounded alphabets and of Pietrzak [48] for constant size alphabets also carry on to our problems, implying that upper bounds of the form $f(k) \cdot n^c$ for a constant c independent of k and n are unlikely. Huang [31] strengthens Pietrzak's reduction from k-clique to the shortest common superstring problem and shows that $n^{o(k)}$ algorithms for our problems are not possible under the plausible Exponential Time Hypothesis [32].

These negative results deliver an important message to biologists, showing that an efficient algorithm for optimally aligning a hundred strings is not likely to exist, yet another pressing question remains widely open: is there an algorithm running in time $O(n^{k-1})$ or even $O(n^{k/5})$ for MLA?[4] Such algorithms would imply a major advance in our ability to analyze biological data.

We extend our reduction from CNF-SAT to show that Strong ETH implies a negative answer to our question, when we are interested in k-wise scoring, even when the strings are binary.

Theorem 3. *If for some $\varepsilon > 0$ the MLA on k binary strings of length n with k-wise scoring can be solved in time $O(n^{k-\varepsilon})$, then Strong ETH is false.*

As is stressed in Gusfield's book [29], the less general case of SP scoring has more applications in Bioinformatics. Our reduction from CNF-SAT requires the computation of an OR function, which is easy with k-wise scoring yet does not seem possible with SP scoring. We show, however, that the Weighted-k-Clique problem can explain the hardness of getting faster algorithms even for the SP case of MLA.

[3] In a more general alignment, one can align alphabet symbols with spaces, and the scoring function can take that into account. In this paper, we prove hardness even for the easier alignment problem where no spaces are allowed.

[4] In the reduction of Bodlaender et al. [11] k increases to k^2 and in Pietrzak's [48] reduction n increases to n^7.

Weighted k-Clique. The Max-Weight k-Clique problem is as follows. Given a graph $G = (V, E)$ with integer edge weights, find a k-clique of maximum total weight, or determine that no k-clique exists. Since k-Clique is a special case of the problem, the Max-Weight k-Clique problem is NP-complete and $W[1]$-hard. The unweighted k-Clique admits an $O(n^{0.792k})$ time algorithm using matrix multiplication [44]. When the edge weights are small, one can obtain $O(n^{k-\varepsilon})$ time algorithms for Max-Weight k-Clique as well, by reducing to a small number of instances of k-Clique. However, when the weights are larger than n^k, the trivial $O(n^k)$ algorithm is essentially the best known (ignoring $n^{o(1)}$ improvements).

We show a tight reduction from Max-Weight $2k$-Clique to MLA on k strings with SP scoring, showing that improving on n^k time for MLA would also imply that Max-Weight $2k$-Clique has faster than n^{2k} algorithms.

Theorem 4. *If for some $\varepsilon > 0$ the MLA on k strings of length n over an alphabet of size $\sqrt{n}$ with SP scoring can be solved in time $O(n^{k-\varepsilon})$, then Max-Weight $2k$-Clique on n nodes graphs can be solved in time $O(n^{2k-\varepsilon})$.*

We prove the above theorem in Section 4. An immediate corollary of this theorem is a conditional lower bound for the Local Alignment problem for two strings, based on the assumption that Max-Weight 4-Clique does not have improved algorithms.

Corollary 1. *If for some $\varepsilon > 0$ the Local Alignment on two strings of length n over an alphabet of size $\sqrt{n}$ can be solved in time $O(n^{2-\varepsilon})$, then Max-Weight 4-Clique on n nodes graphs can be solved in time $O(n^{4-\varepsilon})$.*

We note that the special case of Max-Weight k-clique for $k = 3$ is especially interesting. The Max-Weight 3-Clique problem on n-node graphs is known to be essentially equivalent to the All Pairs Shortest Paths (APSP) problem, in the sense that if Max-Weight 3-Clique has a truly subcubic algorithm, so does APSP, and vice versa. It is a longstanding open problem whether APSP on n-node graphs can be solved in $O(n^{3-\varepsilon})$ time [58]. It is thus a major open problem whether Max-Weight 3-Clique is in $O(n^{3-\varepsilon})$ time for some $\varepsilon > 0$. Our current reductions show that a $O(n^{1.5-\varepsilon})$ time algorithm for Local Alignment on two strings would give $O(n^{3-\varepsilon})$ time for APSP, but we suspect that they can be strengthened to show a tighter relationship

Extensions. In Section 5 we also show quadratic lower bounds for well-known generalizations of the *Global* Alignment problem like Alignment with Gap penalties [28] and Alignment with moves [39], and for other string problems like Normalized LCS [5,23] and Partial Match [15,50,10].

2 From CNF-SAT to Alignment

In this section we give a reduction from CNF-SAT on n variables and m clauses to the longest common substring with don't cares problem on binary strings of length $N = O(2^{n/2} \cdot m)$ in $O^*(2^{n/2} \cdot m)$ time. Thus, given an algorithm for this problem that runs in time $O(N^{2-\varepsilon})$ for some $\varepsilon > 0$, we can solve CNF-SAT in time $O^*((2^{n/2} \cdot m)^{2-\varepsilon}) = O^*(2^{(1-\varepsilon/2)n} \cdot \text{poly}\ (m))$, refuting Strong ETH. We then explain how to get a reduction to Local Alignment on binary strings, proving Theorem 2, and give the extension to MLA on k strings to prove Theorem 3.

Our reduction follows the split and list technique introduced by Williams [57]. In particular, our reduction from CNF-SAT to the longest common substring problem can be obtained by combining

his reduction from CNF-SAT to the *orthogonal vectors* problem on binary vectors and a simple reduction from the latter problem to the longest common substring problem. Below we present a direct reduction from CNF-SAT that makes our extension to MLA on k strings follow more clearly.

Lemma 1. *CNF-SAT over formulas on n variables and m clauses can be reduced to the longest common substring with don't cares problem over strings of length $O(2^{n/2} \cdot m)$ and constant-size alphabet in $O^*(2^{n/2} \cdot m)$ time.*

Proof. Let $V = \{x_1, \ldots, x_n\}$ be the n variables of the input CNF formula φ. We split V into two sets of $n/2$ variables, $U = \{x_1, \ldots, x_{n/2}\}$ and $V \setminus U$. Let $A = \{\alpha_1, \ldots, \alpha_N\}$ and $B = \{\beta_1, \ldots, \beta_N\}$ be the sets of all $N = 2^{n/2}$ partial assignments of boolean values to the variables in U and $V \setminus U$, respectively. Combining two partial assignments $\alpha \in A$ and $\beta \in B$ gives an assignment $(\alpha \cdot \beta)$ to all the variables in the formula. We say that a partial assignment α satisfies a clause C if α either assigns TRUE to a variable that appears in C as a positive literal or it assigns FALSE to a variable that appears in C as a negative literal. The key idea of the reduction is the following simple observation, which gives a way of checking the satisfiability of ϕ: *The formula ϕ is satisfiable if and only if there are two partial assignments $\alpha \in A, \beta \in B$ such that for every clause C in the formula at least one of α and β satisfies the clause C.*

The reduction will generate two strings S and T. The string S will have N segments of length $5m$ corresponding to the N partial assignments in A and each segment will encode which clauses are satisfied by the partial assignment. Similarly, T will encode the partial assignments in B. A common substring of S and T will have to be entirely contained in these segments, and it will be able to be the whole segment only if the two corresponding assignments satisfy all the clauses of the formula. Therefore, the longest common substring will be of length $5m$ if and only if ϕ is satisfiable.

Let $C_1, \ldots, C_m$ be the clauses of our CNF formula ϕ. For $\alpha \in A$ we define the segment string S_α to contain a different symbol in the $(5j-2)^{th}$ position $S_\alpha[j]$ according to whether α satisfies C_j. Then, $\forall j \in [m]$:

$$S_\alpha[(5j-4)\ldots(5j)] = [01x_j01], \text{where } \; x_j = 1 \text{ if } \alpha \text{ satisfies } C_j, \text{ and 0 otherwise.}$$

Similarly, for $\beta \in B$ we define the segment T_β as follows. $\forall j \in [m]$:

$$T_\beta[(5j-4)\ldots(5j)] = [\star 1 y_j \star 1], \text{where } \; y_j = \star \text{ if } \beta \text{ satisfies } C_j, \text{ and 1 otherwise.}$$

Note that we can construct these strings for every $\alpha \in A$ and $\beta \in B$ given ϕ in $O^*(2^{n/2}m)$ time. Finally, we create S by concatenating all the segment strings S_{α_i} for all $\alpha_i \in A$ and placing "unmatchable" [000] segments between them, and we create T similarly by concatenating the T_{β_i} segment strings and placing [111] segments between them.

$$S = S_{\alpha_1} \circ 0^3 \circ S_{\alpha_2} \circ \cdots \circ 0^3 \circ S_{\alpha_N}, \; T = T_{\beta_1} \circ 1^3 \circ T_{\beta_2} \circ \cdots \circ 1^3 \circ T_{\beta_N}$$

Claim 1 *The longest common substring of S and T is of length $5m$ iff there are $\alpha \in A, \beta \in B$ such that every clause C_j in ϕ is satisfied by at least one of α, β.*

Proof. For the first direction, assume that there are some $\alpha \in A$ and $\beta \in B$ such that every clause is satisfied by one of them. Observe that in this case, S_α and T_β match, since by construction, every coordinate matches except for maybe the $(5j-2)^{th}$ for $j \in [m]$, but these coordinates must also match because of the following. For any $j \in [m]$, either $T_\beta[5j-2] = 1$ and β does not satisfy C_j in

which case α must satisfy C_j and $S_\alpha[5j-2] = 1$ as well, or $T_\beta[5j-2] = \star$ in which case $T_\beta[5j-2]$ matches $S_\alpha[5j-2]$. Therefore, S_α is a substring of S of length $5m$ that appears in T. By noting that any substring of S of length $5m+1$ cannot appear in T, we get that in this case, the longest common substring is of length exactly $5m$.

For the other direction, assume that there is a substring X of S that appears in T and has length $|X| = 5m$. Our careful construction implies that X cannot contain any letter from the "unmatchable" region of S and it cannot appear in T in any part that contains an "unmatchable" region. Therefore, X must correspond to a segment S_α for some $\alpha \in A$ that appears in T as a segment T_β for some $\beta \in B$. We show that this can only happen if the pair α, β satisfies all the clauses. This follows since for every $j \in [m]$ there are two cases: either β satisfies C_j, or $T_\beta[5j-2] = 1$ which then implies that $S_\alpha[5j-2]$ must be 1 as well and α satisfies C_j. □

This completes the proof of Lemma 1. □

2.1 The reduction to Local Alignment on binary strings

To get a reduction from CNF-SAT to Local Alignment on binary strings we note that in our reduction, the string T does not have any 0 letters, and therefore we can replace the $\star$ symbols by 0's, while treating them as a don't cares in our scoring function. Thus, the scoring function will be defined as $s(0,0) = s(1,1) = s(1,0) = 1$ and $s(0,1) = -\infty$. It is easy to verify that the optimal local alignment corresponds to the longest common substring. If one cares about the symmetry of the scoring function, then it seems that the alphabet needs to be of size 3 for the reduction to Local Alignment to work.

2.2 Generalizing to MLA with k-wise scoring

To prove Theorem 3 we reduce CNF-SAT to MLA on k binary strings of length $N = O(2^{n/k}m)$, showing that an algorithm that runs in time $O(N^{k-\varepsilon})$ for some $\varepsilon > 0$ implies an algorithm that runs in time $O^*(2^{(1-\varepsilon/k)n}\text{poly}\ (m))$ for CNF-SAT, refuting Strong ETH. The scoring scheme needs to be k-wise.

The reduction is quite similar to the reduction to the longest common substring with don't cares problem, but here we split the variables of our CNF formula to k sets of size n/k and we define k sets of $2^{n/k}$ partial assignments $A_1, \ldots, A_k$. The task is to find k partial assignments $\alpha_1 \in A_1, \ldots, \alpha_k \in A_k$ such that every clause in our formula is satisfies by at least one of them. Again, we define a segment S_α for every partial assignment α. This time the segments will be of length $3m$ and will be defined as follows, $\forall j \in [m]$:

$$S_\alpha[(3j-2)\ldots(3j)] = [1x_j1], \text{where}\ \ x_j = 1 \text{ if } \alpha \text{ satisfies } C_j, \text{ and 0 otherwise.}$$

The strings that we give to our instance of MLA will be made of concatenating the segments with "unmatchable" [000] segments between them. The scoring function will make sure that these parts are in fact unmatchable, and that aligned substring correspond to partial assignments for which at least one satisfies the current clause, by setting the score $s(0,0,\ldots,0)$ to be $-\infty$. For any other combination of letters the score will be set to $+1$. It is not hard to verify that the score of the optimal local alignment of these k strings is exactly $3m$ if and only if there is a satisfying assignment to our formula.

3 From 3-SUM to Alignment

In this section we prove Theorem 1 by proving the following lemma.

Lemma 2. *For any $0 < \delta < 1$, the 3-SUM problem on n numbers can be reduced in $n^{2-\delta+o(1)}$ time to $n^{\delta+o(1)}$ instances of the local alignment problem on two strings of length $\tilde{O}(n)$ over an alphabet Σ of size $\tilde{O}(n^{1-\delta})$.*

Proof. Given three lists $A, B, C \subseteq \{-n^3, \ldots, n^3\}$ of n numbers each, we want to find a triple of numbers $a \in A, b \in B, c \in C$ that sum to 0. We start by applying a hashing scheme that is due to Dietzfelbinger [22] and that has been used in recent works on 3-SUM [7,45,3,55]. Let $R = n^{1-\delta}$. There is a simple family $\mathcal{H}$ of hash functions $h : \{-n^3, \ldots, n^3\} \to [R]$ such that if we pick a function $h \in \mathcal{H}$ from the family at random, and hash each element x in our input sets $A \cup B \cup C$ to the bucket $B(h(x))$, we get the following properties:[5]

- *Almost linearity.* For any three numbers a, b, c, if $a+b+c = 0$ then $(h(a)+h(b)+h(c))$ modulo R is either 0 or 1.
- *Good load balancing.* The average number of elements x hashed into a bucket $B(x)$ is $3n/R$ and, in expectation, at most $O(R)$ elements are hashed to buckets with load exceeding $9n/R$.

The reduction picks a random hash function $h \in \mathcal{H}$ and hashes each element x in our lists to a bucket $B(h(x))$. For the $O(R)$ elements that fall in overloaded buckets we can run a brute force check to see if they participate in a 3-sum in $O(nR)$ time. Therefore, we can assume that we have at most R buckets $B(1), \ldots, B(R)$, each containing at most $t = 9n/R = O(n^{\delta})$ elements. We order the elements in these buckets $B(i) = \{x_1, \ldots, x_t\}$, and for each index j from 1 to t, we will have a separate stage. In stage j we check whether there is any element c in C such that c is the j^{th} element of its bucket $B(h(c))$ and $a + b + c = 0$ for some $a \in A, b \in B$. By the "almost linearity" property, it is enough to search for the pair $a \in A, b \in B$ among the elements a, b for which either $h(a) + h(b) = -h(c)$ or $h(a) + h(b) = -h(c) + 1$ (modulo R). To do these checks in every stage j, we create $n^{o(1)}$ instances of the local alignment problem, as described below. The total number of instances is therefore $t \cdot n^{o(1)} = n^{\delta+o(1)}$.

In a recent result by Abboud, Lewi and Williams (Lemma 3.1 in [1]), the authors show that we can construct a set of simple $N = n^{O(1/\log\log n)} = n^{o(1)}$ mappings $f_1, \ldots, f_N$ from numbers in $\{-n^3, \ldots, n^3\}$ to vectors in $\{-p, \ldots, p\}^d$ where $p = \log n$ and $d = O(\log n / \log\log n)$ with the following useful property[6]. If three elements $a, b, c \in \{-n^3, \ldots, n^3\}$ sum to 0, then for some $i \in [N]$, the vectors $f_i(a), f_i(b), f_i(c)$ will sum to the all-zero vector $\mathbf{0}$, while if three numbers a, b, c do not sum to 0, then for every $i \in [N]$, the vectors $f_i(a), f_i(b), f_i(c)$ will not sum to the all-zero vector. Note that the entries in each coordinate of our vectors are very small since $p = \log n$.

For every triple of numbers (i, j, z) where $i \in [N], j \in [t], z \in \{0, 1\}$ we create an instance of the Local Alignment problem, i.e. two strings S, T over alphabet Σ and a scoring function $s(\cdot, \cdot)$. The optimal solution to the (i, j, z) local alignment instance will determine whether there are three numbers $a \in A, b \in B, c \in C$ such that

1. c is the j^{th} element in its bucket $B(h(c))$,

[5] The value $h(x)$ is computed by taking a random odd integer a and returning the $\log R$ most significant bits from the number $a \cdot x$.

[6] The idea is simple: each number is mapped to a vector containing the base-p representation of the number, then enumerate over all guesses for the carries when summing k numbers in their base-p representation.

2. $h(a) + h(b) + h(c) = z \pmod R$, and
3. $f_i(a) + f_i(b) + f_i(c) = \mathbf{0}$.

If we find a triple satisfying these conditions then we have found a 3-sum, since by the property of the mappings from numbers to vectors described above, condition 3 can only happen if $a+b+c = 0$. On the other hand, if there is a 3-sum $a \in A, b \in B, c \in C, a + b + c = 0$ in our input set, then for some $(i, j, z) \in [N] \times [t] \times \{0, 1\}$, the above three conditions will hold and we will find this 3-sum. To see this, note that by the property of the mappings there must exist an $i \in [N]$ for which condition 3 holds, and by choosing $j \in [t]$ to be such that c is the j^{th} element in its bucket $B(h(c))$ we satisfy condition 1, and finally, by the "almost linearity" property of our hash function, $z = h(a) + h(b) + h(c) \pmod R$ is in the set $\{0, 1\}$ and condition 2 holds as well.

We now describe the Local Alignment instances that we generate for each triple $(i, j, z) \in [N] \times [t] \times \{0, 1\}$. Our alphabet Σ will contain a letter (h, y) for every pair of integers $h \in [R]$ (which will be used to indicate the value of a hash of a number) and $y \in \{-p, \ldots, p\}$ (which will represent the value in a coordinate of our vectors). We also add two symbols $\$_1$ and $\$_2$ to the alphabet. Note that by our choices of $p = \log n$ and $R = n^{1-\delta}$, we get that $|\Sigma| = \tilde{O}(n^{1-\delta})$. As in the reduction from CNF-SAT, the strings will be composed of segments. For every number a in A we create the segment S_a which will have length d and in its ℓ^{th} coordinate it will contain the letter $(h(a), f_i(a)[\ell])$. This letter encodes both the hash of a and the value in the ℓ^{th} entry of the vector $f_i(a)$ (corresponding to a in our current mapping i). Similarly, for every number b in B we define the segment T_b so that $T_b[\ell] = (h(b), f_i(b)[\ell])$ for every $\ell \in [d]$. The strings S, T of our instance (i, j, z) are constructed by concatenating the segments with \$ signs between them. Let $A = \{a_1, \ldots, a_n\}$ and $B = \{b_1, \ldots, b_n\}$, then $S = S_{a_1} \circ \$_1 \circ S_{a_2} \circ \$_1 \circ \cdots \circ \$_1 \circ S_{a_n}$ and $T = T_{b_1} \circ \$_2 \circ T_{b_2} \circ \$_2 \circ \cdots \circ \$_2 \circ T_{b_n}$.

The scoring function $s(\cdot, \cdot)$ is defined as follows. Given two letters (h_1, y_1) and (h_2, y_2), the scoring function will lookup $c \in C$, where c is the j^{th} element in bucket number $-(h_1 + h_2) + z \pmod R$, and return a score of 1 if $f_i(c) = -(y_1 + y_2)$. In any other case, the returned score is $-\infty$. Formally, for any pair of letters $(h_1, y_1), (h_2, y_2) \in [R] \times \{-p, \ldots, p\}$,

$$s\left((h_1, y_1), (h_2, y_2)\right) = \begin{cases} 1 & \text{if } c \in C \text{ is the } j^{th} \text{ element in } B((-(h_1 + h_2) + z) \mod R) \\ & \text{and } f_i(c) = -(y_1 + y_2). \\ -\infty & \text{otherwise} \end{cases}$$

We also disallow \$ symbols and gaps in the optimal alignment by giving a score of $-\infty$ to any pair containing them.

We now prove the following claim which shows that our construction will find a 3-sum if it exists, which completes the proof.

Claim 2 *There are two substrings of S, T in the (i, j, z) instance achieving a score of d if and only if there is a triple $a \in A, b \in B, c \in C$ satisfying conditions 1 to 3 above.*

Proof. For the first direction, let $a \in A, b \in B, c \in C$ be a triple satisfying conditions 1 to 3, and consider the score achieved by S_a and T_b under the scoring function $s(\cdot, \cdot)$ in the (i, j, z) instance. In each coordinate $\ell \in [d]$ of the segments, the score is

$$s(S_a[\ell], T_b[\ell]) = s((h(a), f_i(a)[\ell]), (h(b), f_i(b)[\ell])).$$

By condition 1 we know that c is the j^{th} element in its bucket, but by condition 2, we know that c is hashed to bucket $B(h(c))$ where $h(c) = -(h(a)+h(b))+z$ modulo R and therefore c is the j^{th} element

of bucket $-(h(a)+h(b))+z$. Thus, by definition of our scoring function $s(\cdot,\cdot)$, for every dimension $\ell \in [d]$, the score $s((h(a), f_i(a)[\ell]), (h(b), f_i(b)[\ell]))$ is equal to 1 whenever $f_i(c) = -(f_i(a) + f_i(b))$ which is true for every $\ell \in [d]$ by condition 3. Therefore, the total score achieved by S_a and T_b is d.

For the other direction, let X, Y be two substrings achieving a score of d. The "unmatchable" \$ signs imply that the lengths of X and Y cannot be larger than d, and since the maximum score for a pair of letters under $s(\cdot,\cdot)$ is 1, the lengths of our substrings must be larger than $(d-1)$. Therefore, X must correspond to a segment S_a for some $a \in A$ and Y must correspond to a segment T_b for some $b \in B$. Now, consider c, the j^{th} element of the bucket $-(h(a)+h(b))+z$ modulo R. Since the total score is d we know that in every coordinate $\ell \in [d]$ the contribution is 1,

$$s(S_a[\ell], T_b[\ell]) = s((h(a), f_i(a)[\ell]), (h(b), f_i(b)[\ell])) = 1,$$

which by definition of the scoring function implies that

$$f_i(c)[\ell] = -(f_i(a)[\ell] + f_i(b)[\ell]).$$

Therefore, in every dimension $\ell \in [d]$ we have that $f_i(a)[\ell] + f_i(b)[\ell] + f_i(c)[\ell] = 0$, and condition 3 holds for our triple $a \in A, b \in B$ and c. Note that c must be in C since otherwise all the scores would be $-\infty$. Moreover, conditions 1 and 2 clearly hold by our choice of c, and we found a triple satisfying conditions 1 to 3, as claimed. □

This completes the proof of Lemma 2. □

4 From Weighted Clique to Alignment

In this section we obtain efficient reductions for all $k \geq 2$ from the Max-Weight $2k$-Clique problem to the MLA on k strings *with SP scoring*. We can assume that the input graph is complete, by making each nonedge have weight $-\infty$.

An interesting observation about our reduction is that the length of the substrings in the optimal alignment is only $O(k+\log M)$, which is quite short, while one could have hoped for faster algorithms for the restricted problem in which we are only looking for short substrings. With more work, one can strengthen the reduction to make the length of the optimal substrings only $O(\log k + \log M)$. Theorem 4 in the introduction is an immediate consequence of the following Lemma.

Lemma 3. *Max-Weight $2k$-Clique on a weighted graph with n nodes and m edges can be reduced in $O(mk)$ time to MLA on k strings of length $O(m(k+\log M))$ and alphabet of size $O(n+\log M)$.*

Proof. Recall that the Max-Weight k-Clique problem is as follows. Given a graph $G = (V, E)$ with integer edge weights, find a k-clique of maximum total weight, or determine that no k-clique exists. Notice that we can assume that the input graph is complete, by making each nonedge have weight $-\infty$. We can replace all $-\infty$ occurrences by $-k^3M$ and this won't change our proof.

The alphabet. For each vertex u in the graph we have two types of letters: u and u'. We have the letters \$ and Λ from before. We also have a letter (b, i) for every $b \in \{0, 1\}$ and $i \in \{0, \ldots, \log M\}$. Now, any $\log M$ bit integer r can be represented as a string of letters as follows. Let $r[0], \ldots, r[\log M]$ be the bit representation of r. Then the string corresponding to r would be $(r[0], 0), \ldots, (r[\log M], \log M)$.

The scoring function $s(\cdot,\cdot)$. Intuitively, we want the score between any two strings corresponding to integers, to be their sum. We accomplish this by setting $s((b,i),(b',i)) = (b+b')\cdot 2^i$, and $s((b,i),(b',j)) = -\infty$ if $i \neq j$. The score between an integer letter and a vertex letter is $-\infty$.

We set the scores between vertex letters as follows. $s(u',u) = s(u',u') = s(u,u) = -\infty$, and for $u \neq v$, $s(u',v') = s(u',v) = w(u,v)$, and $s(u,v) = w(u,v)/(k-1)$. (We can assume all edge weights are in fact divisible by $(k-1)\cdot(k-2)$ by premultiplying them by $(k-1)\cdot(k-2)$. This won't increase our string length by more than a constant factor.) For \$ and Λ we set the scores as follows: $s(\$,a) = s(\Lambda,a) = -\infty$ for all $a \neq \Lambda$ and $s(\Lambda,\Lambda) = k^2M$.

The segments. For string j (from 1 to k), we have a segment S_{uv} of length $k+1$ for each ordered pair (u,v) of vertices. The segment is as follows: $S_{uv}[j] = u'$, $S_{uv}[0] = u'$, and $S_{uv}[\ell] = v$ otherwise. At the end of the segment, we have the string corresponding to the integer $w(u,v)/(k-1)$. We start and end each segment with Λ. Between any two segments in a string, we put a letter \$.

The weight of taking a segment from each string. Consider taking for string j, the segment corresponding to edge (u_j,v_j). By construction, if for some $i \neq j$, we have that the nodes u_i,v_i,u_j,v_j are not distinct, then the weight of taking these segments is $-\infty$. Hence we can assume that the nodes u_i,v_i,u_j,v_j are distinct. Consider the $2k$-clique formed by them. We'll show that its weight is the same as the value of the chosen segment alignment. The score of the segment alignment is formed as follows. For any of the $\binom{k}{2}$ choices of the k strings, i,j, the score from aligning their segments is obtained from aligning
(1) the three pairs u_i',v_j, u_j',v_i and u_i',u_j' appearing in positions i,j and 0 in the alignment; these give weight $w(u_i,v_j)+w(v_i,u_j)+w(u_i,u_j)$;
(2) the $k-2$ pairs v_i,v_j appearing in positions ℓ with $k \in \{1,\ldots,i-1\}\cup\{i+1,\ldots,j-1\}\cup\{j+1,k\}$; these have weight $(k-2)\cdot w(v_i,v_j)/(k-2) = w(v_i,v_j)$;
(3) the two number strings after position k in the segments; these give weight $w(u_i,v_i)/(k-1) + w(u_j,v_j)/(k-1)$.

The score of the alignment is thus exactly the sum of the edge weights in the $2k$ clique:

$$\sum_{ij} w(u_i,u_j) + \sum_{ij} w(v_i,u_j) + \sum_{ij} w(v_i,v_j) + \sum_i \sum_{j\neq i} w(u_i,v_i)/(k-1) = \sum_{ij}[w(u_i,u_j)+w(u_i,v_j)+w(u_j,v_i)+w(v_i,v_j)].$$

The score of any alignment of any substrings of blocks that do not use both Λs is no more than $\binom{k}{2}\cdot k^2M + kM + \binom{k}{2}(k+1)M < 2\binom{k}{2}k^2M$ which is the score of any alignment that uses both Λs, and so the optimum alignment uses both Λs. Thus full segments are used, and the optimal value of an alignment is $2\binom{k}{2}k^2M$ plus the maximum weight of a $2k$-clique in G. □

5 Extensions

In this section we show how Local Alignment can be replaced by other natural string problems in our reductions from CNF-SAT to conclude with tight lower bounds for these problems under Strong ETH.

5.1 Indexing with don't cares

The following problem (also known as the *Partial Match* problem) is a generalization of classical pattern matching in which the pattern might contain don't care letters, $\star$'s which can be treated as either 0 or 1.

Definition 2 (The indexing with don't cares problem). *Given a string T of length n over alphabet $\Sigma = \{0,1\}$, preprocess T in order to answer, given a query pattern P of length m over $\Sigma \cup \{\star\}$, whether P appears in T.*
A $\star$ in T can be treated as either 0 or 1

Notice that if don't care letters are not allowed, the problem can be solved optimally with $O(n)$ preprocessing and $O(m)$ query using a suffix tree. Without preprocessing, checking whether P appears in T can be done in $O(n \log n)$ time with algorithms based on the Fast Fourier Transform [26,33,35,42,16], and it is known that the problem is at least as hard as boolean convolution [43]. With any polynomial-time preprocessing of T, it is currently not known how to answer query P in $n^{1-\varepsilon}$ time in the worst case for any $\varepsilon > 0$. Solutions that can achieve $n^{1-\varepsilon}$ query are only known for restricted cases such as small number of don't care letters [15,50,10]. As for lower bounds, Pătraşcu [46] proved (a cell-probe lower bound) that with polynomial space the query-time must be at least logarithmic. Williams [57] proved that a similar problem requires $n^{1-o(1)}$ query time if allowed $O(n^{2-\varepsilon})$ preprocessing time, under Strong ETH. We show that even after $O(n^c)$ preprocessing time, for a constant $c > 1$, the query time must $n^{1-o(1)}$ under Strong ETH.

Theorem 5. *If for some $\varepsilon > 0$ there is an algorithm for indexing with don't cares that preprocesses a string T of length n over a binary alphabet $\{0,1\}$ in polynomial time and then answers whether a given query string P of length $m = O(\log n)$ over $\{0,1,\star\}$ appears in T in $O(n^{1-\varepsilon})$ time, then Strong ETH is false.*

Proof. The proof is similar to the proof of Lemma 1 with a few changes. Suppose T is preprocessed in $O(n^t)$ time for some integer t. Choose the smallest number k for which $k > t/(1-\varepsilon)$ and n/k is an integer, and let $U = \{x_1, \ldots, x_{n/k}\} \subseteq V$ be the set of first n/k variables in our formula ϕ. As in previous proofs, define the sets A and B to be the sets of partial assignments to the variables in U and $V \setminus U$, respectively. Note that now, however, the sizes $|A| \neq |B|$. In particular, $|A| = 2^{n/k}$ and $|B| = 2^{(1-1/k)n}$. Again, we are looking for $\alpha \in A$ and $\beta \in B$ such that all the clauses are satisfied. Here, we define the following segments T_α and P_β of length $2m$.

$$\forall j \in [m]: \quad T_\alpha[2j-1] = 0, \quad T_\alpha[2j] = \begin{cases} 1 & \text{if } \alpha \text{ satisfies } C_j \\ 0 & \text{otherwise} \end{cases}$$

$$\forall j \in [m]: \quad P_\beta[2j-1] = 0, \quad P_\beta[2j] = \begin{cases} \star & \text{if } \beta \text{ satisfies } C_j \\ 1 & \text{otherwise} \end{cases}$$

We construct the string T of length $2^{n/k} \cdot (2m+2)$ as follows.

$$T = T_{\alpha_1} \circ 1^2 \circ T_{\alpha_2} \circ 1^2 \circ \cdots \circ 1^2 \circ T_{\alpha_{2^{n/k}}}$$

We check the satisfiability of ϕ by checking whether for any $\beta \in B$, the string P_β appears in T.

Claim 3 *The formula ϕ is satisfiable if and only if for some $\beta \in B$ the string P_β appears in T.*

Proof. For the first direction, assume that there are some $\alpha \in A$ and $\beta \in B$ such that every clause is satisfied by one of them. Observe that in this case, T_α and P_β match, since for every $j \in [m]$, $P_\beta[2j-1] = T_\alpha[2j-1]$, and either $P_\beta[2j] = 1$ and β does not satisfy C_j in which case α must

satisfy C_j and $T_\alpha[2j] = 1$ as well, or $P_\beta[2j] = \star$ in which case $P_\beta[2j]$ matches $T_\alpha[2j]$. Therefore, P_β is a substring of P of length $2m$ that appears in T.

For the other direction, assume that for some $\beta \in B$ the string P_β appears in T. We claim that either P_β appears in T as a segment T_α or P_β is a substring that starts on the second letter of a segment and ends with a single '1' symbol (i.e., $P_\beta = T_\alpha[2 \dots (2m-1)] \circ 1$). This is because every alternating symbol in P_β is a '0' symbol . We show that in either case, we found a pair α, β such that every clause is satisfied by one of them. If $P_\beta = T_\alpha$, then we have that for every clause C_j, either β satisfies it or $P_\beta[2j] = 1$ which implies that $T_\alpha[2j] = 1$ and α satisfies it. In the other case, $P_\beta = T_\alpha[2 \dots (2m-1)] \circ 1$ and so for every $j \in [m-1]$, $P_\beta[2j] = T_\alpha[2j+1]$ and $T_\alpha[2j+1] = 0$, but $P_\beta[2j]$ can be either a $\star$ or a '1', and therefore it must be that $P_\beta[2j] = \star$ for every $j \in [m]$ which implies that β satisfies all the clauses on its own, and we are done. □

Now, if there is an algorithm that can preprocess T in $O(|T|^t)$ time for some integer t and then answer whether a given string P appears in T in $O(|T|^{1-\varepsilon})$, we can solve CNF-SAT by constructing and preprocessing the above T and querying the algorithm about P_β for every $\beta \in B$. The total running time is

$$\begin{aligned} |T|^t + |B| \cdot |T|^{1-\varepsilon} &= (2^{n/k}m)^t + 2^{(1-1/k)n} \cdot (2^{n/k}m)^{1-\varepsilon} \\ &\leq 2^{(1-\varepsilon)n} m^t + 2^{(1-\varepsilon/k)n} \cdot m = O^*(2^{(1-\varepsilon')n}), \end{aligned}$$

where $\varepsilon' = \varepsilon/k > 0$, which contradicts Strong ETH. This completes the proof of Theorem 5. □

5.2 Normalized LCS

The well known longest common subsequence (LCS) problem asks to find, given two strings T, P, the length of the longest string that is a subsequence of both T and P. This problem also has a natural $O(n^2)$-time dynamic programming solution. The local version of LCS has $\tilde{O}(n^2)$-time [5,23] solutions and is defined as follows.

Definition 3 (The normalized LCS problem). *Given two strings T, P over alphabet Σ and a positive integer M, find a substring X of T and a substring Y of P such that $LCS(X,Y) \geq M$ and $LCS(X,Y)/(|X| + |Y|)$ is maximized, where $LCS(X,Y)$ is the length of the longest common subsequence of X and Y.*

Lemma 4. *CNF-SAT over formulas on n variables and m clauses can be reduced to the normalized LCS problem over strings of length $O(2^{n/2} \cdot m)$ and constant-size alphabet in $O^*(2^{n/2} \cdot m)$ time.*

Proof. The reduction will follow the same steps as in the proof of Lemma 1, except that now the strings T and P that we construct will be such that their normalized LCS value will tell us whether the input CNF formula ϕ is satisfiable. Again, we split the variables into two sets U and $V \setminus U$ each of size $n/2$ and we let A and B be the sets of all $N = 2^{n/2}$ partial assignments to U and $V \setminus U$ respectively. We can check the satisfiability of ϕ by looking for a pair $\alpha \in A$ and $\beta \in B$ that satisfy all the clauses.

For any $\alpha \in A$, $\beta \in B$ we define the following two segments of length $4m$.

$$\forall j \in [m]: \quad T_\alpha[4j-3, \dots, 4j] = \begin{cases} 6416 & \text{if } \alpha \text{ satisfies } C_j \\ 6406 & \text{otherwise} \end{cases}$$

$$\forall j \in [m]: \quad P_\beta[4j-3,\ldots,4j] = \begin{cases} 6016 & \text{if } \beta \text{ satisfies } C_j \\ 6516 & \text{otherwise} \end{cases}$$

Then, we construct the following strings in $O^*(2^{n/2} \cdot m)$ time.

$$T = T_{\alpha_1} \circ 2^m \circ T_{\alpha_2} \circ 2^m \circ \cdots \circ 2^m \circ T_{\alpha_N}$$
$$P = P_{\beta_1} \circ 3^m \circ P_{\beta_2} \circ 3^m \circ \cdots \circ 3^m \circ P_{\beta_N}$$

We set $M = 3m$ and prove the following claim which completes the proof.

Claim 4 *There are two substrings X, Y of T, P with $LCS(X,Y) \geq M = 3m$ and $LCS(X,Y)/(|X|+|Y|) \geq 3/8$ if and only if ϕ is satisfiable.*

Proof. For the first direction, let $\alpha \in A$ and $\beta \in B$ be two partial assignments that satisfy all the clauses. We show that the segments T_α and P_β are substrings of T, P that satisfy $LCS(T_\alpha, P_\beta) = 3m = M$ and $LCS(T_\alpha, P_\beta)/(|T_\alpha| + |P_\beta|) = 3/8$. To see that $LCS(T_\alpha, P_\beta) = 3m$, consider the subsequence S defined as

$$S[3j-2] = T_\alpha[4j-3], S[3j-1] = T_\alpha[4j-1], S[3j] = T_\alpha[4j], \text{ for all } j \in [m].$$

Clearly, S is a subsequence of T_α. To see that S is also a subsequence of P_β, notice that whenever $S[3j-1] = 1$ we can pick $P_\beta[4j-1]$ which is always 1 and whenever $S[3j-1] = 0$ we know that α does not satisfy the clause C_j and therefore β must satisfy it and so $P_\beta[4j-2]$ is 1

For the other direction, assume that X, Y are substrings of T, P for which $LCS(X,Y) \geq 3m$ and $LCS(X,Y)/(|X|+|Y|) \geq 3/8$. We show that X, Y must correspond to two segments T_α, P_β for which α, β satisfies all the clauses. First, we note that we can assume w.l.o.g. that the first and last letters in X and Y match, since otherwise we can decrease the length of one of them by 1, This will not change the value of the LCS but it will increase the ratio so we will have another pair of substrings X', Y' that satisfy our assumptions. Second, we claim that the length of both X and Y cannot be larger than $4m$, since otherwise they will contain the '2' and '3' areas which cannot be matched and the ratio $LCS(X,Y)/(|X|+|Y|)$ will be less than $3/8$. Therefore, X and Y are some segments $X = T_\alpha$ and $Y = P_\beta$. Now we note that any subsequence of length $3m$ of two segments T_α, P_β must match all the '6' symbols to one another, and then for every $j \in [m]$, $T_\alpha[4j-1]$ needs to be matched to either $P_\beta[4j-2]$ or $P_\beta[4j-1]$. Therefore, by our construction of the segments, we have that $LCS(X,Y) \geq 3m$ can only happen if whenever α does not satisfy a clause C_j (i.e., $T_\alpha[4j-1] = 0$), $P_\beta[4j-2]$ must be 0 too, which means that β satisfies C_j. □

This completes the proof of Lemma 4. □

5.3 Edit distance with gaps

For global alignment (or edit distance), we can show an $n^{2-o(1)}$ lower bound for two extended definitions of alignment. The first is global alignment with gap penalties that can be solved in $O(n^2)$ time [28]. This is an extension of global alignment in which aligning k consecutive letters to spaces costs $f(k)$ for some linear function of k.

Definition 4 (The edit distance with gaps problem). *Given two strings T, P over alphabet Σ, compute their global alignment with a scoring function that allows gap penalties.*

To get a reduction to edit distance with gaps we define segment strings of length m as before:

$$\forall j \in [m]: \quad T_\alpha[j] = \begin{cases} 1 & \text{if } \alpha \text{ satisfies } C_j \\ 0 & \text{otherwise} \end{cases}$$

Then we let the strings T, P of length $O(Nm)$, where $N = 2^{n/2}$, over alphabet $\Sigma = \{0, 1, \$, \phi\}$ be:

$$\begin{aligned} T &= \$\ T_{\alpha_1}\ \$ \cdots \$\ T_{\alpha_N}\ \$ \qquad \phi^{N(m+1)} \qquad \$ \\ P &= \qquad \phi^{N(m+1)} \qquad \$\ T_{\beta_1}\ \$\ \cdots \$\ T_{\beta_N}\ \$ \end{aligned}$$

The scoring function is defined so that ϕ does not affect the score $s(\phi, \cdot) = 0$, the \$ signs cannot be aligned with 0 or 1 $s(\$, \$) = 0, s(\$, 0) = s(\$, 1) = -\infty$, and then we implement an OR function $s(0, 0) = -\infty, s(0, 1) = s(1, 0) = s(1, 1) = +1$. The gap scoring function is defined so that starting a gap sequence costs $-(m-1)/2$ but the length of the gap does not matter $f(k) = -(m-1)/2$. Finally, observe that the maximal score of an alignment of T and P is $+1$ if there is a satisfying assignment $(\alpha \cdot \beta)$ to our CNF-formula and 0 otherwise.

5.4 Edit distance with one move

The second extension of global alignment, called edit distance with moves, allows (at a given cost) to move entire substrings of the input strings. For unbounded number of moves, the problem is known to be NP-hard [39]. We prove a lower bound for a very restricted version of this problem.

Definition 5 (The one move distance problem). *Given two strings T, P over alphabet Σ, is there a string T' that can be obtained from T by moving one block such that the global alignment of T' and P has non-negative score?*

To get a reduction to one move distance we define the segments T_α exactly as in the previous construction, but now the strings T, P of length $O(Nm)$ are different. Below, we abuse notation and let α_i denote the segment T_{α_i} and let β_i denote T_{β_i}.

$$\begin{aligned} T &= \alpha_1\$\alpha_1\ \$ \cdots \$\ \alpha_{N-1}\$\alpha_{N-1}\ \$\ \alpha_N\$\alpha_N\ \$\ 1^N\$1^N\ \$ \cdots \$\ 1^N\$1^N \\ P &= 1^N\$1^N\ \$ \cdots \$ \quad 1^N\$1^N \quad \$\ \ \beta_1\$\beta_1\ \ \$\ \ \beta_2\$\beta_2\ \ \$ \cdots \$\ \beta_N\$\beta_N \end{aligned}$$

The scoring function is defined so that \$ can only be aligned with \$ while two letters in $\{0, 1\}$ can be aligned if their OR is 1: $s(\$, \$) = 0$ and $s(\$, x) = s(x, \$) = -\infty$ if $x \in \{0, 1\}$, while $s(0, 0) = -\infty, s(1, 0) = s(0, 1) = s(1, 1) = 0$. Observe that if there a satisfying assignment $(\alpha \cdot \beta)$ then we can move the block $(\alpha\$\alpha)$ to be right on top of $(\beta\$\beta)$ and get an alignment with cost 0, and otherwise there is no way to move one block in T and get a score that is not $-\infty$.

The above discussion and Lemma 4 allow us to conclude with the following lower bounds.

Theorem 6. *If for some $\varepsilon > 0$, normalized LCS, edit distance with gaps, or one move distance, on two strings and constant size alphabet can be solved in time $O(n^{2-\varepsilon})$, then Strong ETH is false.*

6 Discussion

We showed that under plausible assumptions, our upper bounds for Local Alignment are optimal, but many important questions remain unsolved. Perhaps the most important one is whether the Edit Distance problem might be easier, or whether one can prove that a subquadratic algorithm implies unexpected consequences. It is not hard to show that the local alignment problem is at least as hard as the global alignment and a very interesting open question is whether one can show the opposite direction (i.e., reduce the local alignment problem to the global alignment problem). In fact, any reduction from the problems in Section 5 to Edit distance would imply that our lower bounds hold for Edit Distance.

The scoring function. Our reductions rely on our ability to choose the scoring function for the problem. It could be that faster algorithms are achievable when the scoring function is restricted to be of some sort, and it would be of interest to understand the complexity of the alignment problems when instantiated with the standard scoring functions, e.g. Levenstein Distance [37] (where the score is 0 for matching identical characters and 1 otherwise).

Acknowledgements. We would like to thank Alex Bishara, Kevin Lewi, Ryan Williams and the anonymous reviewers for helpful discussions and comments. The first and second authors were supported by a Stanford School of Engineering Hoover Fellowship, NSF Grant CCF-1417238 and BSF Grant BSF:2012338. The third author was supported by Israel Science Foundation grant 794/13.

References

1. A. Abboud, K. Lewi, and R. Williams. On the parameterized complexity of k-sum. *CoRR*, abs/1311.3054, 2013.
2. A. Abboud and V. Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. *arXiv*, arXiv:1402.0054, 2014.
3. Amir Abboud and Kevin Lewi. Exact weight subgraphs and the k-sum conjecture. In *ICALP (1)*, pages 1–12, 2013.
4. S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. Basic local alignment search tool. *Journal of molecular biology*, 215(3):403–410, 1990.
5. A.N. Arslan, Ö. Eğecioğlu, and P.A. Pevzner. A new approach to sequence comparison: Normalized sequence alignment. *Bioinformatics*, 17(4):327–337, 2001.
6. David J. Bacon and Wayne F. Anderson. Multiple sequence alignment. *Journal of Molecular Biology*, 191(2):153 – 161, 1986.
7. Ilya Baran, Erik D. Demaine, and Mihai Pătraşcu. Subquadratic algorithms for 3SUM. *Algorithmica*, 50(4):584–596, 2008. See also WADS'05.
8. Gill Barequet and Sariel Har-Peled. Some variants of polygonal containment and minimum hausdorff distance undertranslation are 3SUM-hard. In *Proc. SODA*, pages 862–863, 1999.
9. P. Bille and M. Farach-Colton. Fast and compact regular expression matching. *Theoretical Computer Science,*, 409(3):486–496, 2008.
10. Philip Bille, Inge Li Gørtz, Hjalte Wedel Vildhøj, and Søren Vind. Less space: Indexing for queries with wildcards. In *SWAT*, pages 283–294, 2012.
11. Hans L Bodlaender, Rodney G Downey, Michael R Fellows, and Harold T Wareham. The parameterized complexity of sequence alignment and consensus. *Theoretical Computer Science*, 147(1):31–54, 1995.
12. C. Calabro, R. Impagliazzo, and R. Paturi. The complexity of satisfiability of small depth circuits. In *Parameterized and Exact Computation*, pages 75–85. Springer, 2009.
13. K. Chen, P. Hsu, and K. Chao. Approximate matching for run-length encoded strings is 3sum-hard. In *CPM*, pages 168–179. Springer, 2009.
14. Otfried Cheong, Alon Efrat, and Sariel Har-Peled. On finding a guard that sees most and a shop that sells most. In *Proc. SODA*, pages 1098–1107, 2004.

15. Richard Cole, Lee-Ad Gottlieb, and Moshe Lewenstein. Dictionary matching and indexing with errors and don't cares. In *STOC*, pages 91–100, 2004.
16. Richard Cole and Ramesh Hariharan. Verifying candidate matches in sparse and wildcard matching. In *STOC*, pages 592–601, 2002.
17. M. Crochemore, G.M. Landau, and M. Ziv-Ukelson. A subquadratic sequence alignment algorithm for unrestricted scoring matrices. *SIAM Journal on Computing*, 32:1654–1673, 2003.
18. M. Cygan, H. Dell, D. Lokshtanov, D. Marx, J. Nederlof, Y. Okamoto, R. Paturi, S. Saurabh, and M. Wahlstrom. On problems as hard as CNFSAT. In *Proc. CCC*, pages 74–84, 2012.
19. Marek Cygan, Stefan Kratsch, and Jesper Nederlof. Fast Hamiltonicity checking via bases of perfect matchings. In *STOC*, pages 301–310, 2013.
20. E. Dantsin and A. Wolpert. On moderately exponential time for SAT. In *Proc. 13th International Conference on Theory and Applications of Satisfiability Testing*, pages 313–325, 2010.
21. Mark de Berg, Marko de Groot, and Mark H. Overmars. Perfect binary space partitions. *Computational Geometry: Theory and Applications*, 7(81):81–91, 1997.
22. Martin Dietzfelbinger. Universal hashing and k-wise independent random variables via integer arithmetic without primes. In *STACS*, pages 569–580, 1996.
23. N. Efraty and G.M. Landau. Sparse normalized local alignment. In *CPM*, pages 333–346, 2004.
24. J. Erickson. New lower bounds for convex hull problems in odd dimensions. *SIAM Journal on Computing*, 28(4):1198–1214, 1999.
25. Jeff Erickson. Bounds for linear satisfiability problems. *Chicago J. Theor. Comput. Sci.*, 1999, 1999.
26. M.J. Fischer and M.S. Paterson. String matching and other products. *SIAM-AMS Proc.*, 7:113–125, 1973.
27. A. Gajentaan and M. Overmars. On a class of $o(n^2)$ problems in computational geometry. *Computational Geometry*, 5(3):165–185, 1995.
28. O. Gotoh. An improved algorithm for matching biological sequences. *J. Mol. Biol.*, 162:705–708, 1982.
29. Dan Gusfield. *Algorithms on strings, trees and sequences: computer science and computational biology*. Cambridge University Press, 1997.
30. E. A. Hirsch. Two new upper bounds for SAT. In *Proc. SODA*, pages 521–530, 1998.
31. Xiuzhen Huang. *Parameterized complexity and polynomial-time approximation schemes*. PhD thesis, Citeseer, 2004.
32. R. Impagliazzo and R. Paturi. On the complexity of k-sat. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001.
33. P. Indyk. Faster algorithms for string matching problems: Matching the convolution bound. In *Proc. FOCS*, pages 166–, 1998.
34. Z. Jafargholi and E. Viola. 3sum, 3xor, triangles. *Electronic Colloquium on Computational Complexity (ECCC)*, 20:9, 2013.
35. A. Kalai. Efficient pattern-matching with don't cares. In *Proc. SODA*, pages 655–656, 2002.
36. C. E. Lawrence, S. F. Altschul, M. S. Boguski, J. S. Liu, A. F Neuwald, and J. C Wootton. Detecting subtle sequence signals: a gibbs sampling strategy for multiple alignment. *science*, 262(5131):208–214, 1993.
37. V. I. Levenshtein. Binary codes capable of correcting deletions insertions and reversals. *Soviet Physics-Doklandy*, 10(8):707–710, 1966.
38. Daniel Lokshtanov, Dániel Marx, and Saket Saurabh. Known algorithms on graphs on bounded treewidth are probably optimal. In *SODA*, pages 777–789, 2011.
39. Daniel Lopresti and Andrew Tomkins. Block edit models for approximate string matching. *Theoretical Computer Science*, 181:159–179, 1997.
40. J. Erickson M. Soss and M. H. Overmars. Preprocessing chains for fast dihedral rotations is hard or even impossible. *Computational Geometry: Theory and Applications*, 26(3):235–246, 2002.
41. W.J. Masek and M.S. Paterson. A faster algorithm computing string edit distances. *Journal of Computer and System Sciences*, 20, 1980.
42. S. Muthukrishan and K. K. Palem. Non-standard stringology: Algorithms and complexity. In *STOC*, pages 770–779, 1994.
43. S. Muthukrishan and H. Ramesh. String matching under a general matching relation. *Information and Computation*, 122(1):140 – 148, 1995.
44. J. Nešetřil and S. Poljak. On the complexity of the subgraph problem. *Comment. Math. Univ. Carolin.*, 26(2):415–419, 1985.
45. Mihai Pătraşcu. Towards polynomial lower bounds for dynamic problems. In *Proc. 42nd ACM Symposium on Theory of Computing (STOC)*, pages 603–610, 2010.
46. Mihai Pătraşcu. Unifying the landscape of cell-probe lower bounds. *SIAM Journal on Computing*, 40(3):827–847, 2011. See also FOCS'08, arXiv:1010.3783.

47. R. Paturi, P. Pudlák, M. E. Saks, and F. Zane. An improved exponential-time algorithm for k-SAT. *J. ACM*, 52(3):337–364, 2005.
48. Krzysztof Pietrzak. On the parameterized complexity of the fixed alphabet shortest common supersequence and longest common subsequence problems. *Journal of Computer and System Sciences*, 67(4):757–771, 2003.
49. M. Pătraşcu and R. Williams. On the possibility of faster SAT algorithms. In *Proc. SODA*, pages 1065–1075, 2010.
50. M. S. Rahman, C. Iliopoulos, I. Lee, M. Mohamed, and W.F. Smyth. Finding patterns with variable length gaps or don't cares. In *COCOON*, pages 146–155, 2006.
51. L. Roditty and V. Vassilevska Williams. Fast approximation algorithms for the diameter and radius of sparse graphs. In *Proc. STOC*, pages 515–524, 2013.
52. U. Schöning. A probabilistic algorithm for k-SAT and constraint satisfaction problems. In *Proc. FOCS*, pages 410–414, 1999.
53. T.F. Smith and M.S. Waterman. The identification of common molecular subsequences. *Journal of Molecular Biology*, 147:195–197, 1981.
54. V. Vassilevska and R. Williams. Finding, minimizing, and counting weighted subgraphs. In *Proc. STOC*, pages 455–464, 2009.
55. Joshua Wang. Space-efficient las vegas algorithms for k-sum. *CoRR*, abs/1303.1016, 2013.
56. Lusheng Wang and Tao Jiang. On the complexity of multiple sequence alignment. *Journal of computational biology*, 1(4):337–348, 1994.
57. Ryan Williams. A new algorithm for optimal constraint satisfaction and its implications. In *ICALP*, pages 1227–1237, 2004.
58. V. Vassilevska Williams and R. Williams. Subcubic equivalences between path, matrix and triangle problems. In *Proc. FOCS*, pages 645–654, 2010.
59. G. J. Woeginger. Space and time complexity of exact algorithms: Some open problems. In *Proc. IWPEC, LNCS 3162*, pages 281–290, 2004.