The Complexity of Problems in P Given Correlated Instances*

Shafi Goldwasser[1] and Dhiraj Holden[2]

1 **Massachusetts Institute of Technology, Cambridge, USA**
shafi@theory.csail.mit.edu
2 **Massachusetts Institute of Technology, Cambridge, USA**
dholden@mit.edu

Abstract

Instances of computational problems do not exist in isolation. Rather, multiple and correlated instances of the same problem arise naturally in the real world. The challenge is how to gain computationally from correlations when they can be found. [DGH, ITCS 2015] showed that significant computational gains can be made by having access to auxiliary instances which are correlated to the primary problem instance *via the solution space*. They demonstrate this for constraint satisfaction problems, which are *NP-hard* in the general worst case form.

Here, we set out to study the impact of having access to correlated instances on the complexity of *polynomial time problems*. Namely, for a problem P that is conjectured to require time n^c for $c > 0$, we ask whether access to a few instances of P that are correlated in some natural way can be used to solve P on one of them (the designated "primary instance") faster than the conjectured lower bound of n^c.

We focus our attention on a number of problems: the Longest Common Subsequence (LCS), the minimum Edit Distance between sequences, and Dynamic Time Warping Distance (DTWD) of curves, for all of which the best known algorithms achieve $\tilde{O}(n^2)$ runtime via dynamic programming. These problems form an interesting case in point to study, as it has been shown that a $O(n^{2-\epsilon})$ time algorithm for a worst-case instance would imply improved algorithms for a host of other problems as well as disprove complexity hypotheses such as the *Strong Exponential Time Hypothesis*.

We show how to use access to a logarithmic number of auxiliary correlated instances, to design novel $o(n^2)$ time algorithms for LCS, EDIT, DTWD, and more generally improved algorithms for *computing any tuple-based similarity measure* - a generalization which we define within on strings. For the multiple sequence alignment problem on k strings, this yields an $O(nk \log n)$ algorithm contrasting with classical $O(n^k)$ dynamic programming.

Our results hold for several correlation models between the primary and the auxiliary instances. In the most general correlation model we address, we assume that the primary instance is a worst-case instance and the auxiliary instances are chosen with uniform distribution subject to the constraint that their allignments are ϵ-correlated with the optimal allignment of the primary instance. We emphasize that optimal solutions for the auxiliary instances will not generally coincide withan optimal solutions for the worst case primary instance.

We view our work as pointing out a new avenue for looking for significant improvements for sequence alignment problems and computing similarity measures, by taking advantage of access to sequences which are correlated through natural generating processes. In this first work we show how to take advantage of mathematically inspired simple clean models of correlation – the intriguing question, looking forward, is to find correlation models which coincide with evolutionary models and other relationships and for which our approach to multiple sequence alignment gives provable guarantees.

* This work was supported by an Akamai Presidential Fellowship, NSF MACS - CNS-1413920, and SIMONS Investigator award Agreement Dated 6-5-12

8th Innovations in Theoretical Computer Science Conference (ITCS 2017).
Editor: Christos H. Papadimitrou; Article No. 13; pp. 13:1–13:19

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1998 ACM Subject Classification F.2.2 Nonnumerical Algorithms and Problems

Keywords and phrases Correlated instances, Longest Common Subsequence, Fine-grained complexity

Digital Object Identifier 10.4230/LIPIcs.ITCS.2017.13

1 Introduction

A recent work of Dinur, Goldwasser, and Lin [6] which appeared in ITCS 2015 puts forth the following thesis: "Instances of computational problems of interest rarely exist in isolation, rather, multiple and correlated instances of the same problem arise naturally in the real world. The challenge is not to find settings which present correlated instances, but how to take advantage of correlations when they exist." Obvious examples are in biology where learning genotype to phenotype mappings of an organism can draw on data from multiple specimens with highly correlated DNA; or when multiple files are stored on (or transmitted over) the same error-prone medium, then error patterns in one file may be correlated to error patterns in other files which can be useful for error detection and recovery.

Indeed [DGH, ITCS 2015] show that significant computational gains can be made in solving instances of intractable combinatorial problems, when additional auxiliary instances are available which are *correlated to the primary instance via the solution space*. In particular, they show for several correlation models on the auxiliary instances polynomial time algorithms for constraint satisfaction problems which are NP-hard in the traditional worst case single-instance setting. Similarly, the work of Heninger et al [8] provides examples of number-theoretic problems where correlated cryptographic keys for the RSA function chosen due to poorly designed pseudorandom number generators enable efficient factorization of the RSA moduli involved. The works of [6] and [8] show that access to a polynomial number of auxiliary (but correlated) instances can enable us to find a polynomial time solution to an otherwise **intractable problem**.

In this paper, we set out to study the impact of having access to correlated instances on the complexity of **tractable problems**. Namely, for a polynomial time problem P, we ask whether we can use access to multiple instances of P that are correlated in some natural way, to solve P on one of them – the designated "primary instance" of size n– faster than the conjectured lower bound of n^c.

The question of which polynomial time problems to study is non-obvious. To draw meaningful conclusions from an algorithmic improvement to a polynomial time problem, one should choose problems which are at the same time classical and well-studied and for which there is hopefully some formal evidence to the complexity barrier they face, possibly in the form of a conditional lower bound. An approach for showing conditional lower bounds on a problem P which has gained significant prominence in the last few years and dates back to 1995[7], is to assume a lower bound on the complexity of one long standing open problem Q such as 3SUM or orthogonal-vectors and show a fine-grained reduction of Q to P which implies that any significant algorithmic improvement to P will imply an algorithmic improvement to Q. Another approach has been to show exact polynomial time lower bounds on P under complexity conjectures such as the Strong Exponential Time Hypothesis(SETH) which was introduced by Impagliazzo and Paturi[10] and asserts that solving satisfiability cannot be done significantly faster than exhaustive search.

This "conditional lower bound approach" has been successfully applied, as we detail below, [4, 1, 5] to various problems which compute similarity measures between strings, sequences

and curves, including computing the Longest Common Subsequence (LCS) between pairs and k-tuples of strings (kLCS), computing the minimal Edit Distance (EDIT) between pairs of strings and the dynamic time warping distance (DTWD) between curves. All these problems are currently addressed by dynamic programming algorithms which run in *quadratic time*.

These particular problems are not only the focus of intense theoretical study in the last few years, but arise in natural real-world settings according to some natural generating processes which is suggestive of the availability of multiple correlated instances. For example, the LCS and EDIT sequence alignment problems are often used in identifying conserved sequence regions across a group of sequences of DNA, RNA or proteins hypothesized to be evolutionarily related, or as aid in establishing evolutionary relationships by constructing phylogenetic trees. The dynamic time warping distance applied to signals over time can compare temporal data such as video and audio to detect whether an audio or video sequence is a sped up version of another audio or video sequence. Finally, we remark that to the best of our knowledge there are no known sub-quadratic algorithms for solving LCS, EDIT or DTWD on uniform (vs. worst case) input distributions nor are conditional lower bounds known, although this fascinating challenge has been raised in a multitude of works.

Looking ahead, in this work we will study and demonstrate how the availability of a logarithmic number of auxiliary instances which are correlated to primary instances of the LCS, EDIT and DTWD problems for several natural correlation modes enables us to overcome the quadratic complexity barrier.

But which correlation model over auxiliary instances should be assumed? In this first work we show how to take advantage of mathematically inspired simple and yet natural and clean models of correlation. The intriguing question, looking forward, is to find correlation models which coincide with evolutionary models and other relationships and for which our approach to multiple sequence alignment gives provable guarantees. We view our work as pointing out a novel direction to look for significant improvements, by taking advantage of access to sequences which are correlated through natural processes.

Our algorithmic approach departs from the dynamic programming approaches. It builds on the primary and auxiliary instances to construct new instances over a slightly larger alphabet for which solving the LCS (EDIT and DTWD respectively) can be done in sub-quadratic time and from which the optimal answer for the primary instance can be extracted. Rather than develop a different algorithm for each problem, we define a general notion of a tuple-based similarity measure which captures the LCS, EDIT and DTWD similarity measures on strings as special cases. We will show an algorithm for computing any tuple-based similarity measure over strings as long as access to correlated instances is available.

2 The New Results

We set out to study the impact of access to correlated instances on the complexity of polynomial time problems. The immediate questions which comes to mind is which problems and for which correlations

2.1 The problems: Any tuple-based Similarity Measure

We address the following problems.

- *Longest Common Subsequence* (LCS): Given two strings x and y over an alphabet Σ, find the maximum length of not necessarily consecutive substring common to both.
- *Minimum Edit Distance* (EDIT): Given two strings x and y over alphabet Σ, find the minimum number of insert character, delete character and replace character by another

character operations to apply to x to obtain y.

- *Dynamic Time Warping Distance (DTWD)*: Given two input strings x and y over an alphabet Σ compute the minimal cost of any allowed traversal of the two strings from first character to last where at each traversal step, one advances forward on at least x or y, and the cost is the sum over all time steps of the distance between the current entries. Namely, the sum of the distances between each pair of the traversal When the distances are restricted to $\{0,1\}$ we refer to the problem as $DTWD_{0,1}$
- *k-Longest Common Subsequence* (k-LCS) : Given k strings over alphabet Σ, find the maximum length not necessarily consecutive substring common to all k strings

Each of the above problems computes a different measure of similarity between pairs (or tuples) of strings, but all share a common feature which makes them amenable to speedups using correlated instances. Namely, the value of each of these similarity measures is the optimum over a set of values which only depend on the locations of the alignment. When this is the case, we will show that we can use a correlation model where the alignment stays the same (or with higher probability than uniform stays the same) in the correlated instances, to give a faster algorithm for finding this optimal alignment.

Formally, we define *tuple based similarity measure* which generalizes of all of the above and design an algorithm for any tuple based similarity measure, instead of treating each problem separately.

Main Definition: Let Σ denote an alphabet and let $f : \Sigma^{m_1} \times \Sigma^{m_2} \times, ..., \times \Sigma^{m_k} \to \mathbb{R}$ be a real valued function over k strings of lengths $m_1, ..., m_k$ whose output we informally think of as a measure of similarity between the strings. We say that f is a **tuple-based similarity measure** if there exists:

- a set of allowable k-tuples $\mathcal{S} \subseteq \bigcup_{m_1,m_2,...,m_k} \mathcal{P}([m_1] \times [m_2] \times ... \times [m_k])$ where $\mathcal{P}$ denotes the power set, (these can be viewed as the set of allowable "pseudo-alignments" where multiple positions in one string can match to the same position in another string) and
- a cost function $\mathcal{C}$ defined on pseudo-alignments $S \in \mathcal{S}$ s.t. $\mathcal{C}(S)$ is computable in time $\tilde{O}(|S|)$ and $f(x_1, x_2, ..., x_k) = \max\{\mathcal{C}(S)\text{: } S \in \mathcal{S} \text{ s.t. } \forall (i_1, i_2, ..., i_k) \in S, x_1[i_1] = x_2[i_2] = ... = x_k[i_k]\}$
  (or $f(x_1, x_2, ..., x_k) = \min\{\mathcal{C}(S)\text{: } S \in \mathcal{S} \text{ s.t. } \forall (i_1, i_2, ..., i_k) \in S, x_1[i_1] = x_2[i_2] = ... = x_k[i_k]\}$)

We note that this definition provides a generalization of the alignment gadget of [5] (where $k = 2$) which captures what is needed to have an improved algorithm given correlated instances. Bringmann and Kunnemann [5] introduce the notion of an alignment gadget, and show that any problem with an alignment gadget cannot be solved in strongly subquadratic time. Our definition is more general than theirs in that we do not restrict ourselves to looking at alignments: multiple positions in one string can match to the same position in another string. On the other hand, we only look at measures where the function is determined by the positions of the matches, whereas the gadgets of [5] speak of general measures.

It is easy to see that LCS is a pair-based similarity measure when we take $\mathcal{C}(S) = |S|$; the allowed sets of tuples are all possible alignments, and the maximum over all sets of alignments of $\mathcal{C}(S)$ is precisely the length of the longest common subsequence. In section 3, we prove that computing EDIT and DTWD are each a special case of a pair-based similarity measure.

2.2 The Models of Correlation Models over Auxiliary: Exact and Relaxed

The choices of which auxiliary instances (and which correlations) to consider for the LCS, k-LCS, $EDIT$, and $DTWD$ problems are tailor-made to the problems themselves. Indeed, one may argue that this will likely always be the case, as our hope is to consider correlations which would come up via *natural* generating processes of problem instances.

For example, in bioinformatics, sequence alignment is a way of arranging the sequences of DNA, RNA, or proteins to identify regions of similarity that may be a consequence of functional, structural, or evolutionary relationships between the sequences. Although we expect that only certain parts of the genome will be in common between multiple organisms, these parts will have a much higher rate of matching than random chance. Thus, which correlations between genomic sequences one may expect are dictated by evolutionary process which would be highly problem specific.

To describe distributions over the auxiliary instances, we use the framework of generating processes suggested by [6] to describe distributions over instances where correlations are defined between the *solutions* of the primary and auxiliary instance. A generating process $G_{Pri,Aux,t}$ for a problem P proceeds as follows: initialize the process with primary instance I of P drawn from primary distribution Pri. Let S be an underlying solution to I. Then choose t auxiliary instances $\{I_j\}_{j=1,...,t}$ from a distribution Aux conditioned on S in some fashion. The algorithm designer is given the tuple of $t+1$ instances $I, I_1, ..., I_t$ and is tasked with finding a solution to I.

We consider two generating models, which apply for any tuple-based similarity measure, but are easiest to first illustrate for the LCS problem.

Let Pri denote a distribution over pairs of strings of lengths m over alphabet Σ.

Exact LCS Correlation Model GenLCS$_{Pri}(m,t)$: Draw a pair of strings (x,y) from Pri. Let the longest common sub-sequence of (x,y) be at locations $\vec{a} = (a_1, ...a_n)$ in x and locations $\vec{b} = (b_1, ..., b_n)$ in y. Then choose auxiliary instances (x^1, y^1),...,(x^t, x^t) uniformly in $\Sigma^{m_1} \times \Sigma^{m_2}$ subject to the condition that each pair (x^j, y^j) contains a common sub-sequence at locations $\vec{a}$ and $\vec{b}$. Namely, $x^j_{a_i} = y^j_{b_i}$ for all j, for $1 \leq i \leq n$.

We next relax the restriction that each of the auxiliary instance pairs (x^j, y^j) contain a common (although not the longest) subsequence at the locations $\vec{a}$ and $\vec{b}$ of the longest common subsequence of the primary instance (x,y). Instead, we require that in each of the locations in (x^j, y^j) which coincide with locations of the longest common subsequence of x and y, the strings x^j and y^j agree with probability slightly higher than $\frac{1}{2}$.

Let Pri denote a distribution over pairs of strings of lengths m over alphabet Σ.

Relaxed LCS Correlation Model GenLCS2$_{Pri}(m,t,\epsilon)$: Draw a pair of strings (x,y) of length m each from Pri. Let the longest common sub-sequence of (x,y) be at locations $\vec{a} = (a_1, ...a_n)$ in x and locations $\vec{b} = (b_1, ..., b_n)$ in y. Then choose auxiliary instances (x^1, y^1),...,(x^t, x^t) uniformly in $\Sigma^m \times \Sigma^m$. Next, for each pair (x^j, y^j) and for each $1 \leq i \leq n$,set $x^j[a_i] := y^j[b_i]$ with probability ϵ and leave unchanged with probability $1-\epsilon$.

Correlations Models for General Tuple Based Similarity Measures

We are now ready to describe the models stated in terms of *any tuple-based similarity measure on strings*. We again consider two generating processes, an exact and a relaxed one.

The first generating process draws primary instances from a primary distribution, and auxiliary instances from the auxiliary distribution conditioned on the optimal set of tuples for the primary instances matching in each auxiliary instance. The second generating process will be similar to the first generating process, but instead of making the contents of the locations indicated by the optimum tuple always match in the auxiliary instances, there will only be a higher probability of matching in these locations.

Let f be a tuple-based similarity measure with allowed sets of tuples $\mathcal{S}$ and cost function $\mathcal{C}$.

Exact General Generating Model $Gen_{Pri,Aux}(f,k,\vec{m},t)$**:** First draw from the primary distribution Pri strings $x_1,x_2,...,x_k$ of lengths $\vec{m}=m_1,m_2,...,m_k$ where location tuples $S=\{(a_{11},a_{21},...,a_{k1}),...,(a_{1n},a_{2m},...,a_{kn})\}=\arg\min\{\mathcal{C}(S):S\in\mathcal{S}$ s.t. $\forall(i_1,i_2,...,i_k)\in S,x_1[i_1]=x_2[i_2]=...=x_k[i_k]\}$. Next, draw t auxiliary tuples $(x_1^{(i)},x_2^{(i)},...,x_k^{(i)})$ from distribution Aux conditioned on $\forall(j_1,j_2,...,j_k)\in S,x_1^{(i)}[j_1]=x_2^{(i)}[j_2]=...:=x_k^{(i)}[j_k]$ $\forall 1\leq i\leq t$ and output the primary and auxiliary instances drawn.

The main restriction in the above correlation model is that the optimal set of tuple locations must also induce a matching set of tuples for every auxiliary instance. Let us now relax this restriction to give faster algorithms when the optimal set of tuples only has a higher probability of inducing a matching set of tuples in the auxiliary instances.

Relaxed Generating Model $Gen_{Pri,Aux}(f,k,\vec{m},t,\epsilon)$**:** First draw from the primary distribution Pri strings $x_1,x_2,...,x_k$ of lengths
$\vec{m}=m_1,m_2,...,m_k$ where location tuples $S=\{(a_{11},a_{21},...,a_{k1}),...,(a_{1n},a_{2m},...,a_{kn})\}=\arg\min\{\mathcal{C}(S):S\in\mathcal{S}$ s.t. $\forall(i_1,i_2,...,i_k)\in S,x_1[i_1]=x_2[i_2]=...=x_k[i_k]\}$. Next, draws t auxiliary tuples $(x_1^{(i)},x_2^{(i)},...,x_k^{(i)})$ from distribution Aux Finally, set $x_1^{(i)}[j_1]=x_2^{(i)}[j_2]=...:=x_k^{(i)}[j_k]$ with probability ϵ for all tuples $(j_1,j_2,...,j_k)\in S$ and output the primary and auxiliary instances drawn.

Important Discussion on on Correlation Models: A word of caution is in order. One should be careful not to consider correlations which trivialize the problem.

- An example of such trivializations would be access to "too many" auxiliary instances. Whereas in [6] "too many" would correspond to more than a polynomial number of instances, in the current work it would be more than a linear number of auxiliary instances. Indeed, a logarithmic number of additional instances suffice to get a marked improvement.
- Another forbidden trivialization would be access to auxiliary "easy" instances for which the correlation enables trivial extraction of the answer for the primary instance. This is avoided in the exact correlation model as follows. Recall that the primary instance (x,y) can be selected in a worst case manner, whereas the auxiliary instances are selected at random subject to the condition that in each auxiliary pair x^j and y^j share a common subsequence at the same locations that primary x and y contain the *longest* common subsequence. However, we emphasize that the longest common subsequence for any of the auxiliary pairs, does not generally coincide with the longest common subsequence for the primary pair (x,y) . Therefore, we can not extract the solution to the LCS of (x,y) from a solution to one of the auxiliary pairs (x^j,y^j). Indeed, our solutions need $l=O(logn)$ auxiliary instances to be able to compute the solution to the primary instance and does not solve the LCS on auxiliary pairs (x^j,y^j). We also emphasize that the actual substring at the $\vec{a}$ locations of x^j does not agree with the substring at the $\vec{a}$ locations of the primary instance x, nor does the substring at the $\vec{b}$ locations of y^j match the substrings at the $\vec{b}$

locations of y, only the substring at locations $\vec{a}$ of x^j is required to match the substring at locations $\vec{b}$ of y^j.[1]

2.3 The New Algorithms in Exact and Relaxed Correlation Models

We start by describing our results for the LCS generating processeses.

Theorem 1 (LCS with exact correlation model) There exists an algorithm which on input $((x,y),(x^1,y^1),...,(x^t,y^t)$ generated according to the exact correlation model $GenLCS_{Pri}(m,t)$ defined above for Pri being a worst case distribution solves the LCS problem on string pair $(x,\ y\ )$ and runs in expected time $O(m\log m)$ for $t=O(\log m)$ where the expectation is taken over the choices of auxiliary inputs in $GenLCS_{Pri}(m,t)$.

The key technical insight for the aforementioned theorem is that it is now possible to construct a new instance of the LCS problem over an extended alphabet where each character corresponds to a vector in Σ^{l+1} and which now has enough structure (as opposed to the primary worst case instance) to be solved in sub-quadratic time. Essentially, each character in a position of the new LCS instance is determined by the characters in the original primary and auxiliary instances at the same position. When the number of auxiliary instances is large enough and the distribution is uniform, it will mean that the probability of any two non-matching positions having the same values for each of the auxiliary instances is small. This means that any positions that match on every auxiliary instance belong to the longest common subsequence with high probability. It is then possible to use hashing on the extended alphabet symbols to find the positions of the longest common subsequence.

The LCS results described are a special case of a general algorithm applied to any tuple-based similarity measure.

Main Theorem 2 (tuple-based similarity measures with exact correlation model) Let f be a tuple-based similarity measure on k strings of lengths $\vec{m}=m_1,...,m_k$ over alphabet Σ with no overlapping tuples. There exists an algorithm that finds $f(x_1,x_2,...,x_k)$ in time $\tilde{O}(kn\log n+|S|)$ where $|S|$ is the size of the largest allowed set of tuples on a primary instance $x_1,x_2,...,x_k$ and auxiliary instances $(x_1^{(1)},x_2^{(1)}...,x_k^{(1)}),...,(x_1^{(t)},...,x_k^{(t)})$ with $t=O(\log n)$ drawn from $Gen_{Pri,Aux}(f,k,\vec{m},t)$ with Pri worst-case and Aux uniform.
Note that in all of the similarity measures of interest, $|S|$ is of linear size in the input length.

Main Corollary 3

- On a primary instance $x_1,...,x_k$ of lengths $\vec{m}$ and $O(\log n)$, where $n=\max m_1,...,m_k$, auxiliary instances of k-LCS drawn from $Gen_{Pri,Aux}(LCS,k,\vec{m},O(\log n))$ with Pri worst-case and Aux uniform, we can find the longest common subsequence of the primary instance in time $\tilde{O}(nk)$.

[1] Interestingly, as we mentioned above, the induced distribution over the auxiliary instances produced by the exact correlation model, coincides with the semi-random model proposed by [3] in their work on the smooth complexity analysis of the EDIT problem. Recall, that they consider perturbing a worst case pair of strings (x,y) to (x',y') by randomly and independently choosing each of the characters of x' and y' in locations which were not part of the longest common subsequence of x and y; whereas for locations of x and y which are part of the longest common subsequence, x' and y' are forced to contain the same randomly chosen character. The existence of a linear time approximation algorithm for longest common subsequence for the (x',y') distribution, as shown by [3], does not seem to be useful directly for finding an approximate (nor exact) solution for LCS on (x,y) .

- On a primary instance x_1, x_2 of lengths m and n and $O(\log \max m, n)$ auxiliary instances of EDIT drawn from $Gen_{Pri,Aux}(EDIT, 2, \{m, n\}, O(\log \max m, n))$, we can find the minimum edit distance between the primary sequences in time $\tilde{O}(m + n)$.
- On a primary instance of x_1, x_2 of lengths m and n and $O(\log \max m, n)$ auxiliary instances of 0-1 DTWD drawn from $Gen_{Pri,Aux}(DTWD_{0,1}, 2, \{m, n\}, O(\log \max m, n))$ with Pri worst-case and Aux uniform, we can find the dynamic time warping distance of the primary instance in time $\tilde{O}(m + n)$.

Using considerably more complex algorithms and in particular techniques to find Hamming nearest neighbors, we extend the techniques to show the following for the relaxed correlation model.

Theorem 4 (LCS with relaxed correlation model) Let $\epsilon \in [0, \frac{1}{2}]$. There exists an algorithm which on a tuple of input instances $(x, y), (x^1, y^1), ..., (x^t, y^t)$ drawn from $GenLCS2_{Pri}(m, t, \epsilon)$ solves the LCS on string pair (x, y) with probability at least $2/3$ for $t = \Omega(\log m)$ and runs in expected time $O(tn^{1+d})$ for $d < 1$ (i.e sub quadratic time). (where the probability and runtime expectation is taken over the choices of auxiliary inputs in$GenLCS2_{Pri}(m, t, \epsilon)$.

Stating the general theorem for similarity measures.

Main Theorem 5 (tuple-based similarity measure with relaxed correlation model) Let f be a tuple-based similarity measure on k strings of lengths $\vec{m} = m_1, ..., m_k$ over alphabet Σ with no overlapping tuples. There exists an algorithm that finds $f(x_1, x_2, ..., x_k)$ in time $O(|S| + n^{2-\Omega(\epsilon^{1/3}/\log(1/\epsilon))})$ where $|S|$ is the size of the largest allowed set of tuples on a primary instance $(x_1, x_2, ..., x_k)$ and auxiliary instances $(x_1^{(1)}, x_2^{(1)} ..., x_k^{(1)}), ..., (x_1^{(t)}, ..., x_k^{(t)})$ drawn from $Gen_{Pri,Aux}(f, k, \vec{m}, t, \epsilon)$ with $t = O(\log n)$,Pri worst-case and Aux uniform.

Main Corollary 6

- On a primary instance $x_1, ..., x_k$ of lengths $\vec{m}$ and $O(\log n)$, where $n = \max m_1, ..., m_k$, auxiliary instances of k-LCS drawn from $Gen_{Pri,Aux}(LCS, k, \vec{m}, O(\log n), \epsilon)$ with Pri worst-case and Aux uniform, we can find the longest common subsequence of the primary instance in time $O(n^{2-\Omega(\epsilon^{1/3}/\log(1/\epsilon))})$.
- On a primary instance x_1, x_2 of lengths m and n and $O(\log \max m, n)$ auxiliary instances of EDIT drawn from
  $Gen_{Pri,Aux}(EDIT, 2, \{m, n\}, O(\log \max m, n), \epsilon)$, we can find the minimum edit distance between the primary sequences in time $O((m + n)^{2-\Omega(\epsilon^{1/3}/\log(1/\epsilon))})$.
- On a primary instance of x_1, x_2 of lengths m and n and $O(\log \max m, n)$ auxiliary instances of 0-1 DTWD drawn from $Gen_{Pri,Aux}(DTWD_{0,1}, 2, \{m, n\}, O(\log \max m, n), \epsilon)$ with Pri worst-case and Aux uniform, we can find the dynamic time warping distance of the primary instance in time $O((m + n)^{\Omega(\epsilon^{1/3}/\log(1/\epsilon))})$.

2.4 Another correlation model and results for EDIT

In addition to the correlation model implied for computing the min edit distance (EDIT), we consider another model which may be more natural to consider and show an efficient algoritme for EDIT in this model.

Consider the following generating process. Let $\pi_1, \pi_2, ..., \pi_k$ be the minimum sequence of edits needed to transform x to y. The auxiliary instances are random pairs of n-bit strings whose minimum edit sequence are the same as for the primary instance. Namely,

for auxiliary pair $x^{(1)}, y^{(1)}$, $y^{(1)}$ is obtained from $x^{(1)}$ using the same edit sequence which transformed x to y. Somewhat more generally, consider a worst case instance (x, y) and auxiliary instances $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), ...$, with $x^{(i)} = x$ with each character changed with probability $\epsilon \in [0, 1]$ and $y^{(i)} = \pi_k(\pi_{k-1}(...(\pi_1(x^{(i)}))...))$. This is the distribution on which our algorithms perform.

Theorem 7(Solve EDIT exactly with $O(\log n)$ instances) There exists an algorithm A which on inputs generated as above solves the $EDIT(x, y)$ for a worst case primary instance and auxiliary instances as above such that on $O(\log n)$ auxiliary instances , the algorithm computes $EDIT(x, y)$ with high probability and runs in expected time $O(n \log n)$.

2.5 Overview of Techniques

The main idea of this paper is a new algorithm for solving similarity measure problems with correlated instances. This algorithm is significantly different than the dynamic programming algorithms that are used in the absence of correlated instances.

Given a tuple-based similarity measure, our goal is to find the set of tuples of locations which form the optimal (e.g. longest in LCS) solution to the primary instance $(x_1, ..., x_k)$. With correlated instances, consider a tuple of locations (e.g. $(i_1, ..., i_k)$ in $(x_1, ..., x_k)$), then in our exact correlation model, if these locations are not part of optimal solution these will be detected by comparing them across all instances, primary and auxiliary. This is the case as the correlation model makes it unlikely that even a pair of locations (i_u, i_v), which are not part of a matching tuple in the primary instance, would have the same value in the primary instances and all auxiliary instances.

Thus, algorithmically we can look at pairs of positions and determine whether they are in the same tuple by comparing them across all instances, primary and auxiliary. We do not need to do this for all pairs; we can use hashing to map pairs to each other, using the fact that similar pairs will be in the same tuple, and thereby find the tuples that determine the value of the similarity measure. The use of hashing proceeds as follows. First, we construct vectors out of the same position for the primary instance and each auxiliary instance for all of the strings. Then, we construct a hash table out of the vectors for the first string, and use the hash function to determine which vectors in the other strings are the same as vectors in the first string.

This idea works as is for the exact correlation model where pairs of locations in tuple of the optimal solution are guaranteed to contain the same value in all auxiliary instances, but fails when we relax this to the model where contents of such location pairs only have a higher probability of being the same.

In this case, if two locations were part of the same tuple, when we consider the vectors obtained from the auxiliary instances at those locations, they will be closer to each other than vectors obtained from positions which are not part of the same tuple. As a result, the nearest neighbors of the vector corresponding to a position are the vectors corresponding to the positions in the same tuple. Then, we can use algorithms that find nearest neighbors faster than brute-force search to find the tuples. Thus we can solve instances generated by the relaxed correlation model not in nearly linear time like the exact correlation model, but still significantly faster than the best known algorithms for the problem without correlated instances.

2.6 Related Works

We review the relevant results known in the literature for the EDIT,LCS, and DTWD problems.

Backurs and Indyk [4] have shown that if EDIT – the minimum number of insertions, deletions, and substitutions needs to transform one string into another – can be computed in time $O(n^{2-\delta})$ for some constant $\delta > 0$, then the satisfiability of CNF formulas with n variables and m clauses can be solved orm one string into anothertime $poly(m)2^{\epsilon n}$ for a constant $\epsilon > 0$. This would violate the Strong Exponential Time Hypothesis (SETH). A $O(n^{2-\delta})$ algorithm for EDIT would also imply a sub-quadratic algorithm for the orthogonal vector problem and a host of others.

Abboud, Backurs and Williams [1] addresses (among other problems) the LCS problem. They show that an $O(n^{2-\epsilon})$ algorithm for the LCS of two sequences of length n over a constant size alphabet would refute SETH and show a sub-quadratic algorithm for the orthogonal vectors problem and a host of others. For finding the longest subsequence common to k input sequences, it is similarly argued that achieving $O(n^{k-\epsilon})$ for any $\epsilon > 0$ is unlikely in [1].

Bringmann and Kunnermann [5] provide a lower bound for dynamic time warping distance, the minimum cost of a traversal of two strings where at each step, at least one of the strings moves forward, and the cost is the sum of the distances at each step. This problem is also solvable using dynamic programming, and [5] show that an $O(n^{2-\epsilon})$ that finds the dynamic time warping distance would refute SETH. Their proof uses the concept of an alignment gadget, which we will generalize with our notion of pair-based similarity measures.

We also mention work on the smoothed complexity of the EDIT distance problem. Recall, that the smoothed complexity of an algorithm by Spielmann and Teng [12], analyzes for an input x the expected behavior of the algorithm on correlated inputs which are the result of subjecting x to perturbations (e.g. flip its bits with a certain probability).

Andoni and Krauthgamer [3] study of the smoothed complexity of EDIT in the following perturbation model. Given two adversarially chosen binary input strings with common longest subsequence A , eacn character of the input strings is replaced independely (with some fixed probability p) with one restriction: the same perturbation is applied in both strings to the charcters in locations of the common longest subsequence A. They then show a constant factor approximation algorithms for EDIT which runs in nearly linear time for such perturbed instances.The existence of a linear time approximation algorithm for the LCS of (x', y') distribution, as shown by [3], does not seem to be useful directly for finding an approximate (nor exact) solution for the LCS of (x, y). Looking ahead, the stringest dostributions which we consider (which we call "exact correlation model") over auxiliary instances to a primary instance x essentially coincides with the [3] perturbation model applied to x.

We remark that the approach of smoothed analysis in general and the work of [3] in particuar, differs from our study and results in several aspects. First, in contrast to smoothed analysis our goal is not an analysis of when the problem becomes easier (or harder) but rather the development of algorithms which take advantage of (and when) extra correlated information is available in addition to an original input. Second, we solve the exact version of the min-EDIT distance problem on the primary worst case input pair itself with high probability (over the perturbed instances) whereas they approximate the min-EDIT distance on a perturbed instance.

Having said that, there is an interesting interplay between the explorations of smoothed analysis and improving algorithmics by access to correlated instances . That is, one may consider any distribution induced by semi-random models over problem instances both as

distributions for which smoothed complexity can be studied, or as distributions over the auxiliary correlated instances which are available for improving algorithms on a primary worst case instance.

Finally, we note that the idea of using larger alphabets to make the longest common subsequence problem easier is related to work of Hunt and Szymanski [9]. In this paper, they show that if r is the number of pairs of locations that are the same in the two sequences, there is an algorithm to compute the longest common subsequence of two strings of length n in time $O((r+n)\log n)$. Thus, if the number of matching pairs is small, the longest common subsequence can be computed easily. This idea is present in our work as well; when we consider correlated instances to make a new pair of sequences over a larger alphabet, the matching pairs will just be the pairs that are part of the longest common subsequence of the primary instance. Although the algorithm we use to find the longest common subsequence is different, the fundamental concept is the same.

2.7 Roadmap to The paper

The rest of the paper will proceed as follows. In Section 2, we will give the definition of a tuple-based similarity measure. We will also introduce the definitions of longest common subsequence, edit distance, and dynamic time warping distance and then show that LCS, EDIT, and a restricted version of DTWD are all tuple-based similarity measures. In Section 3, we define the correlation models used in the rest of the paper. We are then able to proceed to the results in Section 4. Section 4.1 covers the exact correlation model and Section 4.2 covers the relaxed correlation model. We leave the discussion of the other model for edit distance to the appendix.

3 A generalization of similarity measures

The longest common subsequence problem, edit distance problem, and dynamic time warping distance problem all have similar features which make them amenable to speedups using correlated instances. The feature that we will be looking at is the fact that the value of each of these similarity measures is the optimum over a set of values which only depend on the locations of the alignment. If this is the case, we will show that we can use a correlation model where the alignment stays the same to give a faster algorithm for finding this optimal alignment. The following definition provides a criterion which ensures that a similarity measure will have a good algorithm given correlated instances.

▶ **Definition 1.** *Let f be a function that takes k strings and outputs a number, which we think of as a measure of similarity between the strings. f is a tuple-based similarity measure on strings $x_1, ..., x_k$ if there exist $\mathcal{S} \subseteq \bigcup_{m_1,m_2,...,m_k} \mathcal{P}([m_1] \times [m_2] \times ... \times [m_k])$ of allowed sets of tuples, where $m_1, ..., m_k$ correspond to the length of each string, along with a function $c(S)$ computable in time $\tilde{O}(|S|)$ such that*
$f(x_1, x_2, ..., x_k) = \max_{S \in \mathcal{S}, \forall (i_1,i_2,...,i_k) \in S x_1[i_1]=x_2[i_2]=...=x_k[i_k]} c(S)$ *or*
$f(x_1, x_2, ..., x_k) = \min_{S \in \mathcal{S}, \forall (i_1,i_2,...,i_k) \in S x_1[i_1]=x_2[i_2]=...=x_k[i_k]} c(S)$.

▶ **Definition 2.** *A pair-based similarity measure is a tuple-based similarity measure for tuples of size 2.*

The motivation for this definition is to provide a generalization of the alignment gadget of [5] that captures the property we need to achieve faster algorithms given correlated instances. In [5], they introduce the notion of an alignment gadget, and show that any problem with an

alignment gadget cannot be solved in strongly subquadratic time. Our definition is more general than theirs in that we do not restrict ourselves to looking at alignments; multiple positions in one string can match to the same position in another string. However, we only look at measures where the function is determined by the positions of the matching pairs; this is for reasons related to the correlation model that will become apparent.

We denote the ith character of a string x over an alphabet Σ by $x[i]$.

▶ **Definition 3.** *A longest common subsequence of two strings $x, y \in \Sigma^*$ ($LCS(x, y)$) of length m is a largest set of pairs $(a_1, b_1), ..., (a_m, b_m)$ such that $1 \leq a_1 < a_2 < ... < a_m \leq n$, $1 \leq b_1 < b_2 < ... < b_m \leq n$, and $x[a_i] = y[b_i]$ for $i \in \{1, ..., m\}$.*

▶ **Definition 4.** *A longest common subsequence of k strings ($LCS(x[1], x[2], ..., x[k])$) of strings $x[1], x[2], x[3], ..., x[k]$ is a largest set of tuples $(a_1[1], a_2[1], ..., a_k[1]), ..., (a_1[m], a_2[m], ..., a_k[m])$ with $1 \leq a_j[1] < a_j[2] < ... < a_j[m] \leq n$ for all $j \in \{1, ..., k\}$ and $x_1[a_1[i]] = x_2[a_2[i]] = ... = x_k[a_k[i]]$ for all $i \in \{1, ..., m\}$*

It is easy to see that LCS is a tuple-based similarity measure when we take $c(S) = |S|$; the allowed sets of tuples are all possible alignments, and the maximum over all sets of alignments of $c(S)$ is precisely the length of the longest common subsequence.

▶ **Definition 5.** *The edit distance ($EDIT$) between two strings x, y is the minimum number of insertions, deletions, and character substitutions it takes to get from x to y. $EDIT(x, y) = LEV_{x,y}(|x|, |y|)$, where*

$$LEV_{x,y}(i,j) = \begin{cases} \max i, j & \text{if } \min i, j = 0 \\ \min \begin{cases} LEV_{x,y}(i, j-1) \\ LEV_{x,y}(i-1, j) \\ LEV_{x,y}(i-1, j-1) + \mathbb{1}_{a_i \neq b_j} \end{cases} \end{cases}$$

$$\text{and } \mathbb{1}_{a_i \neq b_j} = \begin{cases} 1 & \text{if } a_i \neq b_j \\ 0 & \text{if } a_i = b_j \end{cases}.$$

▶ **Proposition 1.** *$EDIT$ is a pair-based similarity measure.*

Proof. We observe that if every alignment has a unique minimum sequence edit associated with it, then the proposition follows as the set of pairs corresponding to a location unchanged by the minimum sequence of edits in the first string and the location where it moved to in the second string is an alignment, and thus taking the minimum over all alignments of the edit distance that preserves the alignment gives us the minimum edit distance. Thus it suffices to show that this is the case. Suppose that $(a_1, b_1), ..., (a_k, b_k)$ is the set of pairs that are unchanged when the edit operation is applied. We claim that the minimum number of edits leaving these pairs unchanged is equal to $\sum_{i=0}^{k} \max(a_{i+1} - a_i - 1, b_{i+1} - b_i - 1)$ if this is true. To see this, the edit takes $x[a_i]$ to $y[b_i]$, and thus everything between $x[a_i]$ and $x[a_{i+1}]$ must be matched to things between $y[b_i]$ and $y[b_{i+1}]$. This edit distance is equal to $\max(a_{i+1} - a_i - 1, b_{i+1} - b_i - 1)$. Thus, the minimum number of edits leaving this set of pairs unchanged is $\sum \max(a_{i+1} - a_i - 1, b_{i+1} - b_i - 1)$. Therefore the minimum edit distance is the minimum of this function over all sets of matching pairs that satisfy the constraint that for any i, j, either $a_i < a_j$ and $b_i < b_j$ or $a_i > a_j$ and $b_i > b_j$. ◀

▶ **Definition 6.** *A traversal of two sequences x, y of length m, n respectively is a list of pairs $(a_1, b_1), ..., (a_t, b_t)$ such that $(a_1, b_1) = (1, 1)$, $(a_t, b_t) = (m, n)$, and (a_{i+1}, b_{i+1}) is either $(a_i + 1, b_i), (a_i, b_i + 1)$, or $(a_i + 1, b_i + 1)$.*

▶ **Definition 7.** *The dynamic time warping distance (DTWD) between two strings x, y is the minimum cost of a traversal of the two strings. The cost of a traversal is the sum of the distances between each pair of the traversal.*

▶ **Proposition 2.** *DTWD when the distances are 0 and 1 ($DTWD_{0,1}$) is a pair-based similarity measure.*

Proof. Let $(a_1, b_1), ..., (a_k, b_k)$ be the set of pairs with $x[a_i] = y[b_i]$ in order in the traversal with the minimum cost. Similarly to the edit distance situation, the traversal must match everything between $x[a_i]$ and $x[a_{i+1}]$ and $y[b_i]$ and $y[b_{i+1}]$, and the dynamic time warping distance here is $\sum_{i=1}^{k-1} \max b_{i+1} - b_i - 1, a_{i+1} - a_i - 1$ because none of these were included in matching pairs because of the definition of a traversal. Then, determining the minimum dynamic time warping distance is equivalent to determining the minimum of this quantity over all subsets of traversals. This means that $DTWD_{0,1}$ is a pair-based similarity measure. ◀

4 Correlation models for tuple-based similarity measures

In this section we will define the way primary and auxiliary instances are generated by our generating processes. The first generating process draws primary instances from the primary distribution, and auxiliary instances from the auxiliary distribution conditioned on the optimal set of tuples for the primary instances matching for each auxiliary instance. This does not mean that the values at the locations specified by each tuple are the same between the primary and auxiliary instances; it indicates that the contents of the strings in an auxiliary instance at the locations indicated by the tuple will be the same. The second generating process will be similar to the first generating process, but instead of making the locations always match, there will be a higher probability of matching locations being the same.

▶ **Definition 8.** *Suppose that f is a tuple-based similarity measure with allowed sets of tuples $\mathcal{S}$ and cost function c. Let l be the number of correlated instances, k be the number of strings that f is a measure on, and let $m_1, ..., m_k$ be the length of each string.*

The generating process $Gen_{Pri,Aux}(f, k, \vec{m}, l)$ draws $x_1, x_2, ..., x_k$ of length $m_1, m_2, ..., m_k$ respectively from Pri with the allowed set of tuples $S = \{(a_{11}, a_{21}, ..., a_{k1}), ..., (a_{1m}, a_{2m}, ..., a_{km})\} = \arg\min_{S \in \mathcal{S} | \forall (i_1, i_2, ..., i_k) \in S, x_1[i_1] = x_2[i_2] = ... = x_k[i_k]} c(S)$. Then, it outputs l tuples $(x_1^{(i)}, x_2^{(i)}, ..., x_k^{(i)})$ drawn from $Aux^{\otimes n}$ conditioned on $\forall (j_1, j_2, ..., j_k) \in S, x_1^{(i)}[j_1] = x_2^{(i)}[j_2] = ... = x_k^{(i)}[j_k]$.

▶ **Definition 9.** *Suppose that f is a tuple-based similarity measure with allowed sets of tuples $\mathcal{S}$ and cost function c. Let l be the number of correlated instances, k be the number of strings that f is a measure on, ϵ be the probability that a given tuple is matching for an auxiliary instance, and let $m_1, ..., m_k$ be the length of each string.*

The generating process $Gen_{Pri,Aux}(f, k, \vec{m}, l, \epsilon)$ draws $x_1, x_2, ..., x_k$ of length $m_1, m_2, ..., m_k$ from Pri with the allowed set of tuples $S = \{(a_{11}, a_{21}, ..., a_{k1}), ..., (a_{1m}, a_{2m}, ..., a_{km})\} = \arg\min_{S \in \mathcal{S} | \forall (i_1, i_2, ..., i_k) \in S, x_1[i_1] = x_2[i_2] = ... = x_k[i_k]} c(S)$. Then, it outputs l tuples $(x_1^{(i)}, x_2^{(i)}, ..., x_k^{(i)})$ drawn from $Aux^{\otimes n}$ and then we make $x_1^{(i)}[j_1] = x_2^{(i)}[j_2] = ... = x_k^{(i)}[j_k]$ with probability ϵ for all tuples $(j_1, j_2, ..., j_k) \in S$.

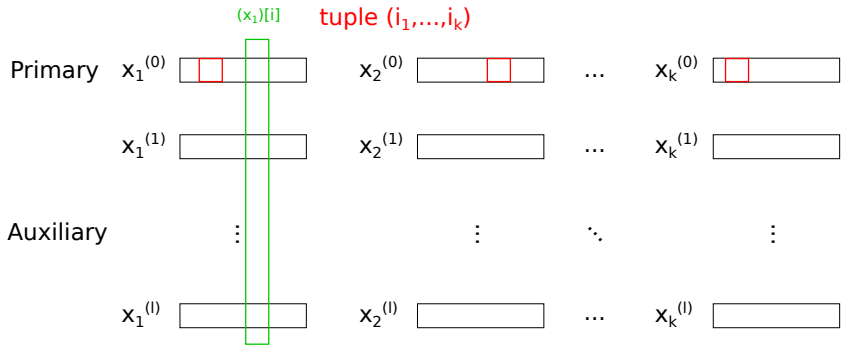

Figure 1 This shows the structure of the primary and auxiliary instances. The instance column is delineated in green, and a tuple in the primary instance is shown in red.

5 Results

5.1 Exact correlation model

What correlations should we expect to provide advantages? For tuple-based similarity measures, we should expect information about what the optimal set of tuples is to give us a speedup. In the model that we are considering, each location tuple in the optimal set of tuples of our primary instance will also be a set of locations with the same value in each auxiliary instance. For example, consider the two sequences 1010010 and 1101100. The longest common subsequence between the two sequences is 10110, so we can expect auxiliary instances to look like $abc^{**}de$ and $^{*}abcde^{*}$, where a, b, c, d, e have the same value and * is a wildcard.

This correlation model is not just solving a random instance of the problem. The goal is to find the optimal set of tuples for the primary instance. There are no guarantees on the distribution for the primary instance; the primary instance is worst-case. As a result, the problem is not equivalent to finding the optimal set of tuples for an auxiliary instance, and thus finding the correlations is not the same as solving a random instance of the original problem.

▶ **Definition 10.** *The instance column $(x)[i]$ is defined as follows. Suppose we have strings $(x_1, x_2, ..., x_k)$ which are primary instances, which we will refer to $(x_1^{(0)}, x_2^{(0)}, ..., x_k^{(0)})$, and auxiliary instances $(x_1^{(1)}, x_2^{(1)}, ..., x_k^{(1)})..., (x_1^{(l)}, x_2^{(l)}, ..., x_k^{(l)})$. Then, we denote $(x_j^{(0)}[i], x_j^{(1)}[i], ..., x_j^{(n)}[i]) = (x_j)[i]$ for any $j \in \{1, ..., k\}$.*

Given the correlation model, we know that two characters in each auxiliary instance will be equal with probability 1 if they belong to a tuple in the optimal set of tuples and with probability less than 1 otherwise. This suggests the following algorithm which runs in $O(kn \log n)$ time, or $O(n \log n)$ if k is constant. If we are given auxiliary instances from $Gen_{Pri,Aux}(f, k, \vec{m}, l)$ with Pri a worst-case distribution and Aux the uniform distribution, we can construct buckets indexed by elements of Σ_{l+1} and the entries of each bucket are the indices of the columns that have the same value as the bucket's index. Then, it looks up the columns of string 1 to see where the matches are. FIND proceeds in three steps; first, for the last $k-1$ strings, it puts each column of the string in the corresponding bucket in the set of buckets for that string. Next, it checks the columns of string 1 against the columns of string

2 to see which positions in string 1 are part of tuples. Then, we construct the set of tuples by looking up the positions in strings 2 through k that match to positions in string 1.

In FIND, the H_j are arrays of buckets where $((x_j)[i], i)$ is stored for $j = 2, ..., n$, and A is the array of k-tuples in the longest common subsequence. $H_j[s]$ denotes the contents of the bucket for the jth string, and is filled iff there exists i such that $(x_j)[i] = s$. $A_{i,j}$ is the index in the jth string of the ith entry of the longest common subsequence.

▶ **Theorem 1.** *Suppose we have a pair-based similarity measure or a k-tuple-based similarity measure with no overlapping tuples. If we are given a primary instance $x_1, x_2, ..., x_k$ and auxiliary instances $x_1^{(1)}, x_2^{(1)} ..., x_k^{(1)}, ..., x_1^{(l)}, ..., x_k^{(l)}$ with $l = O(\log n)$ drawn from $Gen_{Pri,Aux}(f, k, \vec{m}, l)$ with Pri worst-case and Aux uniform, FIND finds $f(x_1, x_2, ..., x_k)$ in time $\tilde{O}(kn \log n + |S|)$, where S is the largest set of tuples in $\mathcal{S}$ and $n = \max m_1, ..., m_k$.*

Proof. If A is the optimal set S^* from the primary instance, we can compute $f(x_1, x_2, ..., x_k) = c(A)$ in $\tilde{O}(|S^*|)$. Then, we will show that given the primary and auxiliary instances, the algorithm finds find the optimal set of tuples for the primary instance. Let $(i_1, ..., i_k) \in S^*$. By the definition of the correlation model, we will have that $(x_1)[i_1] = (x_2)[i_2] = ... = (x_k)[i_k]$ because the tuple is one of the optimal tuples in the primary instance which means that this is a matching tuple in both the primary and auxiliary instances. Suppose that we have a tuple $(i_1, i_2, ..., i_k)$ not in S^*. Then, the probability that $x_1^{(j)}[i_1] = x_2^{(j)}[i_2]$ for a given i is $1/2$, so then the probability of equality for all j is $\frac{1}{2^{O(\log n)}} < \frac{1}{3n^2}$. Taking a union bound over all pairs, we see that the probability of any tuple not in S^* having $(x_1)[i_1] = (x_2)[i_2]$ is at most $1/3$, so therefore with probability at least $2/3$, every tuple with $(x_1)[i_1] = (x_2)[i_2] = ... = (x_k)[i_k]$ will be be in S^*. Thus, with probability at least $2/3$, every hash collision corresponds to a tuple, and therefore A is the optimal set tuples for the primary instance. ◀

Algorithm 1: FIND$(f, x_1, x_2, ..., x_k, x_1^{(1)}, ..., x_k^{(1)}, ..., x_1^{(l)}, ..., x_k^{(l)})$

```
    Put columns into buckets
1   For j = 2, ..., k
2       Construct the array H_j with elements linked lists.
3       For i = 1, ...., n
4           Add ((x_j)[i], i) to H_j[(x_j)[i])].
5   Make the n × k array A.
6   m = 1
    Check which positions in string 1 are part of tuples in S*
7   For i = 1, ..., n
8       For (v, j) in H_2[(x_1)[i]]
9           If (x_1)[i] = v
10              A_{m,1} = i
11              A_{m,2} = j
12              m = m + 1
    Match positions in strings 2 through k to positions in string 1
13  For s = 3, ..., k
14      For l = 1, ..., m
15          i = A_{l,1}
16          If H_s[(x_1)[i]] is empty
17              Remove A_l and skip to the next l
18          For (v, j) in H_s[(x_1)[i]]
19              If (x_1)[i] = v
20                  A_{m,s} = j
21  Output c(A).
```

▶ **Corollary 1.** *If we are given a primary and $O(\log n)$, where $n = \max m_1, ..., m_k$, auxiliary instances of k-LCS drawn from $Gen_{Pri,Aux}(LCS, k, \vec{m}, O(k \log n))$ with Pri worst-case and Aux uniform, we can find the longest common subsequence of the primary instance in time $\tilde{O}(nk)$.*

▶ **Corollary 2.** *If we are given a primary and $O(\log \max m, n)$ auxiliary instance of EDIT drawn from $Gen_{Pri,Aux}(EDIT, 2, \{m, n\}, O(\log \max m, n))$, we can find the minimum edit distance between the primary sequences in time $\tilde{O}(m + n)$.*

▶ **Corollary 3.** *If we are given a primary and $O(\log \max m, n)$ auxiliary instances of $DTWD_{01}$ drawn from*
$Gen_{Pri,Aux}(DTWD_{0,1}, 2, \{m, n\}, O(\log \max m, n))$ with Pri worst-case and Aux uniform, we can find the dynamic time warping distance of the primary instance in time $\tilde{O}(m + n)$.

5.2 Relaxed correlation models

In the preceding section, we established that under a strict correlation model, we are able to find the solution to problems such as longest common subsequence and edit distance much faster than current lower bounds suggest if we did not have auxiliary instances. The main restriction is that the optimal set of tuples must also be a matching set of tuples for every auxiliary instance. When we talk about a matching tuple, we are referring to a tuple such that the characters at the locations specified by the tuple are the same. In this section, we will show how to relax this condition to give faster algorithms when a tuple in the optimal set of tuples only has a higher probability of being a matching tuple.

The main difficulty here is identifying the correct tuples. In the previous case, this was easy, because we could use the fact that every member of a correct tuple would be the same for each auxiliary instance. Then it sufficed to look at locations where the values matched up for each auxiliary instance, and we could use hashing to find these matching locations. In this case, we do not have this guarantee; we only have the guarantee that locations which are in the same tuple will have a higher probability of being the same in an auxiliary instance. This guarantee still allows us to find the elements of a tuple given its first element, as now the Hamming distance between two columns in the same tuple will be less than if they are not in the same tuple with high probability, and then we use faster algorithms for Hamming nearest neighbors such as the one in [2].

The following lemma will guarantee that the nearest neighbors of the column corresponding to the values of a position on the primary instance and auxiliary instances are the columns corresponding to the positions that are part of a tuple in the optimal set of tuples for the primary instance and has the first position in it also. As a result, this means that if we run a Hamming nearest neighbors algorithm on the columns, we will be able to find which tuples were the optimal tuples for the primary instance and thus compute the optimal value of the function for the primary instance.

▶ **Definition 11.** *$\mathbb{1}_{a=b}$ is 1 if $a = b$ and 0 otherwise.*

▶ **Lemma 1.** *Suppose we have x_1, x_2, ..., x_k and $x_1^{(1)}, x_2^{(1)}, ...x_k^{(1)}, ..., x_1^{(l)}, x_2^{(l)}, ..., x_k^{(l)}$ drawn from $Gen_{Pri,Aux}(f, k, \vec{m}, l\epsilon)$ with Pri worst-case and Aux uniform. Let $j \in \{2, ..., k\}, a \in \{1, ..., m_1\}$ and $b \in \{1, ..., m_j\}$. Then, for any δ such that $0 < \delta < 1/2 + \varepsilon$, if $a = i_1$ and $b = i_j$ for some $(i_1, ..., i_k) \in S^*$, the optimal set of tuples for the primary instance,*

$$\Pr[\mathbb{E}_m[\mathbb{1}_{x_1^{(m)}[a]=x_j^{(m)}[b]}] \leq 1/2 + \varepsilon - \delta] \leq e^{-\frac{(\delta/(1/2+\varepsilon))^2 l}{2}}$$

and otherwise

$$\Pr[\mathbb{E}_m[\mathbb{1}_{x_1^{(m)}[a]=x_j^{(m)}[b]}] \geq 1/2+\delta] \leq e^{-4\delta^2 l/3}$$

Proof. We use the form of the Chernoff bound given in [11]: for X a sum of independent random variables $X_1, ..., X_n$ taking values $\{0,1\}$ with expectation μ, we have that $\Pr[X \leq (1-\delta)\mu] \leq e^{-\delta^2\mu/2}$ and $\Pr[X \geq (1+\delta)\mu] \leq e^{-\delta^2\mu/3}$ for $0 < \delta < 1$. The first inequality is proven by plugging in $\mu = (1/2+\varepsilon) * l$ into the first Chernoff bound and noting that the expectation is the sum of the indicator random variables divided by l, and the second inequality follows similarly. ◀

Given this result, we can now use an algorithm to find Hamming nearest neighbors to find the optimal set of tuples. The best algorithm is the one of [2], which runs in time $n^{2-\Omega(\epsilon^{1/3}/\log(1/\epsilon))}$ to find the approximate nearest neighbors. What follows is the algorithm, treating finding the Hamming nearest neighbors as a black box.

FIND-RELAXED$(f, x_1, x_2, ..., x_k, x_1^{(1)}, ..., x_k^{(1)}, ..., x_1^{(l)}, ..., x_k^{(l)}, \epsilon)$

1 Make an $n \times k$ array A.
2 For $i = 1, ..., m_1$
3 For $j = 2, ..., k$ and $t = 1, ..., m_k$, find all vectors $(x_j)[t]$ that are less than $(1/2 - \epsilon/2) * l$ away from $(x_1)[i]$ in Hamming distance using the algorithm for approximate nearest neighbors
4 Add the tuple of all these vectors to A
5 Compute $c(A)$.

▶ **Theorem 2.** *Suppose we have a pair-based similarity measure or a k-tuple-based similarity measure with no overlapping tuples f. Let $l = 1000 \log n$. With probability at least $2/3 - o(1)$, the preceding algorithm computes $f(x_1, ..., x_k)$ given correlated instances from $Gen_{Pri,Aux}(f, k, \vec{m}, l, \epsilon)$ with Pri worst-case and Aux uniform in time $O(|S| + n^{2-\Omega(\epsilon^{1/3}/\log(1/\epsilon))})$, where S is the largest set of tuples in $\mathcal{S}$ and $n = \max m_1, ..., m_k$.*

Proof. Given $1000 \log n$ instances, we can use the lemma to say that for any two columns, with probability $1/n^3$ they do not correspond to a tuple and have Hamming distance less than $1/2 - \epsilon/4$ or they do correspond to a tuple and have Hamming distance more than $1/2 - 3\epsilon/4$. Taking a union bound gets us that the probability that this holds for any pair of columns is less than k/n. This means that the nearest neighbors of every column are the columns that it shares a tuple with, with probability $1 - o(1)$. Then in time $O(n^{2-\Omega(\epsilon^{1/3}/\log(1/\epsilon))})$ using the algorithm of [2] we can find the optimal set of tuples S^* with probability $2/3$. Then in time $|S|$ we can compute $c(S^*) = f(x_1, ..., x_k)$. ◀

▶ **Corollary 4.** *If we are given a primary and $O(\log n)$, where $n = \max m_1, ..., m_k$, auxiliary instances of k-LCS drawn from $Gen_{Pri,Aux}(LCS, k, \vec{m}, O(k \log n), \epsilon)$ with Pri worst-case and Aux uniform, we can find the longest common subsequence of the primary instance in time $O(n^{2-\Omega(\epsilon^{1/3}/\log(1/\epsilon))})$.*

▶ **Corollary 5.** *If we are given a primary and $O(\log \max m, n)$ auxiliary instance of EDIT drawn from $Gen_{Pri,Aux}(EDIT, \{m, n\}, 2, O(\log \max m, n), \epsilon)$, we can find the minimum edit distance between the primary sequences in time $O(n^{2-\Omega(\epsilon^{1/3}/\log(1/\epsilon))})$.*

▶ **Corollary 6.** *If we are given a primary and $O(\log \max m, n)$ auxiliary instances of $DTWD_{01}$ drawn from $Gen_{Pri,Aux}(DTWD_{01}, \{m, n\}, 2, O(\log \max m, n), \epsilon)$ with Pri worst-case and Aux uniform, we can find the dynamic time warping distance of the primary instance in time $O(n^{2-\Omega(\epsilon^{1/3}/\log(1/\epsilon))})$.*

6 Another correlation model for edit distance

▶ **Definition 12.** *The generating process* $Genedit_{Pri,Aux,\epsilon}(n,l)$ *draws* x, y *of length* n *and a sequence of character insertions, deletions, and substitutions* $\pi_1, \pi_2, ..., \pi_k$ *that is the minimum sequence of edits needed to transform* x *to* y *from* Pri*. The output is* $x, y, x^{(1)}, y^{(1)}, x^{(2)}, y^{(2)}, ..., x^{(l)}, y^{(l)}$*, with* $x^{(i)}$ *being* x *with each character changed with probability* ϵ *and* $y^{(i)} = \pi_k(\pi_{k-1}(...(\pi_1(x^{(i)}))...))$.

Suppose that we have two strings x, y for which we want to compute the minimum edit distance. If our auxiliary instances have the same sequence of edit operations, then with $3 \log n$ auxiliary instances the probability that $(x)_i = (y)_j$ and x_i did not to y_j during the sequence of inserts, deletes, and changes is less than $1/n^3$. This means that we can find the parts of the original string that are preserved under the edit and their new positions. Our algorithm works as follows. First, we get $O(\log n)$ auxiliary instances from $Genedit_{Pri,Aux,\epsilon}$, where Pri is worst-case, and then put the $(y)_j$ in buckets and check which $(x)_i$ have $(x)_i = (y)_j$. Then, with the pairs $(a_1, b_1), ..., (a_k, b_k)$, the edit distance is $\sum_{i=0}^{k} \max a_{i+1} - a_i - 1, b_{i+1} - b_i - 1$ with high probability. This algorithm obviously runs in $O(n \log n)$ time.

In FIND-EDIT, the array H of buckets is used to store $(y)_i$, and $COMMONPAIRS$ holds the i, j such that $(x)_i = (y)_j$.

Algorithm 2: FIND-EDIT$(x, y, x^{(1)}, y^{(1)}, ..., x^{(l)}, y^{(l)})$

```
1  Initialize an n × 2 array COMMONPAIRS.
2  Construct an array H of size n of linked lists.
3  For j = 1, ..., n
4        Add ((y)_j, j) to H[(y)_j].
5  For i = 1, ..., n
6        For (v, j) in H[(x)_i]
7              If (x)_i = v add (i, j) to COMMONPAIRS.
8  Output Σ_{i=0}^{k} max (a_{i+1} − a_i − 1, b_{i+1} − b_i − 1),
   where ((a_1, b_1), (a_2, b_2), ..., (a_k, b_k)) = COMMONPAIRS.
```

▶ **Claim 1.** *The algorithm* FIND-EDIT *on* $O(\log n)$ *instances generated by* $Genedit_{Pri,Aux,\epsilon}(n,l)$ *with* Pri *worst-case and* Aux *uniform works and runs in time* $O(n \log n)$ *with probability at least 7/8.* [2]

Proof. Suppose that x_i did not move to y_j by the original sequence of edits. Then, the probability that $x_i^{(k)} = y_j^{(k)}$ is $1/2$ because they were drawn independently uniformly at random. Suppose we have $2 \log n + 3$ auxiliary instances. This means that the probability that $(x)_i = (y)_j$ is at most $1/8n^2$ because each auxiliary instance is independent. Taking a union bound means that the probability that there is a pair x_i, y_j where x_i does not correspond to y_j and $(x)_i = (y)_j$ is at most '/8. Thus, the hashing finds the pairs $(a_1, b_1), ..., (a_k, b_k)$ with probability at least 7/8 in expected time $O(n \log n)$. We claim that the edit distance is equal to $\sum_{i=0}^{k} \max (a_{i+1} - a_i - 1, b_{i+1} - b_i - 1)$ if this is true. To see this, the original edit takes x_{a_i} to y_{b_i}, and thus everything between x_{a_i} and $x_{a_{i+1}}$ must be matched to things between y_{b_i} and $y_{b_{i+1}}$. This edit distance is equal to $\max (a_{i+1} - a_i - 1, b_{i+1} - b_i - 1)$. Suppose that the edit distance was less. Then there must be a coordinate x_c or y_d that was in the original string.

[2] Recall that $Genedit_{Pri,Aux,\epsilon}$ outputs auxiliary instances where the first string is a perturbation of the original string and the second string is obtained by applying the same edits to the first string.

But then, we could extend this to a smaller edit for the entire string than the original edit, which is a contradiction. Thus, the edit distance is at least $\max(a_{i+1} - a_i - 1, b_{i+1} - b_i - 1)$, and this is achieved by the original edit, because none of them are preserved. The running time is $O(n \log n)$ because each loop takes time $O(n \log n)$. ◀

Acknowledgements. We are grateful to Guy Rothblum for early important discussions on this work and the choices of correlation models. Thank you Guy! We also thank Aviv Regev for helpful discussion on correlations in sequence alignment in bioinformatics.

References

1 Amir Abboud, Arturs Backurs, and V. Vassilevska Williams. Tight hardness results for lcs and other sequence similarity measures. In *FOCS 2015*, 2015.

2 Josh Alman, Timothy M Chan, and Ryan Williams. Polynomial representations of threshold functions and algorithmic applications. *arXiv preprint arXiv:1608.04355*, 2016.

3 Alexandr Andoni and Robert Krauthgamer. The smoothed complexity of edit distance. *ACM Transactions on Algorithms (TALG)*, 8(4):44, 2012.

4 Piotr Indyk Arturs Backurs. Edit distance cannot be computed in strongly subquadratic time (unless seth is false). In *STOC15*, 2015.

5 Karl Bringmann and Marvin Künnemann. Quadratic conditional lower bounds for string problems and dynamic time warping. In *Foundations of Computer Science (FOCS), 2015 IEEE 56th Annual Symposium on*, pages 79–97. IEEE, 2015.

6 Irit Dinur, Shafi Goldwasser, and Huijia Lin. The computational benefit of correlated instances. In *Proceedings of the 2015 Conference on Innovations in Theoretical Computer Science, ITCS 2015, Rehovot, Israel, January 11-13, 2015*, pages 219–228, 2015. doi:10.1145/2688073.2688082.

7 Anka Gajentaan and Mark H Overmars. On a class of o(n^2) problems in computational geometry. *Computational geometry*, 5(3):165–185, 1995.

8 Nadia Heninger, Zakir Durumeric, Eric Wustrow, and J Alex Halderman. Mining your ps and qs: Detection of widespread weak keys in network devices. In *Presented as part of the 21st USENIX Security Symposium (USENIX Security 12)*, pages 205–220, 2012.

9 James W Hunt and Thomas G Szymanski. A fast algorithm for computing longest common subsequences. *Communications of the ACM*, 20(5):350–353, 1977.

10 Russell Impagliazzo and Ramamohan Paturi. Complexity of k-sat. In *Computational Complexity, 1999. Proceedings. Fourteenth Annual IEEE Conference on*, pages 237–240. IEEE, 1999.

11 Michael Mitzenmacher and Eli Upfal. *Probability and computing: Randomized algorithms and probabilistic analysis*. Cambridge University Press, 2005.

12 Daniel A Spielman and Shang-Hua Teng. Smoothed analysis. In *Algorithms and data structures*, pages 256–270. Springer, 2003.