Below P vs NP: Fine-Grained Hardness for Big Data Problems

by

Arturs Backurs

B.S., University of Latvia (2012)
S.M., Massachusetts Institute of Technology (2014)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2018

Author .....
Department of Electrical Engineering and Computer Science
August 30, 2018

Certified by .....
Piotr Indyk
Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by .....
Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Department Committee on Graduate Students

Below P vs NP: Fine-Grained Hardness for Big Data Problems

by

Arturs Backurs

Submitted to the Department of Electrical Engineering and Computer Science
on August 30, 2018, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

The theory of NP-hardness has been remarkably successful in identifying problems that are unlikely to be solvable in polynomial time. However, many other important problems do have polynomial-time algorithms, but large exponents in their runtime bounds can make them inefficient in practice. For example, quadratic-time algorithms, although practical on moderately sized inputs, can become inefficient on big data problems that involve gigabytes or more of data. Although for many data analysis problems no sub-quadratic time algorithms are known, any evidence of quadratic-time hardness has remained elusive.

In this thesis we present hardness results for several text analysis and machine learning tasks:

- Lower bounds for edit distance, regular expression matching and other pattern matching and string processing problems.
- Lower bounds for empirical risk minimization such as kernel support vectors machines and other kernel machine learning problems.

All of these problems have polynomial time algorithms, but despite extensive amount of research, no near-linear time algorithms have been found. We show that, under a natural complexity-theoretic conjecture, such algorithms do not exist. We also show how these lower bounds have inspired the development of efficient algorithms for some variants of these problems.

Thesis Supervisor: Piotr Indyk
Title: Professor of Electrical Engineering and Computer Science

Acknowledgments

First and foremost, I would like to thank my advisor Piotr Indyk for the guidance and support that he provided during my graduate studies.

I thank Costis Daskalakis and Virginia Vassilevska Williams for serving on my thesis committee.

I would also like to say thank you to all people from Piotr's group, in particular, Sepideh Mahabadi, Ilya Razenshteyn, Ludwig Schmidt, Ali Vakilian and Tal Wagner for all that I learned from them.

I am grateful to Andris Ambainis who introduced me to theoretical computer science. I thank Krzysztof Onak and Baruch Schieber for hosting me at IBM during my summer internship in 2015.

Finally, I thank my past and current collaborators: Amir Abboud, Kyriakos Axiotis, Mohammad Bavarian, Karl Bringmann, Moses Charikar, Nishanth Dikkala, Thomas Dueholm Hansen, Piotr Indyk, Marvin Künnemann, Cameron Musco, Krzysztof Onak, Mădălina Persu, Eric Price, Liam Roditty, Baruch Schieber, Ludwig Schmidt, Gilad Segal, Anastasios Sidiropoulos, Paris Siminelakis, Christos Tzamos, Ali Vakilian, Tal Wagner, Nicole Wein, Ryan Williams, Virginia Vassilevska Williams, David P. Woodruff, Or Zamir.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Classically, an algorithm is called "efficient" if its runtime is bounded by a polynomial function of its input size. Many well-known computational problems are amenable to such algorithms. For others, such as satisfiability of logical formulas (SAT), no polynomial time algorithms are not known. Although we do not know how to prove that no such algorithm exists, its non-existence can be explained by the fact that SAT is *NP-hard*. That is, a polynomial-time algorithm for SAT would imply that the widely believed $\mathsf{P} \neq \mathsf{NP}$ hypothesis is false. Thus, no polynomial-time algorithm for SAT and other NP-hard problems exists, as long as $\mathsf{P} \neq \mathsf{NP}$.

As the data becomes large, however, even polynomial time algorithms often cannot be considered "efficient". For example, a quadratic-time algorithm can easily take hundreds of CPU-years on inputs of gigabyte size. For even larger (say terabyte-size) inputs, the running time of an "efficient" algorithm must be effectively linear. For many problems such algorithms exist; for others, despite decades of effort, no such algorithms has been discovered so far. Unlike for SAT, however, an explanation for the lack of efficient solutions is hard to come by, even assuming $\mathsf{P} \neq \mathsf{NP}$. This is because the class P puts all problems solvable in polynomial time into *one* equivalence class, not making any distinctions between different exponents in their runtime bounds. Thus, in order to distinguish between, say, linear, quadratic or cubic running times, one needs stronger hardness assumptions, as well as efficient reduction techniques that transfer hardness from the assumptions to the problems. Over the last few years, these research directions has been a focus of an emerging field of *fine-grained complexity*.

The goal of this thesis is to investigate the fine-grained complexity of several fundamental computational problems, by developing conditional lower bounds under strong yet plausible complexity-theoretic assumptions. On a high level, the implications of our results are two-fold. On the one hand, a lower bound that matches a known upper bound gives an evidence that a faster algorithm is not possible or at least demonstrates a barrier for a further improvement. On the other hand, the lack of conditional hardness for a problem suggests that a better algorithm might be possible. Indeed, in this thesis we present several new *algorithms* inspired by this approach.

1.1 Main contributions: an overview

Here we will give an overview of the main contributions. We describe the results in more detail in the subsequent sections.

Sequence similarity and pattern matching Measuring (dis)-similarity between sequences and searching for patterns in a large data corpus are basic building blocks for many data analysis problems, including computational biology and natural language processing. In this thesis we study the fine-grained complexity of these problems. First, we show a conditional (nearly) quadratic-time lower bound for computing the *edit distance* between two sequences. The edit distance (a.k.a. the Levenshtein distance) between two sequences is equal to the minimum number of symbol insertions, deletions and substitutions needed to transform the first sequence into the second. This is a classical computational task with a well-known quadratic-time algorithm based on dynamic programming. Our hardness result is the first complexity-theoretic evidence that this runtime is essentially optimal.

The techniques developed for this problem allows us to show computational hardness for several other similarity measures, such as the *longest common subsequence* and *dynamic time warping*. We also give evidence that some of those computational tasks are hard to solve even *approximately*, at least for deterministic algorithms. Finally, we also study pattern matching with regular expressions. In turn, this study has led to faster algorithms for the *word break* problem, a popular interview question.

This part of the thesis is based on the following papers:

- Arturs Backurs, Piotr Indyk, *Edit Distance Cannot Be Computed in Strongly Subquadratic Time (unless SETH is false)*, appeared at STOC 2015 and HALG 2016 [BI15]. Accepted to the special issue of the SIAM Journal of Computing.

- Arturs Backurs, Amir Abboud, Virginia Vassilevska Williams, *Tight Hardness Results for LCS and Other Sequence Similarity Measures*, appeared at FOCS 2015 [ABVW15].

- Arturs Backurs, Amir Abboud, *Towards Hardness of Approximation for Polynomial Time Problems*, appeared at ITCS 2017 [AB17].

- Arturs Backurs, Piotr Indyk, *Which Regular Expression Patterns are Hard to Match?*, appeared at FOCS 2016 [BI16].

Statistical data analysis and machine learning In the second part of this thesis we study the complexity of a number of commonly used machine learning problems and subroutines, all of which are solvable in polynomial time. These include a collection of *kernel* problems, such as *support vector machines* (SVMs) and *density estimation*, as well as (batch) *gradient evaluation* in neural networks. We show that, in the typical case when the underlying kernel is Gaussian or exponentially decaying, all of those problems require essentially quadratic time to solve up to high accuracy. In contrast,

we give a nearly-linear-time algorithm for density estimation for kernel functions with polynomial decay.

This part of the thesis is based on the following two papers:

- Arturs Backurs, Piotr Indyk, Ludwig Schmidt, *On the Fine-Grained Complexity of Empirical Risk Minimization: Kernel Methods and Neural Networks*, appeared at NIPS 2017 [BIS17].

- Arturs Backurs, Moses Charikar, Piotr Indyk, Paris Siminelakis, *Efficient Density Evaluation for Smooth Kernels*, to appear at FOCS 2018 [BCIS18].

1.2 Hardness assumption

Given a problem that is solvable in quadratic $O(n^2)$ time, how can we argue that it cannot be solved in a strongly sub-quadratic $O(n^{2-\varepsilon})$ time, for a constant $\varepsilon > 0$? As we discussed above, the $\mathsf{P} \neq \mathsf{NP}$ conjecture seems to be insufficient for this purpose, as it does not make a distinction between problems that are solvable in linear time and those that require quadratic time. Thus, we need a stronger hardness assumption to achieve such separations.

One such commonly used hardness assumption is known as the *Strong Exponential Time Hypothesis* (SETH) [IPZ01, IP01]. SETH is a statement about the complexity of the k-SAT problem: decide whether a given conjunctive normal form formula on N variables and M clauses, where each clause has at most k literals, is satisfiable. It postulates that k-SAT cannot be solved in time $O\left(2^{(1-\varepsilon)N}\right)$ where $\varepsilon > 0$ is a constant independent of k. In contrast, despite decades of research, the best known algorithm for k-SAT runs in time $2^{N-N/O(k)}$ (e.g., [PPSZ05]), so the constant in the exponent approaches 1 for large k. Thus, SETH is a strengthening of the $\mathsf{P} \neq \mathsf{NP}$ hypothesis, as the latter merely states that the satisfiability problem cannot be solved in *polynomial* time. Over the last few years, the hypothesis has served as the basis for proving conditional lower bounds for several important computational problems, including the diameter in sparse graphs [RVW13, CLR$^+$14], local alignment [AVWW14], dynamic graph problems [AVW14], Fréchet distance [Bri14], and the approximate nearest neighbor search [Rub18].

Many of the aforementioned are in fact obtained via a reduction from an intermediary problem, known as the *Orthogonal Vectors Problem* (OVP), whose hardness is implied by SETH. It is defined as follows. Given two sets $A, B \subseteq \{0,1\}^d$ such that $|A| = |B| = N$, the goal is to determine whether there exists a pair of vectors $a \in A$ and $b \in B$ such that the dot product $a \cdot b = \sum_{i=1}^{d} a_i b_i$ (taken over reals) is equal to 0, that is, the vectors are orthogonal. An alternative formulation of this problem is: given two collections of N sets each, determine if there is a set in the first collection that does not intersect a set from the second collection.[1] A strongly subquadratic-time algorithm for OVP, i.e., with a running time of $O(d^{O(1)}N^{2-\delta})$, would imply that SETH

[1]Equivalently, after complementing sets from the second collection, determine if there is a set in the first collection that is contained in a set from the second collection.

is false, even in the setting $d = \omega(\log N)$. In this thesis we adopt this approach and obtain our conditional hardness results via reductions from OVP.

1.3 Pattern matching and text analysis

1.3.1 Hardness for sequence alignment problems

The edit distance measures the similarity of two input sequences and is defined as the minimum number of insertions, deletions or substitutions of symbols needed to transform one sequence into another. The metric and its generalizations are widely used in computational biology, text processing, information theory and other fields. In particular, in computational biology it can be used to identify regions of similarity between DNA sequences that may be due to functional, structural, or evolutionary relationships [Gus97]. The problem of computing the edit distance between two sequences is a classical computational task, with a well-known algorithm based on the dynamic programming. Unfortunately, that algorithm runs in quadratic time, which is prohibitive for long sequences.[2] A considerable effort has been invested into designing faster algorithms, either by assuming that the edit distance is bounded, by considering the average case or by resorting to approximation.[3] However, the fastest known exact algorithm, due to [MP80, BFC08, Gra16], has a running time of $O(n^2(\log\log n)/\log^2 n)$ for sequences of length n, which is still nearly quadratic.

In this thesis we provide evidence that the (near)-quadratic running time bounds known for this problem might, in fact, be tight. Specifically, we show that if the edit distance can be computed in time $O(n^{2-\varepsilon})$ for some constant $\varepsilon > 0$, then the Orthogonal Vectors Problem can be solved in $O(d^{O(1)}N^{2-\delta})$ time for a constant $\delta > 0$. The latter result would violate the Strong Exponential Time Hypothesis, as described in the previous section.

How do we reduce the Orthogonal Vectors Problem to the edit distance computation? The key component in the reduction is the construction of a function that maps binary vectors into sequences in a way that two sequences are "close" if and only if the vectors are orthogonal. The first step of our reduction mimics the approach in [Bri14]. We assign a "gadget" sequence for each $a \in A$ and $b \in B$. Then, the gadget sequences $G(a)$ for all $a \in A$ are concatenated together to form the first input sequence, and the gadget sequences $G'(b)$ for all $b \in B$ are concatenated to form the second input sequence. The correctness of the reduction is proven by showing that:

- If there is a pair of orthogonal vectors $a \in A$ and $b \in B$, then one can align the two sequences in a way that the gadgets assigned to a and b are aligned, which implies that the distance induced by the global alignment is "small".

[2] For example, the analysis given in [Fri08] estimates that aligning human and mouse genomes using this approach would take about 95 years.

[3] There is a rather large body of work devoted to edit distance algorithms and we will not attempt to list all relevant works here. Instead, we refer the reader to the survey [Nav01] for a detailed overview of known exact and probabilistic algorithms, and to papers [AKO10, CDG+18] for an overview of approximation algorithms.

- If there is no orthogonal pair, then no such global alignment exists, which implies that the distance induced by any global alignment is "large".

For a fixed ordering $a^1, \ldots, a^N$ of vectors in A and ordering $b^1, \ldots, b^N$ of vectors in B, the edit distance between the final two sequences is (roughly) captured by the formula:

$$\min_{i=1}^{N} \sum_{j=1}^{N} \text{edit}(G(a^j), G'(b^{j+i})),$$

where $b^t = b^{t-N}$ for $t > N$. The edit distance finds an alignment between the sequences of gadgets such that the sum of the distances is minimized. The cost of the optimal alignment is a sum of N distances and we need that it is small if and only if there is a pair of orthogonal vectors.

What requirements does this impose on the construction of the gadgets? First of all, the distance between gadgets should be small if and only if the vectors are orthogonal. Furthermore, when vectors are *not* orthogonal, we need that the distance is *equal* to some quantity C that does not depend on the vectors. Otherwise, if C grew with the number of overlapping 1s between the vectors, the contribution from highly non-orthogonal pairs of vectors could overwhelm a small contribution from a pair of orthogonal vectors. Therefore, we need that there exists a value C such that if two vectors a and b are not orthogonal, the distance between their gadgets is *equal* to C and otherwise it is less than C. Because of this condition, our gadget design and analysis become more involved.

Fortunately, the edit distance is expressive enough to support this functionality. The basic idea behind the gadget construction is to use the fact that the edit distance between two gadget sequences, say G (from the first sequence) and G' (from the second sequence), is the minimum cost over all possible alignments between G and G'. Thus, we construct gadgets that allow two alignment options. The first option results in a cost that is linear in the number of overlapping 1s of the corresponding vectors (this is easily achieved by using substitutions only). On the other hand, the second "fallback" option has a fixed cost (say C_1) that is slightly higher than the cost of the first option when no 1s are overlapping (say, C_0). Thus, by taking the minimum of these two options, the resulting cost is equal to C_0 when the vectors are orthogonal and equal to C_1 ($> C_0$) otherwise, which is what is needed.

Longest common subsequence and dynamic time warping distance We refine the ideas from the above construction to show quadratic hardness for other popular measures, including the longest common subsequence (LCS) and the dynamic time warping (DTW) distance. For the LCS problem we construct gadgets with similar properties as for the edit distance. For the DTW distance we obtain a conditional lower bound by showing how to embed the constructed hard sequences for the edit distance problem into DTW instances.

Approximation hardness for sequence alignment problems The hardness results described above are for algorithms that solve the problems *exactly*. If one

allows approximation, often it is possible to obtain significantly faster algorithms. For example, one can compute a poly-logarithmic approximation to the edit distance in near-linear time [AKO10] and constant approximation in strongly sub-quadratic time [CDG+18]. However, no limitations for the approximate problem are known. In this thesis we make the first step to close this gap by showing quadratic hardness for approximately solving the edit distance and the LCS problems for *deterministic* algorithms, assuming a plausible hypothesis. Furthermore, we show that disproving the hypothesis would imply new circuit lower bounds currently not known to be true.

1.3.2 Regular expression pattern matching and membership

Regular expressions constitute a fundamental notion in formal language theory and are frequently used in computer science to define search patterns. In particular, regular expression pattern matching and membership testing are widely used computational primitives, employed in many programming languages and text processing utilities such as Perl, Python, JavaScript, Ruby, AWK, Tcl and Google RE2. Apart from text processing and programming languages, regular expressions are used in computer networks [KDY+06], databases and data mining [GRS99].

The two key computational problems that involve regular expressions are *pattern matching* and *membership testing*. In pattern matching the goal is to determine whether a given sequence contains a substring that matches a given pattern; in membership testing the goal is to decide if the entire sequence matches the given regular expression. A classic algorithm for these problems constructs and simulates a nondeterministic finite automaton corresponding to the expression, resulting in an $O(mn)$ running time (where m is the length of the pattern and n is the length of the text). This running time can be improved slightly (by a poly-logarithmic factor), but no significantly faster solutions are known. At the same time, much faster algorithms exist for various special cases of regular expressions, including dictionary matching [AC75], wildcard matching [FP74, Ind98, Kal02, CH02], subset matching [CH97, CH02], word break problem [folklore] etc. This raises a natural question: Is it possible to characterize easy pattern matching and membership tasks and separate them from those that are hard?

In this thesis we answer this question affirmatively and show that the complexity of regular expression pattern matching can be characterized based on its *depth* (when interpreted as a formula). Our results hold for expressions involving concatenation, OR, Kleene star and Kleene plus. For regular expressions of depth two (involving any combination of the above operators), we exhibit the following dichotomy: pattern matching and membership testing can be solved in near-linear time, except for "concatenations of stars", which cannot be solved in strongly sub-quadratic time assuming the Strong Exponential Time Hypothesis (SETH). For regular expressions of depth three the picture is more complex. Nevertheless, we prove that all problems can either be solved in strongly sub-quadratic time, or cannot be solved in strongly sub-quadratic time assuming SETH.

An intriguing special case of membership testing involves regular expressions of the form "a star of an OR of concatenations", e.g., $[a|ab|bc]^*$. This corresponds to the

so-called *word break* problem (a popular interview question [Tun11, Lee]), for which a dynamic programming algorithm with a runtime of (roughly) $O(n\sqrt{m})$ is known. We show that the latter bound is not tight and improve the runtime to $O(nm^{0.44\ldots})$. This runtime has been further improved in a follow-up work [BGL17].

1.4 Statistical data analysis and machine learning

1.4.1 Kernel methods and neural networks

Empirical risk minimization (ERM) has been highly influential in modern machine learning [Vap98]. ERM underpins many core results in statistical learning theory and is one of the main computational problems in the field. Several important methods such as support vector machines (SVM), boosting, and neural networks follow the ERM paradigm [SSBD14]. As a consequence, the algorithmic aspects of ERM have received a vast amount of attention over the past decades. This naturally motivates the following basic question: What are the computational limits for ERM algorithms?

In this thesis, we address this question both in convex and non-convex settings. Convex ERM problems have been highly successful in a wide range of applications, giving rise to popular methods such as SVMs and logistic regression. Using tools from convex optimization, the resulting problems can be solved in polynomial time. However, the exact time complexity of many important ERM problems such as kernel SVMs is not yet well understood. In this thesis we show that, assuming SETH, the kernel SVM, kernel ridge regression and kernel principal component analysis as well as training the last layer of a neural net require essentially quadratic time for a sufficiently small approximation factor. All of these methods are popular learning algorithms due to the expressiveness of the kernel or network embedding. Our results show that this expressiveness also leads to an expensive computational problem.

Non-convex ERM problems have also attracted extensive research interest, e.g., in the context of deep neural networks. First order methods that follow the gradient of the empirical loss are not guaranteed to find the global minimizer in this setting. Nevertheless, variants of gradient descent are by far the most common method for training large neural networks. Here, the computational bottleneck is to compute a number of gradients, not necessarily to minimize the empirical loss globally. Although we can compute gradients in polynomial time, the large number of parameters and examples in modern deep learning still makes this a considerable computational challenge. We prove a matching conditional lower bound for (batch) gradient evaluation in neural nets, assuming SETH. In particular, we show that computing (or even approximating, up to polynomially large factors) the norm of the gradient of the top layer in a neural network takes time that is "rectangular". The time complexity cannot be significantly better than $O(n \cdot m)$, where m is the number of examples and n is the number of units in the network. Hence, there are no algorithms that compute batch gradients faster than handling each example individually, unless SETH fails.

Our results hold for a significant range of the accuracy parameter. For kernel methods, our bounds hold for algorithms approximating the empirical risk up to a

factor of $1+\varepsilon$, for $\log(1/\varepsilon) = \omega(\log^2 n)$. Thus, they provide conditional quadratic lower bounds for algorithms with, say, a $\log 1/\varepsilon$ runtime dependence on the approximation error ε. A (doubly) logarithmic dependence on $1/\varepsilon$ is generally seen as the ideal rate of convergence in optimization, and algorithms with this property have been studied extensively in the machine learning community (cf. [BS16]). At the same time, approximate solutions to ERM problems can be sufficient for good generalization in learning tasks. Indeed, stochastic gradient descent (SGD) is often advocated as an efficient learning algorithm despite its polynomial dependence on $1/\varepsilon$ in the optimization error [SSSS07, BB07]. Our results support this viewpoint since SGD sidesteps the quadratic time complexity of our lower bounds.

For other problems, our assumptions about the accuracy parameter are less stringent. In particular, for training the top layer of the neural network, we only need to assume that $\varepsilon \approx 1/n$. Finally, our lower bounds for approximating the norm of the gradient in neural networks hold even if $\varepsilon = n^{O(1)}$, i.e., for *polynomial* approximation factors (or alternatively, a constant additive factor for ReLU and sigmoid activation functions).

1.4.2 Efficient density evaluation for smooth kernels

The work on the kernel problems leads us to the study of the kernel density evaluation problem. Given a kernel function $k(\cdot,\cdot)$ and point-sets $P, Q \subset \mathbb{R}^d$, the kernel density evaluation problem asks to approximate $\sum_{p\in P} k(q,p)$ for every point $q \in Q$. This task has numerous applications in scientific computing [GS91], statistics [RW10], computer vision [GSM03], machine learning [SSPG16] and other fields. Assuming SETH we show that the problem requires essentially $|P| \cdot |Q|$ time for Gaussian kernel $k(q,p) = \exp(-\|q-p\|^2)$ and any constant approximation factor. This hardness result crucially relies on the property that the Gaussian kernel function decays very fast. We complement the lower bound with a better algorithm for "polynomially-decaying" kernel functions. For example, for the Cauchy kernel $k(q,p) = \frac{1}{1+\|q-p\|^2}$ we achieve roughly $|P| + |Q|/\varepsilon^2$ runtime for $1 \pm \varepsilon$ approximation.

The main idea behind the faster algorithm is to combine a randomized dimensionality reduction with a hierarchical partitioning of the space, via multi-dimensional quadtrees. This allows us to construct a randomized data structure such that, given a query point q, we can partition P into a short sequence of sets such that the variance of the kernel with respect to q is small in each of the sets. The overall estimator then can be obtained by computing and aggregating the estimators for individual layers. A convenient property of the overall estimator is that it is *unbiased*, which makes easy to extend the algorithm to other metrics via low-distortion embeddings. This property is due to the fact that we use the dimensionality reduction only to partition the points, while the kernel values are always evaluated in the original space. This means that the distortion induced by the dimensionality reduction only affects the variance, not the mean of the estimator.

Chapter 2

Preliminaries

Our hardness assumption is about the complexity of the SAT problem, which is defined as follows: given a conjunctive normal form formula, decide if the formula is satisfiable. The k-SAT problem is a special case when the input formula has at most k literals in each of the clauses. Throughout the paper we will use the following conjecture.

Conjecture 2.1 (Strong exponential time hypothesis (SETH) [IPZ01, IP01])**.** *For every constant $\varepsilon > 0$ there exists a large enough constant k such that k-SAT problem on N variables cannot be solved in time $O\left(2^{(1-\varepsilon)N}\right)$.*

To show conditional lower bounds that are based on SETH, it is often convenient to perform the reductions from the following intermediary problem.

Definition 2.2 (Orthogonal vectors problem)**.** *Given two sets $A, B \subseteq \{0,1\}^d$ such that $|A| = |B| = N$, determine whether there exists a pair of vectors $a \in A$ and $b \in B$ such that the dot product $a \cdot b = \sum_{i=1}^{d} a_i b_i$ (taken over reals) is equal to 0, that is, the vectors are orthogonal.*

The orthogonal vectors problem has an easy $O(N^2 d)$-time solution. The currently best known algorithm for this problem runs in time $n^{2-1/O(\log c)}$, where $c = d/\log N$ [AWY15, CW16]. The following theorem enables us to use the orthogonal vectors problem as a starting point for our reductions.

Theorem 2.3 ([Wil05])**.** *The orthogonal vectors problem cannot be solved in $O(d^{O(1)} N^{2-\delta})$ time, unless SETH is false. The statement holds for any $d = \omega(\log N)$.*

The orthogonal vectors problem is a special case of the following more general problem.

Definition 2.4 (Almost orthogonal vectors problem)**.** *Given two sets $A, B \subseteq \{0,1\}^d$ such that $|A| = |B| = N$ and an integer $r \in \{0, \ldots, d\}$, determine whether there exists a pair of vectors $a \in A$ and $b \in B$ such that the dot product $a \cdot b = \sum_{i=1}^{d} a_i b_i$ (taken over reals) is at most r. We call any two vectors that satisfy this condition as r-orthogonal.*

For $r = 0$ we ask to determine if there exists a pair of orthogonal vectors, which recovers the orthogonal vectors problem.

Clearly, an $O(d^{O(1)}N^{2-\delta})$ algorithm for the almost orthogonal vectors problem in d dimensions implies a similar algorithm for the orthogonal vectors problem, while the other direction might not be true. In fact, while mildly sub-quadratic algorithms are known for the orthogonal vectors problem when d is poly-logarithmic, with $N^2/(\log N)^{\omega(1)}$ running times [CIP02, ILPS14, AWY15], we are not aware of any such algorithms for the almost orthogonal vectors problem.

The lemma below shows that such algorithms for the almost orthogonal vectors problem would imply new $2^n/n^{\omega(1)}$ algorithms for MAX-SAT Problem on a polynomial number of clauses. Given a conjunctive normal form formula on n variables, the MAX-SAT Problem asks to output the maximum number of clauses that an assignment to the variables can satisfy. While such upper bounds are known for the SAT problem [AWY15, DH09], $2^n/n^{\omega(1)}$ upper bounds are known for MAX-SAT only when the number of clauses is a sufficiently small polynomial in the number of variables [DW06, SSTT17]. The reductions that we present in Chapters 3 and 4 from the almost orthogonal vectors to edit distance, LCS and DTW incur only a poly-logarithmic overhead (for poly-logarithmic dimension d). This implies that shaving a super-poly-logarithmic factor over the quadratic running times for these problems might be difficult.

Lemma 2.5. *If the almost orthogonal vectors problem on N vectors in $\{0,1\}^d$ can be solved in $T(N,d)$ time, then given a conjunctive normal form formula on n variables and m clauses, we can compute the maximum number of satisfiable clauses (MAX-SAT), in $O(T(2^{n/2},m)\cdot \log m)$ time.*

Proof. We will use the split-and-list technique from [Wil05]. Given a conjunctive normal form formula on n variables and m clauses, split the variables into two sets of size $n/2$ and list all $2^{n/2}$ partial assignments to each set. Define a vector $a(\alpha)$ for each partial assignment α to the first half of variables. $a(\alpha)$ contains 0 at coordinate $j \in [m]$ if α sets any of the literals of the j-th clause of the formula to true, and 1 otherwise. In other words, it contains a 0 if the partial assignment satisfies the clause and 1 otherwise. In the same way construct a vector $b(\beta)$ for each partial assignment β to the second half of variables. Now, observe that if α, β is a pair of partial assignments for the first and second set of variables, then the inner product of $a(\alpha)$ and $b(\beta)$ is equal to the number of clauses that the combined assignment α and β does not satisfy. Therefore, to find the assignment that maximizes the number of satisfied clauses, it is enough to find a pair of partial assignments α, β such that the inner product of $a(\alpha), b(\beta)$ is minimized. The latter can be easily reduced to $O(\log m)$ calls to an oracle for the almost orthogonal vectors problem on $N = 2^{n/2}$ vectors in $\{0,1\}^m$ with the standard binary search. □

By the above discussion, a lower bound that is based on the almost orthogonal vectors problem can be considered stronger than one that is based on the orthogonal vectors problem. In our hardness proofs for the kernel problems it will be more convenient to work with the following closely related problem.

Definition 2.6 (Bichromatic Hamming close pair (BHCP) problem)**.** *Given two sets $A, B \subseteq \{0,1\}^d$ such that $|A| = |B| = N$ and an integer $t \in \{0, \ldots, d\}$, determine whether there exists a pair of vectors $a \in A$ and $b \in B$ such that the number of coordinates in which they differ is less than t. Formally, the Hamming distance is less than t: $Hamming(a, b) \triangleq \|a - b\|_1 < t$. If there is such a pair a, b of vectors, we call it a* close pair*.*

It turns out that the best runtimes for the bichromatic Hamming close pair problem and the almost orthogonal vectors problem are the same up to factors polynomial in d as is shown by the following lemma.

Lemma 2.7. *If the bichromatic Hamming close pair problem can be solved in $T(N, d)$ time, then the almost orthogonal vectors problem can be solved in $O(d^2 T(N, d))$ time. Similarly, if the almost orthogonal vectors problem can be solved in $T'(N, d)$, then the bichromatic Hamming close pair problem can be solved in $O(d^2 T'(N, d))$ time.*

Proof. We show how to reduce the almost orthogonal vectors problem to $(d+1)^2$ instances of the bichromatic Hamming close pair problem. Let A and B be the sets of input vectors to the almost orthogonal vectors problem. Let r be the threshold. We split $A = A^0 \cup A^1 \cup \ldots \cup A^d$ into $d+1$ sets such that A^i contains only those vectors that have exactly i entries equal to 1. Similarly, we split $B = B^0 \cup B^1 \cup \ldots \cup B^d$. Let $\bar{B^j}$ be the set of vectors B^j except we replace all entries that are equal to 1 with 0 and vice versa. We observe that A^i and B^j have a pair of vectors that are r-orthogonal (dot product is $\leq r$) if and only if there is a pair of vectors from A^i and $\bar{B^j}$ with the Hamming distance at most $d - i - j + 2r$. Thus, to solve the almost orthogonal vectors problem, we invoke an algorithm for the bichromatic Hamming close pair problem on all $(d+1)^2$ pairs A^i and B^j.

It remains to show how to reduce the Hamming close pair problem to $(d+1)^2$ instances of the bichromatic almost orthogonal vectors problem. As before, we split $A = A^0 \cup A^1 \cup \ldots \cup A^d$ and $B = B^0 \cup B^1 \cup \ldots \cup B^d$. Let t be the distance threshold. We observe that A^i and B^j have a pair of vectors that differ in less than t entries if and only if A^i and $\bar{B^j}$ have a pair of vectors with dot product $< (i - j + t)/2$. Thus, to solve the bichromatic Hamming close pair problem, we invoke an algorithm for the almost orthogonal vectors problem on all $(d+1)^2$ pairs A^i and $\bar{B^j}$. □

Unbalanced orthogonal vectors problem. Some of our reductions are from the unbalanced version of the orthogonal vectors problem in which the sets of the input vectors have different sizes. More precisely, $|A| = N$ and $|B| = M$, where $M = N^\alpha$ for a constant $\alpha > 0$. It turns out that this variant of the problem cannot be solved in $d^{O(1)}(NM)^{1-\delta}$ time unless SETH is false [BK15]. The following short argument proves this. Without loss of generality we assume that $\alpha \leq 1$. Given a *balanced* instance of orthogonal vectors problem with $|A'| = |B'| = N$, we split the set B' into N/M subsets of size M: $B' = B'_1 \cup \ldots \cup B'_{N/M}$. This gives N/M instances of the unbalanced problem: one instance for every pair of sets A' and B'_i. If a faster algorithm for the unbalanced problem exists, then we can solve the balanced problem in time $N/M \cdot d^{O(1)}(NM)^{1-\delta} = d^{O(1)} N^{2-\alpha\delta}$, which contradicts SETH.

The same argument works for the unbalanced version of the bichromatic Hamming close pair problem, which we will use in our hardness proofs for the empirical risk minimization problems.

Notation. We use the $\tilde{O}(\cdot)$ notation to hide poly log factors. For an integer i, we use $[i]$ to denote the set $\{1, \ldots, i\}$. For a symbol (or a sequence) s and an integer i, s^i denotes the symbol (or the sequence) s repeated i times. We use $\sum$-style notation to denote the concatenation of sequences. For sequences $s_1, s_2, \ldots, s_k$, we write

$$\bigcirc_{i=1}^{k} s_i = s_1 \circ s_2 \circ \ldots \circ s_k = s_1\, s_2\, \ldots\, s_k$$

to denote the concatenation of the sequences.

Part I

Pattern matching and text analysis

Chapter 3

Edit distance

Many applications require comparing long sequences. For instance, in biology, DNA or protein sequences are frequently compared using sequence alignment tools to identify regions of similarity that may be due to functional, structural, or evolutionary relationships. In speech recognition, sequences may represent time-series of sounds. Sequences could also be English text, computer viruses, points in the plane, and so on. Because of the large variety of applications, there are many notions of sequence similarity. Some of the most important and widely used notions are the edit distance, the longest common subsequence (LCS) and the dynamic time warping (DTW) distance.

In this section we show conditional lower bound for the edit distance problem. We obtain the hardness by a reduction from the orthogonal vectors problem. In Section 3.2.5 we show how to modify the construction so that the reduction is from the almost orthogonal vectors problem.

3.1 Preliminaries

Edit distance. For any two sequences P and Q over an alphabet Σ, the edit distance $\mathrm{edit}(P, Q)$ is equal to the minimum number of symbol insertions, symbol deletions or symbol substitutions needed to transform P into Q. It is well known that the edit distance induces a metric; in particular, it is symmetric and satisfies the triangle inequality.

In our hardness proof we will use use an equivalent definition of the edit distance that will make the analysis of our reductions easier.

Observation 3.1.1. *For any two sequences P, Q, edit(P, Q) is equal to the minimum, over all sequences T, of the number of deletions and substitutions needed to transform P into T and Q into T.*

Proof. It follows directly from the metric properties of the edit distance that $\mathrm{edit}(P, Q)$ is equal to the minimum, over all sequences T, of the number of *insertions*, deletions and substitutions needed to transform P into T and Q into T. Furthermore, observe that if, while transforming P, we insert a symbol that is later aligned with some symbol of Q, we can instead delete the corresponding symbol in Q. Thus, it suffices to allow deletions and substitutions only. □

Definition 3.1.2. *We define the following similarity distance between sequences P and Q and call it the pattern matching distance between P and Q.*

$$pattern(P, Q) \triangleq \min_{\substack{Q' \text{ is a contiguous} \\ \text{subsequence of } Q}} edit(P, Q').$$

Simplifying assumption. We assume that in the orthogonal vectors problem, for all vectors $b \in B$, $b_1 = 1$, that is, the first entry of any vector $b \in B$ is equal to 1. We can make this assumption without loss of generality because we can always add a 1 to the beginning of each $b \in B$, and add a 0 to the beginning of each $a \in A$. Note that the added entries do not change orthogonality of any pair of vectors.

3.2 Reductions

The key component in the reduction is the construction of a function that maps binary vectors into sequences in a way that two sequences are "close" if and only if the vectors are orthogonal. The first step of our reduction mimics the approach in [Bri14]. We assign a "gadget" sequence for each $a \in A$ and $b \in B$. Then, the gadget sequences $G(a)$ for all $a \in A$ are concatenated together to form the first input sequence, and the gadget sequences $G'(b)$ for all $b \in B$ are concatenated to form the second input sequence. The correctness of the reduction is proven by showing that:

- If there is a pair of orthogonal vectors $a \in A$ and $b \in B$, then one can align the two sequences in a way that the gadgets assigned to a and b are aligned, which implies that the distance induced by the global alignment is "small".
- If there is no orthogonal pair, then no such global alignment exists, which implies that the distance induced by any global alignment is "large".

For a fixed ordering $a^1, \ldots, a^N$ of vectors in A and ordering $b^1, \ldots, b^N$ of vectors in B, the edit distance between the final two sequences is (roughly) captured by the formula:

$$\min_{i=1}^{N} \sum_{j=1}^{N} \text{edit}(G(a^j), G'(b^{j+i})),$$

where $b^t = b^{t-N}$ for $t > N$. The edit distance finds an alignment between the sequences of gadgets such that the sum of the distances is minimized. The cost of the optimal alignment is a sum of N distances and we need that it is small if and only if there is a pair of orthogonal vectors.

3.2.1 Vector gadgets

We now describe vector gadgets as well as provide some intuition behind the construction.

We will construct sequences over an alphabet $\Sigma = \{0, 1, 2, 3, 4, 5, 6\}$. We start by defining an integer parameter $l_0 \triangleq 10d$, where d is the dimensionality of the vectors

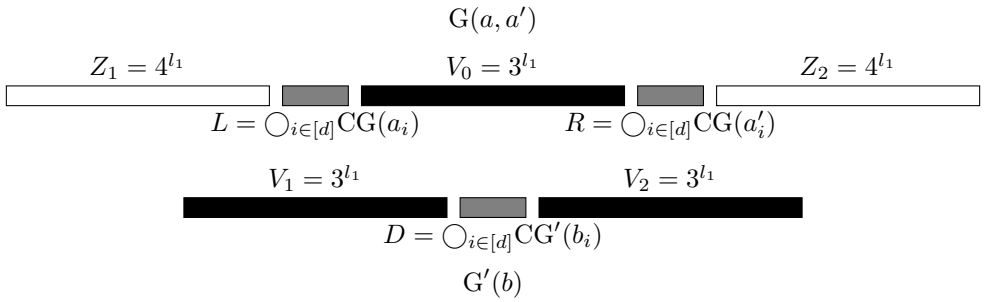

Figure 3-1: A visualization of the vector gadgets. A black rectangle denotes a run of 3s, while a white rectangle denotes a run of 4s. A gray rectangle denotes a sequence that contains 0s, 1s and 2s. A short rectangle denotes a sequence of length l, while a long one denotes a sequence of length l_1.

in the orthogonal vectors problem. We then define *coordinate gadget* sequences CG and CG′ as follows. For an integer $x \in \{0,1\}$ we set

$$\mathrm{CG}(x) \triangleq \begin{cases} 2^{l_0}\,0\,1\,1\,1\,2^{l_0} & \text{if } x = 0; \\ 2^{l_0}\,0\,0\,0\,1\,2^{l_0} & \text{if } x = 1, \end{cases}$$

$$\mathrm{CG}'(x) \triangleq \begin{cases} 2^{l_0}\,0\,0\,1\,1\,2^{l_0} & \text{if } x = 0; \\ 2^{l_0}\,1\,1\,1\,1\,2^{l_0} & \text{if } x = 1. \end{cases}$$

The coordinate gadgets were designed so that they have the following properties. For any two integers $x, x' \in \{0,1\}$,

$$\mathrm{edit}(\mathrm{CG}(x), \mathrm{CG}'(x')) = \begin{cases} 1 & \text{if } x \cdot x' = 0; \\ 3 & \text{if } x \cdot x' = 1. \end{cases}$$

Further, we define another parameter $l_1 \triangleq (10d)^2$. For vectors $a, a', b \in \{0,1\}^d$, we define the *vector gadget* sequences as

$$\mathrm{G}(a,a') \triangleq Z_1\, L(a)\, V_0\, R(a')\, Z_2 \quad \text{and} \quad \mathrm{G}'(b) \triangleq V_1\, D(b)\, V_2,$$

where we set

$$V_0 = V_1 = V_2 \triangleq 3^{l_1}, \quad Z_1 = Z_2 \triangleq 4^{l_1},$$
$$L(a) \triangleq \bigcirc_{i\in[d]} \mathrm{CG}(a_i), \quad R(a') \triangleq \bigcirc_{i\in[d]} \mathrm{CG}(a'_i), \quad D(b) \triangleq \bigcirc_{i\in[d]} \mathrm{CG}'(b_i).$$

In what follows we skip the arguments of L, R and D. We denote the length of L, R and D by $l \triangleq |L| = |R| = |D| = d(4 + 2l_0)$.

We visualize the defined vector gadgets in Fig. 3-1.

Intuition behind the construction. Before going into the analysis of the gadgets in Section 3.2.2, we will first provide some intuition behind the construction. Given three vectors $a, a', b \in \{0,1\}^d$, we want that $\mathrm{edit}(\mathrm{G}(a,a'), \mathrm{G}'(b))$ grows linearly in the

minimum of $a \cdot b$ and $a' \cdot b$. More precisely, we want that

$$\text{edit}(\mathrm{G}(a, a'), \mathrm{G}'(b)) = c + t \cdot \min(a \cdot b, a' \cdot b), \tag{3.1}$$

where the integers $c, t > 0$ are values that depend on d only. In fact, we will have that $t = 2$. To realize this, we construct our vector gadgets G and G′ such that there are only two possibilities to achieve small edit distance. In the first case, the edit distance grows linearly in $a \cdot b$. In the second case, the edit distance grows linearly in $a' \cdot b$. Because the edit distance is equal to the minimum over all possible alignments, we take the minimum of the two inner products. After taking the minimum, the edit distance will satisfy the properties stated in Eq. (3.1). More precisely, we achieve the minimum edit distance cost between G and G′ by following one of the following two possible sequences of operations:

- Case 1. Delete Z_1 and L. Substitute Z_2 with V_2. This costs $c' \triangleq |Z_1| + |L| + |Z_2| = 2l_1 + l$. Transform R and D into the same sequence by transforming the corresponding coordinate gadgets into the same sequences. By the construction of the coordinate gadgets, the cost of this step is $d + 2 \cdot (a' \cdot b)$. Therefore, this case corresponds to edit distance cost $c' + d + 2 \cdot (a' \cdot b) = c + 2 \cdot (a' \cdot b)$ for $c \triangleq c' + d$.

- Case 2. Delete R and Z_2. Substitute Z_1 with V_1. This costs c'. Transform L and D into the same sequence by transforming the corresponding coordinate gadgets. Similarly as before, the cost of this step is $d + 2 \cdot (a \cdot b)$. Therefore, this case corresponds to edit distance cost $c' + d + 2 \cdot (a \cdot b) = c + 2 \cdot (a \cdot b)$.

Taking the minimum of these two cases yields the desired Eq. (3.1).

In the reduction given in Section 3.2.3, we ensure that the dot product $a' \cdot b$ is always equal to 1 by setting a' to be equal to a fixed vector. This gives us a gadget $\mathrm{G}(a) \triangleq \mathrm{G}(a, a')$ with the property that $\text{edit}(\mathrm{G}(a), \mathrm{G}'(b))$ is small (equal to C_0) if the vectors a and b are orthogonal, and is slightly larger (equal to C_1) otherwise. That is:

$$\text{edit}(\mathrm{G}(a), \mathrm{G}'(b)) = \begin{cases} C_0 & \text{if } a \cdot b = 0 \\ C_1 & \text{otherwise} \end{cases}$$

for $C_1 > C_0$. This property is crucial for our construction, as it guarantees that the sum of several terms $\text{edit}(\mathrm{G}(a), \mathrm{G}'(b))$ is smaller than some threshold if and only if $a \cdot b = 0$ for at least one pair of vectors a and b. This enables us to detect whether such a pair exists. In contrast, this would not hold if $\text{edit}(\mathrm{G}(a), \mathrm{G}'(b))$ depended linearly on the value of $a \cdot b$.

3.2.2 Properties of the vector gadgets

Theorem 3.2.1. *For any vectors $a, a', b \in \{0,1\}^d$,*

$$\textit{edit}(\, G(a, a'), G'(b)) = 2l_1 + l + d + 2 \cdot \min\left(a \cdot b, a' \cdot b\right).$$

Proof. Follows from Lemmas 3.2.2 and 3.2.3 below. □

Lemma 3.2.2. *For any vectors $a, a', b \in \{0,1\}^d$,*

$$\mathit{edit}(\,G(a,a'),\,G'(b)) \leq 2l_1 + l + d + 2 \cdot \min\left(a \cdot b, a' \cdot b\right).$$

Proof. Without loss of generality, $a \cdot b \leq a' \cdot b$. We delete R and Z_2 from $\mathrm{G}(a,a')$. This costs $l_1 + l$. We transform $Z_1\,L\,V_0$ into $V_1\,D\,V_2$ by using substitutions only. This costs $l_1 + d + 2 \cdot (a \cdot b)$. We get the upper bound on the edit cost and this finishes the proof. □

Lemma 3.2.3. *For any vectors $a, a', b \in \{0,1\}^d$,*

$$\mathit{edit}(\,G(a,a'),\,G'(b)) \geq X \triangleq 2l_1 + l + d + 2 \cdot \min\left(a \cdot b, a' \cdot b\right).$$

Proof. Consider an optimal transformation of $\mathrm{G}(a,a')$ and $\mathrm{G}'(b)$ into the same sequence. Every symbol (say x) in the first sequence is either substituted, preserved or deleted in the process. If a symbol is not deleted but instead is preserved or substituted by another symbol (say y), we say that x is *aligned* with y, or that x and y have an *alignment*.

We state the following fact without a proof.

Fact 3.2.4. *Suppose we have two sequences P and Q of symbols. Let $i < j$ and $i' < j'$ be four positive integers. If P_i is aligned with $Q_{j'}$, then P_j cannot be aligned with $Q_{i'}$.*

From now on we proceed by considering three cases.

Case 1. The subsequence D has alignments with both $Z_1\,L$ *and* $R\,Z_2$. In this case, the cost induced by symbols from Z_1 and Z_2, and V_0 is l_1 for each one of these sequences because the symbols must be deleted or substituted. This implies that $\mathrm{edit}(\mathrm{G}(a,a'),\mathrm{G}'(b)) \geq 3l_1$, which contradicts an easy upper bound. We have an upper bound $\mathrm{edit}(\mathrm{G}(a,a'),\mathrm{G}'(b)) \leq 2l_1 + 3l$ and it is obtained by deleting L, R, D, Z_1 and replacing Z_2 with V_2 symbol by symbol. Remember that $l_0 = 10d$ and $l_1 = (10d)^2$, and $l = d(4 + l_0)$. Thus, $l_1 > 3l$ and the lower bounds contradicts the upper bound. Therefore, this case cannot occur.

Case 2. D does not have any alignments with $Z_1\,L$. We will show that, if this case happens, then

$$\mathrm{edit}(\mathrm{G}(a,a'),\mathrm{G}'(b)) \geq 2l_1 + l + d + 2 \cdot (a' \cdot b).$$

We start by introducing the following notion. Let P and Q be two sequences that decompose as $P = P_1P_2$ and $Q = Q_1Q_2$. Consider two sequences $\mathcal{T}$ and $\mathcal{R}$ of deletions and substitutions that transform P into S and Q into S, respectively. An operation in $\mathcal{T}$ or $\mathcal{R}$ is called *internal to* P_2 and Q_2 if it is either a (1) deletion of a symbol in P_2 or Q_2, or (2) a substitution of a symbol in P_2 so that it aligns with a symbol in Q_2, or vice versa. All other operations, including substitutions that align with symbols in P_2 (Q_2, resp.) to those outside of Q_2 (P_2, resp.) are called *external to* P_2 and Q_2.

We state the following fact without a proof.

Fact 3.2.5. *Let $P_1\,P_2$ and $Q_1\,Q_2$ be sequences such that $P_2 = 4^t$, $Q_2 = 3^t$ for an integer t and P_1 and Q_1 are arbitrary sequences over an arbitrary alphabet not including 3 or 4. Consider edit$(P_1\,P_2, Q_1\,Q_2)$ and the corresponding operations minimizing the distance. Among those operations, the number of operations that are internal to P_2 and Q_2 is at least t.*

Given that $|Z_2| = |V_2| = l_1$ and Z_2 consists only of 4s and V_2 consists of only 3s, Fact 3.2.5 implies that the number of operations that are internal to Z_2 and V_2 is at least $S_1 \triangleq l_1$.

Because D does not have any alignments with $Z_1\,L$, we must have that every symbol in $Z_1\,L$ gets deleted or substituted. Thus, the total contribution from symbols in $Z_1\,L$ to an optimal alignment is $S_2 \triangleq |Z_1\,L| = l_1 + l$. Now we will lower bound the contribution to an optimal alignment from symbols in sequences R and D. First, observe that both R and D have d runs of 1s. We consider the following two subcases.

Case 2.1. There exist $i, j \in [d]$ with $i \neq j$ such that the ith run in D has alignments with the jth run in R. The number of symbols of type 2 to the right of the ith run in D and the number of symbols of type 2 to the right of the jth run in R differ by at least $2l_0$. Therefore, the induced edit cost of symbols of type 2 in R and D is at least $2l_0 \geq S_3 \triangleq d + 2 \cdot (a' \cdot b)$, from which we conclude that

$$\begin{aligned}\text{edit}(\mathrm{G}(a, a'), \mathrm{G}'(b)) &\geq S_1 + S_2 + S_3 \\ &= l_1 + (l_1 + l) + (d + 2 \cdot (a' \cdot b)) \\ &= X.\end{aligned}$$

In the inequality we used the fact that the contributions from S_1, S_2 and S_3 are disjoint. This follows from the definitions of the quantities. More precisely, the contribution from S_1 comes from operations that are *internal* to V_2 and Z_2. Thus, it remains to show that the contributions from S_2 and S_3 are disjoint. This follows from the fact that the contribution from S_2 comes from symbols $Z_1\,L$ and the assumption that D does not have any alignments with $Z_1\,L$.

Case 2.2. (The complement of Case 2.1.) Consider any $i \in [d]$. If a symbol of type 1 from the ith run in D is aligned with a symbol of type 1 in R, then the symbol of type 1 comes from the ith run in R. Define the set Z as the set of all numbers $i \in [d]$ such that the ith run of 1s in D has alignment with the ith run of 1s in R.

For all $i \in Z$, the ith run in R aligns with the ith run in D. By the construction of coordinate gadgets, the ith run in R and D incur edit cost $\geq 1 + 2a'_i b_i$.

For all $i \notin Z$, the ith run in D incurs edit cost at least 2 (since there are at least two symbols of type 1). Similarly, the ith run in R incurs edit cost at least 1 (since there is at least one symbol of type 1). Therefore, for every $i \notin Z$, the ith run in R and D incur edit cost $\geq 1 + 2 \geq 1 + 2a'_i b_i$.

We get that the total contribution to the edit cost from the d runs in D and the d runs in R is

$$\sum_{i \in Z} (1 + 2a'_i b_i) + \sum_{i \in [d] \setminus Z} 3 \geq \sum_{i=1}^{d} (1 + 2a'_i b_i) = S_4 \triangleq d + 2 \cdot (a' \cdot b)\,.$$

We conclude:

$$\begin{aligned}\mathrm{edit}(\mathrm{G}(a,a'),\mathrm{G}'(b)) &\geq S_1+S_2+S_4\\ &= l_1+(l_1+l)+(d+2\cdot(a'\cdot b))\\ &= X.\end{aligned}$$

We used the fact that the contributions from S_1, S_2 and S_4 are disjoint. The argument is analogous as in the previous case.

Case 3. The symbols of D are not aligned with any symbols in $R\,Z_2$. If this case happens, then

$$\mathrm{edit}(\mathrm{G}(a,a'),\mathrm{G}'(b)) \geq 2l_1+l+d+2\cdot(a\cdot b) = X.$$

The analysis of this case is analogous to the analysis of Case 2. More concretely, for any sequence P, define reverse(P) to be the sequence Q of length $|P|$ such that $Q_i = P_{|P|+1-i}$ for all $i = 1, 2, \ldots, |P|$. Now we repeat the proof in Case 2 but for

$$\mathrm{edit}(\mathrm{reverse}(\mathrm{G}(a,a')),\mathrm{reverse}(\mathrm{G}'(b))).$$

This yields exactly the lower bound that we need.

The proof of the lemma follows. We showed that Case 1 cannot happen. By combining lower bounds corresponding to Cases 2 and 3, we get the lower bound stated in the lemma. □

We set $a' \triangleq 1\,0^{d-1}$, that is, a' is a binary vector of length d such that $a'_1 = 1$ and $a'_i = 0$ for $i = 2, \ldots, d$. We define

$$\mathrm{G}(a) \triangleq \mathrm{G}(a,a').$$

Theorem 3.2.6. *Let $a \in \{0,1\}^d$ be any binary vector and $b \in \{0,1\}^d$ be any binary vector that starts with 1, that is, $b_1 = 1$. Then,*

$$\mathit{edit}(\,G(a),\,G'(b)) = \begin{cases} C_0 \triangleq 2l_1+l+d & \text{if } a\cdot b = 0;\\ C_1 \triangleq 2l_1+l+d+2 & \text{if } a\cdot b \geq 1.\end{cases}$$

Proof. Follows from Theorem 3.2.1 by setting $a' = 1\,0^{d-1}$ and observing that $a'\cdot b = 1$ because $b_1 = 1$. □

3.2.3 Hardness for pattern matching

We proceed by concatenating vector gadgets into sequences.

We note that the length of the vector gadgets G depends on the dimensionality d of the vectors but not on the entries of the vectors. The same is true about G′. We set l' to be the maximum of the two lengths. Furthermore, we set $l_2 \triangleq 10dl'$. We define $\hat{\mathrm{G}}(a) \triangleq 5^{l_2}\,\mathrm{G}(a)\,5^{l_2}$ and $\hat{\mathrm{G}}'(a) \triangleq 5^{l_2}\,\mathrm{G}'(a)\,5^{l_2}$. Let b' be a vector consisting of d entries equal to 1, that is, $b'_i = 1$ for $i = 1, \ldots, d$.

Let A and B be sets from the orthogonal vectors instance. We define sequences

$$P \triangleq \bigcirc_{a \in A} \hat{\mathrm{G}}(a),$$

$$Q \triangleq \left(\bigcirc_{i=1}^{|A|-1} \hat{\mathrm{G}}'(b')\right) \left(\bigcirc_{b \in B} \hat{\mathrm{G}}'(b)\right) \left(\bigcirc_{i=1}^{|A|-1} \hat{\mathrm{G}}'(b')\right).$$

Theorem 3.2.7. *Let $C \triangleq |A|C_1$. If there are two orthogonal vectors, one from set A, another from set B, then pattern$(P, Q) \leq C - 2$; otherwise we have pattern$(P, Q) = C$.*

Proof. Follows from Lemmas 3.2.8 and 3.2.9 below. □

Lemma 3.2.8. *If there are two orthogonal vectors, one from A, another from B, then*

$$\textit{pattern}(P, Q) \leq C - (C_1 - C_0) = C - 2.$$

Proof. Let $a \in A$ and $b \in B$ be vectors such that $a \cdot b = 0$.

We can choose a contiguous subsequence Q' of Q consisting of a sequence of $|A|$ vector gadgets $\hat{\mathrm{G}}'$ such that Q' has the following property: transforming the vector gadgets $\hat{\mathrm{G}}$ from P and their corresponding vector gadgets $\hat{\mathrm{G}}'$ from Q' into the same sequence one by one as per Theorem 3.2.6, we achieve a cost smaller than the upper bound. We use the fact that at least one transformation is cheap because $a \cdot b = 0$ and we choose Q' so that $\hat{\mathrm{G}}(a)$ and $\hat{\mathrm{G}}'(b)$ get transformed into the same sequence. □

Lemma 3.2.9. *If there are no two orthogonal vectors, one from A, another from B, then*

$$\textit{pattern}(P, Q) = C.$$

Proof. Consider a graph $(V \cup V', E)$ with vertices $v(a) \in V$, $a \in A$, $v'(b) \in V'$, $b \in B$. We also add $2|A| - 2$ copies of $v'(b')$ to the set V' corresponding to $2|A| - 2$ vectors b' in sequence V'. Consider an optimal transformation of P and a subsequence of Q into the same sequence according to Definition 3.1.2. We connect two vertices $v(a)$ and $v'(b)$ if and only if $\mathrm{G}(a)$ and $\mathrm{G}'(b)$ have an alignment in the transformation.

We want to claim that every vector gadget $\mathrm{G}(a)$ from P contributes a cost of at least C_1 to the final cost of pattern(P, Q). This will give pattern$(P, Q) \geq |A|C_1 = C$. We consider the connected components of the graph. We will show that a connected component that has $r \geq 1$ vertices from V, contributes $\geq rC_1$ to the final cost of pattern(P, Q). From the case analysis below we will see that these contributions for different connected components are separate. Therefore, by summing up the contributions for all the connected components, we get pattern$(P, Q) \geq |A|C_1 = C$.

Consider a connected component of the graph with at least one vertex from V. We examine several cases.

Case 1. The connected component has only one vertex from V. Let $v(a)$ be the vertex.

Case 1.1. $v(a)$ is connected to more than one vertex. In this case, $\mathrm{G}(a)$ induces a cost of at least $2l_2 > C_1$ (this cost is induced by symbols of type 5).

Case 1.2. $v(a)$ (corresponding to vector gadget $\mathrm{G}(a)$) is connected to only one vertex $v'(b)$ (corresponding to vector gadget $\mathrm{G}'(b)$). Let Q' be a contiguous substring of Q that achieves the minimum of $\mathrm{edit}(P, Q')$ (see Definition 3.1.2).

Case 1.2.1. The vector gadget $\mathrm{G}'(b)$ is fully contained in the substring Q'. We claim that the contribution from symbols in the sequences $\mathrm{G}(a)$ and $\mathrm{G}'(b)$ is at least $\mathrm{edit}(\mathrm{G}(a), \mathrm{G}'(b))$. This is sufficient because we know that $\mathrm{edit}(\mathrm{G}(a), \mathrm{G}'(b)) \geq C_1$ from Theorem 3.2.6. If no symbol in $\mathrm{G}(a)$ or $\mathrm{G}'(b)$ is aligned with a symbol of type 5, the claim follows directly by applying Theorem 3.2.6. Otherwise, every symbol that is aligned with a symbol of type 5 contributes cost 1 to the final cost. The contribution from symbols in the sequences $\mathrm{G}(a)$ and $\mathrm{G}'(b)$ is at least $\mathrm{edit}(\mathrm{G}(a), \mathrm{G}'(b))$ because we can transform the sequences $\mathrm{G}(a)$ and $\mathrm{G}'(b)$ into the same sequence by first deleting the symbols that are aligned with symbols of type 5 (every such alignment contributes cost 1) and then transforming the remainders of the sequences $\mathrm{G}(a)$ and $\mathrm{G}'(b)$ into the same sequence.

Case 1.2.2. The complement of Case 1.2.1. We need to consider this case because of the following reason. We could potentially achieve a contribution of $\mathrm{G}(a)$ to $\mathrm{pattern}(P, Q)$ that is smaller than C_1 by transforming $\mathrm{G}(a)$ and a *contiguous substring* of $\mathrm{G}'(b)$ into the same string (instead of transforming $\mathrm{G}(a)$ and $\mathrm{G}'(b)$ into the same string). In the next paragraph we show that this cannot happen.

$\mathrm{G}'(b)$ shares symbols with Q' and is not fully contained in Q'. $\mathrm{G}'(b)$ must be the left-most (right-most, resp.) vector gadget in Q' but then l_2 left-most (right-most, resp.) symbols of type 5 of $\hat{\mathrm{G}}(a)$ induce a cost of at least $l_2 > C_1$ since the symbols of type 5 cannot be preserved and must be substituted or deleted.

Case 1.3. $v(a)$ is connected to no vertex. We get that $\mathrm{G}(a)$ induces cost of at least $|\mathrm{G}(a)| > C_1$.

Case 2. The connected component has $r > 1$ vertices $v(a)$ from V. In this case, the cost induced by the vector gadgets $\mathrm{G}(a)$ corresponding to the vertices from V in the connected component is at least $2l_2(r-1) > rC_1$ (this cost is induced by symbols of type 5).

This finishes the argument that $\mathrm{pattern}(P, Q) \geq C$. It remains to argue that we can achieve cost C (to show that $\mathrm{pattern}(P, Q) \leq C$) and it can be done as in Lemma 3.2.8. □

3.2.4 Hardness for edit distance

We set $P' \triangleq 6^{|Q|}\, P\, 6^{|Q|}$ and $Q' \triangleq Q$. The following theorem immediately implies hardness for the edit distance problem.

Theorem 3.2.10. *Let $C' \triangleq 2|Q'| + C$. If there are no two orthogonal vectors, then $\mathit{edit}(P', Q') = C'$; otherwise $\mathit{edit}(P', Q') \leq C' - 2$.*

Proof. Follows from Lemmas 3.2.11 and 3.2.12 below. □

Lemma 3.2.11. *If there are two orthogonal vectors, then $\mathit{edit}(P', Q') \leq C' - 2$.*

Proof. We transform P and a subsequence of Q' into the same sequence as in Lemma 3.2.8. We replace the remaining prefix and suffix of Q' with the symbols of type 6 and delete the excess of symbols of type 6 from P'. □

Lemma 3.2.12. *If there are no two orthogonal vectors, then*

$$\mathit{edit}(P', Q') = C'.$$

Proof. We can easily check that $\mathrm{edit}(P', Q') \leq C'$ as in Lemma 3.2.11. It remains to prove the opposite inequality.

P' contains $2|Q'|$ symbols of type 6. Those will incur a cost of at least $2|Q'|$. P' has the remaining subsequence P, which will incur cost at least $\mathrm{pattern}(P, Q')$. Using Lemma 3.2.9, we finish the proof. □

3.2.5 Reduction from the almost orthogonal vectors problem

We do the following changes to the above reduction to obtain a reduction from the almost orthogonal vectors problem. First, we change the simplifying assumption in Section 3.1. Instead of making an assumption that the first entry of the vector b is 1, we make an assumption that the first $r+1$ entries are equal to 1. Remember that r is the threshold in the almost orthogonal vectors instance and our goal is to determine if there is a pair of vectors with the dot product at most r. Next, at the end of Section 3.2.2 we set $a' \triangleq 1^{r+1}\,0^{d-r-1}$ (instead of $1\,0^{d-1}$). Finally, in Theorem 3.2.6 we set $C_0 \triangleq 2l_1 + l + d + 2r$ and $C_1 \triangleq 2l_1 + l + d + 2r + 2$. The rest of the construction goes through by replacing the condition that the vectors are orthogonal with the condition that the vectors are r-orthogonal.

Chapter 4

Longest common subsequence and dynamic time warping

In this section we extend hardness results described in Chapter 3 to other similarity measures, including longest common subsequence and dynamic time warping.

Longest common subsequence. This problem asks to output the length of the longest common subsequence between two given sequences of length n. LCS has attracted an extensive amount of research, both due to its mathematical simplicity and to its large number of important applications, including data comparison programs (e.g., **diff** in UNIX) and bioinformatics (e.g., [JP04]). There are many algorithms for LCS, beyond the classical quadratic-time dynamic programming solution, in many different settings, e.g., [Hir75, HS77] (see [BHR00] for a survey). Nevertheless, the best algorithms on arbitrary sequences are only slightly sub-quadratic and have an $O(n^2/\log^2 n)$ running time [MP80] if the alphabet size is constant, and $O(n^2(\log\log n)/\log^2 n)$ otherwise [BFC08, Gra16].

Dynamic time warping distance. DTW distance assumes a distance measure between any two symbols and is defined in terms of a "best" joint traversal of the sequences. The traversal places a marker at the beginning of each sequence and during each step one or both markers are moved forward one symbol, until the end of both sequences is reached. Each step aligns two symbols, one from each sequence. DTW defines the quality of the traversal to be the sum of distances between all pairs of aligned symbols.

Dynamic time warping is useful in scenarios in which one needs to cope with differing speeds and time deformations of time-dependent data. Because of its generality, DTW has been successfully applied in a large variety of domains: automatic speech recognition [RJ93], music information retrieval [Mül07] , bioinformatics [AC01], medicine [CPB+98], identifying songs from humming [ZS03], indexing of historical handwriting archives [RM03], databases [RK05, KR05] and more.

DTW compares sequences over an arbitrary feature space, equipped with a distance function for any pair of symbols. The sequences may represent time series or features sampled at equidistant points in time. The cost function differs with the application.

For instance, if the features are real numbers, then the distance could be ℓ_p. A simple cost function which is useful when comparing text is to have the cost between two letters be 1 if they are different and 0 if they are the same (See Example 4.2. in [Mül07] for this version).

A simple dynamic programming algorithm solves DTW in $O(n^2)$ time and the best known algorithm takes $O(n^2(\log\log\log n)/\log\log n)$ time on the worst-case case inputs [GS17]. Besides algorithms for the worst-case inputs, many heuristics were designed in order to obtain faster runtimes in practice (see Wang et al. [WDT+10] for a survey).

Unfortunately, despite substantial research, the current best algorithms for LCS and DTW problems are only mildly sub-quadratic—one can shave small poly-logarithmic factors, but there is no known strongly sub-quadratic, $O(n^{2-\varepsilon})$ time algorithm, for a constant $\varepsilon > 0$. In this part of the thesis we show that neither LCS nor DTW admits strongly sub-quadratic algorithms, unless SETH fails. Our lower bounds hold when the input sequences are over a constant size alphabet.

Related work. In [BK15] a quadratic lower bound was independently obtained for LCS and DTW. [AHVWW16] showed that it is possible to relax the hardness assumption and replace SETH with a weaker assumption. Essentially, hardness of satisfiability of circuits as opposed of conjunctive normal form formulas. This was further strengthened in [AB18]. In [BK18] tight conditional lower bounds were shown for various parameterization of the LCS problem.

4.1 Preliminaries

Longest common subsequence. For two sequences P and Q of length n over an alphabet Σ, the longest sequence X that appears in both P and Q as a subsequence is the *longest common subsequence* (LCS) of P, Q and we say that $\mathrm{LCS}(P,Q) \triangleq |X|$. The LCS problem asks to output $\mathrm{LCS}(P,Q)$.

Dynamic time warping distance. For two sequences P and Q of n points from a set Σ and a distance function $d : \Sigma \times \Sigma \to [0,\infty)$, the *dynamic time warping distance*, denoted by $\mathrm{DTW}(P,Q)$, is the minimum cost of a (monotone) *traversal* of P and Q.

A traversal of the two sequences P, Q has the following form. We have two markers and initially one is located at the beginning of P and the other is located at the beginning of Q. At every step, one or both of the markers simultaneously move one point forward in their corresponding sequences. At the end, both markers must be located at the last point of their corresponding sequence.

To determine the *cost* of a traversal, we consider all the $O(n)$ steps of the traversal, and add up the following quantities to the final cost. Let the configuration of a step be the pair of symbols p and q that the first and second markers are pointing at, respectively. The contribution of this step to the final cost is $d(p,q)$.

The DTW problems asks to output $\mathrm{DTW}(P,Q)$.

In the rest we will be interested in the following special case of DTW.

DTW over symbols. The DTW problem over sequences of symbols, is the special case of DTW in which the points come from an alphabet Σ and the distance function is such that for any two symbols $p, q \in \Sigma$, $d(p,q) = 1$ if $p \neq q$ and $d(p,q) = 0$ if $p = q$.

4.2 Hardness for longest common subsequence

In this section we show a conditional lower bound for the longest common subsequence problem.

4.2.1 Weighted LCS

It will be more convenient to work with the following more general version of the LCS problem.

Definition 4.2.1 (Weighted longest common subsequence (WLCS))**.** *For two sequences P and Q of length n over an alphabet Σ and a weight function $w : \Sigma \to [K]$, let X be the sequence that appears in both P, Q as a subsequence and maximizes the total weight (weighted length) $W(X) \triangleq \sum_{i=1}^{|X|} w(X_i)$. We say that X is the weighted longest common subsequence of P, Q and write $WLCS(P,Q) \triangleq W(X)$. The WLCS problem asks to output $WLCS(P,Q)$.*

Note that a common subsequence X of two sequences P, Q can be thought of as an alignment or a matching $M = \{(p_i, q_i)\}_{i=1}^{|X|}$ between the two sequences, so that for all $i \in [|X|] : P_{p_i} = Q_{q_i}$, and $p_1 < \ldots < p_{|X|}$ and $q_1 < \ldots < q_{|X|}$. Clearly, the weight $\sum_{i=1}^{|X|} P_{p_i} = \sum_{i=1}^{|X|} Q_{q_i}$ of the matching M corresponds to the weight length $W(X)$ of the common subsequence X.

In our proofs, we will find useful the following relation between pairs of indices. For a pair (p, q) and a pair (p', q') of indices we say that they are in *conflict* or they *cross* if $p < p'$ and $q > q'$ or $p > p'$ and $q < q'$.

The following simple reduction from WLCS to LCS gives a way to translate a lower bound for WLCS to a lower bound for LCS, and allows us to simplify our proofs.

Lemma 4.2.2. *Computing WLCS of two sequences of length n over Σ with weights $w : \Sigma \to [K]$ can be reduced to computing the LCS of two sequences of length $O(Kn)$ over (unweighted) Σ.*

Proof. The reduction simply copies each symbol $s \in \Sigma$ in each of the sequences $w(s)$ times. That is, we define a mapping f from symbols in Σ to sequences of length up to K so that for any $s \in \Sigma$, $f(s) \triangleq x^{w(s)} \in \Sigma^{w(s)}$. For a sequence S of length m over Σ, let $f(S) \triangleq \bigcirc_{i=1}^{m} f(S_i)$. That is, replace the i-th symbol S_i with the sequence $f(S_i)$ as defined above. Note that $|f(P)| \leq K|P|$.

The reduction follows from the next claim.

Claim 4.2.3. *For any two sequences P, Q of length n over Σ, the mapping f satisfies:*

$$WLCS(P,Q) = LCS(f(P), f(Q)).$$

Proof. For brevity of notation, we let $P' \triangleq f(P)$ and $Q' \triangleq f(Q)$.

First, observe that $\mathrm{WLCS}(P,Q) \leq \mathrm{LCS}(P',Q')$, since for any common subsequence X of P and Q, the sequence $f(X)$ is a common subsequence of P' and Q' and has length $|f(X)| = \sum_{i=1}^{m} |f(X_i)| = \sum_{i=1}^{m} w(X_i) = W(X)$, where $m \triangleq |S|$.

In the remainder of this proof, we show that $\mathrm{WLCS}(P,Q) \geq \mathrm{LCS}(P',Q')$. Let X be the LCS of P', Q' and consider a corresponding matching M as defined above.

We say that a symbol s in P' at index $i \in [|P'|]$ belongs to interval $I_P(i) \in [|P|]$ if and only if this symbol was generated when mapping $P_{I_P(i)}$ to the subsequence $f(s)$. Moreover, we say that it is at index $J_P(i) \in [w(s)]$ in interval $I_P(i)$ if and only if it is the $J_P(i)$-th symbol in that interval (in the subsequence $f(s)$). We define $I_Q(i)$ and $J_Q(i)$ in the same manner.

We will go over the symbols $s \in \Sigma$ of the alphabet in an arbitrary order, and perform the following modifications to X and the corresponding matching M for each such symbol in turn.

Go over the indices i of P' that are matched in M to some index j of Q', and for which $P'_i = s$, in increasing order. Consider the intervals $I_P(i)$ and $I_Q(j)$, both of which contain the symbol s, $w(s)$ times. Throughout our scan, we maintain the invariant that: i is the first index to be matched to the interval $I_Q(j)$.

If $J_P(i) = J_Q(j) = 1$, and the next $w(s)-1$ pairs in our matching M are matching the rest of the interval $I_P(i)$ to the interval $I_Q(j)$, we do not need to modify anything, and we move on to the next index i' that is not a part of this interval $I_P(i)$ and is matched to some index j'—note that at this point, i' satisfies the invariant, since it cannot also be matched to the interval $I_Q(j)$, and therefore $I_Q(j') > I_Q(j)$ and i' is the first index to be matched to the interval $I_Q(j')$.

If $J_P(i) = J_Q(j) = 1$ does not hold, we modify the matching M so that now the whole intervals $I_P(i)$ and $I_Q(j)$ are matched to one another: for each i', j' such that $I_P(i') = I_P(i), I_Q(j') = I_Q(j)$, and $J_P(i') = J_Q(j')$, we add pair (i', j') to the matching M, and remove any conflicting pairs from M. We claim that we obtain a matching of at least the original size, since we add $w(s)$ pairs and we remove only up to $w(s)$ pairs. To see this, note that for a pair (i'', j'') to be in conflict with one of the pairs we added, it must be one of the following three types: (1) $I_P(i'') = I_P(i)$ and $I_Q(j'') = I_Q(j)$, or (2) $I_P(i'') = I_P(i)$ but $I_Q(j'') > I_Q(j)$, or (3) $I_Q(j'') = I_Q(j)$ but $I_P(i'') > I_P(i)$. Here, we use the invariant to rule out pairs for which $I_P(i'') < I_P(i)$ or $I_Q(j'') < I_Q(j)$. However, in any matching M, there cannot be both pairs of type (2) and pairs of type (3), since any such two pairs would cross. Therefore, we conclude that all conflicting pairs either come from the interval $I_P(i)$ or they all come from the interval $I_Q(j)$, and in any case, there are only $w(s)$ of them. After this modification, we move on to the next index i' that is not a part of this interval $I_P(i)$ and is matched (in the new matching M) to some index j'—as before, this i' satisfies the invariant.

After we are done with all these modifications, we end up with a matching M of size at least $|X|$ in which complete intervals are aligned to each other. Now, we can define a matching M' between P and Q that contains all pairs $(I_P(i), I_Q(j))$ for which $(i,j) \in M$. In words, we contract the intervals of P', Q' to the original symbols of P, Q. Finally, M' corresponds to a common subsequence X' of P_1, P_2, and $W(X') = |M| \geq |X|$ since each matched interval corresponds to some symbol s and

contributes $w(s)$ matches to M and a single match of weight $w(s)$ to M'. □

□

4.2.2 Reducing the almost orthogonal vectors problem to LCS

We are now ready to present our reduction from the almost orthogonal vectors problem to the longest common subsequence problem.

We will proceed in two steps. First, we will show that WLCS is at least as hard as the almost orthogonal vectors problem. Second, given that the symbols in the constructed WLCS instance will have small weights, an application of Lemma 4.2.2 will allow as to conclude that LCS is at least as hard as the almost orthogonal vectors problem. Our alphabet will be $\Sigma = \{0, 1, 2, 3, 4, 5\}$.

We start with the reduction from the almost orthogonal vectors to WLCS. We define *coordinate gadget* sequence CG as follows. For an integer $x \in \{0, 1\}$ we set

$$\mathrm{CG}(x) \triangleq \begin{cases} 1\,0\,1 & \text{if } x = 0; \\ 1\,1 & \text{if } x = 1. \end{cases}$$

We set the weight function $w(0) \triangleq 1$ and $w(1) \triangleq l_0 \triangleq 10d$.

This gadget satisfy the following equalities:

$$\mathrm{WLCS}(\mathrm{CG}(x), \mathrm{CG}(x')) = \begin{cases} 2l_0 + 1 & \text{if } x \cdot x' = 0; \\ 2l_0 & \text{if } x \cdot x' = 1. \end{cases}$$

Now, we define the *vector gadgets* as a concatenation of the coordinate gadgets. For vectors $a, b \in \{0, 1\}^d$ we set

$$\mathrm{G}(a) \triangleq 2 \circ \bigcirc_{i \in [d]} \mathrm{CG}(a_i),$$

$$\mathrm{G}'(b) \triangleq \left(\bigcirc_{i \in [d]} \mathrm{CG}(b_i)\right) \circ 2.$$

The weight of the symbol of type 2 is $w(2) = l_1 - 1$, where $l_1 \triangleq d \cdot 2X + d - r$. It is now easy to prove the following two claims.

Claim 4.2.4. *If two vectors a and b, are r-orthogonal ($a \cdot b \leq r$), then*

$$\mathit{WLCS}(G(a), G'(b)) \geq l_1.$$

Proof. For each $i \in [d]$, match $\mathrm{CG}(a_i)$ with $\mathrm{CG}(b_i)$ optimally to get a weight at least $l_1 = r \cdot 2X + (d - r)(2X + 1)$. □

Claim 4.2.5. *If two vectors a and b, are not r-orthogonal ($a \cdot b \geq r + 1$), then*

$$\mathit{WLCS}(G(a), G'(b)) = l_1 - 1.$$

Proof. $\mathrm{WLCS}(\mathrm{G}(a), \mathrm{G}'(b)) \geq l_1 - 1$ is true because we can match the symbols of type 2, which gives cost $l_1 - 1$.

Now we prove that $\mathrm{WLCS}(\mathrm{G}(a), \mathrm{G}'(b)) \leq l_1 - 1$. If we match the symbols of type 2 symbols, then we cannot match any symbols which are not of type 2 and the inequality is true. Thus, we assume that the symbols of type 2 are not matched.

Now we can check that, if the symbols of type 1 in $\mathrm{G}(a)$ and $\mathrm{G}'(b)$ are not matched, then we cannot achieve weight $l_1 - 1$ even if we match all the other symbols (except for the symbols of type 2). Therefore, we assume that all the symbols of type 1 are matched. As there are at least $r+1$ coordinates where a and b both are 1 (the vectors are not r-orthogonal), by the construction of the coordinate gadgets it follows that $\mathrm{WLCS}(\mathrm{G}(a), \mathrm{G}'(b)) \leq (r+1) \cdot 2X + (d - (r+1))(2X+1) = l_1 - 1$ as required. □

Finally, we combine the vector gadgets into two sequences. Let $\hat{\mathrm{G}}(a) \triangleq 3 \circ \mathrm{G}(a) \circ 4$ and $\hat{\mathrm{G}}'(b) \triangleq 3 \circ \mathrm{G}(b) \circ 4 \circ 5$. Let b' be a vector consisting of d entries equal to 1, that is, $b'_i = 1$ for $i = 1, \ldots, d$.

$$P \triangleq 5^{|Q|} \circ \left(\bigcirc_{a \in A} \hat{\mathrm{G}}(a) \right) \circ 5^{|Q|},$$

$$Q \triangleq 5 \circ \left(\bigcirc_{i \in [N-1]} \hat{\mathrm{G}}'(b') \right) \circ \left(\bigcirc_{b \in B} \hat{\mathrm{G}}'(b) \right) \circ \left(\bigcirc_{i \in [N-1]} \hat{\mathrm{G}}'(b') \right).$$

Note that the number of symbols of type 3 in the first sequence P is defined in terms of the number of symbols in the second sequence Q. This is well defined because the second sequence does not use the construction of the first sequence.

And set the weights so that $w(5) \triangleq l_2 \triangleq l_1^2$ and $w(3) = w(4) \triangleq l_3 \triangleq l_2^2$. Additionally we define useful quantities $l' \triangleq 2l_3 + l_1 - 1$ and $l \triangleq Nl' + 2Nl_2$.

The following two lemmas prove that there is a gap in the WLCS score of our two sequences between cases when there is a pair of vectors that are r-orthogonal and when there is no such pair.

Lemma 4.2.6. *If there is a pair of vectors that are r-orthogonal, then $WLCS(P, Q) \geq l + 1$.*

Proof. Let a and b be a pair of vectors that are r-orthogonal. Let $i \in [N]$ be the index of the gadget $\hat{\mathrm{G}}(a)$ in P corresponding to the vector a. Match $\hat{\mathrm{G}}(a)$ and $\hat{\mathrm{G}}'(b)$ to get a weight of at least $2l_3 + l_1 \geq l' + 1$. Match the $i-1$ vector gadgets to the left of $\hat{\mathrm{G}}(a)$ to the $i-1$ corresponding vector gadgets immediately to the left of $\hat{\mathrm{G}}'(b)$, and similarly, match the $N - i$ gadgets to the right. The total additional weight we get is at least $(N-1)l'$. Finally, note that after the above matches, only $N-1$ out of the $3N-1$ symbols of type 3 in the sequence Q are surrounded by matched symbols. The remaining $2N$ symbols of type 3 can be matched, giving an additional weight of $2Nl_2$. The total weight is at least $(l'+1) + (N-1)l' + 2Nl_2 = l + 1$. □

Lemma 4.2.7. *If there are no pairs of vectors that are r-orthogonal, then the upper bound $WLCS(P, Q) \leq l$ holds.*

Proof. The main part of the proof will be dedicated to showing that if the N vector gadgets in P are matched to a substring of N' vector gadgets from Q, then N' must

be equal to N. This will follow since: if $N' < N$, then at least one of the symbols of type 3 or 4 in P will remain unmatched, and, if $N' > N$, then less than $2N$ of the symbols of type 3 in Q can be matched. The large weights we gave to symbols of type 3,4 and 5 make this impossible in an optimal matching. It will be easy to see that in any matching in which $N = N'$, the total weight is at most l.

Now, we introduce some notation. Let $1 \leq L, L'$ be two integers and define $Z(L, L')$ to be the maximum score of matching two sequence P', Q' where P' is composed of L vector gadgets $\hat{\mathrm{G}}(a)$ and Q' is composed of L' vector gadgets $\hat{\mathrm{G}}'(b)$, where no pair a, b is r-orthogonal. Define $Z_0(L, L')$ similarly, except that we restrict the matchings so that all symbols of type 3 and 4 in P' must be matched. In the following two claims we prove an upper bound on $Z(L, L')$ by first establishing an upper bound on $Z_0(L, L')$. Below we assume $1 \leq L \leq L'$; analogous proof shows that $Z_0(L, L') \leq L'l' + (L - L')(l_2 - 1)$ for $1 \leq L' \leq L$.

Claim 4.2.8. *For any integers $1 \leq L \leq L'$, we have upper bound $Z_0(L, L') \leq Ll' + (L' - L)(l_2 - 1)$.*

Proof. Let P', Q' be two sequences with L, L' vector gadgets $\hat{\mathrm{G}}, \hat{\mathrm{G}}'$, respectively. Consider an optimal matching of P' and Q' in which all the symbols of type 3 and 4 of P' are matched. Let Z_0' be the total score; our goal is to show that $Z_0' \leq Ll' + (L' - L)(l_2 - 1)$. Note that in such a matching, each vector gadget of P' must be matched completely within one or more vector gadget of Q', and each vector gadget of Q' has matches to at most one vector gadget from P' (otherwise, it must be the case that some 3 or 4 symbols in P' are not matched, which contradicts our assumption).

Let x be the number of vector gadgets of P' that contribute at most l' to the weight of our optimal matching. Note that any of the $L - x$ other vector gadgets must be matched to a substring of Q' that contains at least two vector gadgets for the following reason. The 3 and 4 symbols of the vector gadget of Q' must be matched, and, if the matching stays within a single vector gadget of Q' and has more than l' weight, then we have a pair which is r-orthogonal because of Claim 4.2.5. Thus, using the fact that there are only L' vector gadgets in Q', we get the condition,

$$x + 2(L - x) \leq L'. \tag{4.1}$$

We now give an upper bound on the weight of our matching, by summing the contributions of each vector gadget of P': there are x vector gadgets contributing $\leq l'$ weight, and there are $(L - x)$ vector gadgets matched to Q' with unbounded contribution, but we know that even if all the symbols of an vector gadget are matched, it can contribute at most $l'' \triangleq 2l_3 + (l_1 - 1) + d(2l_0 + 1)$. Therefore, the total weight of the matching can be upper bounded by

$$Z_0' \leq xl' + (L - x)l''. \tag{4.2}$$

We claim that no matter what $x \geq 0$ is, as Eqs. (4.1) and (4.2) hold, this expression is less than $Ll' + (L' - L)(l_2 - 1)$, which is what we need.

To maximize the right hand side of Eq. (4.2), we choose the smallest possible integer x that satisfies Eq. (4.1), since $l'' > l'$. This implies that $x = \max(0, 2L - L')$. A key inequality, which we will use multiple times in the proof, following from the fact that the symbols of type 3 and 4 are much more important than the rest, is that $l'' < l' + l_2 - 1$, which follows since $l'' - l' = d(2l_0 + 2) < 100d^2 < l_2$.

First, consider the case where $L \leq L'/2$, and therefore $x = 0$, which means that all the vector gadgets of P' might be fully matched. Using that $l'' < l' + l_2 - 1$ and that $L' - L \geq L'/2 \geq L$, we get the desired upper bound:

$$Z'_0 \leq Ll'' \leq L(l' + l_2 - 1) \leq Ll' + (L' - L)(l_2 - 1).$$

Now, assume that $L > L'/2$, and therefore $x = 2L - L'$. In this case the upper bound becomes:

$$Z'_0 \leq (2L - L')l' + (L' - L)l'' = Ll' + (L' - L)(l'' - l'),$$

which is less than $Ll' + (L' - L)(l_2 - 1)$, since $l'' < l' + l_2 - 1$. □

Next, we prove by induction that leaving the symbols of type 3 and 4 in sequence P' unmatched will only worsen the weight of the optimal matching. Below we assume $1 \leq L \leq L'$; analogous proof shows that $Z_0(L, L') \leq L'l' + (L - L')(l_2 - 1)$ for $1 \leq L' \leq L$.

Claim 4.2.9. *For any integers $1 \leq L \leq L'$, we have upper bound $Z(L, L') \leq Ll' + (L' - L)(l_2 - 1)$.*

Proof. We will prove by induction on integer $i \geq 2$ that: for all $L' \geq L \geq 1$ with $L + L' \leq i$, $Z(L, L') \leq Ll' + (L' - L)(l_2 - 1)$. The base case is when $i = 2$ and $L = L' = 1$. Then $Z(1, 1) = l'$ and we are done.

For the inductive step, assume that the statement is true for all $i' \leq i - 1$ and we will prove it for i. Let L, L' be so that $1 \leq L \leq L'$ and $L + L' = i$ and let P', Q' be sequences with L, L' vector gadgets, respectively. Consider the optimal matching between P' and Q'. Let Z' be the total score; our goal is to show that $Z' \leq Ll' + (L' - L)(l_2 - 1)$.

If every symbol of type 3 and 4 in P' is matched then, by definition, the weight cannot be more than $Z_0(L, L')$, and by Claim 4.2.8 we are done. Otherwise, consider the first unmatched symbol of type 3 or 4, call it s, and there are two cases.

Symbol s is of type 3. If s is the first symbol of type 3 in P', then the first symbol of type 3 in Q' must be matched to some symbol of type 3 after s (otherwise we can add this pair to the matching without violating any other pairs). This implies that none of the symbols in the vector gadget starting with the symbol s can be matched, since such matches will be in conflict with the pair containing the first symbol of type 3 in the sequence Q'. If s is not the first symbol of type 3 in P', consider the symbol of type 4 that appears right before s and note that it must be matched to some symbol s' of type 4 in Q', by our choice of symbol s as the first unmatched symbol of type 3 or 4. Now, there are two possibilities: either there are no more vector gadgets in Q' after the symbol s', or there is a symbol of type 3 right

after the symbol s' in Q' that is matched to a symbol of type 3 in P' that is after the symbol s (from a later vector gadget in P'). Note that in all considered cases above, the interval vector gadget starting at symbol s (and ending with the next symbol of type 4 after s) is completely unmatched in our matching. Therefore, in this case, we let P'' be the sequence with $L-1$ vector gadgets which is obtained from P' by removing the vector gadget starting with s. The weight of our matching will not change if we look at it as a matching between Q' and P'' instead of P', which implies that $Z' \leq Z(L-1, L')$. Using our inductive hypothesis we conclude that $Z' \leq (L-1)l' + (L'-L+1)(l_2-1) \leq Ll' + (L'-L)(l_2-1)$, since $l' > l_2$, and we are done.

Symbol s is of type 4. The symbol of type 3 at the start of the vector gadget that the symbol s is part of must have been matched to some symbol s' in Q'. Let symbol s'' be the symbol of type 4 at the end of the vector gadget that s' is part of. Note that the symbol s'' must be matched to some symbol s''' of type 4 in P' after s, since otherwise, we can add the pair that matches symbols s and s'' to the matching, gaining a cost of l_3, and the only possible conflicts we would create is with pairs containing a symbol between the symbols between s' and s'' or inside vector gadget containing s, and if we remove all such pairs, we would lose at most $2(l_1-1+d(2l_0+1))$ in the cost, which is much less than the gain of l_3—implying that our matching could not have been optimal. Therefore, there are $x \geq 2$ vector gadgets in P' that are matched to a single vector gadget in Q': all the vector gadgets starting at the symbol of type 3 right before s and ending at the symbol s''' are matched to the vector gadget starting with s' and ending with s''. We denote the subsequence of P' consisting of the x vector gadget by P'''. Let P'' be the reminder of P'—consisting of the prefix and the suffix. We denote the subsequence of Q' consisting of the single vector gadget by Q'''. Let Q'' be the reminder of Q'—consisting of the prefix and the suffix. By the above discussion, $Z' \leq \mathrm{WLCS}(P'', Q'') + \mathrm{WLCS}(P''', Q''')$. The contribution of the latter part to the weight of the matching can be at most the weight of all the symbols (that are not of type 5) in Q'''; this can be upper bounded by l''. By the inductive hypothesis, we know that any matching of P'' and Q'' can have weight at most $Z(L-c, L'-1) \leq (L-x)l' + (L'-1-L+x)(l_2-1)$. Summing up the two bounds on the contributions, we get that the total weight of the matching is at most:

$$\begin{aligned} Z' &\leq l'' + (L-x)l' + (L'-1-L+x)(l_2-1) \\ &\leq Ll' + (L'-L)(l_2-1) + (x-1)(l_2-1) + l'' - xl' \end{aligned}$$

However, note that $l'' < 1.1l'$ and that $(x-1)l_2 < 10(x-1.1)l_2 < (x-1.1)l'$, which implies that Z' can be upper bounded by $Ll' + (L'-L)(l_2-1)$, and we are done. □

We are now ready to complete the proof of the lemma. Consider the optimal matching of P and Q. Let s and s' be the first and last symbols of type 5 in Q that are not matched, respectively. Note that there cannot be any matched symbols of type 5 between s and s', since otherwise we could match either s or s' and gain extra weight without incurring any loss. Moreover, note that s cannot be the first symbol in Q and s' cannot be the last one, since those must be matched in an optimal

alignment. Let N' be the number of vector gadgets starting with the vector gadget that the symbol s is part of and ending with the vector gadget immediately right from the symbol s'. We have $2 \leq N' \leq 3N - 2$. We note that it cannot be that all symbols of type 5 are matched as the contribution from all such symbols and at most one pair of vector gadgets is less than what we could get by matching all symbols of type 3 and 4 from the first sequence.

We can now bound the total weight of the matching by the sum of the maximum possible contribution of these N' vector gadgets, and the contribution of the rest of Q. The prefix of Q that ends at the symbol of type 5 to the left of s and the suffix of Q that start at the symbol of type 5 to the right of s' can only contribute symbols of type 5 to the matching, and they contain exactly $(3N - 1 - (N' - 1))$ such symbols of type 5, giving a contribution of $(3N - N')l_2$. To bound the contribution of the N' vector gadgets, we use Claim 4.2.9: we obtain two sequences P', Q' composed of N, N' vector gadgets, respectively, in which no pair is r-orthogonal. The contribution of the P', Q' part, depends on N, N':

If $N' \leq N$, then by the above claims, the contribution of P' and Q' is at most $N'l' + (N - N')(l_2 - 1)$ and the total weight of our matching can be upper bounded by

$$(3N - N')l_2 + (N'l' + (N - N')(l_2 - 1)),$$

which is maximized when N' is as large as possible, since $l' > 2l_2 - 1$. Thus, setting $N' = N$, we get the upper bound $2Nl_2 + Nl' = l$, which is what we wanted.

Otherwise, if $N' > N$, then by the above claims, we get that the contribution is at most $Nl' + (N' - N)(l_2 - 1)$, and the total weight of our matching can be upper bounded by

$$(3N - N')l_2 + (Nl' + (N' - N)(l_2 - 1)) = Nl' + 2Nl_2 - (N' - N) < l.$$

□

To conclude our reduction, we note that the largest weight used in our weight function is polynomial in d, and therefore the reduction of Lemma 4.2.2 gives two unweighted sequences $f(P), f(Q)$ of length $d^{O(1)}n$, for which $\mathrm{LCS}(f(P), f(Q)) = \mathrm{WLCS}(P, Q)$.

4.3 Hardness for dynamic time warping

In this section we show conditional hardness for the problem of computing the dynamic time warping distance. We obtain the lower bound by a reduction from the almost orthogonal vectors problem. To simplify the argument, we do not do a direct reduction from the almost orthogonal vectors to the DTW problem. Instead, we show a reduction from the edit distance problem. In Chapter 3 (see Section 3.2.5) we already provided a conditional lower bound for the edit distance by a reduction from the almost orthogonal vectors. Below we show that there is a simple transformation f of sequences such that $\mathrm{edit}(P, Q) = \mathrm{DTW}(f(P), f(Q))$ for any two sequences P and Q. This implies

the following theorem.

Theorem 4.3.1. *If DTW can be computed in time $O(n^{2-\delta})$ for some $\delta > 0$ on two sequences of length n over an alphabet of size 8, then the almost orthogonal vectors problem with $A, B \subseteq \{0,1\}^d$ and $|A| = |B| = N$ can be solved in time $d^{O(1)}N^{2-\delta}$.*

We note that a quadratic lower bound for DTW between two sequences of symbols over an alphabet of size 8 implies a quadratic lower lower bound for DTW between two sequences of points from 8-dimensional ℓ_p space for any $p \geq 1$. This follows from the observation that we can choose 8 (appropriately scaled) vectors from the standard basis such that the distance between any two distinct vectors is 1.

In this section we prove Theorem 4.3.1. We start by defining the transformation f. Given an arbitrary sequence T over an alphabet Σ and a symbol $s \notin \Sigma$ that does not belong to the alphabet, let

$$f(T) \triangleq s\,T_1\,s\,T_2\,s\,\ldots\,s\,T_{|T|}s$$

be the transformation that inserts the symbol s before every symbol in T and also adds the symbol s at the end. Theorem 4.3.1 follows from Lemmas 4.3.2 and 4.3.3 below. Since in Section 3.2.5 we obtain sequences over an alphabet of size 7 and f introduces a new symbol, the final alphabet size for the hard DTW instances is 8.

Lemma 4.3.2. *For any two sequences P and Q over an alphabet Σ and $s \notin \Sigma$ we have the inequality*

$$\mathit{edit}(P,Q) \leq \mathit{DTW}(f(P), f(Q)).$$

Proof. We do the proof inductively on $i \triangleq |P| + |Q|$. We observe that if $|P| = 0$ or $|Q| = 0$, then we have the equality $\mathrm{edit}(P,Q) = \mathrm{DTW}(f(P), f(Q))$ and there is nothing to prove. This also deals with the base case $i = 1$. Thus, we assume that $|P|, |Q| \geq 1$. Fix an optimal traversal of P and Q that achieves the cost $\mathrm{DTW}(f(P), f(Q))$. It is equal to the total cost of all configurations during the traversal. We call a configuration *good* if both markers point to a symbol of type s. Suppose that there exists a good configuration during the traversal that aligns two symbols of type s and at least one of them do not start or end sequence P or Q. In this case we can write $P = P_1\,P_2$ and $Q = Q_1\,Q_2$ such that $i > |P_1| + |Q_1|$ and $i > |P_2| + |Q_2|$ and $\mathrm{DTW}(f(P), f(Q)) = \mathrm{DTW}(f(P_1), f(Q_1)) + \mathrm{DTW}(f(P_2), f(Q_2))$. We use the inductive assumption and prove the required inequality:

$$\begin{aligned}\mathrm{DTW}(f(P), f(Q)) &= \mathrm{DTW}(f(P_1), f(Q_1)) + \mathrm{DTW}(f(P_2), f(Q_2))\\ &\geq \mathrm{edit}(P_1, Q_1) + \mathrm{edit}(P_2, Q_2)\\ &\geq \mathrm{edit}(P, Q).\end{aligned}$$

It remains to consider the case in which all internal symbols of type s in sequences P and Q do not participate in any good configuration. The number of such internal symbols of type s is $(|P| - 1) + (|Q| - 1)$. These symbols contribute cost $|P| + |Q| - 2$ to the final DTW cost. Without loss of generality we assume that $|P| \geq |Q| \geq 1$. We consider two subcases.

Case $P_1 \neq Q_1$. That is, sequences P and Q start with different symbols. This implies that the second configuration in the traversal of $f(P)$ and $f(Q)$ will contribute cost 1 to the final DTW score. Thus, we can lower bound the DTW score by $(|P| + |Q| - 2) + 1 = |P| + |Q| - 1 \geq |P| \geq \text{edit}(P, Q)$ as required.

Case $P_1 = Q_1$. In this case we can lower bound the DTW score by $|P| + |Q| - 2 \geq |P| - 1 \geq \text{edit}(P, Q)$. □

Lemma 4.3.3. *For any two sequences P and Q over an alphabet Σ and $s \notin \Sigma$ we have the inequality*

$$\textit{edit}(P, Q) \geq \textit{DTW}(f(P), f(Q)).$$

Proof. Fix any two sequences P and Q and consider an optimal alignment between the sequences that achieves the edit distance cost $\text{edit}(P, Q)$. We will show how to transform the alignment into a traversal of the sequences $f(P)$ and $f(Q)$ that achieves the DTW cost $\text{edit}(P, Q)$. This is sufficient to show that $\text{edit}(P, Q) \geq \text{DTW}(f(P), f(Q))$ as the optimal traversal can only be cheaper.

Initially both markers are at the first symbols of $f(P)$ and $f(Q)$. We work through the alignment of P and Q from the beginning to the end. If the alignment performs a substitution or matches two equal symbols, we advance both markers by two symbols in $f(P)$ and $f(Q)$. If the alignment deletes a symbol from the first sequence, we advance the marker in $f(P)$ by two symbols. If the alignment inserts a symbol in the first sequence, we advance the marker in $f(Q)$ by two symbols. We can check the DTW cost is 1 in all cases except when we match two equal symbols, in which case the DTW cost is 0. This gives a traversal of $f(P)$ and $f(Q)$ with DTW cost $\text{edit}(P, Q)$. □

Chapter 5

Regular expressions

A regular expression (regexp) is a formula that describes a set of words over some alphabet Σ. It consists of individual symbols from Σ, as well as operators such as *OR* "|" (an alternative between several pattern arguments), *Kleene star* "$*$" (which allows 0 or more repetitions of the pattern argument), *Kleene plus* "+" (which allows 1 or more repetitions of the pattern argument), wildcard "." (which matches an arbitrary symbol), etc. For example, $[a|b]^+$ describes any sequence of symbols a and b of length at least 1. See Preliminaries for the formal definition.

In addition to being a fundamental notion in formal language theory, regular expressions are widely used in computer science to define search patterns. Formally, given a regular expression (pattern) P of size m and a sequence of symbols (text) T of length n, the goal of regular expression matching is to check whether a substring of T can be derived from P. A closely related problem is that of membership testing where the goal is to check whether the text T itself can be derived from P. Regular expression matching and membership testing are widely used computational primitives, employed in several programming languages and text processing utilities such as Perl, Python, JavaScript, Ruby, AWK, Tcl and Google RE2. Apart from text processing and programming languages, regular expressions are used in computer networks [KDY+06], databases and data mining [GRS99], computational biology [NR03], human-computer interaction [KHDA12] etc.

A classic algorithm for both problems constructs and simulates a non-deterministic finite automaton corresponding to the expression, resulting in the "rectangular" $O(mn)$ running time. A sequence of improvements, first by Myers [Mye92] and then by [BT09], led to an algorithm that achieves roughly $O(mn/\log^{1.5} n)$ running time. The latter result constitutes the fastest algorithm for this problem known to date, despite an extensive amount of research devoted to this topic. The existence of faster algorithms is a well-known open problem ([Gal85], Problem 4).

However, significantly faster algorithms are known for various well-studied special cases of regular expressions. For example:

1. If the pattern is a concatenation of symbols (i.e., we search for a specific sequence of symbols in the text), the pattern matching problem corresponds to the "standard" string matching problem and can be solved in linear time, e.g., using the Knuth-Morris-Pratt algorithm [KMP77].

2. If the pattern is of the form $P_1|P_2|\ldots|P_k$ where P_i are sequences of symbols, then the pattern matching problem corresponds to the *dictionary matching* problem that can be solved in linear time using the Aho-Corasick algorithm [AC75].

3. If the pattern is a concatenation of symbols and single character wildcards "." matching any symbol, the pattern matching problem is known as the *wildcard matching* and can solved in (deterministic) $O(n \log m)$ time using convolutions [FP74, Ind98, Kal02, CH02].

4. More generally, if the pattern is a concatenation of *single* character ORs of the form $s_1|s_2|\ldots|s_k$ for $s_1, \ldots, s_k \in \Sigma$ (e.g., $[a|b][a|c][b|c]$), the pattern matching problem is known as *superset matching* and can be solved in (deterministic) $O(n \log^2 m)$ time [CH97, CH02].

5. Finally, if the goal is to test whether a text T can be derived from a pattern P of the form $(P_1|P_2|\ldots|P_k)^+$ where P_i are sequences of symbols, the problem is known as the *word break* problem. It is a popular interview question [Tun11, Lee], and the known solutions can be implemented to run in $\sqrt{mn}\log^{O(1)} n$ time.

The first two examples were already mentioned in [Gal85] as a possible reason why a faster algorithm for the general problem might be possible. Despite the existence of such examples, any super-poly-logarithmic improvements to the algorithms of [Mye92, BT09] in the general case remain elusive. Furthermore, we are not aware of any systematic classification of regular expressions into "easy" and "hard" cases for the pattern matching and membership testing problems. The goal of this part of the thesis is to address this gap.

Our results. Our main result is a classification of the computational complexity of regular expression matching and membership checking for patterns that involve operators "|", "+", "$*$" and concatenation "$\circ$", based on the pattern *depth*. Our classification enables us to distinguish between the cases that are solvable in sub-quadratic time (including the five problems listed above) and the cases that do not have strongly sub-quadratic time algorithms under SETH. Our results therefore demonstrate a non-trivial dichotomy for the complexity of these problems.

To formulate our results, we consider pattern formulas that are *homogeneous*, i.e., in which the operators at the same level of the formula are equal (note that the five aforementioned problems involve patterns that satisfy this condition). We say that a homogeneous formula of depth k has *type* $o_1 o_2 \ldots o_k$ if for all levels i the operators at level i are equal to o_i (note that, in addition to the operators, a level might also contain leaves, i.e., symbols; for example, the expression $[a|b]a[b|c]$ is a depth-2 formula of type "$\circ|$"). We will assume that no two consecutive operators in the type descriptor are equal, as otherwise they can be collapsed into one operator.

Our results are described in Table 5.1 (for depth-2 expressions) and Table 5.2 (for depth-3 expressions). The main findings can be summarized as follows:

1. Almost all pattern matching and membership problems involving depth-2 expressions can be solved in near-linear time. The lone exception involve patterns of type "$\circ*$", for which we show that matching and membership problems cannot be solved in time $O((mn)^{1-\delta})$ for any constant $\delta > 0$ and $m \leq n$ assuming SETH. Interestingly, we show that pattern matching with a very similar depth-2 type, namely "$\circ+$", can be solved in $O(n \log^2 m)$ time.

2. Pattern matching problems with depth-2 expressions contain a "high density" of interesting algorithmic problems, with non-trivial algorithms existing for types "$\circ+$" (our result), "$\circ|$" [CH02] , "$|\circ$" [AC75] and "$+\circ$" (essentially solved in [KMP77], since + can be dropped). In contrast, membership problems with depth-2 expressions have a very restrictive structure that makes them mostly trivially solvable in linear time, with the aforementioned exception for the "$\circ*$" type.

3. Pattern matching problems with depth-3 expressions have a more diversified structure. All types starting with $\circ$ are SETH-hard; all types starting with $|$ are either-SETH hard (if followed by $\circ$) or easily solvable in linear time; all types starting from $*$ are trivially solvable in linear time (since $*$ allows zero repetitions); all types starting from + inherit their complexity from the last two operators in the type description (since + allows exactly one repetition).

4. Finally, membership checking with depth-3 expressions presents the most complex picture. As before, all types starting with $\circ$ are SETH-hard. On the other hand, types starting with $|$ (except for $|\circ*$) have linear time algorithms, with difficulty levels that range from trivial observations to undergraduate-level exercises. Types starting with $*$ or + include "$*|\circ$" and "$+|\circ$", which correspond to the aforementioned word break problem [Tun11, Lee]. This is the only problem in the table whose (conditional) complexity is not determined up to logarithmic factors. However, we show that the running time of the standard dynamic-programming based algorithm can be improved, from roughly $nm^{0.5}$ to roughly $nm^{0.5-1/18}$. This runtime has been further improved in a follow-up work [BGL17].

Our techniques. Our upper bounds for depth-2 expressions follow either from known near-linear time algorithms for specific variants of regular expressions, or relatively simple constructions of such algorithms. In particular, we observe that type "$\circ|$" expressions (concatenations of ORs) correspond to superset matching, type "$|\circ$" expressions (OR of sequences) correspond to dictionary matching and type "$+\circ$" reduces to "$\circ$" and thus corresponds to the standard pattern matching problem. Furthermore, we give a near-linear time algorithm for pattern matching with type "$\circ+$" expressions, where patterns are concatenations of expressions of the form $s^{\geq k}$ or s^k, where $s^{\geq k}$ denotes a sequence of symbols s repeated at least $k \geq 1$ times.[1] We show that this

[1]For example, the expression aa^+bc^+ generates all words of the form $a^{\geq 2}b^1c^{\geq 1}$.

Type	Example	Pattern matching	Membership
$\circ+$	a^+ab^+	$O(n \log^2 m)$ (Section 5.4.2)	$O(n+m)$ (immediate)
$\circ*$	a^*ab^*	$\Omega((mn)^{1-\alpha})$ (Section 5.2.3)	$\Omega((mn)^{1-\alpha})$ (Section 5.3.1)
$\circ\|$	$[a\|b][b\|c]$	$O(n \log^2 m)$ (**superset matching** [CH02])	$O(n+m)$ (immediate)
$\|\circ$	$ab\|c$	$O(n+m)$ (**dictionary matching** [AC75])	$O(n+m)$ (immediate)
$\|*$	$a^*\|a\|b^*$	$O(n+m)$ (immediate)	$O(n+m)$ (immediate)
$\|+$	$a^+\|a\|b^+$	$O(n+m)$ (reducible to "|")	$O(n+m)$ (immediate)
$*\circ$	$[ab]^*$	$O(n+m)$ (immediate)	$O(n+m)$ (immediate)
$*+$	$[a^+]^*$	$O(n+m)$ (immediate)	$O(n+m)$ (reducible to "+")
$*\|$	$[a\|b]^*$	$O(n+m)$ (immediate)	$O(n+m)$ (immediate)
$+\circ$	$[ab]^+$	$O(n+m)$ (**string matching** [KMP77])	$O(n+m)$ (equivalent to "$*\circ$")
$+\|$	$[a\|b]^+$	$O(n+m)$ (reducible to "|")	$O(n+m)$ (equivalent to "$*\|$")
$+*$	$[a^*]^+$	$O(n+m)$ (immediate)	$O(n+m)$ (reducible to "+")

Table 5.1: Classification of the complexity of the pattern matching and the membership test problems for depth-2 expressions. All lower bounds assume SETH and $m \leq n$. Some upper bounds use randomization (notably hashing).

problem can be solved in near-linear time by reducing it to one instance of subset matching and one instance of wildcard matching. All other problems can be solved in linear time, with the exception of type "$\circ*$". The latter expressions correspond to patterns obtained by concatenating patterns of the form $s^{\geq k}$ and s^k. Unlike in the "$\circ+$" case, however, here we cannot assume that $k \geq 1$, since each symbol could be repeated zero times. We show that this simple change makes the problem SETH-hard. This is accomplished by a reduction from the unbalanced version of the orthogonal vectors problem, see the preliminaries (Chapter 2).[2]

Our results for depth-3 expression pattern matching are multi-fold. First, all types starting from $*$ are trivially solvable in linear time, since $*$ allows zero repetitions. Second, all types starting from $+$ inherit their complexity from the last two operators in the type description, since $+$ allows exactly one repetition. Third, all types starting from $|*$ or $|+$ have simple linear time solutions.

The remaining cases lead to SETH-hard problems. For six types this follows immediately from the analogous result for type "$\circ*$". For the six remaining types the hardness is shown via individual reductions from the orthogonal vectors problem. For some types, such a reduction is immediate. For example, for type "$|\circ|$" expressions (ORs of concatenations of ORs), we form the text by concatenating all vectors in B (separated by some special symbol), and we form the pattern by taking an OR of the vectors in A, modified by replacing 0 with [0|1] and 1 with 0. A similar approach works for type "$|\circ+$" expressions.

The remaining four types are grouped into two classes: "$\circ+\circ$" is grouped with

[2]The reduction is somewhat complex, so we will not outline it here. However, we give an overview of other reductions from the orthogonal vectors problem in the next few paragraphs.

Type	Example	Pattern matching	Membership
$\circ\|\circ$	$[a\|bb][ba\|b]$	$\Omega((mn)^{1-\alpha})$ (Section 5.2.5)	$\Omega((mn)^{1-\alpha})$ (Section 5.3.3)
$\circ\|*$	$[a^*\|b^*][c^*\|b]$	$\Omega((mn)^{1-\alpha})$ (from "$\circ*$")	$\Omega((mn)^{1-\alpha})$ (from "$\circ*$")
$\circ\|+$	$[a^+\|b^+][c^+\|b]$	$\Omega((mn)^{1-\alpha})$ (Section 5.2.7)	$\Omega((mn)^{1-\alpha})$ (Section 5.3.5)
$\circ+\circ$	$[ab]^+[bca]^+$	$\Omega((mn)^{1-\alpha})$ (Section 5.2.4)	$\Omega((mn)^{1-\alpha})$ (Section 5.3.2)
$\circ+\|$	$[a\|b]^+[a\|c\|d]^+$	$\Omega((mn)^{1-\alpha})$ (Section 5.2.6)	$\Omega((mn)^{1-\alpha})$ (Section 5.3.4)
$\circ+*$	$[a][a^+]^*[b^+]$	$\Omega((mn)^{1-\alpha})$ (from "$\circ*$")	$\Omega((mn)^{1-\alpha})$ (from "$\circ*$")
$\circ*\circ$	$[ab]^*[bca]^*$	$\Omega((mn)^{1-\alpha})$ (from "$\circ*$")	$\Omega((mn)^{1-\alpha})$ (from "$\circ*$")
$\circ*\|$	$[a\|b]^*[a\|b\|c]^*$	$\Omega((mn)^{1-\alpha})$ (from "$\circ*$")	$\Omega((mn)^{1-\alpha})$ (from "$\circ*$")
$\circ*+$	$[a^*]b[b^+]^*$	$\Omega((mn)^{1-\alpha})$ (from "$\circ*$")	$\Omega((mn)^{1-\alpha})$ (from "$\circ*$")
$\|\circ\|$	$[(a\|b)(b\|c)]\|b$	$\Omega((mn)^{1-\alpha})$ (Section 5.2.1)	$O(n+m)$ (immediate)
$\|\circ*$	$[a^*b^*]\|[b^*c^*]$	$\Omega((mn)^{1-\alpha})$ (from "$\circ*$")	$\Omega((mn)^{1-\alpha})$ (from "$\circ*$")
$\|\circ+$	$[a^+b^+]\|[b^+c^+]$	$\Omega((mn)^{1-\alpha})$ (Section 5.2.2)	$O(n+m)$ (Section 5.4.4)
$\|*\circ$	$[abc]^*\|[bc]^*$	$O(n+m)$ (reducible to "$\|$")	$O(n+m)$ (Section 5.4.3)
$\|*\|$	$[a\|b\|c]^*\|[b\|c]^*$	$O(n+m)$ (reducible to "$\|$")	$O(n+m)$ (immediate)
$\|*+$	$[a^+]^*\|[b^+]^*$	$O(n+m)$ (reducible to "$\|$")	$O(n+m)$ (immediate)
$\|+\circ$	$[abc]^+\|[bc]^+$	$O(n+m)$ (reducible to "$\|\circ$")	$O(n+m)$ (Section 5.4.3)
$\|+\|$	$[a\|b\|c]^+\|[b\|c]^+$	$O(n+m)$ (reducible to "$\|$")	$O(n+m)$ (same as "$\|*\|$")
$\|+*$	$[a^*]^+\|[b^*]^+$	$O(n+m)$ (reducible to "$\|$")	$O(n+m)$ (immediate)
$*\circ\|$	$[[a\|b][b\|c]]^*$	$O(n+m)$ (immediate)	$O(n+m)$ (immediate)
$*\circ*$	$[a^*b^*c^*]^*$	$O(n+m)$ (immediate)	$\Omega((mn)^{1-\alpha})$ (from "$\circ*$")
$*\circ+$	$[a^+b^+c^+]^*$	$O(n+m)$ (immediate)	$O(n+m)$ (Section 5.4.4)
$*\|\circ$	$[a\|ab\|bc]^*$	$O(n+m)$ (immediate)	$O(nm^{0.44\ldots})$ (**word break**—Section 5.4.1)
$*\|*$	$[a^*\|b^*\|c^*]^*$	$O(n+m)$ (immediate)	$O(n+m)$ (immediate)
$*\|+$	$[a^+\|b^+\|c^+]^*$	$O(n+m)$ (immediate)	$O(n+m)$ (immediate)
$*+\circ$	$[[abcd]^+]^*$	$O(n+m)$ (immediate)	$O(n+m)$ (immediate)
$*+\|$	$[[a\|b\|c\|d]^+]^*$	$O(n+m)$ (immediate)	$O(n+m)$ (immediate)
$*+*$	$[[a^*]^+]^*$	$O(n+m)$ (immediate)	$O(n+m)$ (immediate)
$+\circ\|$	$[[a\|b][b\|c]]^+$	$O(n\log^2 m)$ (reducible to "$\circ\|$")	$O(n+m)$ (same as "$*\circ\|$")
$+\circ*$	$[a^*b^*c^*]^+$	$\Omega((mn)^{1-\alpha})$ (from "$\circ*$")	$\Omega((mn)^{1-\alpha})$ (same as "$*\circ*$")
$+\circ+$	$[a^+b^+c^+]^+$	$O(n\log^2 m)$ (reducible to "$\circ+$")	$O(n+m)$ (same as "$*\circ+$")
$+\|\circ$	$[a\|ab\|bc]^+$	$O(n+m)$ (reducible to "$\|\circ$")	$O(nm^{0.44\ldots})$ (**word break**—Section 5.4.1)
$+\|*$	$[a^*\|b^*\|c^*]^+$	$O(n+m)$ (reducible to "$\|*$")	$O(n+m)$ (same as "$*\|*$")
$+\|+$	$[a^+\|b^+\|c^+]^+$	$O(n+m)$ (reducible to "$\|+$")	$O(n+m)$ (same as "$*\|+$")
$+*\circ$	$[[abcd]^*]^+$	$O(n+m)$ (reducible to "$*\circ$")	$O(n+m)$ (same as "$*\circ$")
$+*\|$	$[[a\|b\|c\|d]^*]^+$	$O(n+m)$ (reducible to "$*\|$")	$O(n+m)$ (same as "$*\|$")
$+*+$	$[[a^+]^*]^+$	$O(n+m)$ (reducible to "$*+$")	$O(n+m)$ (same as "$*+$")

Table 5.2: Classification of the complexity of the pattern matching and the membership test problems for depth-3 expressions. See Fig. 5-1 for the visualization of the table.

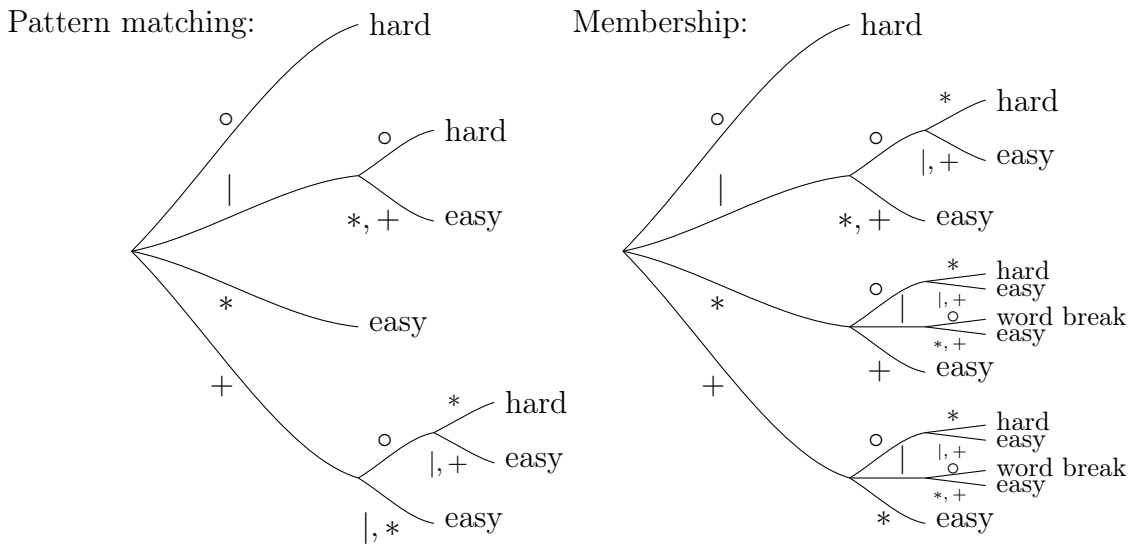

Figure 5-1: Tree diagrams visualizing Table 5.2. Depth-3 types are classified as "easy" (near-linear time), "hard" (near-quadratic time, assuming SETH), or "word break" (whose complexity is not determined). The leftmost operators in each tree correspond to the leftmost operators in type descriptions.

"∘|∘" and "∘+|" is grouped with "∘|+". For each group, we first show hardness of the first type in the group (i.e., of "∘+∘" and "∘+|", respectively). We then show that the second type in each group is hard by making changes to the hardness proof for the first type.

The hardness proof for "∘+∘" proceeds as follows. We form the pattern P by concatenating (appropriately separated) pattern vector gadgets for each vector in A, and form the text T by concatenating (appropriately separated) text vector gadgets for each vector in B. We then show that if there is a pair of orthogonal vectors $a^i \in A, b^j \in B$ then P can be matched to a substring of T, and vice versa. To show this, we construct P and T so that any pair of gadgets (in particular the gadgets for a^i and b^j) can be aligned. We then show that (i) each vector gadget for a vector in A can be matched with "most" of the gadget for the corresponding $b \in B$ (ii) matching the gadgets for *orthogonal* vectors a^i and b^j allows us to make a "smaller step", i.e., to match the gadget for a^i with a smaller part of the gadget for b^j, and (iii) at least one "smaller step" is necessary to completely derive a substring of T from P. We then conclude that there is a pair of orthogonal vectors $a^i \in A, b^j \in B$ if and only if P can be matched to a substring of T. The hardness proof for "∘+|" follows a similar general approach, although the technical development is different. In particular, we construct the gadgets such that the existence of orthogonal vectors makes it possible to make a "bigger step", i.e., to derive a bigger part of T, and that one bigger step is necessary to complete the derivation.

To show hardness of the second type in each group, we adapt the arguments for the first type in the group. In particular, to show hardness for type "∘|∘" , we construct P and T as in the reduction for type "∘+∘" and then transform P into a

type "$\circ|\circ$" regular expression P'. The transformation has the property that P' is *less* expressive than P (i.e., the language corresponding to P is a *superset* of the language corresponding to P'), but the specific substrings of the text T needed for the reduction can be still derived from P'. The hardness proof for "$\circ|+$" is obtained via a similar transformation of the hardness proof for "$\circ+|$".

Finally, consider the membership checking problem for depth-3 expressions. As before, all types starting with $\circ$ are shown to be SETH-hard. The reductions are similar to those for the pattern matching problem, but in a few cases require some modifications. On the other hand, types starting with $|$ (with the exception of $|\circ*$) have linear time algorithms. The algorithms are not difficult, but require the use of basic algorithmic notions, such as periodicity (for types "$|*\circ$" and "$|+\circ$") and run-length encoding (for type "$|\circ+$"). Types starting with $*$ are mostly solvable in linear time, with two exceptions: type "$*\circ*$" inherits the hardness from "$\circ*$",[3] while the type "$*|\circ$" corresponds to the aforementioned word break problem which we discuss in the next paragraph. Finally, types starting with $+$ are analogous to those starting with $*$.

The word break problem is the only problem in the table whose (conditional) complexity is not determined up to logarithmic factors. There are several known solutions to this problem based on dynamic programming [Tun11, Lee]. A careful implementation of those algorithms (using substring hashing and pruning) leads to a runtime of $O(nm^{0.5}\log^{O(1)} n)$. However, we show that this bound is not tight, and can be further improved to roughly $O(nm^{0.5-1/18})$. Our new algorithm speeds up the dynamic program by using convolutions to pre-compute information that is reused multiple times during the execution of the algorithm. We note that the algorithm is randomized and has a one-sided error.

Related work. Our hardness results come on the heels of several recent works demonstrating quadratic hardness of sequence alignment problems assuming SETH or other conjectures. The technical development in this chapter is, however, quite different, since regular expression matching is not defined by a sequence similarity measure. Instead, our gadget constructions are tailored to the specific sets of operators and expression types defining the problem variants. Furthermore, we exploit the similarity between related expression types (such as "$\circ+\circ$" and "$\circ|\circ$") and show how to convert a hardness proof for one type into a hardness proof for the other type.

The reduction in Section 5.2.1 has been independently discovered by Kasper Larsen and Raphael Clifford (personal communication). Conditional lower bounds (via reductions from 3SUM) for certain classes of regular expressions have been investigated in [AKL$^+$16]. Estimating the complexity of regular expression matching using *specific* algorithms has also been a focus of several papers. See, e.g., [WvdMBW16] and the references therein. The work [BGL17] extends the classification to higher depth

[3]Let a regular expression P and a text T be a hard instance for the "$\circ*$" membership problem. Let s be a symbol that does not appear in P or T. Then $P' \triangleq (s \circ P \circ s)^*$ and $T' \triangleq s\,T\,s$ is a hard instance for the membership problem of type "$*\circ*$". Since T' starts and ends with the unique symbol, we must use the argument regular expression $s \circ P \circ s$ exactly once. Thus we get "$\circ*$" membership problem.

regular expressions as well as provided more general and faster algorithms. See also the discussion at the beginning of Section 5.4.1. In [ARW17] the authors show conditional lower bounds for approximately solving the membership problem. This is further strengthened in [CGL$^+$18]. The later work also shows circuit lower bound consequences for solving the matching and membership problems faster.

5.1 Preliminaries

Subset matching problem. In the subset matching problem, we are given a pattern string P and a text string T where each pattern and text location is a set of symbols drawn from some alphabet. The pattern is said to occur at the text position i if the set P_j is a subset of the set T_{i+j} for all j. The goal of the problem is find all positions where P occurs in T. The problem can be solved in deterministic $O(N \log^2 N)$ time, where $N \triangleq \sum_i |T_i| + \sum_i |P_i|$ [CH02].

Superset matching problem. This problem is analogous to subset matching except that we require that P_j is a *superset* of the set T_{i+j} for all j. The aforementioned algorithm of [CH02] applies to this problem as well.

Wildcard matching problem. In the wildcard matching problem, we are given a pattern string P and a text string T where each pattern and text location is an element from $\Sigma \cup \{.\}$, where "." is the special wildcard symbol. The pattern is said to occur at the text position i if for all j we have that (i) one of the symbols P_j and T_{i+j} is equal to ".", or (ii) $P_j = T_{i+j}$. The goal of the problem is find all positions where P occurs in T. The problem can be solved in deterministic $O(n \log n)$ time [CH02].

Regular expressions. Regular expressions over a symbol set Σ and an operator set $O \triangleq \{\circ, |, +, *\}$ are defined recursively as follows:

- a is a regular expression, for any $a \in \Sigma$;
- if R and S are regular expressions then so are $[R]|[S]$, $[R] \circ [S]$, $[R]^+$ and $[R]^*$.

For the sake of simplicity, in the rest of this section we will typically omit the concatenation operator $\circ$, and also omit some of the parenthesis if the expression is clear from the context.

A regular expression P determines a language $L(P)$ over Σ. Specifically, for any regular expressions R, S and any $a \in \Sigma$, we have: $L(a) = \{a\}$; $L(R|S) = L(R) \cup L(S)$; $L(R \circ S) = \{uv : u \in L(R), v \in L(S)\}$; $L(R^+) = \cup_{i \geq 1} \bigcirc_{j=1}^{i} L(R)$; and $L(R^*) = L(R^+) \cup \{\epsilon\}$, where ϵ denotes the empty sequence.

Regular expressions can be viewed as rooted labeled trees, with internal nodes labeled with operators from O and leaves labeled with symbols from Σ. Note that the number of children of an internal node is not fixed, and can range between 1 (for $+$ and $*$) and m. We say that a regular expression is *homogeneous* if all internal node labels at the same tree level are equal. Note that this definition does not preclude

expressions such as aa^+ where not all leaves have the same depth. A homogeneous formula of depth k has *type* $o_1 o_2 \dots o_k$, $o_i \in O$, if for all levels i the operators at level i are equal to o_i. For example, aa^+ has type "$\circ+$".

To state our results, it is convenient to identify depth-3 homogeneous regular expressions that are equivalent to some depth-2 regular expressions. In particular, in all types starting from $+$, the operator $+$ can be removed, since P^+ occurs in the text T if and only if P occurs in T. Similarly, in all types starting from $|+$, the operator $+$ can be removed, for the same reasons.

Notation. Given an integer d, 0_d (1_d, resp.) denotes the vector with all entries equal to 0 (1, resp.) in d dimensions.

Simplifying assumptions. To simplify our proofs, we will make several simplifying assumptions about the orthogonal vectors instance (different assumptions are used in different proofs). In what follows, we describe what kind of assumptions we make, and how to satisfy them:

1. The number of vectors M in set A is odd or even (depending on the proof): this can be achieved without loss of generality since we can add a vector to the set A consisting only of 1s.
2. The dimensionality d of vectors from sets A and B is odd or even (depending on the proof): this can be achieved without loss of generality since we can add new coordinate to every vector and set this coordinate to 0.
3. Let $A \triangleq \{a^1, \dots, a^M\}$ and $B \triangleq \{b^1, \dots, b^N\}$. If there are i, j such that $a^i \cdot b^j = 0$, then there are i', j' such that $a^{i'} \cdot b^{j'} = 0$ and $i' = j' \pmod 2$: this can be assumed without loss of generality since we can define a set $A' \triangleq \{a^M, a^1, a^2, a^3, \dots, a^{M-1}\}$ and perform two reductions, one on the pair of sets A and B and another one on the pair of sets A' and B. If a pair of orthogonal vectors exists, one of these reductions will detect it.
4. The dimensionality d is greater than 100: we can assume this since otherwise Orthogonal Vectors problem can be solved in $O(2^d N) = O(N)$ time.
5. d is odd and $b_1^t = b_d^t = 0$ for all $t \in [N]$: first, we make d odd as described above. Then we add two entries for every vector from A or B, one at the beginning and one at the end, and set both entries to 0.
6. The first vector a^1 from the set A is not orthogonal to all vectors from B: first, we can detect whether this is the case in $O(dN)$ time. If a^1 is orthogonal to a vector from B, we have found a pair of orthogonal vectors. Otherwise we proceed with the reduction.

5.2 Reductions for the pattern matching problem

We start this section by showing hardness for regular expressions of type "$|\circ|$" and "$|\circ+$". These hardness proofs are quite simple, and will help us introduce notation

used in more complex reductions presented later. Next we present the hardness proof for regular expressions of type "$\circ*$". After that we present hardness proofs for the other cases.

5.2.1 Hardness for type "$|\circ|$"

Theorem 5.2.1. *Given sets $A \triangleq \{a^1, \ldots, a^M\} \subseteq \{0,1\}^d$ and $B \triangleq \{b^1, \ldots, b^N\} \subseteq \{0,1\}^d$, we can construct a regular expression P and a sequence of symbols T, in $O(Nd)$ time, such that a substring of T can be derived from P if and only if there are $a \in A$ and $b \in B$ such that $a \cdot b = 0$. Furthermore, P has type "$|\circ|$", $|P| \leq O(Md)$ and $|T| \leq O(Nd)$.*

Proof. First, we will construct our pattern P. For an integer $v \in \{0,1\}$, we construct the following pattern coordinate gadget

$$\mathrm{CG}(v) \triangleq \begin{cases} [0|1] & \text{if } v = 0; \\ [0] & \text{if } v = 1. \end{cases}$$

For a vector $a \in \{0,1\}^d$, we define a pattern vector gadget

$$\mathrm{G}(a) \triangleq \mathrm{CG}(a_1)\,\mathrm{CG}(a_2)\,\mathrm{CG}(a_3)\,\ldots\,\mathrm{CG}(a_d).$$

Our pattern P is then defined as "$|$" of all pattern vector gadgets:

$$P \triangleq \mathrm{G}\left(a^1\right) | \mathrm{G}\left(a^2\right) | \mathrm{G}\left(a^3\right) | \mathrm{G}\left(a^4\right) | \ldots | \mathrm{G}\left(a^{M-1}\right) | \mathrm{G}\left(a^M\right).$$

Now we construct the text T. First, for a vector $b \in \{0,1\}^d$, we define text vector gadget as concatenation of all entries of b: $\mathrm{G}'(b) \triangleq b_1\, b_2\, b_3\, b_4\, \ldots\, b_d$. Note that we can derive $\mathrm{G}'(b)$ from $\mathrm{G}(a)$ if and only if $a \cdot b = 0$. Our text T is defined as a concatenation of all text vector gadgets with a symbol of type 2 in between any two neighboring vector gadgets:

$$T \triangleq \mathrm{G}'\left(b^1\right)\, 2\, \mathrm{G}'\left(b^2\right)\, 2\, \mathrm{G}'\left(b^3\right)\, 2\, \mathrm{G}'\left(b^4\right)\, 2\, \ldots\, 2\, \mathrm{G}'\left(b^{N-1}\right)\, 2\, \mathrm{G}'\left(b^N\right).$$

We need to show that we can derive a substring of T from P if and only if there are two orthogonal vectors in A and B. This follows from Lemmas 5.2.2 and 5.2.3 below. □

Lemma 5.2.2. *If there are two vectors $a \in A$ and $b \in B$ that are orthogonal, then a substring of T can be derived from P.*

Proof. Suppose that $a^i \cdot b^j = 0$ for some $i \in [M]$, $j \in [N]$. We choose a pattern vector gadget $\mathrm{G}(a^i)$ from the pattern P and transform it into a text vector gadget $\mathrm{G}'(b^j)$. This is possible because of the orthogonality and the construction of the vector gadgets. □

Lemma 5.2.3. *If a substring of T can be derived from P, then there are two orthogonal vectors.*

Proof. By the construction of pattern P, we have to choose one pattern vector gadget, say, $\mathrm{G}(a^i)$, that is transformed into a binary substring of T of length d. The text T has the property that it is a concatenation of binary strings of length d separated by symbols 2. This means that $\mathrm{G}(a^i)$ will be transformed into binary string $\mathrm{G}'(b^j)$ for some j. This implies that $a^i \cdot b^j = 0$ by the construction of the vector gadgets. □

5.2.2 Hardness for type "|∘+"

Theorem 5.2.4. *Given sets $A \triangleq \{a^1, \ldots, a^M\} \subseteq \{0,1\}^d$ and $B \triangleq \{b^1, \ldots, b^N\} \subseteq \{0,1\}^d$, we can construct a regular expression P and a sequence of symbols T, in $O(Nd)$ time, such that a substring of T can be derived from P if and only if there are $a \in A$ and $b \in B$ such that $a \cdot b = 0$. Furthermore, P has type "|∘+", $|P| \leq O(Md)$ and $|T| \leq O(Nd)$.*

Proof. First, we will construct our pattern. For an integer $v \in \{0,1\}$, we construct the following pattern coordinate gadget

$$\mathrm{CG}(v) \triangleq \begin{cases} x^+ & \text{if } v = 0; \\ x^+\, x^+ & \text{if } v = 1. \end{cases}$$

For a vector $a \in \{0,1\}^d$, we define a pattern vector gadget as concatenation of coordinate gadgets for all coordinates with a symbol of type y in between every two neighboring coordinate gadgets:

$$\mathrm{G}(a) \triangleq \mathrm{CG}(a_1)\, y\, \mathrm{CG}(a_2)\, y\, \mathrm{CG}(a_3)\, y\, \ldots\, y\, \mathrm{CG}(a_d).$$

Our pattern P is then defined as an OR ("|") of all the pattern vector gadgets:

$$P \triangleq \mathrm{G}\left(a^1\right) \mid \mathrm{G}\left(a^2\right) \mid \mathrm{G}\left(a^3\right) \mid \mathrm{G}\left(a^4\right) \mid \ldots \mid \mathrm{G}\left(a^{M-1}\right) \mid \mathrm{G}\left(a^M\right).$$

Now we proceed with the construction of our text T. For an integer $v \in \{0,1\}$, we define the following text coordinate gadget

$$\mathrm{CG}'(v) \triangleq \begin{cases} xx & \text{if } v = 0; \\ x & \text{if } v = 1. \end{cases}$$

For vector $b \in \{0,1\}^d$, we define the text vector gadget as

$$\mathrm{G}'(b) \triangleq \mathrm{CG}'(b_1)\, y\, \mathrm{CG}'(b_2)\, y\, \mathrm{CG}'(b_3)\, y\, \ldots\, y\, \mathrm{CG}'(b_d).$$

Note that we can derive $\mathrm{G}'(b)$ from $\mathrm{G}(a)$ if and only if $a \cdot b = 0$. Our text T is defined as a concatenation of all text vector gadgets with a symbol of type z in between any two neighboring vector gadgets:

$$T \triangleq \mathrm{G}'(b^1)\, z\, \mathrm{G}'(b^2)\, z\, \mathrm{G}'(b^3)\, z\, \mathrm{G}'(b^4)\, z\, \ldots\, z\, \mathrm{G}'(b^{N-1})\, z\, \mathrm{G}'(b^N).$$

We need to show that we can derive a substring of T from P if and only if there are two orthogonal vectors. This follows from Lemmas 5.2.5 and 5.2.6 below. □

Lemma 5.2.5. *If there are two vectors $a \in A$ and $b \in B$ that are orthogonal, then a substring of T can be derived from P.*

Proof. Suppose that $a^i \cdot b^j = 0$ for some $i \in [M]$, $j \in [N]$. We choose a pattern vector gadget $\mathrm{G}(a^i)$ from the pattern P and transform it into a text vector gadget $\mathrm{G}'(b^j)$. This is possible because of the orthogonality and the construction of the vector gadgets. □

Lemma 5.2.6. *If a substring of T can be derived from P, then there are two orthogonal vectors.*

Proof. We call a sequence of symbols *nice* if and only if it can be derived from the regular expression $x^+\, y\, x^+\, y\, x^+\, y\, \ldots\, y\, x^+$, where "$x^+$" appears d times.

By the construction of the pattern P, we have to choose one pattern vector gadget, say, $\mathrm{G}(a^i)$, that is transformed into a nice sequence. The text T has the property that it is a concatenation of nice sequences separated by symbols of type z. This means that $\mathrm{G}(a^i)$ will be transformed into a sequence $\mathrm{G}'(b^j)$ for some j. This implies that $a^i \cdot b^j = 0$ by the construction of vector gadgets. □

5.2.3 Hardness for type "$\circ *$"

Theorem 5.2.7. *Given sets $A \triangleq \{a^1, \ldots, a^M\} \subseteq \{0,1\}^d$ and $B \triangleq \{b^1, \ldots, b^N\} \subseteq \{0,1\}^d$ with $M \leq N$, we can construct the regular expression P and the text T in time $O(Nd)$, such that a substring of T can be derived from P if and only if there are $a \in A$ and $b \in B$ such that $a \cdot b = 0$. Furthermore, P is of type "$\circ *$", $|P| \leq O(Md)$ and $|T| \leq O(Nd)$.*

Proof. Without loss of generality we can assume that $M = 1 \pmod 2$ and $d = 1 \pmod 2$, $d \geq 100$. Also, if there are $i \in [M]$, $j \in [N]$ such that $a^i \cdot b^j = 0$, then there are $i' \in [M]$, $j' \in [N]$ such that $a^{i'} \cdot b^{j'} = 0$ and $i' = j' \pmod 2$. Furthermore, we assume $b^j_1 = b^j_d = 0$ for all $j \in [N]$ and that a^1 is not orthogonal to any vector b^j.

First, we will construct our pattern. For an integer $v \in \{0,1\}$ and an integer $i \in [d]$, we construct the following pattern coordinate gadget

$$\mathrm{CG}(v,i) \triangleq \begin{cases} yy^* & \text{if } v = 0 \text{ and } i = 1 \pmod 2; \\ yyyy^* & \text{if } v = 1 \text{ and } i = 1 \pmod 2; \\ xx^* & \text{if } v = 0 \text{ and } i = 0 \pmod 2; \\ xxxx^* & \text{if } v = 1 \text{ and } i = 0 \pmod 2. \end{cases}$$

For a vector $a \in \{0,1\}^d$, we define a pattern vector gadget

$$\mathrm{G}(a) \triangleq \mathrm{CG}(a_1, 1)\, \mathrm{CG}(a_2, 2)\, \mathrm{CG}(a_3, 3)\, \ldots\, \mathrm{CG}(a_d, d).$$

We also need another pattern vector gadget $\mathrm{G}_0 \triangleq (y^*\, x^*)^{d+10}\, y^*$.

Our pattern is then defined as follows:

$$P \triangleq y^6 \bigcirc_{j\in[M-1]} \left(x^{10}\,\mathrm{G}(a^j)\,x^{10}\,\mathrm{G}_0\right)\, x^{10}\,VG(a^M)\,x^{10}\,y^6.$$

Now we proceed with the construction of our text. For integers $v \in \{0,1\}, i \in [d]$, we define the following text coordinate gadget

$$\mathrm{CG}'(v,i) \triangleq \begin{cases} yyy & \text{if } v = 0 \text{ and } i = 1 \pmod 2; \\ y & \text{if } v = 1 \text{ and } i = 1 \pmod 2; \\ xxx & \text{if } v = 0 \text{ and } i = 0 \pmod 2; \\ x & \text{if } v = 1 \text{ and } i = 0 \pmod 2. \end{cases}$$

For a vector $b \in \{0,1\}^d$ and an integer $j = 1 \pmod 2$, we define the text vector gadget as

$$\mathrm{G}'(b,j) \triangleq \mathrm{CG}'(b_1,1)\,\mathrm{CG}'(b_2,2)\,\mathrm{CG}'(b_3,3)\,\ldots\,\mathrm{CG}'(b_d,d).$$

We also define $\mathrm{G}'(b,j)$ when $j = 0 \pmod 2$. In this case, $\mathrm{G}'(b,j)$ is equal to $\mathrm{G}'(b,1)$ except that we replace every occurrence of the substring y^3 with the substring y^6.

One can verify that for any *vectors* $a, b \in \{0,1\}^d$ and any integer i, $\mathrm{G}'(b,i)$ can be derived from $\mathrm{G}(a)$ if and only if $a \cdot b = 0$.

We will also need an additional text vector gadget

$$\mathrm{G}'_0 \triangleq y^3\,(x^3\,y^3)^{(d-1)/2}.$$

Our text is then defined as follows:

$$T \triangleq \bigcirc_{j=-2N}^{3N} \left(x^{10}\,\mathrm{G}'_0\,x^{10}\,\mathrm{G}'(b^j,j)\right),$$

where we assume $b^j \triangleq 0111\ldots1110$ for $j \notin [N]$. That is, for $j \notin [N]$, all entries of b^j are equal to 1 except the first and the last entry, which are equal to 0.

We have to show that we can derive a substring of T from P if and only if there are two orthogonal vectors. This follows from Lemmas 5.2.8 and 5.2.9 below. □

Lemma 5.2.8. *If there are two vectors $a \in A$ and $b \in B$ that are orthogonal, then a substring of T can be derived from P.*

Proof. Without loss of generality we have that $a^k \cdot b^k = 0$ for some $k \in [M]$. The proof for the case when $a^k \cdot b^r = 0$, $k \in [M]$, $r \in [N]$ and $k = r \pmod 2$ is analogous.

The pattern P starts with y^6. We transform it into $\mathrm{CG}'(b^0_d, d)$ appearing in $\mathrm{G}'(b^0, 0)$. We can do this since $b^0_d = 0$.

For $j = 1, 2, \ldots, k-2$ we transform $x^{10}\,\mathrm{G}(a^j)\,x^{10}\,\mathrm{G}_0$ into $x^{10}\,\mathrm{G}'_0\,x^{10}\,\mathrm{G}'(b^j,j)$ by transforming $\mathrm{G}(a^j)$ into G'_0 and G_0 into $\mathrm{G}'(b^j,j)$.

Next, we transform

$$x^{10}\,\mathrm{G}(a^{k-1})\,x^{10}\,\mathrm{G}_0\,x^{10}\,\mathrm{G}(a^k)\,x^{10}\,\mathrm{G}_0$$

into

$$x^{10}\,\mathrm{G}'_0\,x^{10}\,\mathrm{G}'(b^{k-1},k-1)\,x^{10}\,\mathrm{G}'_0\,x^{10}\,\mathrm{G}'(b^{k},k)\,x^{10}\,\mathrm{G}'_0\,x^{10}\,\mathrm{G}'(b^{k+1},k+1)$$

Notice that we use the fact that $k \geq 2$ (we assumed that a^1 is not orthogonal to any vector from B). Note that $\mathrm{G}'(b^{k+1}, k+1)$ appears in the text T even if $k = N$. This is because in the definition of the text T, integer j ranges from $-2N$ up to $3N$.

We perform the transformation by performing the following steps:

1. transform $\mathrm{G}(a^{k-1})$ into G'_0;
2. transform G_0 into $\mathrm{G}'(b^{k-1}, k-1)\,x^{10}\,\mathrm{G}'_0$;
3. transform $\mathrm{G}(a^k)$ into $\mathrm{G}'(b^k, k)$ (we can do this since $a^k \cdot b^k = 0$);
4. transform G_0 into $\mathrm{G}'_0\,x^{10}\,\mathrm{G}'(b^{k+1}, k+1)$.

Now, for $j = k+1, \ldots, M-1$ transform $x^{10}\,\mathrm{G}(a^j)\,x^{10}\,\mathrm{G}_0$ into $x^{10}\,\mathrm{G}'_0\,x^{10}\,\mathrm{G}'(b^{j+1}, j+1)$ similarly as before. Next, transform $x^{10}\,\mathrm{G}(a^M)\,x^{10}$ into $x^{10}\,\mathrm{G}'_0\,x^{10}$. Finally, transform y^6 into $\mathrm{CG}'(b_1^{M+1})$ appearing in $\mathrm{G}'(b^{M+1}, M+1)$. We can do this since $b_1^{M+1} = 0$. □

Lemma 5.2.9. *If a substring of T can be derived from P, then there are two orthogonal vectors.*

Proof. By the construction, every substring x^{10} from P must be mapped to a unique substring x^{10} in T (there are no substrings of T that have more than 10 symbols x). Because of this, every $\mathrm{G}(a^i)$ must be mapped to G'_0 or $\mathrm{G}'(b^j, j)$ for some j. If the latter case occurs, the corresponding vectors are orthogonal and we are done. It remains to consider the case that *all* vector gadgets $\mathrm{G}(a^i)$ get mapped to G'_0. Consider any vector gadget G_0 in P. To the left of it we have the sequence x^{10} and to the right of it we have the sequence x^{10}. Each one of these two sequences x^{10} in P gets mapped to a unique sequence x^{10} in T. We call the vector gadget G_0 *nice* if the two unique sequences x^{10} are neighboring in T, that is, there is no other sequence x^{10} in T between the two unique sequences. We consider two cases below.

Case 1. There is a vector gadget G_0 in P that is *not* nice. Take any vector gadget G_0 that is not nice and denote it by V. The gadget V is immediately to the right of the expression $\mathrm{G}(a^{i'})\,x^{10}$ in P for some $i' \in [M]$. $\mathrm{G}(a^{i'})$ is mapped to G'_0 (otherwise, we have found an orthogonal pair of vectors, as per the discussion above) and this G'_0 is to the left of the substring $x^{10}\,\mathrm{G}'(b^{j''}, j'')$ in T for some j''. Because V is *not* nice, a prefix of it must map to $\mathrm{G}'(b^{j''}, j'')\,x^{10}\,\mathrm{G}'_0$. We claim that *entire* V gets mapped to $\mathrm{G}'(b^{j''}, j'')\,x^{10}\,\mathrm{G}'_0$. If this is not the case, then a prefix of V must be mapped to $\mathrm{G}'(b^{j''}, j'')\,x^{10}\,\mathrm{G}'_0\,x^{10}\,\mathrm{G}'(b^{j''+1}, j''+1)$ (since the sequence x^{10} in P to the right of V must be mapped to x^{10}). A prefix of V cannot be mapped to $\mathrm{G}'(b^{j''}, j'')\,x^{10}\,\mathrm{G}'_0\,x^{10}\,\mathrm{G}'(b^{j''+1}, j''+1)$ since $V = (y^*\,x^*)^{d+10}\,y^*$ can produce sequence with at most $d+11$ substrings of maximal length consisting entirely of symbols y but

the sequence $\mathrm{G}'(b^{j''}, j'')\, x^{10}\, VG'_0\, x^{10}\, VG'(b^{j''+1}, j''+1)$ has $3^{\frac{d+1}{2}} > d+11$ (if $d \geq 100$) subsequences of maximal length consisting entirely of y. Therefore, we are left with the case that the entire V is mapped to $\mathrm{G}'(b^{j''}, j'')\, x^{10}\, VG'_0$. The gadget V is to the left of the vector gadget $\mathrm{G}(a^{i'+1})$ and $\mathrm{G}'(b^{j''}, j'')\, x^{10}\, VG'_0$ is to the left of the vector gadget $\mathrm{G}'(b^{j''+1}, j''+1)$. We conclude that $\mathrm{G}(a^{i'+1})$ must be mapped to $\mathrm{G}'(b^{j''+1}, j''+1)$. This implies that $a^{i'+1} \cdot b^{j''+1} = 0$ and we are done.

Case 2. All vector gadgets G_0 in P are nice. The pattern P starts with y^6 followed immediately by x^{10}. This means that y^6 is mapped to $\mathrm{CG}'(b_d^{j'}, j')$ for some *even* j' (by the construction of the coordinate gadgets CG'). Consider vector gadgets in P from the left to the right. We must have that $\mathrm{G}(a^1)$ is mapped to G'_0, that G_0 is mapped to $\mathrm{G}'(b^1, 1)$ (since every G_0 in P is nice), that $\mathrm{G}(a^2)$ is mapped to G'_0, that G_0 is mapped to $\mathrm{G}'(b^2, 2)$ (since every G_0 in P is nice) and so forth. Since M is odd, we have that $\mathrm{G}(a^M)$ is mapped to G'_0 and that this vector gadget G'_0 is followed by $x^{10}\, \mathrm{G}'(b^{j'+M}, j'+M)$. $\mathrm{G}(a^M)$ is followed by $x^{10}\, y^6$ and this means that y^6 is mapped to the beginning of $\mathrm{G}'(b^{j'+M}, j'+M)$. This is impossible since $\mathrm{G}'(b^{j'+M}, j'+M)$ does not contain a substring of length 6 or more consisting of symbols y (observe that $j'+M$ is odd and see the construction of the vector gadget G'). We get that this case (Case 2) cannot happen. □

5.2.4 Hardness for type "∘+∘"

Theorem 5.2.10. *Given sets $A \triangleq \{a^1, \ldots, a^M\} \subseteq \{0,1\}^d$ and $B \triangleq \{b^1, \ldots, b^N\} \subseteq \{0,1\}^d$ with $M \leq N$, we can construct a regular expression P and a sequence of symbols T, in $O(Nd)$ time, such that a substring of T can be derived from P if and only if there are $a \in A$ and $b \in B$ such that $a \cdot b = 0$. Furthermore, P is a concatenation of "+" of sequences, $|P| \leq O(Md)$ and $|T| \leq O(Nd)$.*

Proof. Without loss of generality d is even. First, we will construct our pattern P. For an integer $v \in \{0,1\}$ and an integer $i \in [d]$, we construct the following pattern coordinate gadget

$$\mathrm{CG}(v,i) \triangleq \begin{cases} [x]^+ & \text{if } v = 0 \text{ and } i = 1 \pmod 2; \\ [xx]^+ & \text{if } v = 1 \text{ and } i = 1 \pmod 2; \\ [y]^+ & \text{if } v = 0 \text{ and } i = 0 \pmod 2; \\ [yy]^+ & \text{if } v = 1 \text{ and } i = 0 \pmod 2. \end{cases}$$

For a vector $a \in \{0,1\}^d$, we define a pattern vector gadget as $[x^4]^+\,[y^4]^+$ followed by the concatenation of all coordinate gadgets:

$$\mathrm{G}(a) \triangleq [x^4]^+\,[y^4]^+\,\mathrm{CG}(a_1,1)\,\mathrm{CG}(a_2,2)\,\mathrm{CG}(a_3,3)\,\ldots\,\mathrm{CG}(a_d,d).$$

We also need two other pattern vector gadgets

$$\mathrm{G}_0 \triangleq [x^4\, y^4]^+\,([x]^+\,[y]^+)^{d/2}; \qquad \mathrm{G}_1 \triangleq [x^4]^+\,[y^4]^+\,([x]^+\,[y^8]^+)^{d/2}.$$

Our pattern is then defined as follows:

$$P \triangleq \mathrm{G}_1 \left(\bigcirc_{j\in[M]} \left(\mathrm{G}(1_d)\,\mathrm{G}_0\,\mathrm{G}(a^j)\,\mathrm{G}_0\right)\right) \mathrm{G}(1_d)\,\mathrm{G}(1_d)\,\mathrm{G}_1.$$

Now we proceed with the construction of our text. For integers $v \in \{0,1\}, i \in [d]$, we define the following text coordinate gadget

$$\mathrm{CG}'(v,i) \triangleq \begin{cases} xx & \text{if } v = 0 \text{ and } i = 1 \pmod 2; \\ x & \text{if } v = 1 \text{ and } i = 1 \pmod 2; \\ yy & \text{if } v = 0 \text{ and } i = 0 \pmod 2; \\ y & \text{if } v = 1 \text{ and } i = 0 \pmod 2. \end{cases}$$

For vector $b \in \{0,1\}^d$, we define the text vector gadget as $x^4\,y^4$ followed by the concatenation of all coordinate gadgets:

$$\mathrm{G}'(b) \triangleq x^4\,y^4\,\mathrm{CG}'(b_1,1)\,\mathrm{CG}'(b_2,2)\,\mathrm{CG}'(b_3,3)\,\ldots\,\mathrm{CG}'(b_d,d).$$

In what follows, we will use the following important property of vector gadgets G and G′. First, observe that for integers $u, v \in \{0,1\}, i \in [d]$, we can derive $\mathrm{CG}'(v,i)$ from $\mathrm{CG}(v,i)$ if and only if $uv = 0$. It means that for any *vectors* $a, b \in \{0,1\}^d$, $\mathrm{G}'(b)$ can be derived from $\mathrm{G}(a)$ if and only if $a \cdot b = 0$.

We will also need additional text vector gadgets

$$\mathrm{G}'_0 \triangleq x^4\,y^4\,(x^4\,y^4)^{d/2}; \qquad\qquad \mathrm{G}'_1 \triangleq x^4\,y^4\,(x\,(y)^8)^{d/2}.$$

Our text is then defined as follows:

$$T \triangleq \bigcirc_{j=-5N}^{6N} \left(\mathrm{G}'(0_d)\,\mathrm{G}'_0\,\mathrm{G}'(b^j)\,\mathrm{G}'(0_d)\,\mathrm{G}'_0\,\mathrm{G}'_1\right),$$

where we assume $b^j \triangleq 1_d$ for $j \notin [N]$. That is, b^j is a vector with d entries all equal to 1 for $j \notin [N]$.

We have to show that we can derive a substring of T from P if and only if there are two orthogonal vectors in A and B. This follows from Lemmas 5.2.11 and 5.2.12 below. □

Lemma 5.2.11. *If there are two vectors $a \in A$ and $b \in B$ that are orthogonal, then a substring of T can be derived from P.*

Proof. We assume that $a^k \cdot b^r = 0$ for some $k \in [M]$, $r \in [N]$.

Observe that the pattern P starts with a prefix G_1, which is then followed by the sequence

$$\mathrm{G}(1_d)\,\mathrm{G}_0\,\mathrm{G}(a^i)\,\mathrm{G}_0$$

repeated M times for different $i \in [M]$ and ends with the suffix $\mathrm{G}(1_d)\,\mathrm{G}(1_d)\,\mathrm{G}_1$. We refer to

$$\mathrm{G}(1_d)\,\mathrm{G}_0\,\mathrm{G}(a^i)\,\mathrm{G}_0$$

corresponding to a specific $i \in [M]$ as the i-th *group* of P. Similarly, we observe that the text T is concatenation of sequences

$$\mathrm{G}'(0_d)\,\mathrm{G}'_0\,\mathrm{G}'(b^j)\,\mathrm{G}'(0_d)\,\mathrm{G}'_0\,\mathrm{G}'_1$$

corresponding to different $j = -5N, \ldots, 6N$. We refer to that sequence corresponding to particular $j = -5N, \ldots, 6N$ as the j-th *group* of T.

We transform P into the following substring T' of T:

$$T' \triangleq \mathrm{G}'_1 \bigcirc_{j=1+r-k}^{M+r-k} \left(\mathrm{G}'(0_d)\,\mathrm{G}'_0\,\mathrm{G}'(b^j)\,\mathrm{G}'(0_d)\,\mathrm{G}'_0\,\mathrm{G}'_1\right).$$

Note that T' starts with the prefix G'_1 which is a suffix of the $(r-k)$-th group of T. Then it consists of groups $1+r-k$ to $M+r-k$ of T. We transform P into T' from the left to the right. First, we transform G_1 into G'_1 in the unique way. Then, for $i = 1, \ldots, k-1$, we transform the i-th group of P into the $(i+r-k)$-th group of T'. We do that as follows. For each $i \in \{1, \ldots, k-1\}$, we perform the following transformations: transform $\mathrm{G}(1_d)$ into $\mathrm{G}'(0_d)$; transform G_0 into $\mathrm{G}'_0\,\mathrm{G}'(b^{i+r-k})$; transform $\mathrm{G}(a^i)$ into $\mathrm{G}'(0_d)$; transform G_0 into $\mathrm{G}'_0\,\mathrm{G}'_1$.

Now we transform the k-th and the $(k+1)$-th group of P into the prefix

$$\mathrm{G}'(0_d)\,\mathrm{G}'_0\,\mathrm{G}'(b^r)\,\mathrm{G}'(0_d)\,\mathrm{G}'_0\,\mathrm{G}'_1\,\mathrm{G}'(0_d)\,\mathrm{G}'_0\,\mathrm{G}'(b^{r+1})$$

of the remainder of T'. This is done by transforming $\mathrm{G}(1_d)$ into $\mathrm{G}'(0_d)$, G_0 into G'_0, $\mathrm{G}(a^k)$ into $\mathrm{G}'(b^r)$ (which can be done because $a^k \cdot b^r = 0$), G_0 into $\mathrm{G}'(0_d)$, $\mathrm{G}(1_d)$ into G'_0, G_0 into G'_1, $\mathrm{G}(a^{k+1})$ into $\mathrm{G}'(0_d)$ and G_0 into $\mathrm{G}'_0\,\mathrm{G}'(b^{r+1})$. The remainder of the pattern P consists of groups $k+2, \ldots, M$ and the suffix $\mathrm{G}(1_d)\,\mathrm{G}(1_d)\,\mathrm{G}_1$, while the remainder of T' is

$$T'' \triangleq \left(\bigcirc_{j=r+2}^{M+r-k} \left(\mathrm{G}'(0_d)\,\mathrm{G}'_0\,\mathrm{G}'_1\,\mathrm{G}'(0_d)\,\mathrm{G}'_0\,\mathrm{G}'(b^j)\right)\right)\,\mathrm{G}'(0_d)\,\mathrm{G}'_0\,\mathrm{G}'_1.$$

We transform group $i = k+2, \ldots, M$ of P into a substring $\mathrm{G}'(0_d)\,\mathrm{G}'_0\,\mathrm{G}'_1\,\mathrm{G}'(0_d)\,\mathrm{G}'_0\,\mathrm{G}'(b^{i+r-k})$, as follows. We transform $\mathrm{G}(1_d)$ into $\mathrm{G}'(0_d)$, G_0 into $\mathrm{G}'_0\,\mathrm{G}'_1$, $\mathrm{G}(a^i)$ into $\mathrm{G}'(0_d)$ and G_0 into $\mathrm{G}'_0\,\mathrm{G}'(b^{i+r-k})$. It remains to transform the suffix $\mathrm{G}(1_d)\,\mathrm{G}(1_d)\,\mathrm{G}_1$ of P into the suffix $\mathrm{G}'(0_d)\,\mathrm{G}'_0\,\mathrm{G}'_1$ of T'', which can be done in a non-ambiguous way. □

Lemma 5.2.12. *If a substring of T can be derived from P, then there are two vectors $a \in A$ and $b \in B$ that are orthogonal.*

Proof. Notice that P starts and ends with G_1 and that this vector gadget does not appear anywhere else in the pattern. G_1 contains $d/2$ patterns $[y^8]^+$. This means that it must map to a substring of T containing $d/2$ substrings consisting of only symbols y of length divisible by 8. The text T contains vector gadgets G'_1 that has substrings of symbols y of length divisible by 8. No other vector gadget in T has this property. Therefore, both vector gadgets G_1 of P must map to G'_1 in T. Consider G_1 at the beginning of P. Suppose that it maps to G'_1 that comes from the r-th group of T, for some r. The pattern P contains $2M$ vector gadgets G_0. Each one of them starts with $[x^4\,y^4]^+$. In the proof of Lemma 5.2.11 we transform $[x^4\,y^4]^+$ into $x^4\,y^4$ so

that G_0 maps to a single corresponding vector gadget in T or we transform $[x^4\,y^4]^+$ into $(x^4\,y^4)^{2+d/2}$ so that G_0 maps to two vector gadgets in T. Here we consider the first copy V of the vector gadget G_0 of P such that the corresponding $[x^4\,y^4]^+$ *does not* map to $(x^4\,y^4)^{2+d/2}$. If there is no such a vector gadget, we can check that the j-th group of P must be transformed into the $(j+r)$-th group of T for all $t \in [M]$. That means that the suffix $\mathrm{G}(1_d)\,\mathrm{G}(1_d)\,\mathrm{G}_1$ of P must be transformed into the prefix of the $(M+r+1)$-st group of T. This implies that G_1 is transformed into $\mathrm{G}'(b^{M+r+1})$, which is impossible by the construction of the vector gadgets. Therefore, there must be vector gadget V with the stated properties.

Suppose that V comes from the i-th group in P. For all vector gadgets of type G_0 to the left of V, the prefix $[x^4\,y^4]^+$ got mapped to $(x^4\,y^4)^{2+d/2}$. This means that group $k = 1, \ldots, i-1$ of P is transformed into the group $k+r$ of T. There are two cases to consider, depending on whether V comes before or after the vector gadget $\mathrm{G}(a^i)$ in the group i of P.

Case 1: V comes after $\mathbf{G}(a^i)$. We show that this case cannot occur. We start by observing that $\mathrm{G}(a^i)$ maps to $\mathrm{G}'(0_d)$. The sequence $[x^4\,y^4]^+$ in V cannot map to $3+d/2$ or more copies of $x^4\,y^4$ because it would imply that the first substring $x\,(y)^8$ of G'_1 can be derived from $[x^4\,y^4]^+$, which is impossible. It follows that $[x^4\,y^4]^+$ maps to $1+d/2$ or fewer copies of $x^4\,y^4$. Note that V is followed by $\mathrm{G}(1_d)$ and $\mathrm{G}(1_d)$ starts with $[x^4]^+\,[y^4]^+$. This means that $[x^4\,y^4]^+$ of V must map to only one copy of $x^4\,y^4$ (i.e., G_0 is mapped to G'_0), so that $[x^4]^+\,[y^4]^+$ at the beginning of $\mathrm{G}(1_d)$ can map to $x^4\,y^4$ at the beginning of G'_1. This in turn implies that $\mathrm{G}(1_d)$ must be transformed into a suffix of G'_1, which is again impossible since G'_1 contain a single symbol x surrounded by symbols y on both sides and the regular expression $\mathrm{G}(1_d)$ cannot produce such a substring. Thus, this case (Case 1) cannot occur.

Case 2: V comes before $\mathbf{G}(a^i)$. The argument is similar to the previous paragraph except we will conclude that $a^i \cdot b^{i+r} = 0$. Similarly as before, we can conclude that $[x^4\,y^4]^+$ in V cannot be mapped to $3+d/2$ or more copies of $x^4\,y^4$. This is because $\mathrm{G}'(b^{i+r})$ starts with $x^4\,y^4\,\mathrm{CG}'(b_1^{i+r}, 1)$ and, if $[x^4\,y^4]^+$ in V was mapped to $3+d/2$ or more copies of $x^4\,y^4$, $\mathrm{CG}'(b_1^{i+r}, 1)$ would be a prefix of a sequence that can be derived from $[x^4\,y^4]^+$. We can verify that it is impossible. Assume that $[x^4\,y^4]^+$ maps to $1+d/2$ or fewer copies of $x^4\,y^4$. Because $\mathrm{G}(a^i)$ starts with $[x^4]^+\,[y^4]^+$, we have that $\mathrm{G}(a^i)$ is transformed into $\mathrm{G}'(b^{i+r})$. This is possible only if $a^i \cdot b^{i+r} = 0$ by the construction of the coordinate gadgets. □

5.2.5 Hardness for type "∘|∘"

Theorem 5.2.13. *Given sets $A \triangleq \{a^1, \ldots, a^M\} \subseteq \{0,1\}^d$ and $B \triangleq \{b^1, \ldots, b^N\} \subseteq \{0,1\}^d$ with $M \leq N$, we can construct a regular expression P and a sequence of symbols T, in $O(Nd)$ time, such that a substring of T can be derived from P if and only if there are $a \in A$ and $b \in B$ such that $a \cdot b = 0$. Furthermore, P is of type "∘|∘", $|P| \leq O(Md)$ and $|T| \leq O(Nd)$.*

Proof. We will modify the construction for type "∘+∘" so that it gives a hardness proof for type "∘|∘". The text T remains the same. We will modify pattern P as follows. First, recall that P is a repeated concatenation of regular expressions of the form $[x]^+, [x\,x]^+, [y]^+, [y\,y]^+, [x^4]^+, [y^4]^+, [x^4\,y^4]^+, [y^8]^+$ in some order. In the proof of Lemma 5.2.11, all of those sequences get repeated at most 8 times, except for the sequence $x^4\,y^4$ which gets repeated once or $2 + d/2$ times. Therefore, we replace $[S]^+$ with $[S \mid S^2 \mid S^3 \mid S^4 \mid S^5 \mid S^6 \mid S^7 \mid S^8]$ for all S, except when $S = x^4\,y^4$. In the latter case we replace $[x^4\,y^4]^+$ with $[x^4\,y^4 \mid (x^4\,y^4)^{2+d/2}]$. The proof of Lemma 5.2.11 goes through as before. The proof of Lemma 5.2.12 also goes through because, after these modifications, if a sequence can be derived from the modified pattern, it can be also derived from the original pattern. □

5.2.6 Hardness for type "∘+|"

Theorem 5.2.14. *Given sets $A \triangleq \{a^1, \ldots, a^M\} \subseteq \{0,1\}^d$ and $B \triangleq \{b^1, \ldots, b^N\} \subseteq \{0,1\}^d$ with $M \le N$, we can construct a regular expression P and a sequence of symbols T, in $O(Nd)$ time, such that a substring of T can be derived from P if and only if there are $a \in A$ and $b \in B$ such that $a \cdot b = 0$. Furthermore, P has type "∘+|", $|P| \le O(Md)$ and $|T| \le O(Nd)$.*

Proof. Without loss of generality $M = 0 \pmod 2$ and $d = 0 \pmod 2$. Also, if there are two orthogonal vectors, then there are $a^i \in A$, $b^j \in B$ with $a^i \cdot b^j = 0$ and $i = j \pmod 2$.

First, we will construct our pattern. We need the following coordinate gadget construction. For integers $v \in \{0,1\}$ and $i \in [d]$ define

$$\mathrm{CG}(v,i) \triangleq \begin{cases} [x \mid y \mid 0 \mid 1]^+ & \text{if } v = 0 \text{ and } i = 1 \pmod 2; \\ [x \mid y \mid 0]^+ & \text{if } v = 1 \text{ and } i = 1 \pmod 2; \\ [x \mid y \mid 0' \mid 1']^+ & \text{if } v = 0 \text{ and } i = 0 \pmod 2; \\ [x \mid y \mid 0']^+ & \text{if } v = 1 \text{ and } i = 0 \pmod 2. \end{cases}$$

For a vector $a \in \{0,1\}^d$, we define pattern vector gadget $\mathrm{G}(a)$ as concatenation of all coordinate gadgets for entries of the vector:

$$\mathrm{G}(a) \triangleq \mathrm{CG}(a_1, 1)\,\mathrm{CG}(a_2, 2)\,\mathrm{CG}(a_3, 3)\,\ldots\,\mathrm{CG}(a_d, d).$$

We also need another vector gadget

$$\mathrm{G}_0 \triangleq ([0|1]^+\,[0'|1']^+)^{d/2},$$

that is, G_0 is equal to the vector gadget $\mathrm{G}(0_d)$ except it cannot produce symbols x and y. Our pattern P is then defined as follows:

$$P \triangleq x^+\ \left(\bigcirc_{i\in[M]} \left(\mathrm{G}_0\,[x|y]^+\,\mathrm{G}(a^i)\,[x|y]^+\right)\right)\ \mathrm{G}_0\ x^+.$$

Now we construct our text T. First, for a vector $b \in \{0,1\}^d$, we define text vector

gadget

$$\mathrm{G}'(b) \triangleq b_1\, b'_2\, b_3\, b'_4\, b_5\, \ldots\, b'_d,$$

that is, it is concatenation of all entries and we add $'$ for every second entry. Note that we can derive $\mathrm{G}'(b)$ from $\mathrm{G}(a)$ if and only if $a \cdot b = 0$. Also, we can derive $\mathrm{G}'(b)$ from G_0 for any b. Our text T is defined as

$$T \triangleq \bigcirc_{j=-9N}^{10N} \left(x^{d+10}\, \mathrm{G}'(b^{2j})\, y^{d+10}\, \mathrm{G}'(b^{2j+1})\right),$$

where, for $j \notin [N]$, we set $b^j \triangleq 1_d$ (vector consisting of 1s only).

We need to show that we can derive a substring of T from P if and only if there are two orthogonal vectors. This follows from Lemmas 5.2.15 and 5.2.16 below. □

Lemma 5.2.15. *If there are two vectors $a \in A$ and $b \in B$ that are orthogonal, then a substring of T can be derived from P.*

Proof. We assume that $a^k \cdot b^k = 0$ for some $k \in [M]$. In the general case when $a^k \cdot b^r = 0$ for $k \in [M]$, $r \in [N]$ and $k = r \pmod 2$, the proof is analogous. Furthermore, we assume that $k = 1 \pmod 2$ (the case $k = 0 \pmod 2$ is analogous).

We transform P into the following substring T' of T:

$$T' \triangleq x\, \mathrm{G}'(b^0)\, y^{d+10}\, \mathrm{G}'(b^1)\, x^{d+10}\, \mathrm{G}'(b^2)\, y^{d+10}\, \mathrm{G}'(b^3)\, x^{d+10}\, \mathrm{G}'(b^4)\, y^{d+10}\, \mathrm{G}'(b^5)\, x^{d+10} \ldots x^{d+10}\, \mathrm{G}'(b^M)\, y^{d+10}\, \mathrm{G}'(b^{M+1})\, x.$$

Note that T' starts and ends with x, because $M = 0 \pmod 2$.

To show how to transform P into T', it is helpful to write pattern P as $P = P''\, P'''$, where

$$P'' \triangleq x^+\, \mathrm{G}_0\, [x|y]^+\, \mathrm{G}(a^1)\, [x|y]^+\, \mathrm{G}_0\, [x|y]^+\, \mathrm{G}(a^2)\, [x|y]^+ \ldots\, \mathrm{G}(a^{k-1})\, [x|y]^+\, \mathrm{G}_0,$$

$$P''' \triangleq [x|y]^+\, \mathrm{G}(a^k)\, [x|y]^+\, \mathrm{G}_0\, [x|y]^+\, \mathrm{G}(a^{k+1})\, [x|y]^+\, \mathrm{G}_0 \ldots\, \mathrm{G}_0\, [x|y]^+\, \mathrm{G}(a^M)\, [x|y]^+\, \mathrm{G}_0\, x^+$$

and text T as $T = T''\, T'''$, where

$$T'' \triangleq x\, \mathrm{G}'(b^0)\, y^{d+10}\, \mathrm{G}'(b^1)\, x^{d+10}\, \mathrm{G}'(b^2)\, y^{d+10}\, \mathrm{G}'(b^3)\, x^{d+10}\, \mathrm{G}'(b^4) \ldots\, \mathrm{G}'(b^{k-3})\, y^{d+10}\, \mathrm{G}'(b^{k-2})\, x^{d+10}\, \mathrm{G}'(b^{k-1}),$$

$$T''' \triangleq y^{d+10}\, \mathrm{G}'(b^k)\, x^{d+10}\, \mathrm{G}'(b^{k+1})\, y^{d+10}\, \mathrm{G}'(b^{k+2})\, x^{d+10}\, \mathrm{G}'(b^{k+3})\, y^{d+10}\, \mathrm{G}'(b^{k+4})\, x^{d+10} \ldots x^{d+10}\, \mathrm{G}'(b^M)\, y^{d+10}\, \mathrm{G}'(b^{M+1})\, x.$$

Now we transform P into T in two steps—transform P'' into T'' and P''' into T'''.

Transform P'' into T''. P'' starts with x^+, which we transform into x. For the rest of P'', we make transformations according to the following rules. We make transformations starting from the beginning of P''. If we see G_0, we transform it into

the corresponding $\mathrm{G}'(b^j)$. If we see $[x|y]^+$, we transform it into x^5 or y^5 (according to which symbols are in the corresponding positions in T''). If we see $\mathrm{G}(a^i)$, we transform it into x^d or y^d (according to which symbols are in the corresponding positions in T'').

Transform P''' into T'''. Notice that T''' starts with y. This is because $k = 1 \pmod 2$. Now we make the following 3 transformations to the prefix of P''': transform $[x|y]^+$ into y^{d+10}, $\mathrm{G}(a^k)$ into $\mathrm{G}'(b^k)$ (we can do this because $a^k \cdot b^k = 0$) and $[x|y]^+$ into x^{d+10}. Now we transform the rest of P''' into the remainder of T'''. We do that starting from the beginning of the remainder of P'''. If we see G_0, we transform it into the corresponding $\mathrm{G}'(b^j)$. If we see $[x|y]^+$, we transform it into x^5 or y^5 (depending on which symbols are in the corresponding positions in T). If we see $\mathrm{G}(a^i)$, we transform it into x^d or y^d (depending on which symbols are in the corresponding positions in T). Finally, to finish the transformation, we transform x^+ into x. □

Lemma 5.2.16. *If a substring of T can be derived from P, then there are two orthogonal vectors.*

Proof. Recall that the pattern P consists of $M+1$ vector gadgets G_0. We enumerate the gadgets with integers $0, 1, 2, 3, \ldots, M$. Each of them consists of d symbols from the alphabet $\{0, 1, 0', 1'\}$. Therefore, by the construction of T, it must be the case that every vector gadget G_0 transforms into $\mathrm{G}'(b^j)$ for some j. Assume that the 0-th G_0 transforms into $\mathrm{G}'(b^0)$. If it transforms into $\mathrm{G}'(b^j)$ for some other j and $j = 0 \pmod 2$, the proof is analogous. The modularity constraint on j holds because P starts with x and the symbol x must precede $\mathrm{G}'(b^j)$. We consider two cases below:

Case 1. There exists $t \in \{0, 1, 2, \ldots, M\}$ such that the t-th G_0 is not transformed into $\mathrm{G}'(b^t)$. Pick smallest such t. This means that $\mathrm{G}'(b^t)$ has been derived from $\mathrm{G}(a^t)$. From the construction of $\mathrm{G}(a^t)$ and $\mathrm{G}'(b^t)$, we conclude that $a^t \cdot b^t = 0$.

Case 2. For every $t \in \{0, 1, 2, \ldots, M\}$, the t-th G_0 is transformed into $\mathrm{G}'(b^t)$. Consider the M-th vector gadget G_0. It is transformed into $\mathrm{G}'(b^M)$. The M-th vector gadget G_0 is followed by x^+ and $\mathrm{G}'(b^M)$ is followed by y. This means that we cannot derive this substring of T from P. Thus, this case (Case 2) cannot happen. □

5.2.7 Hardness for type "∘|+"

Theorem 5.2.17. *Given sets $A \triangleq \{a^1, \ldots, a^M\} \subseteq \{0,1\}^d$ and $B \triangleq \{b^1, \ldots, b^N\} \subseteq \{0,1\}^d$ with $M \leq N$, we can construct a regular expression P and a sequence of symbols T, in $O(Nd)$ time, such that a substring of T can be derived from P if and only if there are $a \in A$ and $b \in B$ such that $a \cdot b = 0$. Furthermore, P has type "∘|+", $|P| \leq O(Md)$ and $|T| \leq O(Nd)$.*

Proof. We will modify the construction for "∘+|" so that it gives the hardness proof for "∘|+". Whenever we have a regular expression in P of the form $[s_1|s_2|s_3|\ldots|s_l]^+$ for $l \geq 1$ symbols $s_1, s_2, \ldots, s_l$, we replace it with $[s_1^+|s_2^+|s_3^+|\ldots|s_l^+]$. Let P' be the new

regular expression that we obtain this way. The text T remains unchanged. Observe that, if we can derive some sequence S from P', we were able to derive S from P as well. Because of this, if a subsequence of T can be derived from P, then we can conclude that there are two orthogonal vectors between A and B. It remains to show that, if there are two orthogonal vectors, then a subsequence of T can be derived from P'. This follows from the proof of Lemma 5.2.15. In particular, we observe that in the proof of Lemma 5.2.15, if we derive a sequence from $[s_1|s_2|s_3|\ldots|s_l]^+$, such sequence is of the form s_i^j for some $j \geq 1$ and $i \in [l]$. □

5.3 Reductions for the membership problem

5.3.1 Hardness for type "∘∗"

Theorem 5.3.1. *Given sets $A \triangleq \{a^1, \ldots, a^N\} \subseteq \{0,1\}^d$ and $B \triangleq \{b^1, \ldots, b^N\} \subseteq \{0,1\}^d$, we can construct a regular expression P and a sequence of symbols T, in $O(Nd)$ time, such that T can be derived from P if and only if there are $a \in A$ and $b \in B$ such that $a \cdot b = 0$. Furthermore, P is of type "∘∗", $|P|, |T| \leq O(Nd)$.*

Proof. We slightly modify the construction from Theorem 5.2.7. We instantiate the construction from Theorem 5.2.7 with $M = N$. We obtain a pattern P' and a text T such that a substring of T can be derived from P' if and only if there are two orthogonal vectors. We define the pattern P as follows:

$$P \triangleq \left(\bigcirc_{j=1}^{|T|}(x^*\, y^*)\right) \circ P' \circ \left(\bigcirc_{j=1}^{|T|}(x^*\, y^*)\right).$$

We claim that T can be derived from P if and only if there are two orthogonal vectors. This follows from the construction of P and Theorem 5.2.7. We know that a substring of T can be derived from P' if and only if there are two orthogonal vectors. The expressions $\bigcirc_{j=1}^{|T|}(x^*\, y^*)$ allow us to derive the remaining prefix and suffix of T. □

5.3.2 Hardness for type "∘+∘"

Theorem 5.3.2. *Given sets $A \triangleq \{a^1, \ldots, a^N\} \subseteq \{0,1\}^d$ and $B \triangleq \{b^1, \ldots, b^N\} \subseteq \{0,1\}^d$, we can construct a regular expression P and a sequence of symbols T, in $O(Nd)$ time, such that T can be derived from P if and only if there are $a \in A$ and $b \in B$ such that $a \cdot b = 0$. Furthermore, P is a concatenation of "+" of sequences, $|P|, |T| \leq O(Nd)$.*

Proof. We will adapt the construction from Theorem 5.2.10. We instantiate the construction from Theorem 5.2.10 with $M = N$. We obtain a pattern P' and a text T such that a substring of T can be derived from P' if and only if there are two orthogonal vectors. The final pattern $P \triangleq P_1 \circ P' \circ P_2$ is a concatenation of three expressions P_1, P', P_2. Each one of the expressions P_1, P' and P_2 is a concatenation of "+" of sequences. Clearly, if T can be derived from P, then a substring of T can be derived from P'. By the statement of Theorem 5.2.10, there must be two orthogonal

vectors in this case. Therefore, our goal is to construct P_1 and P_2 such that the text T can be derived from $P = P_1 \circ P' \circ P_2$ if there are two orthogonal vectors. In the rest of the proof we achieve this goal.

If there are two orthogonal vectors, then by the proof of Theorem 5.2.10, the text

$$T = \bigcirc_{j=-5N}^{6N} \left(\mathrm{G}'(0_d)\,\mathrm{G}'_0\,\mathrm{G}'(b^j)\,\mathrm{G}'(0_d)\,\mathrm{G}'_0\,\mathrm{G}'_1\right)$$

can be written as $T = T_1\,T_2\,T'\,T_3\,T_4$, where the sequences T_1, T_2, T', T_3 and T_4 have the following properties.

- $$T_1 \triangleq \bigcirc_{j=-5N}^{w-N} \left(\mathrm{G}'(0_d)\,\mathrm{G}'_0\,\mathrm{G}'(1_d)\,\mathrm{G}'(0_d)\,\mathrm{G}'_0\,\mathrm{G}'_1\right).$$

- $$T_2 \triangleq \left(\bigcirc_{j=w-N+1}^{w-1} \left(\mathrm{G}'(0_d)\,\mathrm{G}'_0\,\mathrm{G}'(b^j)\,\mathrm{G}'(0_d)\,\mathrm{G}'_0\,\mathrm{G}'_1\right)\right) \circ \mathrm{G}'(0_d)\,\mathrm{G}'_0\,\mathrm{G}'(b^w)\,\mathrm{G}'(0_d)\,\mathrm{G}'_0.$$

- $$T' \triangleq \mathrm{G}'_1 \bigcirc_{j=1+w}^{N+w} \left(\mathrm{G}'(0_d)\,\mathrm{G}'_0\,\mathrm{G}'(b^j)\,\mathrm{G}'(0_d)\,\mathrm{G}'_0\,\mathrm{G}'_1\right)$$
  for some $w \in \{1-N, \ldots, N-1\}$ and t' can be derived from p'.

- $$T_3 \triangleq \bigcirc_{j=N+w+1}^{w+2N-1} \left(\mathrm{G}'(0_d)\,\mathrm{G}'_0\,\mathrm{G}'(b^j)\,\mathrm{G}'(0_d)\,\mathrm{G}'_0\,\mathrm{G}'_1\right).$$

- $$T_4 \triangleq \bigcirc_{j=w+2N}^{6N} \left(\mathrm{G}'(0_d)\,\mathrm{G}'_0\,\mathrm{G}'(1_d)\,\mathrm{G}'(0_d)\,\mathrm{G}'_0\,\mathrm{G}'_1\right).$$

Our goal is to construct expressions P_1 and P_2 such that $T_1\,T_2$ can be derived from P_1 (independently of the value w) and $T_3\,T_4$ can be derived from P_2 (independently of the value w). We construct P_1 and P_2 as follows.

- We note that the expression
  $$\hat{P} \triangleq \bigcirc_{j=1}^{N} \left(\mathrm{G}'(0_d)\,\mathrm{G}'_0\,\mathrm{G}'(1_d)\,\mathrm{G}'(0_d)\,\mathrm{G}'_0\,\mathrm{G}'_1\right)^+$$
  can derive the sequence T_1 independently of the value w. Let S be an arbitrary sequence of symbols x and y. As long as there are two neighboring symbols x (y, resp.), we replace those two symbols by one copy of symbol x (y, resp.). Let S' be the resulting sequence. We define the expression $f(S)$ as follows:
  $$f(S) \triangleq \bigcirc_{j=1}^{|S'|} (S'_j)^+.$$
  That is, in the expressions $f(S)$ we allow to repeat any symbol in S' one or more times. Note that $f(T_2)$ can derive T_2 independently of the value w and that $f(T_2)$ does not depend on vectors b^j. This implies that the expression $P_1 \triangleq \hat{P} \circ f(T_2)$ can derive $T_1\,T_2$ which is what we needed.

- We define $P_2 \triangleq f(T_3) \circ \hat{P}$. We can check that $f(T_3)$ and $\hat{P}$ do not depend on the value w and vectors b^j. We can also check that we can derive $T_3\,T_4$ from P_2.

□

5.3.3 Hardness for type "∘|∘"

Theorem 5.3.3. *Given sets $A \triangleq \{a^1, \ldots, a^N\} \subseteq \{0,1\}^d$ and $B \triangleq \{b^1, \ldots, b^N\} \subseteq \{0,1\}^d$, we can construct a regular expression P'' and a sequence of symbols T, in $O(Nd)$ time, such that T can be derived from P'' if and only if there are $a \in A$ and $b \in B$ such that $a \cdot b = 0$. Furthermore, P'' is of type "∘|∘", $|P''|, |T| \leq O(Nd)$.*

Proof. We will modify the construction for "∘+∘" (Theorem 5.3.2) so that it gives a hardness proof for "∘|∘". The text T remains the same. We will modify pattern P as follows. First, recall that $P = P_1 \circ P' \circ P_2$. We transform P into an expression of type "∘|∘" in two steps.

- We transform P' into a sequence of type "∘|∘" in the same way as it is done in the proof of Theorem 5.2.13.

- Expressions P_1 and P_2 are repeated concatenations of expressions

$$[x]^+,\ [y]^+,\ [\mathrm{G}'(0_d)\,\mathrm{G}'_0\,\mathrm{G}'(1_d)\,\mathrm{G}'(0_d)\,\mathrm{G}'_0\,\mathrm{G}'_1]^+\,.$$

  In the proof of Theorem 5.3.2 we can repeat each one of argument expressions

$$x,\ y,\ \mathrm{G}'(0_d)\,\mathrm{G}'_0\,\mathrm{G}'(1_d)\,\mathrm{G}'(0_d)\,\mathrm{G}'_0\,\mathrm{G}'_1$$

  at most 8 times so that we are still able to derive sequences $T_1\,T_2$ and $T_3\,T_4$ from P_1 and P_2, respectively. Thus, we replace $[S]^+$ by $[S \mid S^2 \mid S^3 \mid S^4 \mid S^5 \mid S^6 \mid S^7 \mid S^8]$ for each

$$S = x,\ y,\ \mathrm{G}'(0_d)\,\mathrm{G}'_0\,\mathrm{G}'(1_d)\,\mathrm{G}'(0_d)\,\mathrm{G}'_0\,\mathrm{G}'_1$$

  in P_1 and P_2.

Let P'' be the resulting expression. If the sequence T can be derived from P, it can still be derived from P''. This follows from Theorem 5.2.13 and the construction of T. It remains to argue that if T cannot be derived from P, then it cannot be derived from P''. This is true by the transformation above and Theorem 5.2.13. □

5.3.4 Hardness for type "∘+|"

Theorem 5.3.4. *Given sets $A \triangleq \{a^1, \ldots, a^M\} \subseteq \{0,1\}^d$ and $B \triangleq \{b^1, \ldots, b^N\} \subseteq \{0,1\}^d$ with $M \leq N$, we can construct a regular expression P and a sequence of symbols T, in $O(Nd)$ time, such that T can be derived from P if and only if there are $a \in A$ and $b \in B$ such that $a \cdot b = 0$. Furthermore, P has type "∘+|", $|P| \leq O(Md)$ and $|T| \leq O(Nd)$.*

Proof. We adapt the hardness proof from Theorem 5.2.14. We instantiate the construction from Theorem 5.2.14 and we obtain a pattern P' and a text T such that a substring of T can be derived from P' if and only if there are two orthogonal vectors. We define the new pattern P as follows:

$$P \triangleq [0 \mid 1 \mid 0' \mid 1' \mid x \mid y]^+ \circ P' \circ [0 \mid 1 \mid 0' \mid 1' \mid x \mid y]^+.$$

We claim that T can be derived from P if and only if there are two orthogonal vectors. If T can be derived from P, then a substring of T can be derived from P' and by Theorem 5.2.14 there are two orthogonal vectors. Conversely, if there are two orthogonal vectors then by Theorem 5.2.14 we can derive a substring of T from P'. We derive the remaining prefix and suffix of T from the expressions $[0 \mid 1 \mid 0' \mid 1' \mid x \mid y]^+$. □

5.3.5 Hardness for type "∘|+"

Theorem 5.3.5. *Given sets $A \triangleq \{a^1, \ldots, a^N\} \subseteq \{0,1\}^d$ and $B \triangleq \{b^1, \ldots, b^N\} \subseteq \{0,1\}^d$, we can construct a regular expression P and a sequence of symbols T, in $O(Nd)$ time, such that T can be derived from P if and only if there are $a \in A$ and $b \in B$ such that $a \cdot b = 0$. Furthermore, P has type "∘|+", $|P|, |T| \leq O(Nd)$.*

Proof. We adapt the hardness proof from Theorem 5.2.17. We instantiate the construction from Theorem 5.2.17 with $M = N$. We obtain a pattern P' and a text T such that a substring of T can be derived from P' if and only if there are two orthogonal vectors. From the proof of Theorem 5.2.14 we have that

$$T = \bigcirc_{j=-9N}^{10N} \left(x^{d+10}\, \mathrm{G}'(b^{2j})\, y^{d+10}\, \mathrm{G}'(b^{2j+1}) \right).$$

Let $k \triangleq |x^{d+10}\, \mathrm{G}'(b^{2j})\, y^{d+10}\, \mathrm{G}'(b^{2j+1})|$ be the length of the sequence $x^{d+10}\, \mathrm{G}'(b^{2j})\, y^{d+10}\, \mathrm{G}'(b^{2j+1})$. Notice that k does not depend on j. Our new sequence P is constructed as follows:

$$P \triangleq \left(\bigcirc_{j=1}^{7Nk} [0^+ \mid 1^+ \mid [0']^+ \mid [1']^+ \mid x^+ \mid y^+] \right) \circ P' \circ \bigcirc_{j=1}^{7Nk} [0^+ \mid 1^+ \mid [0']^+ \mid [1']^+ \mid x^+ \mid y^+].$$

If T can be derived from P, then a substring of T can be derived from P' and by Theorem 5.2.17 there are two orthogonal vectors. Conversely, if there are two orthogonal vectors, then we can derive a substring of T from P'. We can easily check that we can derive the remaining prefix and suffix of T from the expressions $\bigcirc_{j=1}^{7Nk} [0^+ \mid 1^+ \mid [0']^+ \mid [1']^+ \mid x^+ \mid y^+]$. □

5.4 Algorithms

5.4.1 Algorithm for the word break problem

Word break problem. Given a binary sequence T of length $n \triangleq |T|$ and a collection of binary[4] sequences P with total length $m \triangleq \sum_{P' \in P} |P'|$, decide if the sequence T can be written as a concatenation $T = T_1 \dots T_k$ such that $T_i \in P$ for every $i \in [k]$. If T can be written in such a way, we call T *decomposable*.

We will solve this problem in time $\tilde{O}(nm^{0.4444\dots}) = \tilde{O}\left(nm^{0.5-1/18}\right)$. The runtime has been further improved to $\tilde{O}(nm^{1/3})$ in a follow-up work [BGL17]. They also provided a matching conditional lower bound for a certain class of algorithms.

As a warm-up, we first solve the problem in time $\tilde{O}(n\sqrt{m})$ and then we present the $\tilde{O}\left(nm^{0.5-1/18}\right)$ time algorithm.

$\tilde{O}(n\sqrt{m})$ time algorithm for the word break problem

Let $d(P) \triangleq \{|P'| : P' \in P\}$ be the set of distinct lengths of the pattern sequences $P' \in P$. We will show how to solve the word break problem in time $\tilde{O}(n|d(P)|)$. Since $\sum_{P' \in P} |P'| = m$, we have that $|d(P)| \leq O(\sqrt{m})$, which implies the upper bound.

We will use the following lemma.

Lemma 5.4.1. *We can randomly choose a hash function $h : \{0,1\}^* \to \mathbb{N}$ and preprocess T in $\tilde{O}(n)$ time so that the following holds:*

- *Given any (contiguous) substring T' of T, we can compute the hash $h(T')$ in $\tilde{O}(1)$ time.*
- *For any two sequences $T'' \neq T'$ (not necessarily substrings of T), $\Pr[h(T'') = h(T')] \leq 1/n^{10}$.*
- *For any sequence T' (necessarily substring of T), we can compute $h(T')$ in time $\tilde{O}(|T'|)$.*

Proof. E.g., use Rabin-Karp rolling hash. □

Theorem 5.4.2. *The word break problem can be solved in time $\tilde{O}(n|d(P)|)$.*

Proof. Preprocess T according to Lemma 5.4.1 and compute $h(P) \triangleq \{h(P') : P' \in P\}$.

We solve the problem using dynamic programming. We use the table $D : [n+1] \to \{0,1\}$. The algorithm determines the values $D(n), D(n-1), \dots, D(1)$ (in this order). We set $D(i) = 1$ if and only if the sequence $T_i\, T_{i+1} \dots T_n$ is decomposable.

- Set $D(i) = 0$ for all $i = 1, \dots, n$ and set $D(n+1) = 1$.
- For $i = n, \dots, 1$, set $D(i) = 1$ if and only if there exists $j > i$ such that $D(j) = 1$ and $(j - i) \in d(P)$, and $h(T_i \dots T_{j-1}) \in h(P)$.

[4]Without loss of generality we assume that all sequences are binary. If this is not so, we encode every symbol of the alphabet using a binary sequence of length $\lceil \log w \rceil$ where w is the size of the alphabet. This increases the lengths of the sequences by a logarithmic multiplicative factor.

- Out that T is decomposable if and only if $D(1) = 1$.

□

$\tilde{O}\left(nm^{0.5-1/18}\right)$ time algorithm for the word break problem

Similarly as in the $\tilde{O}(n\sqrt{m})$ time algorithm, we fill out the table $D : [n+1] \to \{0,1\}$:

- Set $D(i) = 0$ for all $i = 1, \ldots, n$ and set $D(n+1) = 1$.
- For every $i = n, \ldots, 1$ in this order, set $D(i) = 1$ if and only if the sequence $T_i\, T_{i+1}\, \ldots\, T_n$ is decomposable.
- Output that T is decomposable if and only if $D(1) = 1$.

We will show that the second step can be performed in $\tilde{O}(nm^{0.5-\alpha})$ time for a sufficiently small constant $\alpha > 0$. We will later show that we can set $\alpha = 1/18$. For now we can think of $\alpha > 0$ as a sufficiently small constant, say, $\alpha = 0.01$. We will make the following two assumptions, justified by the next two lemmas.

Lemma 5.4.3. *For all $P' \in P$, $|P'| \geq m^{0.5-\alpha}$.*

Proof. Let $\hat{P} \triangleq \{P' \in P : |P'| < m^{0.5-\alpha}\}$. Clearly, we have that $|d(\hat{P})| < m^{0.5-\alpha}$. Therefore, as we perform the second step of the algorithm, for every $i = n, \ldots, 1$, we set $D(i) = 1$ if there exists $j > i$ with $D[j] = 1$ and $T_i\, \ldots\, T_{j-1} \in \hat{P}$ in the same way as it is done in the proof of Theorem 5.4.2. For every i this takes $\tilde{O}(|d(\hat{P})|) = \tilde{O}(m^{0.5-\alpha})$ time. Therefore, the total runtime corresponding to processing sequences in $\hat{P}$ is $\tilde{O}(nm^{0.5-\alpha})$. □

Lemma 5.4.4. *For all $P' \in P$, $|P'| \leq m^{0.5+\alpha}$.*

Proof. Let $\hat{P} \triangleq \{P' \in P : |P'| > m^{0.5+\alpha}\}$. Since $\sum_{P' \in P} |P'| = m$, we have $|d(\hat{P})| \leq |\hat{P}| \leq m^{0.5-\alpha}$. Similarly as in the proof of Lemma 5.4.3 we can set $D(i) = 1$ if there exists $j > i$ with $D(j) = 1$ and $T_i\, \ldots\, T_{j-1} \in \hat{P}$. Therefore, the total runtime corresponding to processing sequences in $\hat{P}$ is $\tilde{O}(n|\hat{P}|) \leq \tilde{O}(nm^{0.5-\alpha})$. □

In the rest of the section we will show that the second step of the algorithm can be implemented in $\tilde{O}(nm^{0.5-\alpha})$ time if $\alpha > 0$ is a sufficiently small constant. By Lemmas 5.4.3 and 5.4.4, we can assume that for all $P' \in P$, $m^{0.5-\alpha} \leq |P'| \leq m^{0.5+\alpha}$.

We build a tree data structure for P. It is a binary tree t where each node has two children. Each node corresponds to a prefix of a sequence in P. The root node corresponds to an empty sequence. If a node u corresponds to the sequence S and has two children then one of the children corresponds to the sequence $S\,0$ (S followed by 0) and the other corresponds to the sequence $S\,1$. If u has only one child, it corresponds to either $S\,0$ or $S\,1$. If a node u corresponds to a sequence of length $i \geq 0$, u has *depth* i. If a node u corresponds to a sequence P' and $P' \in P$, then we call the node u *marked*. We preprocess t in such a way that for any node u we have a pointer to node v such that v is a marked ancestor of u of maximal depth. A node is not its own ancestor. This data structure can be constructed in $O(m)$ time. Because $\sum_{P' \in P} |P'| = m$ and for all $P' \in P$, $m^{0.5-\alpha} \leq |P'|$, we have the number of marked nodes in the tree is upper bounded by $|P| \leq m^{0.5+\alpha}$.

Preprocessing of the tree t. We further preprocess the tree t and the sequence T such that given any index $i = 1, \ldots, n$, we can answer the following query in $\tilde{O}(1)$ time. Specifically, the query algorithm outputs the maximal $j > i$ such that $T_i\, T_{i+1} \ldots T_{j-1}$ is a prefix of a sequence in P, and reports the node u corresponding to $T_i\, T_{i+1} \ldots T_{j-1}$. We build a data structure that stores the hash values for all non-empty prefixes of all sequences in P and supports lookups in $\tilde{O}(1)$ time. We use Rabin-Karp rolling hash to compute the hashes. This takes $\tilde{O}(m)$ time. We also preprocess the sequence T so that we can compute Rabin-Karp rolling hash value in $\tilde{O}(1)$ time for every substring. This takes $\tilde{O}(n)$. The total runtime of the preprocessing steps is $\tilde{O}(m+n)$. Given i, the query algorithm does the binary search to find the largest $j > i$ such that the hash of sequence $T_i\, T_{i+1} \ldots T_{j-1}$ is in the table. Since we can do lookups in the table in $\tilde{O}(1)$ and there are $\tilde{O}(1)$ binary search steps, and we can compute the rolling hash value for every substring in $\tilde{O}(1)$ time, we get the required upper bound $\tilde{O}(1)$ on the query time. We can easily augment the data structure so that we can output the corresponding node u from the tree t.

For $j = 1, \ldots, n/m^{0.5-\alpha}$ we call the sequence $D(n - jm^{0.5-\alpha} + 1) \ldots D(n - (j-1)m^{0.5-\alpha})$ the j-th *chunk* of the table D. We will show how to determine the values in the j-th chunk in time $\tilde{O}(m^{1-2\alpha})$ assuming that we have all values for chunks with indices smaller than j. This gives the required upper bound on the runtime because there are $n/m^{0.5-\alpha}$ chunks and $\tilde{O}(m^{1-2\alpha})n/m^{0.5-\alpha} = \tilde{O}(nm^{0.5-\alpha})$. Since for all $P' \in P$, $m^{0.5-\alpha} \leq |P'| \leq m^{0.5+\alpha}$, the only previously computed values of D that are needed to compute the j-th chunk are:

$$D(n - (j-1)m^{0.5-\alpha} + 1) \ldots D(n - (j-1)m^{0.5-\alpha} + m^{0.5+\alpha}).$$

To simplify the notation, we relabel the table D by defining the table D'. Specifically, we identify $D(n-jm^{0.5-\alpha}+1) \ldots D(n-(j-1)m^{0.5-\alpha}+m^{0.5+\alpha})$ with $D'(1) \ldots D'(m^{0.5-\alpha}+m^{0.5+\alpha})$. Thus, our goal is to find the values $D'(1) \ldots D'(m^{0.5-\alpha})$ knowing the values $D'(m^{0.5-\alpha}+1) \ldots D'(m^{0.5-\alpha}+m^{0.5+\alpha})$. Let T' be the corresponding substring of T of length $m^{0.5-\alpha} + m^{0.5+\alpha}$.

Intuition. Suppose that we want to determine value of $D'(i)$. We look for the largest $j > i$ such that $T'_i \ldots T'_{j-1}$ is in P. If $D'(j) = 1$, we set $D'(i) = 1$ and move to determine $D'(i-1)$. However, it might be that $D'(j) = 0$ and there are integers j' such that $i < j' < j$ and $T'_i \ldots T'_{j'-1}$ is in P. For every such j' we have to check whether $D'(j') = 1$ and set $D'(i) = 1$ if this happens. If there are not too many such j', we can work through all of them. It might happen that there are many such j'. In this case we build a characteristic vector of the set of such j's, i.e., set the entry corresponding to each such j' to 1. We then convolve the characteristic vector with D'. Although this does not reduce the runtime when working with $D'(i)$, the saving will occur in the future if we will need to determine $D'(i')$ such that $i' < i$ and $T'_i\, T'_{i+1} \ldots$ and $T'_{i'}\, T'_{i'+1} \ldots$ share long prefixes. We will make this more precise below.

We determine the unknown values of D' in two phases—preprocessing phase and online phase. In the preprocessing phase we preprocess the known part of D' together with t in $\tilde{O}(m^{1-2\alpha})$ time. In the online phase we determine the unknown values $D'(i)$

for $i = m^{0.5-\alpha}, \ldots, 1$ in this order. We spend time $O(m^{0.5-\alpha})$ for every $D'(i)$.

Preprocessing phase. In the following we will define a subset of the marked nodes that we call *special*. Initially the set of special nodes is empty. We will keep the invariant that if a node is special, then all its marked ancestors are also special. Since $|P| \leq m^{0.5+\alpha}$, there are at most $m^{0.5+\alpha}$ leaves in the tree t. Fix an arbitrary ordering of the leaves and consider the leaves one by one. Let ℓ be the current leaf that we consider. Let c be the number of marked ancestors of ℓ that are not special. We can determine c in time $O(c)$ because every node in t keeps a pointer to the marked ancestor of maximal depth. We distinguish two cases.

Case 1: $c > m^{0.5-3\alpha}$**.** We mark ℓ and all its marked ancestors as special. Since $|P| \leq m^{0.5+\alpha}$, the number of marked nodes is at most $m^{0.5+\alpha}$. This means that we happen to be in this case at most $m^{0.5+\alpha}/m^{0.5-3\alpha} = m^{4\alpha}$ times. Let d denote the depth of the node ℓ. Since for every $P' \in P$, $|P'| \leq m^{0.5+\alpha}$, we have that $d \leq m^{0.5+\alpha}$. Let $u_0, u_1, \ldots, u_d = \ell$ be the nodes on the path from the root of t to ℓ. The node u_0 is the root of t. Let $l \triangleq m^{0.5-\alpha}$. We define $\lfloor d/l \rfloor$ binary vectors $r_1, \ldots, r_{\lfloor d/l \rfloor} \in \{0,1\}^l$ as follows. For $i = 1, \ldots, \lfloor d/l \rfloor$ and $j = 1, \ldots, l$ we set the j-th entry $r_i(j)$ of the vector r_i to be equal to $r_i(j) = 1$ if the node $u_{(i-1)l+j}$ is marked and equal to $r_i(j) = 0$ if the node is not marked. For every $i = 1, \ldots, \lfloor d/l \rfloor$, we compute the convolution between the binary vector r_i and the binary vector $D'(m^{0.5-\alpha}+1) \ldots D'(m^{0.5-\alpha}+m^{0.5+\alpha})$ in the following sense. We output a binary vector c_i with $m^{0.5-\alpha} + m^{0.5+\alpha} + l - 1$ entries such that for $j = 1, \ldots, m^{0.5-\alpha} + m^{0.5+\alpha} + l - 1$,

$$c_i(j) \triangleq \sum_{k=1}^{l} \left(r_i(k) \cdot D'(k + j + m^{0.5-\alpha} - l) \right).$$

When computing $c_i(j)$, if we need to access an entry $D'(z)$ with $z \leq m^{0.5-\alpha}$ or $z \geq m^{0.5-\alpha} + m^{0.5+\alpha} + 1$, we assume that it is equal to 0. Computing the convolution c_i for r_i takes $O(m^{0.5+\alpha} \log m)$ time using the Fast Fourier Transform. Since we have to compute the convolution for $i = 1, \ldots, \lfloor d/l \rfloor$, in total it takes $O(m^{0.5+\alpha} \log m) d/l = O(m^{0.5+3\alpha} \log m)$ time. Since we happen to be in this case at most $m^{4\alpha}$ times, the total runtime corresponding to this case is bounded by $O(m^{0.5+7\alpha} \log m) = \tilde{O}(m^{1-2\alpha})$ assuming that $\alpha \leq 1/18$.

Case 2: $c \leq m^{0.5-3\alpha}$**.** In this case we do not do anything. Since there are at most $m^{0.5+\alpha}$ leaves, the total time corresponding to this case is upper bounded by $m^{0.5+\alpha} O(c) \leq O(m^{1-2\alpha})$ which is what we wanted.

See Figure 5-2 for an example run of the preprocessing phase.

Notice that after the preprocessing phase, the set of marked and special nodes of t form a rooted subtree among all marked nodes of t.

Online phase. We determine the values of $D'(i)$ for $i = m^{0.5-\alpha}, \ldots, 1$ in this order. Fix i for which we want to determine $D'(i)$. Let $j > i$ be the largest integer such that

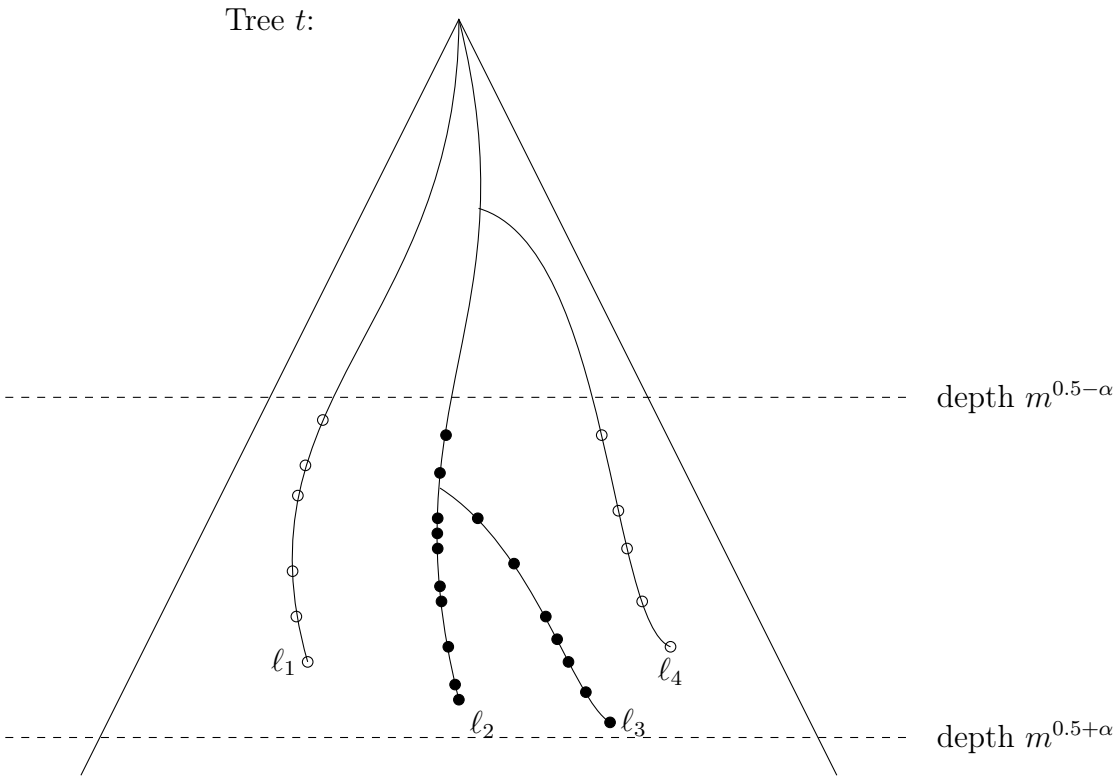

Figure 5-2: An example of the preprocessing phase of tree t and vector D'. Each white circle denotes a marked node that is not special. Each black circle denotes a marked node that is special. We do not depict the nodes that are not marked. Each solid line is a sequence of marked or unmarked nodes. Notice that all marked nodes are at depth at least $m^{0.5-\alpha}$ and at most $m^{0.5+\alpha}$. The tree has 4 leaves $\ell_1, \ell_2, \ell_3, \ell_4$. We process them in order $\ell_1, \ell_2, \ell_3, \ell_4$:

- Leaf ℓ_1 has $c = 5$ marked ancestors. We assume that $m^{0.5-3\alpha} = 5$ and therefore we happen to be in case $c \leq m^{0.5-3\alpha}$. We do not make the marked ancestors special.

- Leaf ℓ_2 has $c = 9$ marked ancestors. We are in $c > m^{0.5-3\alpha}$ case and we make ℓ_2 and all marked ancestors of ℓ_2 special. We split the path from the root to ℓ_2 into shorter paths of length l. For every shorter path we construct a binary characteristic vector where we set entry to be equal to 1 if the corresponding node is marked. Then we convolve each characteristic vector with binary vector D'.

- Leaf ℓ_3 has $c = 6$ marked ancestors that are not special and two marked ancestors that are special. We are in $c = 6 > m^{0.5-3\alpha}$ case and we make ℓ_3 and all marked ancestors of ℓ_3 special. Then we split the path from the root to ℓ_3 into shorter paths of length l and proceed similarly as when processing leaf ℓ_2.

- Leaf ℓ_4 has $c = 4$ marked ancestors. We are in $c \leq m^{0.5-3\alpha}$ case and we do not make any of the marked ancestors special.

$T'_i \dots T'_{j-1}$ corresponds to a node u in tree t. We find j using the query algorithm described before, in $\tilde{O}(1)$ time. Let a be the ancestor of u which is marked and special and whose depth d is the largest. Let c be the number of marked ancestors of u that are marked but not special. By the preprocessing phase, $c \leq m^{0.5-3\alpha}$. For every such marked ancestor which is not special we want to determine whether the corresponding entry of D' is equal to 1. This takes at most $O(m^{0.5-3\alpha})$ total time. If we found such an entry of D' equal to 1, we set $D'(i) = 1$ and move to determining the value of $D'(i-1)$. By the choice of the node a, there are at most l marked ancestors of u that are special and are of depth more than $d' \triangleq \lceil d/l \rceil l$. We can determine whether the corresponding entry of D' is equal to 1 for any of those nodes and set $D'(i) = 1$ if this happens. This takes at most $O(l)$ total time. It remains to consider the marked ancestors of u of depth at most d'. For this we use the convolutions that we performed in the preprocessing step. Let $u_0, u_1, \dots, u_{d'} = a$ be the nodes on the path from the root of t to l. The node u_0 is the root of t. Let $r_1, \dots, r_{d'/l}$ be the binary vectors of length l constructed as follows. For $k = 1, \dots, d'/l$ and $j = 1, \dots, l$ we set the j-th entry $r_k(j)$ of the vector r_k to be equal to $r_k(j) = 1$ if the node $u_{(k-1)l+j}$ is marked and equal to $r_k(j) = 0$ if the node is not marked. We want to determine whether there is a marked node corresponding to an entry equal to 1 in r_k such that the corresponding entry in D' is equal to 1. We can determine whether this is the case in $O(1)$ because this we can check whether the corresponding entry of c_k is equal to 0 or at least 1. If the entry is at least 1, we set $D'(i)$ to be equal to 1 and continue with $D'(i-1)$. We spend $O(1)$ for each $k = 1, \dots, d'/l$. The whole process takes $O(d'/l)$ time. The total runtime spent on computing the value of $D'(i)$ is bounded by $O(m^{0.5-3\alpha}) + O(l) + O(d'/l) \leq O(m^{0.5-\alpha})$.

The runtime of the algorithm. We have seen that the runtime of the algorithm is bounded by $\tilde{O}(nm^{0.5-\alpha})$ for any constant $0 < \alpha \leq 1/18$. We obtain the required runtime $\tilde{O}\left(nm^{0.5-1/18}\right)$ by setting $\alpha = 1/18$.

5.4.2 Algorithm for type "$\circ+$"

Theorem 5.4.5. *Let P be a regular expression of type "$\circ+$" and T be a text. In time $O(|P| + |T|)$ we can reduce the pattern matching problem on P and T to one instance of the subset matching problem and one instance of the wildcard matching problem.*

Proof. Below we reduce this regular expression pattern matching problem to the subset matching and to the wildcard matching problems. Then we combine the two outputs and solve the initial regexp problem.

Reduction to the subset matching problem. We partition the pattern P into substrings of maximal length such that all symbols in each substring are equal, i.e., each substrings is a concatenation of copies of a^+ or a for a symbol a. Consider one particular substring and suppose that it contains l copies of a or a^+. We replace this substring with a set $\{(a,1), (a,2), \dots, (a,l)\}$. We perform this operation for every substring of maximal length and concatenate the resulting sets to obtain the

pattern P' for subset matching. Similarly, we partition the text T into substrings of maximal length such that all symbols in each substring are equal. As before, every such substring of length l is replaced with a set $\{(a,1),(a,2),\ldots,(a,l)\}$. This yields a text t'.

Now we run a subset matching algorithm on the pattern P' and the text T'. For each position i of T', we find whether P' matches the substring of T' that starts at i.

Reduction to the wildcard matching problem. Similarly as before, we partition pattern P into substrings of maximal length such that all symbols in each substring are equal, i.e., each substrings is a concatenation of copies of a^+ or a. There are two kinds of substrings. If the substring contains only l symbols a (i.e., there are no a^+s), we replace this substring with the symbol (a,l). If the substring has at least one a^+, we replace it with a wildcard. We concatenate all symbols and wildcards into one regular expression P''. If P'' starts with a symbol, we replace it with a wildcard. Also, if P'' ends with a symbol, we replace it with a wildcard. Similarly, we partition the text T into substrings of maximal length such that all symbols in each substring are equal. We replace each such substring, say a^l, with a symbol (a,l). The symbols are concatenated into the text T''.

Now we run a wildcard matching algorithm on the pattern P'' and the text T''. For each position i of T'', we find whether P'' matches the substring of T'' that starts at i.

Combining the results. A substring of T can be derived from P if and only if there is a position i such that two conditions hold: P' matches a substring of T' starting at the position i and P'' matches a substring of T'' starting at position i. The first condition (coming from the subset matching instance) ensures that, if we match P to a substring of T, the number of appropriate symbols in T is *at least* as large as in P. The second condition (coming from the wildcard matching instance) ensures that, if P contains a maximal substring consisting only of symbols a (no a^+s) for some a, then the corresponding position in T contains *exactly* the same number of symbols a. Notice that we do not need to satisfy this condition if P starts or ends with a maximal substring consisting only of as, which is why we start and end P'' with wildcards. □

5.4.3 Algorithms for types "|∗○" and "|+○"

Theorem 5.4.6. *Let P be a regular expression of type "|∗○" or "|+○" and T be a text. In time $O(|P|+|T|)$ we can decide if T can be derived from P.*

Proof. Let $T' \triangleq t\,t$ be the sequence T repeated twice. By [Gus97] (page 196), we can use the suffix tree and the lowest common ancestor data structures to preprocess the sequences T' and T in $O(|T|)$ time such that the following holds. For any $i = 1,\ldots,|T|$, we can in constant $O(1)$ time decide if the substring $T'_i\,T'_{i+1}\,\ldots\,T'_{i+|T|-1}$ of the sequence T' of length $|T|$ is equal to the sequence T.

Let P' be an arbitrary sequence of non-zero length. We notice that the sequence T can be derived from $[P']^*$ (or, equivalently, from $[P']^+$ if $|T| > 0$) if and only if P' is a prefix of T and the substring $T'_{|P'|+1} T'_{|P'|+2} \dots T'_{|P'|+|T|}$ is equal to T, and $|T|$ is divisible by $|P'|$. We can check the first condition in time $O(|P'|)$ and the second condition in time $O(1)$ assuming that we did the preprocessing step.

Let $P \triangleq [P_1]^* \mid [P_2]^* \mid \dots \mid [P_k]^*$ be the input pattern to the membership problem for some integer $k \geq 1$ and sequences $P_1, \dots, P_k$. We do the preprocessing step on T' and T which takes $O(|T|)$. For every P_i, $k \geq i \geq 1$ we decide in $O(|P_i|)$ time if T can be derived from $[P_i]^*$. This yields the required runtime.

The algorithm for $P \triangleq [P_1]^+ \mid [P_2]^+ \mid \dots \mid [P_k]^+$ is the same except we cannot derive the text T if $|T| = 0$. □

5.4.4 Algorithms for types "|∘+" and "∗∘+"

Theorem 5.4.7. *Let P be a regular expression of type "|∘+" or "∗∘+" and T be a text. In time $O(|P| + |T|)$ we can decide if T can be derived from P.*

Proof. We do the run-length encoding of T defined as follows. Set $T' \triangleq T$ and initialize R to be an empty ordered sequence of tuples. While $|T'| > 0$, let a be the first symbol of T' and let $l > 0$ be the largest integer such that a^l (the symbol a repeated l times) is a prefix of T'. Remove the prefix a^l from T' and add the tuple (a, l) at the end of the ordered sequence R, and repeat. This takes $O(|T|)$ time in total. Let $|R|$ be the number of tuples in R.

Type "|∘+". Let $P \triangleq P_1 \mid \dots \mid P_k$ for some integer $k > 0$, where each P_i is a concatenation of a and a^+ for various symbols a. We want to decide if there exists an integer $i = 1, \dots, k$ such that the text T can be derived from the expression P_i. Fix an arbitrary $i = 1, \dots, k$. We do the run-length encoding of the expression P_i and produce a sequence of tuples $R(P_i)$ defined as follows. Set $P'_i \triangleq P_i$ and $R(P_i)$ to be an empty sequence of tuples. While $|P'_i| > 0$, choose the largest integer $l > 0$ such that there exists a prefix of P'_i of form $aa^+a^+aa\dots$ (an arbitrary concatenation of a and a^+) for some symbol a. Let $l' \geq 0$ be such that the prefix of P'_i has l' occurrences of a and $l - l'$ occurrences of a^+. If $l' = l$, we add tuple $(a, = l)$ to the end of the sequence of tuples $R(P_i)$. Otherwise, if $l' < l$, we add tuple $(a, \geq l)$ to the end of the sequence $R(P_i)$. We delete the prefix of P'_i and repeat (until $|P'_i| = 0$). Let $|R(P_i)|$ be the number of tuples in $R(P_i)$. We can derive T from the expression P_i if and only if the following two conditions hold:

- $|R| = |R(P_i)|$.
- For all $j = 1, \dots, |R|$, if $R_j = (a, l)$ (the j-th tuple of R is (a, l)), then $R(P_i)_j = (a, = l)$ or $R(P_i)_j = (a, \geq l')$ for some integer $l' \leq l$.

For every $i = 1, \dots, k$ this takes $O(|P_i|)$ time and the required upper bound on the runtime follows.

Type "$*\circ+$". We have to consider the case when $P = [P']^*$ for some regexp P'. We do the run-length encoding on the sequence P' and get the sequence of tuples $R(P')$ (as described in the case for type "$|\circ+$"). For every tuple $(a, = l)$ or $(a, \geq l)$, let a be its type. Consider two subcases.

The first and the last tuple of $R(P')$ are of different types. We can derive the text T from the expression P if and only if the following two conditions hold.

- $|R|$ is divisible by $|R(P')|$.
- Let $R' \triangleq (R(P'), R(P'), \ldots, R(P'))$, where $R(P')$ is repeated $|R|/|R(P')|$ time in the right hand side. For all $j = 1, \ldots, |R|$, if $R_j = (a, l)$, then $R'_j = (a, = l)$ or $R'_j = (a, \geq l')$ for some integer $l' \leq l$.

The first and the last tuple of $R(P')$ are of the same type. If $|R(P')| = 1$, then we can check if T can be derived from P easily. If there is no integer $k \geq 1$ such that $|R| = k|R(P')| - (k-1)$, then T cannot be derived from P. Otherwise, let R' be $R(P')$ except the last tuple. Let R'' be R except we change the first tuple of R. Let z be the first and the last tuple of R merged (in the natural way). We replace the first tuple of R by z and get R''. We define $R''' = (R', R'', R'', R'', \ldots, R'')$, where R'' is repeated $k-1$ times. Furthermore, we add the last tuple of $R(P')$ at the end of R'''. The text T can be derived from the expression P if and only if for all $j = 1, \ldots, |r|$, if $R_j = (a, l)$, then $R'''_j = (a, = l)$ or $R'''_j = (a, \geq l')$ for some integer $l' \leq l$.

In the both subcases the runtime is upper bounded by $O(|P| + |T|)$. □

Chapter 6

Hardness of approximation

In Chapters 3 and 4 we presented hardness for the edit distance and the longest common subsequence problems. The hardness results hold for algorithms that solve the problems exactly. A natural question arises: How well can we approximate LCS and edit distance in a strongly sub-quadratic (or near-linear) time?

We say that an algorithm c-approximates the edit distance $\mathrm{edit}(P, Q)$ of two given sequences P, Q of length n if it outputs a value x that satisfies $\mathrm{edit}(P, Q) \leq x \leq c \cdot \mathrm{edit}(P, Q)$. Since the edit distance is at most n, an n-approximation is trivial. There are better approximation algorithms known. In [AKO10] it was shown that for any fixed $\varepsilon > 0$, in $O(n^{1+\varepsilon})$ time it is possible to output a poly-logarithmic approximation with $c = (\log n)^{O(1/\varepsilon)}$. Recently, [CDG+18] presented an algorithm that runs in $O(n^{2-\varepsilon})$ time, for a constant $\varepsilon > 0$, and outputs a constant factor approximation to the edit distance. See [AKO10, CDG+18] for an overview of other approximation algorithms.

While LCS and edit distance are closely related, they behave quite differently with respect to approximations. We say that an algorithm c-approximates the LCS of two given sequences P, Q if it outputs a value x that satisfies $\mathrm{LCS}(P, Q)/c \leq x \leq \mathrm{LCS}(P, Q)$.

A simple observation shows that the LCS of binary sequences can be approximated within a factor of 2 in linear time: the longest common subsequence that consists entirely of zeroes or entirely of ones is at least half from the optimal solution. In general, for an alphabet of size s, it is easy to get an s-approximation for the LCS and it is an open question to design an $(s - \delta)$-approximation in near-linear time or even strongly sub-quadratic time for any constant integer $s \geq 2$ and constant $\delta > 0$. See [Nav01, BHR00] for surveys on the algorithms for the LCS problem.

6.1 Our results

The results from Chapters 3 and 4 immediately imply quadratic hardness for approximating LCS and edit distance within the factor of $1 + 1/n$. In this part of the thesis we present an evidence against algorithms that achieve $1 + 1/(\log n)^{\omega(1)}$ or $1 + o(1)$ approximation and run in strongly sub-quadratic and *deterministic* time.

We prove a connection of the following form: if there is a strongly sub-quadratic and deterministic algorithm for LCS that achieves $1 + o(1)$ approximation on two sequences of length n over an alphabet of size n^β for an arbitrary small constant $\beta > 0$, then the complexity class E^{NP} does not have non-uniform linear size series parallel circuits.[1] This consequence (explained in more detail below) is widely believed to be true. However, proving it unconditionally would be a breakthrough in complexity theory and in the study of non-uniform circuit lower bounds. As stated, this is merely a "difficulty" or a "no-pass" result for LCS, not "hardness". It provides a "circuit lower bounds" barrier for designing a fast $(1 + o(1))$-approximation algorithm for LCS: it is at least as difficult as resolving a longstanding (and considered to be difficult) open question in circuit complexity. Furthermore, we prove a stronger result (Theorem 6.1.4 below), which we think should be regarded as a "hardness" result as well, giving evidence that such algorithms for LCS might not exist.

Our result for LCS will be based on the presumed difficulty of solving the following problem in a strongly sub-quadratic deterministic time.

Definition 6.1.1 (Orthogonal row problem)**.** *Given two sets of Boolean matrices $A, B \subseteq \{0,1\}^{K \times D}$ of size $|A| = |B| = N$, we say that a pair $A^i \in A, B^j \in B$ is a "good" pair if there exists a $k \in [K]$ such that the rows $A^i_{k,*}$ and $B^j_{k,*}$ are orthogonal, that is, $\sum_{h \in [D]} A^i_{k,h} B^j_{k,h} = 0$.*

The orthogonal row problem asks to distinguish the following two cases (and if we are in neither, the output can be arbitrary):

1. *(no good pairs) none of the pairs $A^i \in A, B^j \in B$ are good;*
2. *(many good pairs) at least $N^2(1 - 1/\log_2^{10} N)$ pairs $A^i \in A, B^j \in B$ are good.*

A trivial algorithm solves this problem in quadratic deterministic time, by going over all pairs of matrices, but can we do better? Note that if we ask whether at least one good pair exists (without the above promise that either there is none or many) then the problem requires $N^{2-o(1)}$ under SETH (which is conjectured to hold also for randomized algorithms), even when $K = 1$ and D is any $\omega(\log N)$ (see Theorem 2.3).

We introduce the hypothesis that the orthogonal row problem cannot be solved by a deterministic algorithm in a strongly sub-quadratic time in the following sense.

Hypothesis 6.1.2. *There is no $\delta > 0$ and $\alpha > 0$ such that for all constant d we can solve the orthogonal row problem on binary matrices of size $N^\alpha \times d \log N$ in deterministic $O(N^{2-\delta})$ time.*

We observe that a randomized algorithm can quickly solve the problem by sampling a few pairs of matrices. But can a *deterministic* algorithm do anything clever enough to solve the problem in a strongly sub-quadratic time? Such an algorithm is not known and, in fact, Lemma 6.1.3 below suggests that it would be a breakthrough.

To state our results, we use the series-parallel circuits [Val77, Cal08, Vio09, CDL+12]. See Section 6.2 for a formal definition. In [Val77] Valiant introduced

[1] The class E^{NP} or $\mathsf{TIME}[2^{O(n)}]^{\mathsf{NP}}$ is the class of problems solvable in exponential time with access to an NP oracle.

these circuits and argued that most known computer programs fit under this restriction. His hope was that understanding these circuits would be easier than the general case. Four decades later we still do not know how to resolve basic challenges proposed in his paper, like showing an explicit function that does not have *linear size* series-parallel circuits. It is still conceivable that the complexity class E^{NP} can be computed by such circuits, and proving otherwise would be a major achievement. Our first lemma states that refuting Hypothesis 6.1.2 is at least as difficult as showing such results.

Lemma 6.1.3. *If Hypothesis 6.1.2 is false, then the class E^{NP} does not have non-uniform Valiant series-parallel circuits of linear size.*

Thus, we contribute to the growing body of known connections between algorithm design and circuit lower bounds (see the recent survey [Wil14]). We stress that the circuit lower bounds consequence is only meant to show that the hypothesis is hard to refute. As an evidence that the hypothesis is *plausible*, we remark that none of the current (e.g., [CW16, GM16]) or conjectured-to-exist derandomization techniques (e.g., if $\mathsf{P} = \mathsf{BPP}$) are enough to refute it.

Reduction to approximate LCS. Our main theorem shows that the orthogonal row problem can be reduced to LCS while creating a multiplicative gap, giving the first nontrivial hardness of approximation for LCS.

Theorem 6.1.4. *If for some $\delta > 0$ and $\beta > 0$ there is a deterministic algorithm that can approximate LCS of two given sequences of length n over an alphabet of size n^β to within a $1 + o(1)$ factor in $O(n^{2-\delta})$ time, then Hypothesis 6.1.2 is false (and the class E^{NP} does not have non-uniform Valiant series-parallel circuits of linear size).*

We remark that our approximation hardness for LCS immediately transfers to other problems. For example, we get that the RNA folding problem [Edd04, BGSW16] cannot be approximated within a $1 + o(1)$ factor in a strongly sub-quadratic time.

Another application of our approach gives a weaker lower bound for the edit distance and LCS on binary sequences. In Section 6.5 we show that a deterministic $(1 + 1/(\log n)^{\omega(1)})$-approximation for these problems in strongly sub-quadratic time implies that E^{NP} does not have non-uniform NC^1 circuits.

Subsequent works. In [AR18] the circuit lower bound connection is improved and the authors show that any constant factor approximation for LCS in $O(n^{2-\delta})$ deterministic time implies that E^{NP} does not have non-uniform linear-size NC^1 circuits. The work [CGL+18] strengthens the connection further and shows barriers for larger approximation factors and bigger classes of circuits.

6.2 Valiant series-parallel circuits

Definition 6.2.1 (Valiant series-parallel graphs [Val77, Cal08, Vio09, CDL+12])**.** *A* multidag $G = (V, E)$ *is a directed acyclic multigraph. Let input(G) be the set of*

vertices of G with in-degree 0. *Let output(G) be the set of vertices of G with out-degree* 0. *We say that the multidag G is a Valiant series parallel (VSP) graph if there exists a* labelling $l : V \to \mathbb{Z}$ *of G with the following properties:*

- *For all directed edges $(u, v) \in E$ we have that $l(u) < l(v)$.*
- *There exists an integer $d \in \mathbb{Z}$ such that for all $v \in$ input(G), $l(v) = d$. The definition from [Cal08] asks that $d = 0$. It is not hard to verify that our definition is equivalent to theirs.*
- *There exists an integer $d' \in \mathbb{Z}$ such that for all $v \in$ output(G), $l(v) = d'$.*
- *There is* no *pair of directed edges $(u, v), (u', v') \in E$ such that the inequality $l(u) < l(u') < l(v) < l(v')$ holds.*

Definition 6.2.2 (Valiant series-parallel circuits [Val77, Cal08, Vio09, CDL+12])**.** *A circuit is a Valiant series-parallel circuit if the underlying multidag is a VSP graph and the fan-in (in-degree) of every gate is at most* 2.

Definition 6.2.3 (Size of a circuit)**.** *The size of a circuit on n input variables is equal to the number of gates in it. We do not count the $n + 2$ input nodes, i.e., the input variables and the two constant values* 0 *and* 1 *(which are assumed to be given as the last two input nodes to the circuit).*

Definition 6.2.4 (VSP_c)**.** *We define class VSP_c to be the set of languages recognizable by VSP circuits of size at most $\leq cn$ where n is the number of input variables. The set of allowed gates is the set of all gates of fan-in at most* 2.

Below we show properties of the class VSP_c that we will use later.
We need the following definition from [BSV14].

Definition 6.2.5 ([BSV14])**.** *Let F_n be a family of functions from $\{0,1\}^n$ to $\{0,1\}$. We say that F_n is* efficiently closed under projections *if functions in F_n have a $n^{O(1)}$-size description and given (the description of) a function $f \in F_n$, indices $i, j \leq n$, and a bit $b \in \{0,1\}$, we can compute in time $n^{O(1)}$ the functions $\neg f$, $f(x_1, \ldots, x_{i-1}, b \text{ XOR } x_j, x_{i+1}, \ldots, x_n)$ and $f(x_1, \ldots, x_{i-1}, b, x_{i+1}, \ldots, x_n)$, all of which are in F_n.*

Lemma 6.2.6. *The class VSP_c is efficiently closed under projections for any $c \geq 1$.*

Proof. From Definition 6.2.4 it follows that the class VSP_c has $n^{O(1)}$-size description: the circuit itself. Consider a function f on n input variables from VSP_c that has a VSP circuit of size at most $\leq cn$. We show that the three functions from the statement of Definition 6.2.5 can be computed in $n^{O(1)}$ time and that all of them are in VSP_c.

Function $\neg f$. Consider the output of f. If it is one of the inputs, we add the NOT gate and remove all the other gates. If the output is none of the inputs, it must be some gate g. We replace it with gate $\neg g$. Since we allow all gates of fan-in at most 2 in the circuit, there is also the gate $\neg g$. The number of gates did not increase and the function $\neg f$ is now in VSP_c. Clearly, the transformation can be done in time $n^{O(1)}$.

Function $f(x_1, \ldots, x_{i-1}, b \text{ XOR } x_j, x_{i+1}, \ldots, x_n)$**.** If $b = 0$, we rewire all gates that used input x_i to use input x_j. If $b = 1$, we rewire all gates to use NOT x_j. Since we have all gates of fan in at most 2, we do not need to introduce the NOT gate. Instead, we replace the gate by another gate that negates the corresponding input. Similarly as before, the transformation can be done in time $n^{O(1)}$ and we did not increase the number of gates. Thus, the resulting function is in VSP_c.

Function $f(x_1, \ldots, x_{i-1}, b, x_{i+1}, \ldots, x_n)$**.** The transformation is similar as in the previous case. Instead of rewiring to x_j, we rewire to the constant input b which is among the inputs. □

Lemma 6.2.7. *Let $f_1, f_2, f_3 \in VSP_c$ be three functions on n variables from the class VSP_c. Then the function*

$$f \triangleq \neg(f_1 \ OR \ f_2 \ OR \ f_3)$$

on the same n variables belongs to VSP_{4c} if $c \geq 4$ and $n \geq 10$.

Proof. For every $i = 1, 2, 3$, let C_i be the VSP circuit of size $\leq cn$ corresponding to the function f_i and let G_i be the underlying VSP multidag of C_i. Let the multidag G be the disjoint union of the underlying VSP multidags G_1, G_2, G_3, and let $\text{input}(G_i) = \{u_i^1, \ldots, u_i^n, u_i^{n+1}, u_i^{n+2}\}$ be the $n+2$ input nodes for C_i, $i = 1, 2, 3$ (see Definition 6.2.3), where the first n nodes $u_i^1, \ldots, u_i^n$ correspond to the n input variables, and u_i^{n+1} and u_i^{n+2} correspond to the two constant inputs 0 and 1, respectively. We have that $\text{input}(G) = \text{input}(G_1) \cup \text{input}(G_2) \cup \text{input}(G_3)$. Moreover, since $|\text{input}(G_i)| = n + 2$, $|\text{input}(G)| = 3n + 6$. Each C_i has only one output gate. Thus, $\text{output}(G_i) = \{o_i\}$ for some node o_i. Therefore, $|\text{output}(G)| = 3$.

A disjoint union of two VSP multidags is a multidag (see the proof of Lemma 3 in [Cal08]). Therefore, the multidag G is a VSP multidag. Let l be the labeling of G according to Definition 6.2.1 of VSP graphs. We construct a circuit C for the function f as follows. First, we let C be the disjoint union of C_1, C_2, C_3 (each C_i has its own $n + 2$ input nodes). Therefore, the underlying graph of C is G. Next, whenever we add a node or an edge to G, we do the same for C, and the other way around. As the circuits C_1, C_2, C_3 do not share their inputs, we add $n+2$ input nodes $u^1, \ldots, u^n, u^{n+1}, u^{n+2}$ to C (and to G). The first n input nodes $u^1, \ldots, u^n$ correspond to the n input variables, and the 2 input nodes u^{n+1} and u^{n+2} correspond to the two constant inputs 0 and 1, respectively. For every $j = 1, \ldots, n + 2$ and $i = 1, 2, 3$, we connect u^j to u_i^j.

For every newly added input node u^j, $j = 1, \ldots, n + 2$, we update the labeling: $l(u^j) = d-1$. As a result, the multidag G has $\text{input}(G) = \{u^1, \ldots, u^n, u^{n+1}, u^{n+2}\}$ and all weights of these nodes are equal to $d - 1$. It remain to verify the fourth property of VSP graphs. Since for every u^j, if (u^j, v) is an edge in G, then $l(v) = d$, the fourth property also holds. Thus G is VSP graph.

Now we have three functions f_1, f_2, f_3 on the same set of $n + 2$ inputs. To get the function $f = \neg(f_1 \text{ OR } f_2 \text{ OR } f_3)$, we add two more gates u_1, u_2 to the circuit C. We set the labeling: $l(u_1) \triangleq d' + 1$ and $l(u_2) \triangleq d' + 2$. u_1 is an OR gate, and it computes the OR of o_1 and o_2 (the outputs of the functions f_1 and f_2). The gate u_2 is a $\neg$OR

gate, and it computes the negation of the OR of u_1 (the OR of f_1 and f_2) and o_3 (the output of the function f_3). Since all gates of fan-in at most two are allowed, we can implement the $\neg$OR gate. We can check that C computes f (the negation of the OR of f_1, f_2, f_3). The size of the circuit C is at most $3cn+|\text{input}(G)|+2 = 3cn+3n+8 \leq 4cn$ as required. It is not hard to verify that the resulting labeling of u_1, u_2 and the rest of the multigraph G satisfies the properties from Definition 6.2.1. Thus, we conclude that the resulting underlying multidag G is a VSP graph and that C is a VSP_{4cn} circuit. □

6.3 VSP circuits and the orthogonal row problem

Circuit lower bounds and derandomization. The connection between derandomizing circuits and lower bounds originates in the works of Impagliazzo, Kabanets, and Wigderson [IKW02] and has been optimized in [Wil13, SW13, BSV14]. These connections rely on "succinct PCP" theorems [Mie09, BSV14], and the recent optimized construction of Ben-Sasson and Viola [BSV14] is crucial to our main result. Our starting point is the following theorem.

Theorem 6.3.1 (Theorem 1.4 in [BSV14])**.** *Let F_n be a family of function from $\{0,1\}^n$ to $\{0,1\}$ that is efficiently closed under projections (see Definition 6.2.5).*

If the fraction of satisfying assignment of a function of the form

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

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

We instantiate this theorem with VSP circuits and then simplify the resulting circuits.

Lemma 6.3.2. *To prove that E^{NP} does not have non-uniform Valiant series parallel circuits of size cn on n input variables, it is enough to show a deterministic algorithm that runs in $2^n/n^{\omega(1)}$ time for the following problem. Given a circuit over n input variables of the form:*

- *OR of fan-in $n^{O(1)}$ of*
- *negations of OR of fan-in 3 of*
- *VSP circuits of size cn,*

distinguish between the case where no assignments satisfy it, versus the case in which at least a $\geq 1 - 1/n^{10}$ fraction of the assignments satisfy it.

Lemma 6.3.2 follows from Theorem 6.3.1 almost directly: by Lemma 6.2.6, the class VSP_c (of functions recognizable by VSP circuits of size $\leq cn$) is efficiently closed under projections. Therefore, we can instantiate Theorem 6.3.1 on VSP_c. Since distinguishing the fraction of satisfying assignments $= 1$ from $\leq 1/n^{10}$ is equivalent to distinguishing the fraction of unsatisfying assignments $= 0$ from $\geq 1 - 1/n^{10}$, we get Lemma 6.3.2 by negating the function which is AND of OR of $F_{n+O(\log n)}$ and using De Morgan's law on the AND. Without loss of generality we replace the number of inputs $n + O(\log n)$ by n.

From derandomizing VSP circuits to the orthogonal row problem. Let C be the circuit on n variables as described in Lemma 6.3.2. We will convert this circuit into a simpler form that will be easier to work with when reducing to other problems. By Lemma 6.2.7, the circuit C can be interpreted as:

- OR of fan-in $n^{O(1)}$ of
- VSP circuits of size $4cn$,

where the $n^{O(1)}$ VSP circuits use the same set of n inputs.

We use the following classical theorem of Valiant to convert each of these VSP circuits into an OR of conjunctive normal form (CNF) formulas. The ideas in the proof are due to Valiant [Val77], but the details were shown by Calabro [Cal08] and Viola [Vio09] (cf. Cygan et al. [CDL+12]).

Theorem 6.3.3 (Depth reduction [Val77])**.** *For all $\ell \geq 1$, we can convert any VSP circuit of size $4cn$ on n variables into an equivalent formula which is OR of $2^{n/\ell}$ CNF formulas with clauses of size k (k-CNFs) on the same n variables, where $k = 2^{2^{\mu c \ell}}$ for some absolute constant $\mu > 0$. The reduction runs in $2^{n/\ell} n^{O(1)}$ time for any constants c and l.*

We will also need to apply the sparsification lemma [IP01, IPZ01].

Lemma 6.3.4. *For all $k \geq 3$ and $\varepsilon > 0$ we can convert a k-CNF formula on n variables into an equivalent OR of $2^{\varepsilon n}$ k-CNF formulas on the same variables where each CNF has $f(\varepsilon, k)n$ clauses and $f(\varepsilon, k) = (k/\varepsilon)^{O(k)}$.*

By combining the transformations of the circuits, we can simplify the circuits given in Lemma 6.3.2 as described in the following claim.

Claim 6.3.5. *Let C be the circuit on n variables as described in Lemma 6.3.2. For all $l \geq 1$ and $\varepsilon > 0$, we can convert C into an equivalent formula C' on the same set of n inputs of the following form:*

- *OR of fan-in $n^{O(1)} \cdot 2^{n/l} \cdot 2^{\varepsilon n}$ of*
- *AND of fan-in $f(\varepsilon, k)n$ where $k = 2^{2^{\mu c l}}$ of*
- *OR of fan-in k of literals.*

Proof. We can think of circuits C as OR of fan-in $n^{O(1)}$ of series parallel circuits of size $\leq 4cn$. We want to decide if C is unsatisfiable or at least a $1 - 1/n^{10}$ fraction of the assignments satisfy it. First, we apply Theorem 6.3.3 on every VSP circuit of size $\leq 4cn$. This produces a formula which is an OR of $2^{n/l}$ k-CNFs. Then, we apply the sparsification of Lemma 6.3.4 on every k-CNF to obtain a circuit as in the statement of the claim. □

The OR of AND of OR formula motivates the definition of the orthogonal row problem (see Definition 6.1.1). Recall that Hypothesis 6.1.2 states that the orthogonal row problem cannot be solved in a strongly sub-quadratic time with a deterministic algorithm. We are now ready to prove that refuting the hypothesis implies a circuit lower bound against linear size VSP circuits, thus establishing a "circuit lower bounds" barrier for refuting the hypothesis. The following claim implies Lemma 6.1.3.

Claim 6.3.6. *For all $c \geq 1$ and $\alpha > 0$, there exists a constant $d \geq 1$ such that if there is a deterministic algorithm that solves the orthogonal row problem on two lists of size N of binary $N^\alpha \times d \log N$ matrices in $N^2/\log^{\omega(1)} N$ time, then E^{NP} does not have non-uniform VSP circuits of size cn. The constant d can be upper bounded by*

$$d \leq 2^{2^{2^{O(c/\alpha)}}}.$$

Proof. By Lemma 6.3.2, to show that E^{NP} does not have non-uniform VSP circuits of size cn, it suffices to solve the derandomization problem (given in Lemma 6.3.2) on a circuit C with n variables in $2^n/n^{\omega(1)}$ time.

First, by Claim 6.3.5, we can transform the circuit C into an equivalent formula C' in the OR of AND of OR form. Below we show a reduction from the derandomization problem on the formula C' to the orthogonal row problem with the promised parameters, as follows.

Let $N \triangleq 2^{n/2}$. We apply the transformation from Claim 6.3.5 to C, with parameters $\varepsilon \triangleq \frac{\alpha}{6}$, $l \triangleq \frac{6}{\alpha}$, and $d \triangleq 2f(\varepsilon, k) \leq (k/\varepsilon)^{O(k)}$, and get an equivalent formula C' of the following form:

- OR of fan-in $n^{O(1)} \cdot 2^{n/\ell} \cdot 2^{\varepsilon n} \leq 2^{\alpha n/2} = N^\alpha$ of
- AND of fan-in $f(\varepsilon, k)n = d \log N \leq (k/\alpha)^{O(k)} n$ where $k = 2^{2^{\mu c \ell}} \leq 2^{2^{O(c/\alpha)}}$ of
- OR of fan-in k of literals.

We think of the formula C' as a disjunction of CNFs with clause size k.

Let us now transform C' to an instance of the orthogonal row problem. The formula C' has n binary input variables $x_1, \ldots, x_n$. We split these variables into two parts: $x_1, \ldots, x_{n/2}$ and $x_{1+(n/2)}, \ldots, x_n$, and construct two sets A and B of matrices for the orthogonal row problem.

Set of matrices A. Consider all $N = 2^{n/2}$ partial assignments of the first half of the variables $x_1, \ldots, x_{n/2}$. We will construct a matrix A^i, $i = 1, \ldots N$, one for each partial assignment p_i of $x_1, \ldots, x_{n/2}$ as follows. For every k-CNF in C' we have a

corresponding row in A^i, such that every clause of the k-CNF has a corresponding column. Thus, for $r = 1, \ldots, N^\alpha$, the r-th row of the matrix A^i corresponds to the r-th k-CNF in C', and every clause of the r-th k-CNF has a corresponding column in the r-th row such that the t-th clause corresponds to the t-th column in the r-th row of A^i. We set $A^i_{r,t}$ to 0 if p_i satisfies the t-th clause of the r-th k-CNF, and to 1 otherwise. A clause is satisfied by a *partial* assignment if and only if the partial assignment sets at least one of the literals in the clause to "true". We assume that the number of k-CNFs is N^α and the number of clauses in each k-CNF is $d \log N$. If this is not the case, then we can add dummy k-CNFs that are not satisfiable, or clauses that are satisfied by any partial assignment.

Set of matrices B. The second set of matrices B is constructed in the same way as the set A but for the second half of variables $x_{1+(n/2)}, \ldots, x_n$.

Our construction satisfies all the parameters of the orthogonal row problem. In particular, $d \leq (k/\varepsilon)^{O(k)} \leq 2^{2^{2^{O(c/\alpha)}}}$.

Correctness of the reduction. To prove the correctness of our reduction, it suffices to show that the fraction of pairs of matrices that form a good pair (see Definition 6.1.1), is the same as the fraction of assignments that satisfy the formula C'. We show that the i-th partial assignment of $x_1, \ldots, x_{n/2}$ and the j-th partial assignment of $x_{1+(n/2)}, \ldots, x_n$ satisfy C' if and only if the matrices A^i and B^j form a good pair. If C' is satisfied, then at least one of the k-CNFs in C' is satisfied. Assume that the r-th k-CNF is satisfied. Our goal is to show that $\sum_{h \in [D]} A^i_{r,h} B^j_{r,h} = 0$. This follows from the fact that the r-th k-CNF is satisfied and from the construction of the matrices A_i and B_j. If, on the other hand, $\sum_{h \in [D]} A^i_{r,h} B^j_{r,h} \geq 1$ for all r, then the i-th partial assignment of $x_1, \ldots, x_{n/2}$ and the j-th partial assignment of $x_{1+(n/2)}, \ldots, x_n$ do not satisfy C'. This follows from the fact that no k-CNF is satisfied in this case and from the construction of A^i and B^j.

We observe that if the number of satisfying assignments to C' is $1 - 1/n^{10}$, then the fraction of good pairs of matrices is $1 - 1/n^{10} \geq 1 - 2^{10}/n^{10} = 1 - 1/\log_2^{10} N$ as required. Since $N = 2^{n/2}$, an $N^2/\log^{\omega(1)} N$ algorithm for the orthogonal row problem implies a $2^n/n^{\omega(1)}$ time algorithm for the derandomization problem. □

6.4 The reduction to approximate LCS

In this section we prove Theorem 6.1.4. In our reduction we will use the weighted longest common subsequence problem (see Definition 4.2.1). We obtain a reduction to LCS problem by applying Lemma 4.2.2. Because of this connection we will write LCS instead of WLCS.

We show that if we have a deterministic algorithm for the LCS problem that runs in $O(n^{2-\delta})$ time and that gives $1 + (\varepsilon/10^5)$ approximation, then we can solve the orthogonal row problem in $O(N^{2-(\delta/2)})$ time on binary matrices of size $N^\alpha \times d \log N$ for $\alpha \triangleq \delta/10$ and $d \triangleq \alpha(1+\varepsilon)/\varepsilon$. The alphabet size of the sequences will be $O(N^{2\alpha}/\varepsilon)$.

This proves Theorem 6.1.4 by observing that for arbitrary small $\varepsilon > 0$, d becomes arbitrary large. Without loss of generality we assume that $\varepsilon \geq 1/\log n$ and $\delta \leq 1/100$.

We provide a deterministic reduction from the orthogonal row problem to the LCS problem. Let $A \triangleq \{A^1, \ldots, A^N\}$ and $B \triangleq \{B^1, \ldots, B^N\}$ be two sets of binary matrices given as an input to the orthogonal row problem. Each matrix is of size $N^\alpha \times d \log N$. We construct a sequence P from the set A and a sequence Q from the set B. The sequence P is of (weighted) length $n \triangleq W(P) \leq O(N^{1+2\alpha}d/\alpha)$ and the sequence Q is of length $W(Q) \leq O(n)$. For a fixed value T the sequences P and Q have the following property. $\mathrm{LCS}(P, Q) \leq T$ if we are in the first case in Definition 6.1.1 and $\mathrm{LCS}(P, Q) \geq (1 + (\varepsilon/10^5))T$ if we are in the second case.

If there exists a fast deterministic approximation algorithm for the LCS problem, we run it and decide in which case we are. Since the reduction is deterministic, this gives a deterministic algorithm for the orthogonal row problem that runs in time

$$O(n^{2-\delta}) \leq O\left(\left(N^{1+2\alpha}d/\alpha\right)^{2-\delta}\right) \leq O\left(\left(N^{1+(\delta/5)}/\varepsilon\right)^{2-\delta}\right),$$

where we use the fact that $\alpha = \delta/10$ and $d = \alpha(1+\varepsilon)/\varepsilon$. Since $\varepsilon \geq 1/\log n$ and $\delta \leq 1/100$, we get that the runtime is upper bounded by $O(N^{(1+(\delta/5))(2-\delta)} \log^2 N) \leq O\left(N^{2-(\delta/2)}\right)$ as required.

Construction of the matrix gadgets. To construct sequences P and Q, we need *matrix gadgets* $\mathrm{G}(A^i)$, $\mathrm{G}'(B^j)$ for every set $A^i \in A$ and $B^j \in B$. We want that there exists fixed value T' such that $\mathrm{G}(A^i)$ and $\mathrm{G}'(B^j)$ satisfy the property: $\mathrm{LCS}(\mathrm{G}(A^i), \mathrm{G}'(B^j)) = T'$ if the pair A^i, B^j is not good and $\mathrm{LCS}(\mathrm{G}(A^i), \mathrm{G}'(B^j)) = (1+\varepsilon)T'$ if the pair A^i, B^j is good. Below we will construct such matrix gadgets with $W(\mathrm{G}(A^i)), W(\mathrm{G}'(B^j)) \leq O(N^{2\alpha}d/\alpha)$. After that we will construct the final sequences P and Q with the promised properties by putting together matrix gadgets for all matrices in A and B.

We construct the matrix gadgets $\mathrm{G}(A^i)$, $\mathrm{G}'(B^j)$ by using the *orthogonality gadgets* $\mathrm{OG}(A^i_{k,*})$, $\mathrm{OG}'(B^j_{k,*})$ for every row $k \in [K]$ of matrices A^i and B^j. The orthogonality gadgets will satisfy the following properties.

- For every $k \in [K]$, $W(\mathrm{OG}(A^i_{k,*})), W(\mathrm{OG}'(B^j_{k,*})) \leq O(N^\alpha d/\alpha)$.
- If $\sum_{h\in[D]} A^i_{k,h}B^j_{k,h} \geq 1$, then $\mathrm{LCS}(\mathrm{OG}(A^i_{k,*}), \mathrm{OG}'(B^j_{k,*})) = T'$. Otherwise we have $\mathrm{LCS}(\mathrm{OG}(A^i_{k,*}), \mathrm{OG}'(B^j_{k,*})) = (1+\varepsilon)T'$.
- For every $k \in [K]$: $\mathrm{OG}(A^i_{k,*}), \mathrm{OG}'(B^j_{k,*}) \in \Sigma_k^*$ and $\Sigma_k \cap \Sigma_{k'} = \emptyset$ for $k \neq k'$.

Given such orthogonality gadgets, we construct matrix gadgets $\mathrm{G}(A^i)$ and $\mathrm{G}'(B^j)$ as follows:

$$\begin{aligned}\mathrm{G}(A^i) &\triangleq \textstyle\bigcirc_{k\in[K]}\mathrm{OG}(A^i_{k,*}) \\ &= \mathrm{OG}(A^i_{1,*})\,\mathrm{OG}(A^i_{2,*})\,\mathrm{OG}(A^i_{3,*})\,\ldots\,\mathrm{OG}(A^i_{K,*}),\end{aligned}$$

$$\begin{aligned}\mathrm{G}'(B^j) &\triangleq \bigcirc_{k\in[K]}\mathrm{OG}'(B^j_{K+1-k,*})\\ &= \mathrm{OG}'(B^j_{K,*})\,\mathrm{OG}'(B^j_{K-1,*})\,\mathrm{OG}'(B^j_{K-2,*})\,\ldots\,\mathrm{OG}'(B^j_{1,*}).\end{aligned}$$

Since $\Sigma_k \cap \Sigma_{k'} = \emptyset$ for $k \neq k'$, the only way to get $\mathrm{LCS}(\mathrm{G}(A^i), \mathrm{G}'(B^j)) > 0$ is by matching symbols in $\mathrm{OG}(A^i_{k,*})$ and $\mathrm{OG}'(B^j_{k,*})$. By the construction of $\mathrm{G}(A^i)$ and $\mathrm{G}'(B^j)$, if we match symbols between $\mathrm{OG}(A^i_{k,*})$ and $\mathrm{OG}'(B^j_{k,*})$, we cannot match symbols between $\mathrm{OG}(A^i_{k',*})$ and $\mathrm{OG}'(B^j_{k',*})$ for $k' \neq k$. This means that

$$\mathrm{LCS}(\mathrm{G}(A^i), \mathrm{G}'(B^j)) = \max_{k\in[K]} \mathrm{LCS}(\mathrm{OG}(A^i_{k,*}), \mathrm{OG}'(B^j_{k,*})).$$

The promised properties of the matrix gadgets follow from the properties of the orthogonality gadgets.

Construction of the orthogonality gadgets. Now we will construct the orthogonality gadgets $\mathrm{OG}(A^i_{k,*}), \mathrm{OG}'(B^j_{k,*})$. $A^i_{k,*}$ is a binary (row) vector of length $d \log N$. We split it into d/α binary vectors $v_t \in \{0,1\}^{\alpha \log N}$ each with $\alpha \log N$ entries: $A^i_{k,*} = (v_1, \ldots, v_{d/\alpha})$. We define

$$\mathrm{OG}(A^i_{k,*}) \triangleq c_k \circ \bigcirc_{t=1}^{d/\alpha} s_{k,t,v_t},$$

where we set the weight of the symbol c_k to $w(c_k) \triangleq (d/\alpha) - 1$. s_{k,t,v_t} are symbols of weight 1 and indexed by rows k, indices of vectors t and vectors v_t. We have $\mathrm{OG}(A^i_{k,*}) \in \Sigma_k^*$, where

$$\Sigma_k \triangleq \{c_k\} \cup \{s_{k,t,v} \mid v \in \{0,1\}^{\alpha \log N} \text{ and } t \in [d/\alpha]\}.$$

Similarly, we split the binary vector $B^j_{k,*}$ of length $d \log N$ into d/α binary vectors $w_t \in \{0,1\}^{\alpha \log N}$ each with $\alpha \log N$ entries: $B^j_{k,*} = (w_1, \ldots, w_{d/\alpha})$. We define

$$\mathrm{OG}'(B^j_{k,*}) \triangleq \left(\bigcirc_{t=1}^{d/\alpha} \bigcirc_{v\,:\,v\cdot w_t=0} s_{k,t,v}\right) \circ c_k,$$

where we do enumerate over all vectors v that are orthogonal $v \cdot w_t$ to w_t. Notice that $W(\mathrm{OG}(A^i_{k,*})) \leq O(d/\alpha)$ and $W(\mathrm{OG}'(B^j_{k,*})) \leq O(N^\alpha d/\alpha)$ as required.

We claim that, if $\sum_{h\in[D]} A^i_{k,h} B^j_{k,h} \geq 1$, then

$$\mathrm{LCS}(\mathrm{OG}(A^i_{k,*}), \mathrm{OG}'(B^j_{k,*})) = T' \triangleq (d/\alpha) - 1$$

and $\mathrm{LCS}(\mathrm{OG}(A^i_{k,*}), \mathrm{OG}'(B^j_{k,*})) = d/\alpha = T' + 1$ otherwise. Since $d = \alpha(1+\varepsilon)/\varepsilon$, we have that T' satisfies the second property of the orthogonality gadgets. We now show the claim.

Clearly, $\mathrm{LCS}(\mathrm{OG}(A^i_{k,*}), \mathrm{OG}'(B^j_{k,*})) \geq (d/\alpha) - 1$ because we can match the symbols c_k. Also, we have the equality if we match the symbols c_k in the optimal alignment. Suppose that we do not match c_k. Then it is not hard to check that

$\mathrm{LCS}(\mathrm{OG}(A^i_{k,*}), \mathrm{OG}'(B^j_{k,*})) = d/\alpha$ if $\sum_{h\in[D]} A^i_{k,h}B^j_{k,h} = 0$ and $\leq (d/\alpha) - 1$ otherwise. Since we have to take maximum between the cases when we match the symbols c_k and when we do not match c_k, we get the required equalities.

Construction of the sequences P and Q. In the remainder of the proof we construct the final sequences P and Q with the promised properties. The sequence P is a concatenation of the matrix gadgets $\mathrm{G}(A^i)$ with some additional symbols. The sequence Q is a concatenation of the matrix gadgets $\mathrm{G}'(B^j)$ with some additional symbols. Each matrix gadget $\mathrm{G}'(B^j)$ appears twice in the second sequence Q.

We define integer values $v_0 < v_1 < v_2 < v_3$ as follows. We set $v_0 \triangleq T'$ (see the construction of the matrix gadgets), $v_1 \triangleq (1+\varepsilon)T'$, $v_2 \triangleq 10v_1$, $v_3 \triangleq 100v_1$. For the simplicity of the notation, we will write A^i instead of $\mathrm{G}(A^i)$ and B^j instead of $\mathrm{G}'(B^j)$. It will be clear from the context whether we refer to A^i (B^j, resp.) or to $\mathrm{G}(A^i)$ ($\mathrm{G}'(B^j)$, resp.). We introduce new symbols of type $0, 1$ and 2 and define the sequence P:

$$P \triangleq 2^{3N} \circ \left(\bigcirc_{i=1}^{N} (0\, A^i\, 1)\right) \circ 2^{3N}.$$

We define the sequence Q:

$$Q \triangleq \left(\bigcirc_{j=1}^{N} \left(2\, 0\, B^j\, 1\right)\right) \circ \left(\bigcirc_{j=1}^{N} \left(2\, 0\, B^j\, 1\right)\right) \circ 2.$$

We set the weight of the symbols of type 0, 1 and 2 as follows: $w(2) \triangleq v_2$ and $w(0) = w(1) \triangleq v_3$.

We have two goals. First, we want to show that if there are many good pairs of matrices A^i and B^j, then the LCS score between P and Q is large: $\mathrm{LCS}(P,Q) \geq (1 + (\varepsilon/10^5))T$. T is some fixed value that we will define later. Second, if there are no good pairs, then the LCS score is small: $\mathrm{LCS}(P,Q) \leq T$. We achieve these two goals via the next two lemmas.

Lemma 6.4.1. *If there are many good pairs (see Definition 6.1.1), then*

$$\mathit{LCS}(P,Q) \geq T'' \triangleq (N+2)v_2 + 2Nv_3 + 0.99Nv_1 + 0.01Nv_0.$$

Proof. We will exhibit N different alignments between P and Q and we will show that at least one of them achieves the LCS score T''. This gives the lower bound on $\mathrm{LCS}(P,Q)$.

For $k = 1, \ldots, N$ we write

$$Q = \left(\bigcirc_{j=1}^{k-1} \left(2\, 0\, B^j\, 1\right)\right) \circ 2 \circ Q_k \circ \left(\bigcirc_{j=k}^{N} \left(2\, 0\, B^j\, 1\right)\right) \circ 2,$$

where

$$Q_k \triangleq (0\, B^k\, 1) \circ \left(\bigcirc_{j=k+1}^{N} \left(2\, 0\, B^j\, 1\right)\right) \circ \left(\bigcirc_{j=1}^{k-1} \left(2\, 0\, B^j\, 1\right)\right).$$

Clearly,

$$\begin{aligned}\mathrm{LCS}(P,Q) \geq\ & \mathrm{LCS}\left(2^{3N}, \left(\bigcirc_{j=1}^{k-1} \left(2\, 0\, B^j\, 1\right)\right) \circ 2\right) \\ & + \mathrm{LCS}\left(\bigcirc_{i=1}^{N} (0\, A^i\, 1), Q_k\right)\end{aligned}$$

$$+ \operatorname{LCS}\left(2^{3N}, \left(\bigcirc_{j=k}^{N} \left(2\,0\,B^j\,1\right)\right) \circ 2\right).$$

The total contribution of the first and the third term on the right hand side is $(N+2)v_2$ because only symbols of type 2 can contribute to the LCS score and there are $N+2$ symbols of type 2. For the middle term we align matrix gadgets in pairs and match all symbols of type 0 and 1. We get the lower bound

$$\operatorname{LCS}(P,Q) \geq (N+2)v_2 + 2Nv_3 + \sum_{i=1}^{N} \operatorname{LCS}(A^i, B^{j_k(i)}),$$

where

$$j_k(i) \triangleq \begin{cases} i+k-1 & \text{if } i \leq N+1-k, \\ i+k-1-N & \text{otherwise.} \end{cases}$$

By averaging the right hand side over all $k = 1, \ldots, N$, we get

$$\begin{aligned} \operatorname{LCS}(P,Q) &\geq \frac{1}{N}\sum_{k=1}^{N}\left((N+2)v_2 + 2Nv_3 + \sum_{i=1}^{N}\operatorname{LCS}(A^i, B^{j_k(i)})\right) \\ &= (N+2)v_2 + 2Nv_3 + \frac{1}{N}\sum_{i,k=1}^{N}\operatorname{LCS}(A^i, B^{j_k(i)}) \\ &= (N+2)v_2 + 2Nv_3 + \frac{1}{N}\sum_{i,j=1}^{N}\operatorname{LCS}(A^i, B^{j}) \\ &\geq (N+2)v_2 + 2Nv_3 + 0.99Nv_1 + 0.01Nv_0, \end{aligned}$$

where in the last inequality we use the fact that there are many good pairs. This finishes the proof of the lemma. □

Lemma 6.4.2. *If there are no good pairs, then*

$$\mathit{LCS}(P,Q) \leq T \triangleq (N+2)v_2 + 2Nv_3 + Nv_0.$$

Proof. We start with the intuition behind the analysis.

Intuition. We saw in the proof of Lemma 6.4.1 that there is an alignment that achieves a large LCS score. In the alignment we match the N matrix gadgets from the first sequence P with N consecutive matrix gadgets from the second sequence Q in pairs. We want to claim that in an optimal alignment, we will do the same: map the N matrix gadgets from the first sequence with N consecutive matrix gadgets from the second sequence in pairs. Intuitively, this is because of the following three reasons:

- We do not want to choose less than N matrix gadgets from the second sequence because otherwise we cannot match all symbols of type 0 and 1 from the first sequence with their counterparts (symbols of type 0 and 1 have the largest weight—we loose a lot by not matching them).

- We do not want to choose more than N matrix gadgets from the second sequence because otherwise we have fewer symbols of type 2 from the second sequence to be matched with their counterparts. Symbols 2 have smaller weight than symbols of type 0 and 1 but still we loose a lot by not matching them.

- Finally, if we choose N matrix gadgets from the second sequence we want to match them in pairs. If we do not do that, we cannot match all symbols of type 0 and 1 which is again expensive.

We proceed to formalize the intuition.

Sequence P starts with $3N$ copies of the symbol of type 2. Suppose that some of those symbols are matched. If this is not the case, we can match one of these symbols with the first symbol of type 2 from Q without decreasing the LCS score. Without loss of generality the matched symbols form a suffix of $\bigcirc_{i=1}^{3N} 2$. Consider the symbol of type 2 from Q that is matched to the last symbol of type 2 from $\bigcirc_{i=1}^{3N} 2$. Consider the symbol to the right of the symbol of type 2 in Q. It is of type 0. Let s be its position in Q. Without loss of generality this symbol of type 0 is matched to the first symbol of type 0 from P. If this is not so, we can make this match and this cannot decrease the LCS score. Analogously we can argue that the last symbol of type 1 from P is matched to a symbol of type 1 in Q. Let t be the location of the symbol of type 1 in Q. Let P' be the substring of P that is to the right of the first symbol of type 0 in P and to the left of the last symbol of type 1 in P. Let Q' be the substring of Q that is to the right of the symbol of type 0 at location s in Q and to the left of the symbol of type 1 at location t in Q. We write $P = P_1\, P'\, P_2$ and $Q = Q_1\, Q'\, Q_2$. We can upper bound the LCS score if we range over all such partitions of P and Q:

$$\mathrm{LCS}(P, Q) \leq \max_{\substack{P=P_1\,P'\,P_2 \\ Q=Q_1\,Q'\,Q_2}} \mathrm{LCS}(P_1, Q_1) + \mathrm{LCS}(P', Q') + \mathrm{LCS}(P_2, Q_2). \tag{6.1}$$

Let $m \geq 1$ denote the number of matrix gadgets in Q'.

Claim 6.4.3.

$$\mathit{LCS}(P_1, Q_1) + \mathit{LCS}(P_2, Q_2) \leq 2v_3 + (2N + 1 - (m-1))v_2.$$

Proof. The total number of symbols of type 2 in Q_1 and Q_2 is $2N + 1 - (m-1)$. The total contribution from all symbols of type 2 is upper bounded by $(2N+1-(m-1))v_2$. We can also match symbol of type 0 in P_1 and symbol of type 1 in P_2. This upper bounds the total contribution from symbols of type 0 and 1 by $2v_3$. There are no other symbols that we can match. The claim follows. □

It remains to give an upper bound on $\mathrm{LCS}(P', Q')$. For any two symbols of type 0 that are matched between P' and Q', their neighboring symbols of type 1 form a match too. If this does not true, we match the symbols of type 1 and this can only increase the LCS score. Similarly, for any two symbols of type 1 that are matched, their neighboring symbols of type 0 are matched too. Let $M \geq 0$ denote the number of pairs of matched symbols of type 0 and 1. This allows us to upper bound the

total contribution from symbols of type 0 and 1 to $\mathrm{LCS}(P', Q')$ by $S \triangleq 2Mv_2$. The M pairs of matched symbols split the sequence P' into $M+1$ maximal substrings $r_1, \ldots, r_{M+1}$. In each one of the $M+1$ substrings r_i the symbols of type 0 or 1 are not matched. Similarly, we split Q' into $M+1$ maximal substrings $p_1, \ldots, p_{M+1}$ so that each p_i does not contain a symbol of type 0 or 1 that is matched. Symbols in r_i can only be matched to symbols in p_i. The only symbols that can be matched from r_i with symbols from p_i come from the matrix gadgets by the definition of r_i and p_i. Let $d_i \geq 1$ denote the number of the matrix gadgets in r_i and $l_i \geq 1$ denote the number of the matrix gadgets in p_i. Clearly, $\sum_{i=1}^{M+1} d_i = N$ and $\sum_{i=1}^{M+1} l_i = m$. We claim that $\mathrm{LCS}(r_i, p_i) \leq (d_i + l_i - 1)v_0$. Since the pairs of matched symbols cannot cross, we can easily check that the total number of pairs of matrix gadgets that can have a match is upper bounded by $d_i + l_i - 1$. Because there are no good pairs, the upper bound $\mathrm{LCS}(r_i, p_i) \leq (d_i + l_i - 1)v_0$ follows. From this we have

$$\mathrm{LCS}(P', Q') \leq S + \sum_{i=1}^{M+1} \mathrm{LCS}(r_i, p_i) \leq 2Mv_2 + \sum_{i=1}^{M+1} (d_i + l_i - 1)v_0.$$

We combine this with the equalities $\sum_{i=1}^{M+1} d_i = N$ and $\sum_{i=1}^{M+1} l_i = m$ and get the following Claim.

Claim 6.4.4.

$$LCS(P', Q') \leq 2Mv_3 + (N+m)v_0 - (M+1)v_0.$$

We combine Eq. (6.1) with Claim 6.4.3 and Claim 6.4.4 and get the following upper bound:

$$\begin{aligned}\mathrm{LCS}(P, Q) \leq\ & 2v_2 + N(2v_2 + v_0) \\ & + (M+1)(2v_3 - v_0) - m(v_2 - v_0).\end{aligned}$$

From $\sum_{i=1}^{M+1} d_i = N$ and $\sum_{i=1}^{M+1} l_i = m$ we get that $M \leq \min(N, m) - 1$. As we increase M, the right hand side of the upper bound only increases. We choose $M = \min(N, m) - 1$. Consider two cases.

- $m \geq N$. We have $M = N-1$ and $\mathrm{LCS}(P, Q) \leq v_2(2N+2) + 2Nv_3 - m(v_2 - v_0) \leq T$.
- $m \leq N$. We have $M = m-1$ and $\mathrm{LCS}(P, Q) \leq v_2(2N+2) + Nv_0 + m(2v_3 - v_2) \leq T$.

□

From the above Lemmas 6.4.1 and 6.4.2 we have that $\mathrm{LCS}(P, Q) \geq T''$ if there are many good pairs and $\mathrm{LCS}(P, Q) \leq T$ if there are no good pairs. From the definition of values v_0, v_1, v_2, v_3 (in particular, $v_1 = (1+\varepsilon)v_0$), we can easily conclude that $T'' \geq (1 + (\varepsilon/10^5))T$ which gives the properties of P and Q that we need.

Harder variants of the orthogonal row problem. In the paragraph "Construction of the orthogonality gadgets" we do the following construction. Given two vectors $z^k \triangleq A^i_{k,*}, w^k \triangleq B^j_{k,*} \in \{0,1\}^{d \log N}$, we construct sequences $\mathrm{OG}(z^k)$ and $\mathrm{OG}(w^k)$ such that LCS between them is $\mathrm{LCS}(z^k, w^k) = d/\alpha$ if the vectors are orthogonal and $\mathrm{LCS}(z^k, w^k) = (d/\alpha) - 1$ otherwise. We split the vector z^k into d/α shorter vectors $z^k_1, \ldots, z^k_{d/\alpha} \in \{0,1\}^{\alpha \log N}$. Similarly, we split the vector w^k into d/α shorter vectors $w^k_1, \ldots, z^k_{d/\alpha} \in \{0,1\}^{\alpha \log N}$. We construct $\mathrm{OG}(z^k)$ by replacing each shorter vector z^k_t by a symbol corresponding to it (indexed by the $\alpha \log N$ binary values) and its position. We construct $\mathrm{OG}(w^k)$ by replacing each shorter vector w^k_t by a sequence of symbols corresponding to all vectors that are orthogonal to w^k_t. This implies that we have a large LCS score if there are many orthogonal pairs z^k_t, w^k_t of short vectors. Instead of replacing w^k_t by a sequence of symbols corresponding to all orthogonal vectors, we can take an arbitrary function $f^k_t : \{0,1\}^{2\alpha \log N} \to \{0,1\}$ and replace w^k_t by a sequence of symbols corresponding to all vectors $u \in \{0,1\}^{\alpha \log N}$ such that $f^k_t(u, w^k_t) = 1$. We recover the orthogonality constraint by choosing functions f^k_t that evaluates to 1 if and only if the two vectors are orthogonal. For arbitrary functions $f^k_1, \ldots, f^k_{d/\alpha} : \{0,1\}^{2\alpha \log N} \to \{0,1\}$, we get that $\mathrm{LCS}(z^k, w^k) = d/\alpha$ if $f^k_1(z^k_1, w^k_1) = \ldots = f^k_{d/\alpha}(z^k_{d/\alpha}, w^k_{d/\alpha}) = 1$ and $\mathrm{LCS}(z^k, w^k) = (d/\alpha) - 1$ otherwise. Clearly, the new version of orthogonal row problem is harder to solve than the one restricted to the orthogonality constraints.

To further increase the hardness of the orthogonal row problem we can define functions $g^k : \{0,1\}^{d/\alpha} \to \{0,1\}$ and require that $\mathrm{LCS}(z^k, w^k) = q$ if

$$g^k(f^k_1(z^k_1, w^k_1), \ldots, f^k_{d/\alpha}(z^k_{d/\alpha}, w^k_{d/\alpha})) = 1$$

and $\mathrm{LCS}(z^k, w^k) = q' < q$ otherwise (for some fixed values q and q'). Notice that previously all functions g^k are AND functions. This modification requires that the gap $(q/q') - 1$ is at least a constant (we have a constant gap for the AND function) and that the functions g^k can be efficiently simulated with LCS.

6.5 Hardness for binary LCS and edit distance

The results in this section follow from simple observations over [AHVWW16] that are easy to make with our framework in mind. We refer the reader to [AHVWW16] for the definition and background on branching programs.

Theorem 6.5.1 (Theorem 2 in [AHVWW16])**.** *There is a reduction from SAT on nondeterministic branching programs on m variables, length T, and width W, to an instance of edit distance or LCS on two binary sequences P and Q of length $n \triangleq 2^{m/2} T^{O(\log W)}$, and the reduction runs in $O(n)$ time.*

See also [AB18] for reductions between circuits and LCS. We need the following additional properties of the reduction from Theorem 6.5.1.

Claim 6.5.2. *Let P be the branching program that we want to reduce.*

If we reduce a branching program X to LCS problem then we have the following two properties:

- *If the branching problem X is not satisfiable, then $LCS(P, Q) \leq C$ for some integer value $C = C(n, m, T, W) \leq n$.*
- *If at least half of the assignments satisfy the branching program X, then $LCS(x, y) \geq C + (2^{m/2}/2)$.*

If we reduce the branching program P to edit distance problem then we have the following two properties:

- *If the branching program X is not satisfiable, then $edit(P, Q) \geq C$ for some integer value $C = C(n, m, T, W) \leq n$.*
- *If at least half of the assignments satisfy the branching program X, then $edit(P, Q) \leq C - (2^{m/2}/2)$.*

Proof. The proof follows from the proof of Claim 9 in [AHVWW16]. We briefly sketch the changes.

Consider the case when X is not satisfiable. The proof does not change—we show that LCS is upper bounded and the edit distance is lower bounded by some fixed quantity C.

Consider the case when X is satisfied by at least half of the assignments. In the proof of Claim 9 the authors choose an integer Δ such that the corresponding alignment pairs up two gadgets that form a satisfying assignment to the branching program X. When there are many satisfying assignments (at least half), we can show that there is an integer such in the corresponding alignment at least half of the assignments are satisfying. By the properties of the gadgets constructed in [AHVWW16], we get the required lower bound on LCS and the required upper bound on the edit distance. □

Theorem 6.5.1 and Claim 6.5.2 combined give the following result.

Theorem 6.5.3. *Given a branching program on m variables, length T and width W, for some $L \leq T^{O(\log W)}$ it is possible to construct two sequences of length $n \leq O(2^{m/2}L)$ over a binary alphabet in $O(n)$ time such that the following holds. If we can approximate LCS between the sequence within the factor of $1 + 1/L$ in time $f(n)$, then in the same time we can distinguish the case when the branching program is not satisfiable from the case when the branching program has more than half of its assignments satisfying. The same statement holds for the edit distance problem.*

From the discussion in [AHVWW16] on the connection between branching programs and NC circuits we obtain the following barrier. A deterministic algorithm that achieves $1 + 1/(\log n)^{\omega(1)}$ approximation for LCS or the edit distance implies that there exists $f \in \mathsf{E}^{\mathsf{NP}}$ such that $f \notin \mathsf{NC}^1$.

Part II

Statistical data analysis and machine learning

Chapter 7

Empirical risk minimization

Empirical risk minimization (ERM) is ubiquitous in machine learning and underlies most supervised learning methods. While there is a large body of work on algorithms for various ERM problems, the exact computational complexity of ERM is still not understood. Since the problems have polynomial time algorithms, the classical machinery from complexity theory (such as NP hardness) is too coarse to apply. Oracle lower bounds from optimization offer useful guidance for convex ERM problems, but the results only hold for limited classes of algorithms. Moreover, they do not account for the *cost* of executing the oracle calls, as they simply lower bound their number. Overall, we do not know if common ERM problems allow for algorithms that compute a high-accuracy solution in sub-quadratic or even nearly-linear time for all instances.[1] Furthermore, we do not know if there are more efficient techniques for computing (mini)-batch gradients than simply treating each example in the batch independently.[2]

We study the exact computational complexity of multiple popular ERM problems including kernel SVMs, kernel ridge regression, and training the final layer of a neural network. In particular, we give conditional hardness results for these problems based on SETH. Under this hardness assumption we show that there are no algorithms that solve the aforementioned ERM problems to high accuracy in sub-quadratic time. We also give similar lower bounds for computing the gradient of the empirical loss, which is the main computational burden in many non-convex learning tasks.

Our hardness results for the gradient computation apply to common activation functions such as ReLU or sigmoid units. We remark that for *polynomial* activation functions (for instance, studied in [LSSS14]), significantly faster algorithms *do exist* (see Section 7.7). Thus, our results can be seen as mapping the "efficiency landscape" of basic machine learning sub-routines. They distinguish between what is possible and (likely) impossible, suggesting further opportunities for improvement.

[1] More efficient algorithms exist if the running time is allowed to be polynomial in the accuracy parameter, e.g., [SSSS07] give such an algorithm for the kernel SVM problem that we consider as well. See also the discussion in Section 7.1.4.

[2] Consider a network with one hidden layer containing n units and a training set with m examples, for simplicity in small dimension $d = O(\log n)$. No known results preclude an algorithm that computes a full gradient in time $O((n + m) \log n)$. This would be significantly faster than the standard $O(n \cdot m \cdot \log n)$ approach of computing the full gradient example by example.

7.1 Our contributions

We obtain our conditional hardness results via reductions from the orthogonal vectors problem and from the bichromatic Hamming close pair problem.

For the orthogonal vectors problem we can assume without loss of generality that all vectors in set B have the same number of entries equal to 1. This can be achieved by appending d entries to every $b \in B$ and setting the necessary number of them to 1 and the rest to 0. We then append d entries to every $a \in A$ and set all of them to 0.

7.1.1 Kernel ERM problems

We provide hardness results for multiple kernel problems. In the following, let $x^1, \ldots, x^n \in \mathbb{R}^d$ be the n input vectors, where $d = \omega(\log n)$. We use $y_1, \ldots, y_n \in \mathbb{R}$ as n labels or target values. Finally, let $k(x, x')$ denote a kernel function and let $K \in \mathbb{R}^{n \times n}$ be the corresponding kernel matrix, defined as $K_{i,j} \triangleq k(x^i, x^j)$ [SS01]. Specifically, we focus on the Gaussian kernel $k(x, x') \triangleq \exp\left(-C\|x - x'\|_2^2\right)$ for some $C > 0$. We note that our results can be generalized to any kernel with exponential tail.

Kernel SVM. For simplicity, we present our result for hard-margin SVMs without bias terms. This gives the following optimization problem.

Definition 7.1.1 (Hard-margin SVM)**.** *A (primal) hard-margin SVM is an optimization problem of the following form:*

$$\begin{aligned} \underset{\alpha_1,\ldots,\alpha_n \geq 0}{\text{minimize}} \quad & \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \, \alpha_j \, y_i \, y_j \, k(x^i, x^j) \\ \text{subject to} \quad & y_i \, f(x^i) \geq 1, \quad i = 1, \ldots, n, \end{aligned} \tag{7.1}$$

where $f(x) \triangleq \sum_{i=1}^{n} \alpha_i \, y_i \, k(x^i, x)$.

The following theorem is our main result for SVMs, described in more detail in Section 7.3.1. In Sections 7.3.2 and 7.3.3 we provide similar hardness results for other common SVM variants, including the soft-margin version.

Theorem 7.1.2. *Let* $k(x, x')$ *be the Gaussian kernel with* $C \triangleq 100 \log n$ *and let* $\varepsilon \triangleq \exp(-\omega(\log^2 n))$*. Then approximating the optimal value of Eq.* (7.1) *within the multiplicative factor* $1 + \varepsilon$ *requires almost quadratic time assuming SETH.*

Kernel Ridge Regression. Next we consider kernel ridge regression, which is formally defined as follows.

Definition 7.1.3 (Kernel ridge regression)**.** *Given a real value* $\lambda \geq 0$*, the goal of the kernel ridge regression is to output*

$$\arg\min_{\alpha \in \mathbb{R}^n} \frac{1}{2}\|y - K\alpha\|_2^2 + \frac{\lambda}{2}\alpha^{\mathsf{T}} K \alpha.$$

This problem is equivalent to computing the vector $(K+\lambda I)^{-1}y$. We focus on the special case where $\lambda = 0$ and the vector y has all equal entries $y_1 = \ldots = y_n = 1$. In this case, the sum of entries of $K^{-1}y$ is equal to the sum of the entries in K^{-1}. Thus, we show hardness for approximating the latter quantity (see Section 7.4).

Theorem 7.1.4. *Let $k(x, x')$ be the Gaussian kernel for any parameter $C \triangleq \omega(\log n)$ and let $\varepsilon \triangleq \exp(-\omega(\log^2 n))$. Then approximating the sum of the entries in K^{-1} up to the multiplicative factor of $1+\varepsilon$ requires almost quadratic time assuming SETH.*

Kernel PCA. Finally, we turn to the kernel PCA problem, which we define as follows [Mur12].

Definition 7.1.5 (Kernel principal component analysis (PCA))**.** *Let 1_n be an $n \times n$ matrix where each entry takes value $1/n$, and define $K' \triangleq (I - 1_n)K(I - 1_n)$. The goal of the kernel PCA problem is to output the n eigenvalues of the matrix K'.*

In the above definition, the output only consists of the eigenvalues, not the eigenvectors. This is because computing all n eigenvectors trivially takes at least quadratic time since the output itself has quadratic size. Our hardness proof applies to the potentially simpler problem where only the eigenvalues are desired. Specifically, we show that computing *the sum* of the eigenvalues (i.e., the *trace* of the matrix) is hard. See Section 7.5 for the proof.

Theorem 7.1.6. *Let $k(x, x')$ be the Gaussian kernel with $C \triangleq 100 \log n$ and let $\varepsilon \triangleq \exp(-\omega(\log^2 n))$. Then approximating the sum of the eigenvalues of $K' \triangleq (I - 1_n)K(I - 1_n)$ within the multiplicative factor of $1+\varepsilon$ requires almost quadratic time assuming SETH.*

All lower bounds for kernel problems in this sections can be further strengthened to any $\varepsilon = n^{-\omega(1)}$ by using [Rub18]. This conditionally rules out algorithms of the form $n^{2-\Omega(1)}/\varepsilon^{o(1)}$.

7.1.2 Neural network ERM problems

We now consider neural networks. We focus on the problem of optimizing the top layer while keeping lower layers unchanged. An instance of this problem is transfer learning with large networks that would take a long time and many examples to train from scratch [RASC14]. We consider neural networks of depth-2, with the sigmoid or ReLU activation function. Our hardness result holds for a more general class of "nice" activation functions S as described later (see Definition 7.6.2).

Given n weight vectors $w^1, \ldots, w^n \in \mathbb{R}^d$ and n weights $\alpha_1, \ldots, \alpha_n \in \mathbb{R}$, consider the function $f : \mathbb{R}^d \to \mathbb{R}$ using a non-linearity $S : \mathbb{R} \to \mathbb{R}$:

$$f(x) \triangleq \sum_{j=1}^{n} \alpha_j S(x \cdot w^j) .$$

This function can be implemented as a neural net that has d inputs, n non-linear activations (units), and one linear output.

To complete the ERM problem, we also require a loss function. Our hardness results hold for a large class of "nice" loss functions, which includes the hinge loss and the logistic loss.[3] Given a nice loss function and m input vectors $x^1, \ldots, x^m \in \mathbb{R}^d$ with corresponding labels y_i, we consider the following problem:

$$\underset{\alpha_1,\ldots,\alpha_n \in \mathbb{R}}{\text{minimize}} \quad \sum_{i=1}^{m} \text{loss}(y_i, f(x^i)). \tag{7.2}$$

Our main result is captured by the following theorem (see Section 7.6 for the proof). For simplicity, we set $m = n$.

Theorem 7.1.7. *For any $d \triangleq \omega(\log n)$, approximating the optimal value in Eq. (7.2) up to a multiplicative factor of $1 + \frac{1}{4n}$ requires almost quadratic time assuming SETH.*

7.1.3 Gradient computation

Finally, we consider the problem of computing the gradient of the loss function for a given set of examples. We focus on the network architecture from the previous section. Formally, we obtain the following result:

Theorem 7.1.8. *Consider the empirical risk in Eq. (7.2) under the following assumptions: (i) The function f is represented by a neural network with n units, nd parameters, and the ReLU activation function. (ii) We have $d \triangleq \omega(\log n)$. (iii) The loss function is the logistic loss or hinge loss. Then approximating the ℓ_p-norm (for any $p \geq 1$) of the gradient of the empirical risk for m examples within a multiplicative factor of n^C for any constant $C > 1$ takes at least $\Omega\left((nm)^{1-o(1)}\right)$ time assuming SETH.*

See Section 7.7 for the proof. We also prove a similar statement for the sigmoid activation function. At the same time, we remark that for *polynomial* activation functions, significantly faster algorithms do exist, using the polynomial lifting argument. Specifically, for the polynomial activation function of the form $S(z) = z^r$ for some integer $r \geq 2$, all gradients can be computed in $O((n+m)d^r)$ time. Note that the running time of the standard backpropagation algorithm is $O(dnm)$ for networks with this architecture. Thus, it is possible to improve over backpropagation for a non-trivial range of parameters, especially for quadratic activation function when $r = 2$. See Section 7.7.

7.1.4 Related work

There is a long line of work on the oracle complexity of optimization problems, going back to [NY83]. We refer the reader to [Nes04] for these classical results.

[3] In the binary setting we consider, the logistic loss is equivalent to the softmax loss commonly employed in deep learning.

The oracle complexity of ERM problems is still subject of active research, e.g., see [AB15, CBMS15, WS16, AS16, AS17]. The work closest to ours is [CBMS15], which gives quadratic time lower bounds for ERM algorithms that access the kernel matrix through an evaluation oracle or a low-rank approximation.

The oracle results are fundamentally different from the lower bounds presented in this section. Oracle lower bounds are typically unconditional, but inherently apply only to a limited class of algorithms due to their information-theoretic nature. Moreover, they do not account for the *cost* of executing the oracle calls, as they merely lower bound their number. In contrast, our results are conditional (based on the SETH and related assumptions), but apply to *any* algorithm and account for the *total* computational cost. This significantly broadens the reach of our results. We show that the hardness is not due to the oracle abstraction but instead inherent in the computational problem. To the best of our knowledge, our result is the first application of this methodology to *continuous* (as opposed to combinatorial) optimization problems.

Finally, we note that our results do not rule out algorithms that achieve a sub-quadratic running time for well-behaved instances, e.g., instances with low-dimensional structure. Indeed, many such approaches have been investigated in the literature, for instance the Nyström method or random features for kernel problems [WS01, RR08]. Our results offer an explanation for the wide variety of techniques. The lower bounds are evidence that there is no "silver bullet" algorithm for solving the aforementioned ERM problems in sub-quadratic time, to high accuracy, and for all instances.

7.2 Preliminaries

In this section we define several notions used later in this section. We start from the soft-margin support vector machine (see [MMR+01]).

Definition 7.2.1 (Support vector machine (SVM))**.** *Let $x^1, \ldots, x^n \in \mathbb{R}^d$ be n vectors and $y_1, \ldots, y_n \in \{-1, 1\}$ be n labels. Let $k(x, x')$ be a kernel function. An optimization problem of the following form is a (primal) SVM.*

$$\begin{aligned} \underset{\substack{\alpha_1,\ldots,\alpha_n \geq 0,\\ \xi_1,\ldots,\xi_n \geq 0}}{\text{minimize}} \quad & \frac{\lambda}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j k(x^i, x^j) \; + \; \frac{1}{n} \sum_{i=1}^{n} \xi_i \\ \text{subject to} \quad & y_i f(x^i) \geq 1 - \xi_i, \quad i = 1, \ldots, n, \end{aligned}$$

where $f(x) \triangleq b + \sum_{i=1}^{n} \alpha_i y_i k(x^i, x)$ and $\lambda \geq 0$ is called the regularization parameter. ξ_i are known as the slack variables.

The dual SVM is defined as

$$\begin{aligned} \underset{\alpha_1,\ldots,\alpha_n \geq 0}{\text{maximize}} \quad & \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j k(x^i, x^j) \\ \text{subject to} \quad & \sum_{i=1}^{n} \alpha_i y_i = 0, \\ & \alpha_1, \ldots, \alpha_n \leq \frac{1}{\lambda n}. \end{aligned}$$

We refer to the quantity b as the bias term. When we require that the bias is $b = 0$, we call the optimization problem as SVM without the bias term. The primal SVM without the bias term remains the same except $f(x) = \sum_{i=1}^{n} \alpha_i y_i k(x^i, x)$. The dual SVM remains the same except we remove the equality constraint $\sum_{i=1}^{n} \alpha_i y_i = 0$.

The (primal) hard-margin SVM defined in the previous section corresponds to soft-margin SVM in the setting when $\lambda \to 0$. The dual hard-margin SVM is defined as follows.

Definition 7.2.2 (Dual hard-margin SVM)**.** *Let $x^1, \ldots, x^n \in \mathbb{R}^d$ be n vectors and $y_1, \ldots, y_n \in \{-1, 1\}$ be n labels. Let $k(x, x')$ be a kernel function. An optimization problem of the following form is a dual hard-margin SVM.*

$$\begin{aligned} \underset{\alpha_1,\ldots,\alpha_n \geq 0}{\text{maximize}} \quad & \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j k(x^i, x^j) \\ \text{subject to} \quad & \sum_{i=1}^{n} \alpha_i y_i = 0. \end{aligned}$$

If the primal hard-margin SVM is without the bias term ($b = 0$), then we omit the inequality constraint $\sum_{i=1}^{n} \alpha_i y_i = 0$ in the dual SVM.

We will use the following fact (see [MMR+01]).

Fact 7.2.3. *If $\alpha_1^*, \ldots, \alpha_n^*$ achieve the minimum in an SVM, then the same $\alpha_1^*, \ldots, \alpha_n^*$ achieve the maximum in the dual SVM. Also, the minimum value and the maximum value are equal.*

7.3 Hardness for kernel SVM

In this section we show conditional lower bounds for several variants of the kernel SVM problem.

7.3.1 Hardness for SVM without the bias term

Let $A \triangleq \{a^1, \ldots, a^n\} \subseteq \{0,1\}^d$ and $B \triangleq \{b^1, \ldots, b^n\} \subseteq \{0,1\}^d$ be the two sets of binary vectors from a BHCP instance with $d \triangleq \omega(\log n)$. Our goal is to determine

whether there is a close pair of vectors. We show how to solve this BHCP instance by reducing it to *three* computations of SVM, defined as follows:

1. We take the first set A of binary vectors, assign label 1 to all vectors, and solve the corresponding SVM on the n vectors:

$$\begin{aligned} \underset{\alpha_1,\ldots,\alpha_n \geq 0}{\text{minimize}} \quad & \frac{1}{2}\sum_{i,j=1}^{n} \alpha_i \alpha_j k(a^i, a^j) \\ \text{subject to} \quad & \sum_{j=1}^{n} \alpha_j k(a^i, a^j) \geq 1, \quad i = 1, \ldots, n. \end{aligned} \tag{7.3}$$

   Note that we do not have y_i in the expressions because all labels are 1.

2. We take the second set B of binary vectors, assign label -1 to all vectors, and solve the corresponding SVM on the n vectors:

$$\begin{aligned} \underset{\beta_1,\ldots,\beta_n \geq 0}{\text{minimize}} \quad & \frac{1}{2}\sum_{i,j=1}^{n} \beta_i \beta_j k(b^i, b^j) \\ \text{subject to} \quad & -\sum_{j=1}^{n} \beta_j k(b^i, b^j) \leq -1, \quad i = 1, \ldots, n. \end{aligned} \tag{7.4}$$

3. We take both sets A and B of binary vectors, assign label 1 to all vectors from the first set A and label -1 to all vectors from the second set B. We then solve the corresponding SVM on the $2n$ vectors:

$$\begin{aligned} \underset{\substack{\alpha_1,\ldots,\alpha_n \geq 0 \\ \beta_1,\ldots,\beta_n \geq 0}}{\text{minimize}} \quad & \frac{1}{2}\sum_{i,j=1}^{n} \alpha_i \alpha_j k(a^i, a^j) + \frac{1}{2}\sum_{i,j=1}^{n} \beta_i \beta_j k(b^i, b^j) - \sum_{i,j=1}^{n} \alpha_i \beta_j k(a^i, b^j) \\ \text{subject to} \quad & \sum_{j=1}^{n} \alpha_j k(a^i, a^j) - \sum_{j=1}^{n} \beta_j k(a^i, b^j) \geq 1, \quad i = 1, \ldots, n\,, \\ & -\sum_{j=1}^{n} \beta_j k(b^i, b^j) + \sum_{j=1}^{n} \alpha_j k(b^i, a^j) \leq -1, \quad i = 1, \ldots, n\,. \end{aligned} \tag{7.5}$$

Intuition behind the construction. To show a reduction from the BHCP problem to SVM computation, we have to consider two cases:

- The YES case of the BHCP problem when there are two vectors that are close in the Hamming distance. That is, there exist $a^i \in A$ and $b^j \in B$ such that $\text{Hamming}(a^i, b^j) < t$.

- The NO case of the BHCP problem when there is no close pair of vectors. That is, for all $a^i \in A$ and $b^j \in B$, we have $\text{Hamming}(a^i, b^j) \geq t$.

We show that we can distinguish between these two cases by comparing the objective value of the first two SVM instances above to the objective value of the third.

Intuition for the NO case. We have $\text{Hamming}(a^i, b^j) \geq t$ for all $a^i \in A$ and $b^j \in B$. The Gaussian kernel then gives the inequality

$$k(a^i, b^j) = \exp(-100 \log n \cdot \|a^i - b^j\|_2^2) \leq \exp(-100 \log n \cdot t)$$

for all $a^i \in A$ and $b^j \in B$. This means that the value $k(a^i, b^j)$ is very small. For simplicity, assume that it is equal to 0, i.e., $k(a^i, b^j) = 0$ for all $a^i \in A$ and $b^j \in B$.

Consider the third SVM Eq. (7.5). It contains three terms involving $k(a^i, b^j)$: the third term in the objective function, the second term in the inequalities of the first type, and the second term in the inequalities of the second type. We assumed that these terms are equal to 0 and we observe that the rest of the third SVM is equal to the sum of the first SVM Eq. (7.3) and the second SVM Eq. (7.4). Thus we expect that the optimal value of the third SVM is approximately equal to the sum of the optimal values of the first and the second SVMs. If we denote the optimal value of the first SVM Eq. (7.3) by $\text{value}(A)$, the optimal value of the second SVM Eq. (7.4) by $\text{value}(B)$, and the optimal value of the third SVM Eq. (7.5) by $\text{value}(A, B)$, then we can express our intuition in terms of the approximate equality

$$\text{value}(A, B) \approx \text{value}(A) + \text{value}(B) \ .$$

Intuition for the YES case. In this case, there is a close pair of vectors $a^i \in A$ and $b^j \in B$ such that $\text{Hamming}(a^i, b^j) \leq t-1$. Since we are using the Gaussian kernel, we have the following inequality for this pair of vectors:

$$k(a^i, b^j) = \exp(-100 \log n \cdot \|a^i - b^j\|_2^2) \geq \exp(-100 \log n \cdot (t-1)) \ .$$

Therefore, we have a large summand in each of the three terms from the above discussion. Thus, the three terms do not (approximately) disappear and there is no reason for us to expect that the approximate equality holds. We can thus expect

$$\text{value}(A, B) \not\approx \text{value}(A) + \text{value}(B) \ .$$

Thus, by computing $\text{value}(A, B)$ and comparing it to $\text{value}(A) + \text{value}(B)$ we can distinguish between the YES and NO instances of BHCP. This completes the reduction. In the rest of the section we formalize the intuition.

We start from the following two lemmas.

Lemma 7.3.1 (NO case)**.** *If for all $a^i \in A$ and $b^j \in B$ we have Hamming$(a^i, b^j) \geq t$, then*

$$\textit{value}(A, B) \leq \textit{value}(A) + \textit{value}(B) + 200n^6 \exp(-100 \log n \cdot t).$$

Lemma 7.3.2 (YES case)**.** *If there exist $a^i \in A$ and $b^j \in B$ such that Hamming(a^i, b^j) $\leq t-1$, then*

$$\textit{value}(A,B) \geq \textit{value}(A) + \textit{value}(B) + \frac{1}{4}\exp(-100\log n \cdot (t-1)).$$

Assuming the two lemmas we can distinguish the NO case from the YES case because

$$200n^6 \exp(-100\log n \cdot t) \ll \frac{1}{4}\exp(-100\log n \cdot (t-1)) \tag{7.6}$$

by our choice of the parameter $C = 100\log n$ for the Gaussian kernel.

Before we proceed with the proofs of the two lemmas, we prove the following auxiliary statement.

Lemma 7.3.3. *Consider SVM Eq.* (7.3)*. Let $\alpha_1^*, \ldots, \alpha_n^*$ be the setting of values for $\alpha_1, \ldots, \alpha_n$ that achieves value(A). Then for all $i = 1, \ldots, n$ we have that $n \geq \alpha_i^* \geq 1/2$.*

Analogous statement holds for SVM Eq. (7.4)*.*

Proof. First we note that value$(A) \leq n^2/2$ because the objective value of Eq. (7.3) is at most $n^2/2$ if we set $\alpha_1 = \ldots = \alpha_n = 1$. Note that all inequalities of Eq. (7.3) are satisfied for this setting of variables. Now we lower bound value(A):

$$\text{value}(A) = \frac{1}{2}\sum_{i,j}^{n} \alpha_i^*\alpha_j^* k(a^i, a^j) \geq \frac{1}{2}\sum_{i=1}^{n}(\alpha_i^*)^2.$$

From value$(A) \geq \frac{1}{2}\sum_{i=1}^{n}(\alpha_i^*)^2$ and value$(A) \leq n^2/2$ we conclude that $\alpha_i^* \leq n$ for all i.

Now we will show that $\alpha_i^* \geq 1/2$ for all $i = 1, \ldots, n$. Consider the inequality

$$\sum_{j=1}^{n} \alpha_j^* k(a^i, a^j) = \alpha_i^* + \sum_{j\,:\,j\neq i} \alpha_j^* k(a^i, a^j) \geq 1,$$

which is satisfied by $\alpha_1^*, \ldots, \alpha_n^*$ because this is an inequality constraint in Eq. (7.3). Note that $k(a^i, a^j) \leq \frac{1}{10n^2}$ for all $i \neq j$ because $C = 100\log n$ and $\|a^i - a^j\|_2^2 =$ Hamming$(a^i, a^j) \geq 1$ for all $i \neq j$. Also, we already obtained that $\alpha_j^* \leq n$ for all j. This gives us the required lower bound for α_i^*:

$$\alpha_i^* \geq 1 - \sum_{j\,:\,j\neq i} \alpha_j^* k(a^i, a^j) \geq 1 - n \cdot n \cdot \frac{1}{10n^2} \geq 1/2.$$

□

Additive precision. For particular threshold t, the sufficient additive precision for solving the three SVMs is $\frac{1}{100}\exp(-100\log n \cdot (t-1))$ to be able to distinguish the NO case from the YES case. Since we want to be able to distinguish the two cases for any $t \in \{2, \ldots, d\}$, it suffices to have an additive precision $\exp(-100\log n \cdot d) \leq$

$\frac{1}{100}\exp(-100\log n\cdot(t-1))$. From [AW15] we know that any $d=\omega(\log n)$ is sufficient to show hardness. Therefore, any additive approximation $\exp(-\omega(\log^2 n))$ is sufficient to show the hardness for SVM.

Multiplicative precision. Consider any $\varepsilon \triangleq \exp(-\omega(\log^2 n))$ and suppose we can approximate within multiplicative factor $(1+\varepsilon)$ quantities value(A), value(B) and value(A,B). From the proof of Lemma 7.3.3 we know that value(A), value$(B) \le n^2/2$. If value$(A,B) \le 10n^2$, then $(1+\varepsilon)$-approximation of the three quantities allows us to compute the three quantities within additive $\exp(-\omega(\log^2 n))$ factor and the hardness follows from the previous paragraph. On the other hand, if value$(A,B) > 10n^2$, then $(1+\varepsilon)$-approximation of value(A,B) allows us to determine that we are in the YES case.

In the rest of the section we complete the proof of the theorem by proving Lemmas 7.3.1 and 7.3.2.

Proof of Lemma 7.3.1. Let $\alpha_1^*,\ldots,\alpha_n^*$ and $\beta_1^*,\ldots,\beta_n^*$ be the optimal assignments to SVMs Eq. (7.3) and Eq. (7.4), respectively. We use the notation $\delta \triangleq \exp(-100\log n\cdot t)$. Note that $k(a^i,b^j)=\exp(-100\log n\cdot\|a^i-b^j\|_2^2)\le\delta$ for all i,j because $\|a^i-b^j\|_2^2=$ Hamming$(a^i,b^j)\ge t$ for all i,j.

We define $\alpha_i' \triangleq \alpha_i^*+10n^2\delta$ and $\beta_i' \triangleq \beta_i^*+10n^2\delta$ for all $i=1,\ldots,n$. We observe that $\alpha_i',\beta_i'\le 2n$ for all i because $\alpha_i^*,\beta_i^*\le n$ for all i (Lemma 7.3.3) and $\delta=\exp(-100\log n\cdot t)\le\frac{1}{10n^2}$. Let V be the value of the objective function in Eq. (7.5) when evaluated on α_i' and β_i'.

We make two claims. We claim that α_i' and β_i' satisfy the inequality constraints in Eq. (7.5). This implies that value$(A,B)\le V$ since Eq. (7.5) is a minimization problem. Our second claim is that $V\le$ value$(A)+$value$(B)+200n^6\delta$. The two claims combined complete the proof of the lemma.

We start with the proof of the second claim. We want to show that $V\le$ value$(A)+$ value$(B)+200n^6\delta$. We get the following inequality:

$$\begin{aligned} V &= \frac{1}{2}\sum_{i,j=1}^n \alpha_i'\alpha_j' k(a^i,a^j) \;+\; \frac{1}{2}\sum_{i,j=1}^n \beta_i'\beta_j' k(b^i,b^j) \;-\; \sum_{i,j=1}^n \alpha_i'\beta_j' k(a^i,b^j) \\ &\le \frac{1}{2}\sum_{i,j=1}^n \alpha_i'\alpha_j' k(a^i,a^j) \;+\; \frac{1}{2}\sum_{i,j=1}^n \beta_i'\beta_j' k(b^i,b^j) \end{aligned}$$

since the third sum is non-negative. Therefore it is sufficient to show two inequalities $\frac{1}{2}\sum_{i,j=1}^n \alpha_i'\alpha_j' k(a^i,a^j) \le$ value$(A)+100n^6\delta$ and $\frac{1}{2}\sum_{i,j=1}^n \beta_i'\beta_j' k(b^i,b^j) \le$ value$(B)+100n^6\delta$ to establish the inequality $V\le$ value$(A)+$value$(B)+200n^6\delta$. We prove the first inequality. The proof for the second inequality is analogous. We use the definition of $\alpha_i'=\alpha_i^*+10n^2\delta$:

$$\frac{1}{2}\sum_{i,j=1}^n \alpha_i'\alpha_j' k(a^i,a^j)$$

$$
\begin{aligned}
&= \frac{1}{2}\sum_{i,j=1}^{n}(\alpha_i^* + 10n^2\delta)(\alpha_j^* + 10n^2\delta)k(a^i, a^j) \\
&\leq \frac{1}{2}\sum_{i,j=1}^{n}\left(\alpha_i^*\alpha_j^* k(a^i, a^j) + 20n^3\delta + 100n^4\delta^2\right) \\
&\leq \text{value}(A) + 100n^6\delta,
\end{aligned}
$$

where in the first inequality we use that $\alpha_i^* \leq n$ and $k(a^i, a^j) \leq 1$.

Now we prove the first claim. We show that the inequality constraints are satisfied by α_i' and β_i'. We prove that the inequality

$$
\sum_{j=1}^{n}\alpha_j' k(a^i, a^j) - \sum_{j=1}^{n}\beta_j' k(a^i, b^j) \geq 1 \tag{7.7}
$$

is satisfied for all $i = 1, \ldots, n$. The proof that the inequality $-\sum_{j=1}^{n}\beta_j' k(b^i, b^j) + \sum_{j=1}^{n}\alpha_j' k(b^i, a^j) \leq -1$ is satisfied is analogous.

We lower bound the first sum of the left hand side of Eq. (7.7) by repeatedly using the definition of $\alpha_i' = \alpha_i^* + 10n^2\delta$:

$$
\begin{aligned}
&\sum_{j=1}^{n}\alpha_j' k(a^i, a^j) \\
&= (\alpha_i^* + 10n^2\delta) + \sum_{j\,:\,j\neq i}\alpha_j' k(a^i, a^j) \\
&\geq 10n^2\delta + \alpha_i^* + \sum_{j\,:\,j\neq i}\alpha_j^* k(a^i, a^j) \\
&= 10n^2\delta + \sum_{j=1}^{n}\alpha_j^* k(a^i, a^j) \\
&\geq 1 + 10n^2\delta.
\end{aligned}
$$

In the last inequality we used the fact that α_i^* satisfy the inequality constraints of SVM Eq. (7.3).

We upper bound the second sum of the left hand side of Eq. (7.7) by using the inequality $\beta_j' \leq 2n$ and $k(a^i, b^j) \leq \delta$ for all i, j:

$$
\sum_{j=1}^{n}\beta_j' k(a^i, b^j) \leq 2n^2\delta.
$$

Finally, we can show that the inequality constraint is satisfied:

$$
\sum_{j=1}^{n}\alpha_j' k(a^i, a^j) - \sum_{j=1}^{n}\beta_j' k(a^i, b^j) \geq 1 + 10n^2\delta - 2n^2\delta \geq 1.
$$

□

Proof of Lemma 7.3.2. To analyze the YES case, we consider the dual SVMs (see Definition 7.2.2) of the three SVMs Eqs. (7.3) to (7.5):

1. The dual SVM of SVM Eq. (7.3):

$$\underset{\alpha_1,\ldots,\alpha_n\geq 0}{\text{maximize}} \quad \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i,j=1}^{n}\alpha_i\alpha_j k(a^i,a^j). \tag{7.8}$$

2. The dual SVM of SVM Eq. (7.4):

$$\underset{\beta_1,\ldots,\beta_n\geq 0}{\text{maximize}} \quad \sum_{i=1}^{n}\beta_i - \frac{1}{2}\sum_{i,j=1}^{n}\beta_i\beta_j k(a^i,a^j). \tag{7.9}$$

3. The dual SVM of SVM Eq. (7.5):

$$\begin{aligned}\underset{\substack{\alpha_1,\ldots,\alpha_n\geq 0\\ \beta_1,\ldots,\beta_n\geq 0}}{\text{maximize}} \sum_{i=1}^{n}\alpha_i \;+\; \sum_{i=1}^{n}\beta_i \;-\; \frac{1}{2}\sum_{i,j=1}^{n}\alpha_i\alpha_j k(a^i,a^j) \;-\; \frac{1}{2}\sum_{i,j=1}^{n}\beta_i\beta_j k(b^i,b^j)\\ +\; \sum_{i,j=1}^{n}\alpha_i\beta_j k(a^i,b^j).\end{aligned} \tag{7.10}$$

Since the optimal values of the primal and the dual SVMs are equal, we have that value(A), value(B) and value(A,B) are equal to the optimal values of the dual SVMs Eq. (7.8), Eq. (7.9) and Eq. (7.10), respectively (see Fact 7.2.3).

Let $\alpha_1^*,\ldots,\alpha_n^*$ and $\beta_1^*,\ldots,\beta_n^*$ be the optimal assignments to dual SVMs Eq. (7.8) and Eq. (7.9), respectively.

Our goal is to lower bound value(A,B). Since Eq. (7.10) is a maximization problem, it is sufficient to show an assignment to α_i and β_j that gives a large value to the objective function. For this we set $\alpha_i=\alpha_i^*$ and $\beta_j=\beta_j^*$ for all $i,j=1,\ldots,n$. This gives the following inequality:

$$\begin{aligned}\text{value}(A,B) \geq\; & \sum_{i=1}^{n}\alpha_i^* \;+\; \sum_{i=1}^{n}\beta_i^* \;-\; \frac{1}{2}\sum_{i,j=1}^{n}\alpha_i^*\alpha_j^* k(a^i,a^j) \;-\; \frac{1}{2}\sum_{i,j=1}^{n}\beta_i^*\beta_j^* k(b^i,b^j)\\ & +\; \sum_{i,j=1}^{n}\alpha_i^*\beta_j^* k(a^i,b^j)\\ \geq\; & \text{value}(A)+\text{value}(B)+\sum_{i,j=1}^{n}\alpha_i^*\beta_j^* k(a^i,b^j),\end{aligned}$$

where we use the fact that value(A) and value(B) are the optimal values of dual SVMs Eq. (7.8) and Eq. (7.9), respectively.

To complete the proof of the lemma, it suffices to show the following inequality:

$$\sum_{i,j=1}^{n} \alpha_i^* \beta_j^* k(a^i, b^j) \geq \frac{1}{4} \exp(-100 \log n \cdot (t-1)). \tag{7.11}$$

Notice that so far we did not use the fact that there is a close pair of vectors $a^i \in A$ and $b^j \in B$ such that $\text{Hamming}(a^i, b^j) \leq t-1$. We use this fact now. We lower bound the left hand side of Eq. (7.11) by the summand corresponding to the close pair:

$$\sum_{i,j=1}^{n} \alpha_i^* \beta_j^* k(a^i, b^j) \geq \alpha_i^* \beta_j^* k(a^i, b^j) \geq \alpha_i^* \beta_j^* \exp(-100 \log n \cdot (t-1)),$$

where in the last inequality we use $\text{Hamming}(a^i, b^j) \leq t-1$ and the definition of the Gaussian kernel.

The proof is completed by observing that $\alpha_i^* \geq \frac{1}{2}$ and $\beta_i^* \geq \frac{1}{2}$ which follows from Fact 7.2.3 and Lemma 7.3.3. □

Hardness for higher approximation factor. The reason for the choice of $\varepsilon = \exp(-\omega(\log^2 n))$ approximation factor is two-fold. First, in Eq. (7.6) we have to set $C \geq \Omega(\log n)$ to be able to distinguish the cases. Second, t can be as large as d and we set d to be any $\omega(\log n)$. In [Rub18] it was shown that approximately solving the Hamming close pair problem with any $1 + o(1)$ factor requires $n^{2-o(1)}$ time for any $d \geq \omega(\log n)$ assuming SETH (by a reduction from the orthogonal vectors problem). This allows us to set C to be any $C \geq \omega(\log n)$ and $t = 1$ (after appropriately scaling the vectors). This gives hardness for any $\varepsilon \leq n^{-\omega(1)}$. Similar argument allows to improve the approximation factor for other variants of the SVM problem as well as the kernel ridge regression and kernel PCA.

7.3.2 Hardness for SVM with the bias term

In the previous section we showed hardness for SVM without the bias term. In this section we show hardness for SVM with the bias term.

Theorem 7.3.4. *Let $x^1, \ldots, x^n \in \{-1, 0, 1\}^{d'}$ be n vectors and let $y_1, \ldots, y_n \in \{-1, 1\}$ be n labels. Let $k(x, x') \triangleq \exp(-C\|x - x'\|_2^2)$ be the Gaussian kernel with $C \triangleq 100 \log n$.*

Consider the corresponding hard-margin SVM with the bias term:

$$\begin{aligned} \underset{\alpha_1,\ldots,\alpha_n \geq 0,\ b}{\text{minimize}} \quad & \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j k(x^i, x^j) \\ \text{subject to} \quad & y_i f(x^i) \geq 1, \quad i = 1, \ldots, n, \end{aligned} \tag{7.12}$$

where $f(x) \triangleq b + \sum_{i=1}^{n} \alpha_i y_i k(x^i, x)$.

Consider any $\varepsilon \triangleq \exp(-\omega(\log^2 n))$*. Approximating the optimal value of Eq.* (7.12) *within the multiplicative factor* $(1+\varepsilon)$ *requires almost quadratic time assuming SETH. This holds for the dimensionality* $d' \triangleq O(\log^3 n)$ *of the input vectors.*

The same hardness result holds for any additive $\exp(-\omega(\log^2 n))$ *approximation factor.*

Proof. Consider a hard instance from Theorem 7.1.2 for SVM without the bias term. Let $x^1, \ldots, x^n \in \{0,1\}^d$ be the n binary vectors of dimensionality $d \triangleq \omega(\log n)$ and $y_1, \ldots, y_n \in \{-1, 1\}$ be the n corresponding labels. For this input consider the dual SVM without the bias term (see Definition 7.2.2):

$$\underset{\gamma_1,\ldots,\gamma_n \geq 0}{\text{maximize}} \quad \sum_{i=1}^{n} \gamma_i \; - \; \frac{1}{2} \sum_{i,j=1}^{n} \gamma_i \gamma_j y_i y_j k(x^i, x^j). \tag{7.13}$$

We will show how to reduce SVM without the bias term Eq. (7.13) to SVM with the bias term. By Theorem 7.1.2 this will give hardness result for SVM with the bias term. We start with a simpler reduction that will achieve almost what we need except the entries of the vectors will not be from the set $\{-1, 0, 1\}$. Then we will show how to change the reduction to fix this.

Consider $2n$ vectors $x^1, \ldots, x^n, -x^1, \ldots, -x^n \in \{-1, 0, 1\}^d$ with $2n$ labels $y_1, \ldots, y_n$, $-y_1, \ldots, -y_n \in \{-1, 1\}$. Consider an SVM with the bias term for the $2n$ vectors, that is, an SVM of the form Eq. (7.12). From Definition 7.2.2 we know that its dual SVM is

$$\begin{aligned} \underset{\substack{\alpha_1,\ldots,\alpha_n \geq 0 \\ \beta_1,\ldots,\beta_n \geq 0}}{\text{maximize}} \quad & \sum_{i=1}^{n} \alpha_i \; + \; \sum_{j=1}^{n} \beta_j \; - \; \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j k(x^i, x^j) \; - \; \frac{1}{2} \sum_{i,j=1}^{n} \beta_i \beta_j y_i y_j k(x^i, x^j) \\ & \qquad + \; \sum_{i,j=1}^{n} \alpha_i \beta_j y_i y_j k(x^i, -x^j) \\ \text{subject to} \quad & \sum_{i=1}^{n} \alpha_i y_i = \sum_{j=1}^{n} \beta_j y_j. \end{aligned} \tag{7.14}$$

Consider any setting of values for α_i and β_j. Notice that if we swap the value of α_i and β_i for every i, the value of the objective function of Eq. (7.14) does not change. This is implies that we can define $\gamma_i \triangleq \frac{\alpha_i + \beta_i}{2}$ and set $\alpha_i = \beta_i = \gamma_i$ for every i. Because of the convexity of the optimization problem, the value of the objective function can only increase after this change. Clearly, the equality constraint will be satisfied. Therefore, without loss of generality, we can assume that $\alpha_i = \beta_i = \gamma_i$ for some γ_i and we can omit the equality constraint.

We rewrite Eq. (7.14) in terms of γ_i and divide the objective function by 2:

$$\underset{\gamma_1,\ldots,\gamma_n \geq 0}{\text{maximize}} \quad \sum_{i=1}^{n} \gamma_i \; - \; \frac{1}{2} \sum_{i,j=1}^{n} \gamma_i \gamma_j y_i y_j k(x^i, x^j) \; + \; \frac{1}{2} \sum_{i,j=1}^{n} \gamma_i \gamma_j y_i y_j k(x^i, -x^j). \tag{7.15}$$

Notice that Eq. (7.15) and Eq. (7.13) are almost the same. The only difference is the third term

$$\frac{1}{2}\sum_{i,j=1}^{n}\gamma_i\gamma_j y_i y_j k(x^i, -x^j)$$

in Eq. (7.15). We can make this term to be equal to 0 and not change the first two terms as follows. We append an extra coordinate to every vector x^i and set this coordinate to be large enough value M. If we set $M = +\infty$, the third term becomes 0. The first term does not depend on the vectors. The second term depends only on the distances between the vectors (which are not affected by adding the same entry to all vectors). Thus, the first two terms do not change after this modification.

We showed that we can reduce SVM without the bias term Eq. (7.13) to the SVM with the bias term Eq. (7.14). By combining this reduction with Theorem 7.1.2 we obtain hardness for SVM with the bias term. This is almost what we need except that the reduction presented above produces vectors with entries that are not from the set $\{-1, 0, 1\}$. In every vector x^i or $-x^i$ there is an entry that has value M or $-M$, respectively. In the rest of the proof we show how to fix this, by bounding M by $O(\log^3 n)$ and distributing its contribution over $O(\log^3 n)$ coordinates.

Final reduction. The final reduction is as follows:

- Take a hard instance for the SVM without the bias term from Theorem 7.1.2. Let $x^1, \ldots, x^n \in \{0, 1\}^d$ be the n binary vectors of dimensionality $d \triangleq \omega(\log n)$ and $y_1, \ldots, y_n \in \{-1, 1\}$ be the n corresponding labels.
- Append $\log^3 n$ entries to each of the vectors x^i, $i = 1, \ldots, n$ and set the entries to be 1.
- Solve SVM Eq. (7.12) on the $2n$ vectors $x^1, \ldots, x^n, -x^1, \ldots, -x^n \in \{-1, 0, 1\}^d$ with $2n$ labels $y_1, \ldots, y_n, -y_1, \ldots, -y_n \in \{-1, 1\}$. Let V be the optimal value of the SVM divided by 2.
- Output V.

Correctness of the reduction From the above discussion we know that we output the optimal value V of the optimization problem Eq. (7.15). Let V' be the optimal value of SVM Eq. (7.13).

By Theorem 7.1.2 it is sufficient to show that $|V - V'| \leq \exp(-\omega(\log^2 n))$ to establish hardness for SVM with the bias term. We will show that $|V - V'| \leq n^{O(1)} \exp(-\log^3 n)$. This gives hardness for additive approximation of SVM with the bias term. However, $|V - V'| \leq \exp(-\omega(\log^2 n))$ is also sufficient to show hardness for multiplicative approximation (see the discussion on the approximation in the proof of Theorem 7.1.2).

In the rest of the section we show that $|V - V'| \leq n^{O(1)} \exp(-\log^3 n)$. Let γ_i' be the assignment to γ_i that achieves V' in SVM Eq. (7.13). Let γ_i^* be the assignment to γ_i that achieves V in Eq. (7.15). We will show that $\gamma_i' \leq O(n)$ for all $i = 1, \ldots, n$.

It is also true that $\gamma_i^* \leq O(n)$ for all $i = 1, \ldots, n$ and the proof is analogous. Since $x^1, \ldots, x^n$ are different binary vectors and $k(x^i, x^j)$ is the Gaussian kernel with the parameter $C = 100 \log n$, we have that $k(x^i, x^j) \leq 1/n^{10}$ for all $i \neq j$. This gives the following upper bound:

$$V' = \sum_{i=1}^{n} \gamma_i' \; - \; \frac{1}{2} \sum_{i,j=1}^{n} \gamma_i' \gamma_j' y_i y_j k(x^i, x^j) \leq \sum_{i=1}^{n} \left(\gamma_i' - \left(\frac{1}{2} - o(1) \right) (\gamma_i')^2 \right).$$

Observe that every non-negative summand on the right hand side is at most $O(1)$. Therefore, if there exists i such that $\gamma_i' \geq \omega(n)$, then the right hand side is negative. This contradicts the lower bound $V' \geq 0$ which follows by setting all γ_i to be 0 in Eq. (7.13).

By plugging γ_i' into Eq. (7.15) and using the fact that $\gamma_i' \leq O(n)$, we obtain the following inequality:

$$V \geq V' + \frac{1}{2} \sum_{i,j=1}^{n} \gamma_i' \gamma_j' y_i y_j k(x^i, -x^j) \geq V' - n^{O(1)} \exp(-\log^3 n). \tag{7.16}$$

In the last inequality we use $k(x^i, -x^j) \leq \exp(-\log^3 n)$ which holds for all $i, j = 1, ..., n$ (observe that each x^i and x^j ends with $\log^3 n$ entries 1 and use the definition of the Gaussian kernel).

Similarly, by plugging γ_i^* into Eq. (7.13) and using the fact that $\gamma_i^* \leq O(n)$, we obtain the following inequality:

$$V' \geq V - \frac{1}{2} \sum_{i,j=1}^{n} \gamma_i^* \gamma_j^* y_i y_j k(x^i, -x^j) \geq V - n^{O(1)} \exp(-\log^3 n). \tag{7.17}$$

Inequalities Eq. (7.16) and Eq. (7.17) combined give the desired inequality $|V - V'| \leq n^{O(1)} \exp(-\log^3 n)$. □

7.3.3 Hardness for soft-margin SVM

Theorem 7.3.5. *Let $x^1, \ldots, x^n \in \{-1, 0, 1\}^d$ be n vectors and let $y_1, \ldots, y_n \in \{-1, 1\}$ be n labels. Let $k(x, x') \triangleq \exp\left(-C\|x - x'\|_2^2\right)$ be the Gaussian kernel with $C \triangleq 100 \log n$.*

Consider the corresponding soft-margin SVM with the bias term:

$$\begin{array}{ll} \underset{\substack{\alpha_1, \ldots, \alpha_n \geq 0, \\ \xi_1, \ldots, \xi_n \geq 0}}{\text{minimize}}_{\; b} & \dfrac{\lambda}{2} \displaystyle\sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j k(x^i, x^j) \; + \; \frac{1}{n} \sum_{i=1}^{n} \xi_i \\ \text{subject to} & y_i f(x^i) \geq 1 - \xi_i, \quad i = 1, \ldots, n, \end{array} \tag{7.18}$$

where $f(x) \triangleq b + \sum_{i=1}^{n} \alpha_i y_i k(x^i, x)$.

Consider any $\varepsilon \triangleq \exp(-\omega(\log^2 n))$ and any $0 < \lambda \leq \frac{1}{Kn^2}$ for a large enough constant $K > 0$. Approximating the optimal value of Eq. (7.18) within the multiplicative

factor $(1+\varepsilon)$ *requires almost quadratic time assuming SETH. This holds for the dimensionality* $d \triangleq O(\log^3 n)$ *of the input vectors.*

The same hardness result holds for any additive $\exp(-\omega(\log^2 n))$ *approximation factor.*

Proof. Consider the hard instance from Theorem 7.3.4 for the hard-margin SVM. The dual of the hard-margin SVM is Eq. (7.14). From the proof we know that the optimal α_i and β_i satisfy $\alpha_i = \beta_i = \gamma_i^* \leq 2Kn$ for some large enough constant $K > 0$ for all $i = 1, \ldots, n$. Thus, without loss of generality we can add these inequalities to the set of constraints. We compare the resulting dual SVM to Definition 7.2.1 and conclude that the resulting dual SVM is a dual of a *soft-margin* SVM with the regularization parameter $\lambda \triangleq \frac{1}{Kn^2}$. Therefore, the hardness follows from Theorem 7.3.4. □

7.4 Hardness for kernel ridge regression

We start by stating helpful definitions and lemmas. We will use the following lemma which is a consequence of the binomial inverse theorem.

Lemma 7.4.1. *Let* X *and* Y *be two square matrices of equal size. Then the following equality holds:*

$$(X+Y)^{-1} = X^{-1} - X^{-1}(I + YX^{-1})^{-1}YX^{-1}.$$

Definition 7.4.2 (Almost identity matrix)**.** *Let* $X \in \mathbb{R}^{n \times n}$ *be a matrix. We call it* almost identity matrix *if* $X = I + Y$ *and* $|Y_{i,j}| \leq n^{-\omega(1)}$ *for all* $i, j = 1, \ldots, n$.

We will need the following two lemmas.

Lemma 7.4.3. *The product of two almost identity matrices is an almost identity matrix.*

Proof. Follows easily from the definition. □

Lemma 7.4.4. *The inverse of an almost identity matrix is an almost identity matrix.*

Proof. Let X be an almost identity matrix. We want to show that X^{-1} is an almost identity matrix. We write $X = I - Y$ such that $|Y_{i,j}| \leq n^{-\omega(1)}$ for all $i, j = 1, \ldots, n$. We have the following matrix equality

$$X^{-1} = (I - Y)^{-1} = I + Y + Y^2 + Y^3 + \ldots$$

To show that X^{-1} is an almost identity, we will show that the absolute value of every entry of $Z \triangleq Y + Y^2 + Y^3 + \ldots$ is at most $n^{-\omega(1)}$. Let $\varepsilon \leq n^{-\omega(1)}$ is an upper bound on $|Y_{i,j}|$ for all $i, j = 1, \ldots, n$. Then $|Z_{i,j}| \leq Z'_{i,j}$, where $Z' \triangleq Y' + (Y')^2 + (Y')^3 + \ldots$ and Y' is a matrix consisting of entries that are all equal to ε. The proof follows since $Z'_{i,j} = \sum_{k=1}^{\infty} \varepsilon^k n^{k-1} \leq 10\varepsilon \leq n^{-\omega(1)}$. □

In the rest of the section we prove Theorem 7.1.4.

Proof of Theorem 7.1.4. We reduce the BHCP problem to the problem of computing the sum of the entries of K^{-1}.

Let A and B be the two sets of binary vectors from the BHCP instance. Let $K \in \mathbb{R}^{2n\times 2n}$ be the corresponding kernel matrix. We can write the kernel matrix K as combination of four smaller matrices $K^{1,1}, K^{1,2}, K^{2,1}, K^{2,2} \in \mathbb{R}^{n\times n}$:

$$K = \left[\begin{array}{c|c} K^{1,1} & K^{1,2} \\ \hline K^{2,1} & K^{2,2} \end{array}\right].$$

$K^{1,1}$ is the kernel matrix for the set of vectors A and $K^{2,2}$ is the kernel matrix for the set of vectors B. We define two new matrices $X, Y \in \mathbb{R}^{2n\times 2n}$: $X \triangleq \left[\begin{array}{c|c} K^{1,1} & 0 \\ \hline 0 & K^{2,2} \end{array}\right]$ and $Y \triangleq \left[\begin{array}{c|c} 0 & K^{1,2} \\ \hline K^{2,1} & 0 \end{array}\right]$.

For any matrix Z, let $s(Z)$ denote the sum of all entries of Z. Using Lemma 7.4.1, we can write K^{-1} as follows:

$$K^{-1} = (X+Y)^{-1} = X^{-1} - X^{-1}(I + YX^{-1})^{-1}YX^{-1}.$$

We note that the matrix X is an almost identity and that $|Y_{i,j}| \leq n^{-\omega(1)}$ for all $i, j = 1, \ldots, 2n$. This follows from the fact that we use the Gaussian kernel function with the parameter $C = \omega(\log n)$ and the input vectors are binary. Combining this with Lemmas 7.4.3 and 7.4.4 allows us to conclude that matrices $X^{-1}(I + YX^{-1})^{-1}$ and X^{-1} are almost identity. Since all entries of the matrix Y are non-negative, we conclude that

$$s(X^{-1}(I + YX^{-1})^{-1}YX^{-1}) = s(Y)(1 \pm n^{-\omega(1)}).$$

We obtain that

$$\begin{aligned} s(K^{-1}) &= s(X^{-1}) - s(X^{-1}(I + YX^{-1})^{-1}YX^{-1}) \\ &= s(X^{-1}) - s(Y)(1 \pm n^{-\omega(1)}) \\ &= s\left((K^{1,1})^{-1}\right) + s\left((K^{2,2})^{-1}\right) - s(Y)(1 \pm n^{-\omega(1)}). \end{aligned}$$

Fix any $\alpha \triangleq \exp(-\omega(\log^2 n))$. Suppose that we can estimate each $s(K^{-1})$, $s\left((K^{1,1})^{-1}\right)$ and $s\left((K^{2,2})^{-1}\right)$ within the additive factor of α. This allows us to estimate $s(Y)$ within the additive factor of 10α. This is enough to solve the BHCP problem. We consider two cases.

Case 1. There are no close pairs, that is, for all $i, j = 1, \ldots, n$ we have $\|a^i - b^j\|_2^2 \geq t$ and $\exp(-C\|a^i - b^j\|_2^2) \leq \delta \triangleq \exp(-Ct)$. Then $s(Y) \leq 2n^2\delta$.

Case 2. There is a close pair. That is, $\|a^{i'} - b^{j'}\|_2^2 \leq t - 1$ for some i', j'. This implies that $\exp(-C\|a^{i'} - b^{j'}\|_2^2) \geq \Delta \triangleq \exp(-C(t-1))$. Thus, $s(Y) \geq \Delta$.

Since $C = \omega(\log n)$, we have that $\Delta \geq 100n^2\delta$ and we can distinguish the two cases assuming that the additive precision $\alpha = \exp(-\omega(\log^2 n))$ is small enough.

Precision. To distinguish $s(Y) \leq 2n^2\delta$ from $s(Y) \geq \Delta$, it is sufficient that $\Delta \geq 100n^2\delta$ and $\alpha \leq \Delta/1000$. We know that $\Delta \geq 100n^2\delta$ holds because $C = \omega(\log n)$. Since $\Delta \leq \exp(-Cd)$, we want to choose C and d such that the $\alpha \leq \Delta/1000$ is satisfied. We can do that because we can pick C to be any $C = \omega(\log n)$ and the BHCP problem requires almost quadratic time assuming SETH for any $d = \omega(\log n)$.

We get that additive ε approximation is sufficient to distinguish the cases for any $\varepsilon = \exp(-\omega(\log^2 n))$. We observe that $s(K^{-1}) \leq O(n)$ for any almost identity matrix K. This means that $(1+\varepsilon)$ multiplicative approximation is sufficient for the same ε. This completes the proof of the theorem. □

7.5 Hardness for kernel PCA

In this section, we present the full proof of quadratic hardness for kernel PCA. It will also be helpful for kernel ridge regression in the next section.

Given a matrix X, we denote its trace (the sum of the diagonal entries) by $\mathrm{tr}(X)$ and the total sum of its entries by $s(X)$. In the context of the matrix K' defining our problem, we have the following equality:

$$\begin{aligned}\mathrm{tr}(K') &= \mathrm{tr}((I - 1_n)K(I - 1_n)) \\ &= \mathrm{tr}(K(I - 1_n)^2) = \mathrm{tr}(K(I - 1_n)) \\ &= \mathrm{tr}(K) - \mathrm{tr}(K1_n) = n - s(K)/n\ .\end{aligned}$$

Since the sum of the eigenvalues is equal to the trace of the matrix and $\mathrm{tr}(K') = n - s(K)/n$, it is sufficient to show hardness for computing $s(K)$. The following lemma completes the proof of the theorem.

Lemma 7.5.1. *Computing $s(K)$ within multiplicative error $1+\varepsilon$ for parameter $\varepsilon = \exp(-\omega(\log^2 n))$ requires almost quadratic time assuming SETH.*

Proof. As for SVMs, we will reduce the BHCP problem to the computation of $s(K)$. Let A and B be the two sets of n binary vectors coming from an instance of the BHCP problem. Let $K_A, K_B \in \mathbb{R}^{n\times n}$ be the kernel matrices corresponding to the sets A and B, respectively. Let $K_{A,B} \in \mathbb{R}^{2n\times 2n}$ be the kernel matrix corresponding to the set $A \cup B$. We observe that

$$\begin{aligned}s &\triangleq (s(K_{A,B}) - s(K_A) - s(K_B))/2 \\ &= \sum_{i,j=1}^{n} k(a^i, b^j) \\ &= \sum_{i,j=1}^{n} \exp(-C\|a^i - b^j\|_2^2).\end{aligned}$$

Now we consider two cases.

Case 1. There are no close pairs, that is, for all $i, j = 1, \ldots, n$ we have $\|a^i - b^j\|_2^2 \geq t$ and $\exp(-C\|a^i - b^j\|_2^2) \leq \delta \triangleq \exp(-Ct)$. Then $s \leq n^2\delta$.

Case 2. There is a close pair. That is, $\|a^{i'} - b^{j'}\|_2^2 \leq t - 1$ for some i', j'. This implies that $\exp(-C\|a^{i'} - b^{j'}\|_2^2) \geq \Delta \triangleq \exp(-C(t-1))$. Thus, $s \geq \Delta$.

Since $C = 100 \log n$, we have that $\Delta \geq n^{10}\delta$ and we can distinguish the two cases.

Precision. To distinguish $s \geq \Delta$ from $s \leq n^2\delta$, it is sufficient that $\Delta \geq 2n^2\delta$. This holds for $C = 100 \log n$. The sufficient additive precision is $\exp(-Cd) = \exp(-\omega(\log^2 n))$. Since $s(K) \leq O(n^2)$ for any Gaussian kernel matrix K, we also get that $(1+\varepsilon)$ multiplicative approximation is sufficient to distinguish the cases for any $\varepsilon = \exp(-\omega(\log^2 n))$. □

7.6 Hardness for training the final layer

We start by formally defining the class of "nice" loss functions and "nice" activation functions.

Definition 7.6.1. *For a label $y \in \{-1, 1\}$ and a prediction $y' \in \mathbb{R}$, we call the loss function $loss(y, y') : \{-1, 1\} \times \mathbb{R} \to [0, \infty)$* nice *if the following three properties hold:*

- *$loss(y, y') = l(yy')$ for some convex function $l : \mathbb{R} \to [0, \infty)$.*
- *For some sufficiently large constant $K > 0$, we have that (i) $l(y) \leq o(1)$ for all $y \geq n^K$, (ii) $l(y) \geq \omega(n)$ for all $y \leq -n^K$, and (iii) $l(y) = l(0) \pm o(1/n)$ for all $y \in \pm O(n^{-K})$.*
- *$l(0) > 0$ is some constant strictly larger than 0.*

We note that the hinge loss function $\text{loss}(y, y') = \max(0, 1 - yy')$ and the logistic loss function $\text{loss}(y, y') = \frac{1}{\ln 2} \ln\left(1 + e^{-yy'}\right)$ are nice loss functions according to the above definition.

Definition 7.6.2. *A* non-decreasing *activation functions $S : \mathbb{R} \to [0, \infty)$ is "nice" if it satisfies the following property: for all sufficiently large constants $T > 0$ there exist $v_0 > v_1 > v_2$ such that $S(v_0) = \Theta(1)$, $S(v_1) = 1/n^T$, $S(v_2) = 1/n^{\omega(1)}$ and $v_1 = (v_0 + v_2)/2$.*

The ReLU activation $S(z) = \max(0, z)$ satisfies these properties since we can choose $v_0 = 1$, $v_1 = 1/n^T$, and $v_2 = -1 + 2/n^T$. For the sigmoid function $S(z) = \frac{1}{1+e^{-z}}$, we can choose $v_1 = -\log(n^T - 1)$, $v_0 = v_1 + C$, and $v_2 = v_1 - C$ for some $C = \omega(\log n)$. In the rest of the proof we set $T \triangleq 1000K$, where K is the constant from Definition 7.6.1.

We now describe the proof of Theorem 7.1.7. We use the notation $\alpha \triangleq (\alpha_1, \ldots, \alpha_n)^\mathsf{T}$. Invoking the first property from Definition 7.6.1, we observe that the optimization problem Eq. (7.2) is equivalent to the following optimization problem:

$$\underset{\alpha \in \mathbb{R}^n}{\text{minimize}} \quad \sum_{i=1}^{m} l(y_i (M\alpha)_i), \tag{7.19}$$

where $M \in \mathbb{R}^{m \times n}$ is the matrix defined as $M_{i,j} \triangleq S(x^i \cdot w^j)$ for $i = 1, \ldots, m$ and $j = 1, \ldots n$. That is, $M_{i,j}$ is equal to the output of the j-th hidden units on the i-th input vector. For the rest of the section we will use $m = \Theta(n)$.[4]

Let $A \triangleq \{a^1, \ldots, a^n\} \subseteq \{0,1\}^d$ and $B \triangleq \{b^1, \ldots, b^n\} \subseteq \{0,1\}^d$ with $d \triangleq \omega(\log n)$ be the input to the orthogonal vectors problem. To show the promised hardness, we define a matrix M as a vertical concatenation of 3 smaller matrices: M_1, M_2 and M_2 (repeated).

$$M \triangleq \begin{bmatrix} M_1 \\ M_2 \\ M_2 \end{bmatrix}.$$

Both matrices $M_1, M_2 \in \mathbb{R}^{n \times n}$ are of size $n \times n$. Thus, the number of rows of M (equivalently, the number of training examples) is $m \triangleq 3n$. We describe the two matrices M_1, M_2 below. Recall that v_0, v_1, and v_2 are given in Definition 7.6.2.

We select the input examples and weights so that the matrices M_1 and M_2 have the following structure.

- $(M_1)_{i,j} \triangleq S\left(v_0 - (v_2 - v_0) a^i \cdot b^j\right)$. For any two real values $v, v' \in \mathbb{R}$ we write $v \approx v'$ if $v = v'$ up to an inversely super-polynomial additive factor. In other words, $|v - v'| \leq n^{-\omega(1)}$. We observe that if two vectors a^i and b^j are orthogonal, then the corresponding entry $(M_1)_{i,j} = S(v_0) = \Theta(1)$ and otherwise $(M_1)_{i,j} \approx 0$. We will show that a $\left(1 + \frac{1}{4n}\right)$-approximation of the optimal value of the optimization problem Eq. (7.19) will allow us to decide whether there is an entry in M_1 that is $S(v_0) = \Theta(1)$. This will give the required hardness.

  It remains to show how to construct the matrix M_1 using a neural network. We set the weights for the j-th hidden unit to be $\begin{pmatrix} b^j \\ 1 \end{pmatrix}$. That is, d weights are specified by the vector b^j, and we add one more input with weight 1. The i-th example (corresponding to the i-th row of the matrix M_1) is the vector $\begin{pmatrix} -(v_2 - v_0) a^i \\ v_0 \end{pmatrix}$. The output of the j-th unit on this example (which corresponds to the entry $(M_1)_{i,j}$) is equal to

$$\begin{aligned} S\left(\begin{pmatrix} -(v_2 - v_0) a^i \\ v_0 \end{pmatrix} \cdot \begin{pmatrix} b^j \\ 1 \end{pmatrix}\right) &= S\left(v_0 - (v_2 - v_0) a^i \cdot b^j\right) \\ &= (M_1)_{i,j} \end{aligned}$$

[4]Note that our reduction does not explicitly construct M. Instead, the values of the matrix are induced by the input examples and weights.

as required.

- $(M_2)_{i,j} \triangleq S\left(v_1 - (v_2 - v_1)\bar{b^i} \cdot b^j\right)$, where $\bar{b^i}$ is a binary vector obtained from the binary vector b^i by complementing all bits. We observe that this forces the diagonal entries of M_2 to be equal to $(M_2)_{i,i} = S(v_1) = 1/n^{1000K}$ for all $i = 1, \ldots, n$ and the off-diagonal entries to be $(M_2)_{i,j} \approx 0$ for all $i \neq j$.[5]

To complete the description of the optimization problem Eq. (7.19), we assign labels to the inputs corresponding to the rows of the matrix M. We assign label 1 to all inputs corresponding to rows of the matrix M_1 and the first copy of the matrix M_2. We assign label -1 to all remaining rows of the matrix M corresponding to the second copy of matrix M_2.

The proof of the theorem is completed by the following two lemmas.

Lemma 7.6.3. *If there is a pair of orthogonal vectors, then the optimal value of Eq.* (7.19) *is upper bounded by* $(3n-1)l(0) + o(1)$.

Proof. To obtain an upper bound on the optimal value in the presence of an orthogonal pair, we set the vector α to have all entries equal to n^{100K}. For this α we have

- $|(M_1\alpha)_i| \geq \Omega(n^{100K})$ for all $i = 1, \ldots, n$ for which there is exists $j = 1, \ldots, n$ with $a^i \cdot b^j = 0$. Let $q \geq 1$ be the number of such i.

- $|(M_1\alpha)_i| \leq n^{-\omega(1)}$ for all $i = 1, \ldots, n$ for which there is no $j = 1, \ldots, n$ with $a^i \cdot b^j = 0$. The number of such i is $n - q$.

By using the second property of Definition 7.6.1, the total loss corresponding to M_1 is upper bounded by

$$\begin{aligned} q \cdot l(\Omega(n^{100K})) + (n-q) \cdot l(n^{-\omega(1)}) &\leq q \cdot o(1) + (n-q) \cdot (l(0) + o(1/n)) \\ &\leq l_1 \triangleq (n-1) \cdot l(0) + o(1). \end{aligned}$$

Finally, the total loss corresponding to the two copies of the matrix M_2 is upper bounded by

$$\begin{aligned} 2n \cdot l(\pm O(n^{-800K})) &= 2n \cdot (l(0) \pm o(1/n)) \\ &\leq l_2 \triangleq 2n \cdot l(0) + o(1). \end{aligned}$$

The total loss corresponding to the matrix M is upper bounded by $l_1 + l_2 \leq (3n - 1) \cdot l(0) + o(1)$ as required. □

Lemma 7.6.4. *If there is no pair of orthogonal vectors, then the optimal value of Eq.* (7.19) *is lower bounded by* $3n \cdot l(0) - o(1)$.

[5]For all $i \neq j$ we have $\bar{b^i} \cdot b^j \geq 1$. This holds because all vectors b^i are distinct and have the same number of 1s.

Proof. We first observe that the total loss corresponding to the two copies of the matrix M_2 is lower bounded by $2n \cdot l(0)$. Consider the i-th row in both copies of matrix M_2. By using the convexity of the function l, the loss corresponding to the two rows is lower bounded by $l((M_2\alpha)_i) + l(-(M_2\alpha)_i) \geq 2 \cdot l(0)$. By summing over all n pairs of rows we obtain the required lower bound on the loss.

We claim that $\|\alpha\|_\infty \leq n^{10^6 K}$. Suppose that this is not the case and let i be the index of the largest entry of α in magnitude. Then the i-th entry of the vector $M_2\alpha$ is

$$\begin{aligned}(M_2\alpha)_i &= \alpha_i (M_2)_{i,i} \pm n \cdot \alpha_i \cdot n^{-\omega(1)} \\ &\geq \frac{\alpha_i}{n^{1000K}} - \alpha_i n^{-\omega(1)} ,\end{aligned}$$

where we recall that the diagonal entries of matrix M_2 are equal to $(M_2)_{i,i} = S(v_1) = 1/n^K$. If $|\alpha_i| > n^{10^6 K}$, then $|(M_2\alpha)_i| \geq n^{1000k}$. However, by the second property in Definition 7.6.1, this implies that the loss is lower bounded by $\omega(n)$ for the i-row (for the first or the second copy of M_2). This contradicts a simple lower bound of $4n \cdot l(0)$ on the loss obtained by setting $\alpha = 0$ to be the all 0s vector. We use the third property of a nice loss function which says that $l(0) > 0$.

For the rest of the proof, we assume that $\|\alpha\|_\infty \leq n^{10^6 K}$. We will show that the total loss corresponding to M_1 is lower bounded by $n \cdot l(0) - o(1)$. This is sufficient since we already showed that the two copies of M_2 contribute a loss of at least $2n \cdot l(0)$.

Since all entries of the matrix M_1 are inversely super-polynomial (there is no pair of orthogonal vectors), we have that $|(M_1\alpha)_i| \leq n^{-\omega(1)}$ for all $i = 1, \ldots, n$. Using the second property again, the loss corresponding to M_1 is lower bounded by

$$\begin{aligned}n \cdot l(\pm n^{-\omega(1)}) &\geq n \cdot (l(0) - o(1/n)) \\ &\geq n \cdot l(0) - o(1)\end{aligned}$$

as required. □

7.7 Gradient computation

Finally, we consider the problem of computing the gradient of the loss function for a given set of examples. We focus on the network architecture as in the previous section. Specifically, let $F_{\alpha,B}(a) \triangleq \sum_{j=1}^{n} \alpha_j S(a, b^j)$ be the output of a neural net for some function $S : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$, where:

- a is an input vector from the set $A \triangleq \{a^1, \ldots, a^m\} \subseteq \{0,1\}^d$;
- $B \triangleq \{b^1, \ldots, b^n\} \subseteq \{0,1\}^d$ is a set of vectors defining the neural network;
- $\alpha \triangleq (\alpha_1, \ldots, \alpha_n)^\mathsf{T} \in \mathbb{R}^n$ is an n-dimensional real-valued vector defining the neural network.

We first prove the following lemma.

Lemma 7.7.1. *For some loss function $l : \mathbb{R} \to \mathbb{R}$, let $l(F_{\alpha,B}(a))$ be the loss for input a when the label of the input a is $+1$. Consider the gradient of the total loss $l_{\alpha,A,B} \triangleq \sum_{a\in A} l(F_{\alpha,B}(a))$ at $\alpha_1 = \ldots = \alpha_n = 0$ with respect to $\alpha_1, \ldots, \alpha_n$. The sum of the entries of the gradient is equal to $l'(0) \sum_{a\in A, b\in B} S(a, b)$, where $l'(0)$ is the derivative of the loss function l at 0.*

Proof.

$$\frac{\partial l_{\alpha,A,B}}{\partial \alpha_j} = \sum_{a\in A} \frac{\partial l(F_{\alpha,B}(a))}{\partial F_{\alpha,B}(a)} S(a, b^j) = l'(0) \sum_{a\in A} S(a, b^j) \qquad (\text{since } F_{\alpha,B}(a) = 0).$$

□

For the hinge loss function, we have that the loss function is $l(y) = \max(0, 1 - y)$ if the label is $+1$. Thus, $l'(0) = -1$. For the logistic loss function, we have that the loss function is $l(y) = \frac{1}{\ln 2} \ln(1 + e^{-y})$ if the label is $+1$. Thus, $l'(0) = -\frac{1}{2 \ln 2}$ in this case. It is important that for both loss functions the derivative is non-zero.

Proof of Theorem 7.1.8. Since all ℓ_p-norms are within a polynomial factor, it suffices to show the statement for the ℓ_1-norm.

We set $S(a, b) \triangleq \max(0, 1-2a\cdot b)$, which can be easily implemented using the ReLU activation function. Using Lemma 7.7.1, we get that the ℓ_1-norm of the gradient of the total loss function is equal to $|l'(0)| \sum_{a\in A, b\in B} [a \cdot b = 0]$, where $[E] = 1$ if the event E happens and $[E] = 0$ otherwise. Since $l'(0) \neq 0$, the expression allows to count the number of orthogonal pairs of vectors and thus we can reduce the orthogonal vectors problem to the gradient computation problem. Note that if there is no orthogonal pair, then the ℓ_1-norm is 0 and otherwise it is a constant strictly greater than 0. Thus, approximating the ℓ_1-norm within any finite factor allows us to distinguish the two cases. □

Sigmoid activation function. We show that our hardness result holds also for the sigmoid activation function.

Theorem 7.7.2. *Consider a neural net with of size n with the sigmoid activation function $\sigma(z) \triangleq \frac{1}{1+e^{-z}}$. Approximating the ℓ_p norm (for any $p \geq 1$) of the gradient of the empirical risk for m examples within the multiplicative factor of n^C for any constant $C > 1$ takes at least $\Omega(nm)^{1-o(1)}$ time assuming SETH.*

Proof. We set $S(a, b) \triangleq \sigma(-10C(\log n)a \cdot b)$. Using Lemma 7.7.1, we get that the ℓ_1 norm of the gradient is equal to $|l'(0)| \sum_{a\in A, b\in B} \frac{1}{1+e^{10C(\log n)a\cdot b}}$. It is easy to show that this quantity is at least $|l'(0)|/2$ if there is an orthogonal pair and at most $|l'(0)|/(2n^C)$ otherwise. Since $l'(0) \neq 0$, we get the required approximation hardness. □

Polynomial activation function. On the other hand, by using the polynomial lifting technique, we can show that changing the activation function can lead to non-trivially faster algorithms:

Theorem 7.7.3. *Consider a neural net with one hidden layer of size n, with the polynomial activation function $S(z) \triangleq z^r$ for some integer $r \geq 2$. Computing the gradients of the empirical loss function for m examples in $\mathbb{R}^d$ can be done in time $O((n+m)d^r)$.*

Note that the running time of the "standard" back-propagation algorithm is $O(dnm)$ for networks with this architecture. Thus our algorithm improves over back-propagation for a non-trivial range of parameters, especially for quadratic activation function when $r = 2$.

We start by defining the network architecture more formally. We consider a neural network computing a function $f : \mathbb{R}^{1\times d} \to \mathbb{R}$ defined as $f(x) \triangleq S(xA)\alpha$, where

- $x \in \mathbb{R}^{1\times d}$ is an input row vector of dimensionality d.
- $A \in \mathbb{R}^{d\times n}$ is a matrix with j-th column specifying weights of edges connecting the input units with the j-th hidden unit.
- S takes as an input a row vectors with n entries and applies the non-linearity $S(z) = z^r$ entry-wise.
- $\alpha \in \mathbb{R}^n$ is column vector with α_j specifying the weight of the edge that connects the j-th hidden unit with the output linear unit.

Let $X \in \mathbb{R}^{m\times d}$ be the matrix specifying m inputs vectors. The i-th row of X specifies the i-th input vector. Let $w \triangleq f(X) \in \mathbb{R}^m$ be the column vector after applying the function f on every row of the input matrix X. Let $l : \mathbb{R}^m \to \mathbb{R}$ be the total loss function defined as $l(w) \triangleq \sum_{i=1}^m l_i(w_i)$ for some functions $l_i : \mathbb{R} \to \mathbb{R}$.

Let

$$\frac{\partial l}{\partial \alpha} \triangleq \left(\frac{\partial l}{\partial \alpha_1}, \ldots, \frac{\partial l}{\partial \alpha_n}\right)^\intercal \in \mathbb{R}^n$$

be the vector of gradients for weights $\alpha_1, \ldots, \alpha_m$. Let $\frac{\partial l}{\partial A} \in \mathbb{R}^{d\times n}$ be the matrix that specifies gradient of l with respect to entries $A_{k,j}$. That is,

$$\left(\frac{\partial l}{\partial A}\right)_{k,j} \triangleq \frac{\partial l}{\partial A_{k,j}}$$

for $k = 1, \ldots, d$ and $j = 1, \ldots, n$.

Theorem 7.7.4. *We can evaluate $\frac{\partial l}{\partial \alpha}$ and $\frac{\partial l}{\partial A}$ in $O((n+m)d^r)$.*

Proof. Let $l'(w) \triangleq \left(\frac{\partial l_1}{\partial w_1}, \ldots, \frac{\partial l_m}{\partial w_m}\right) \in \mathbb{R}^m$ denote the vector that collects all $\frac{\partial l_i}{\partial w_i}$. Let $X^{(r)} \in \mathbb{R}^{m\times d^r}$ and $A^{(r)} \in \mathbb{R}^{d^r\times n}$ be two matrices such that for every $i = 1, \ldots, m$ and every $j = 1, \ldots, n$ we have $(X^{(r)}A^{(r)})_{i,j} = (XA)^r_{i,j}$, which in turn is equal to the output of the j-th hidden unit on the i-th input vector. Such matrices can be easily constructed using the polynomial lifting technique.

We note that

$$\begin{aligned}\frac{\partial l_i}{\partial \alpha_j} &= \frac{\partial l_i}{\partial w_i} \cdot (\text{output of the } j\text{-th hidden unit on the } i\text{-th input vector}) \\ &= l_i'(w_i) \cdot (X^{(r)} A^{(r)})_{i,j}.\end{aligned}$$

This gives

$$\begin{aligned}\frac{\partial l}{\partial \alpha} &= \left(X^{(r)} A^{(r)}\right)^\mathsf{T} l'(w) \\ &= \left(A^{(r)}\right)^\mathsf{T} \left(\left(X^{(r)}\right)^\mathsf{T} l'(w)\right).\end{aligned}$$

The last expression can be evaluated in the required $O((n+m)d^r)$ time.

We note that

$$\begin{aligned}\frac{\partial l}{\partial A_{k,j}} &= \sum_{i=1}^{n} \frac{\partial l_i}{\partial A_{k,j}} \\ &= \sum_{i=1}^{n} X_{i,k} \cdot r \cdot (\text{input to the } j\text{-th hidden unit})^{r-1} \cdot \alpha_j \cdot l_i'(w_i).\end{aligned}$$

For two matrices A and B of equal size let $A \circ B$ be the entry-wise product. We define the column vector $v_k \in \mathbb{R}^m$: $(v_k)_i \triangleq X_{i,k} \cdot r \cdot l_i'(w_i)$ for $k = 1, \ldots, d$. Then the k-th row of $\frac{\partial l}{\partial A}$ is equal to $(v_k^\mathsf{T} X^{(r-1)} A^{(r-1)}) \circ \alpha^\mathsf{T}$. We observe that we can compute

$$(v_k^\mathsf{T} X^{(r-1)} A^{(r-1)}) \circ \alpha^\mathsf{T} = ((v_k^\mathsf{T} X^{(r-1)}) A^{(r-1)}) \circ \alpha^\mathsf{T}$$

in $O((n+m)d^{r-1})$ time. Since we have to do that for every $k = 1, \ldots, d$, the stated runtime follows. □

7.8 Conclusions

We have shown that a range of kernel problems require quadratic time for obtaining a high accuracy solution unless the strong exponential time hypothesis is false. These problems include variants of kernel SVM, kernel ridge regression, and kernel PCA. We also gave a similar hardness result for training the final layer of a depth-2 neural network. This result is general and applies to multiple loss and activation functions. Finally, we proved that computing the empirical loss gradient for such networks takes time that is essentially "rectangular", i.e., proportional to the product of the network size and the number of examples.

We note that our quadratic (rectangular) hardness results hold for *general* inputs. There is a long line of research on algorithms for kernel problems with running times depending on various input parameters, such as its statistical dimension [YPW+17], degrees of freedom [Bac13] or effective dimensionality [MM16]. It would be interesting to establish lower bounds on the complexity of kernel problems as a function of the

aforementioned input parameters.

Our quadratic hardness results for kernel problems apply to kernels with exponential tails. A natural question is whether similar results can be obtained for "heavy-tailed" kernels, e.g., the Cauchy kernel.[6] We note that similar results for the linear kernel do not seem achievable using our techniques.[7]

Several of our results are obtained by a reduction from the (exact) bichromatic Hamming closest pair problem or the orthogonal vectors problem. This demonstrates a strong connection between kernel methods and similarity search, and suggests that perhaps a reverse reduction is also possible. Such a reduction could potentially lead to faster approximate algorithms for kernel methods: although the exact closest pair problem has no known sub-quadratic solution, efficient and practical sub-quadratic time algorithms for the approximate version of the problem do exist (see, e.g., [AI06, Val12, AR15, AIL+15, ACW16]).

[6]For the kernel density evaluation problem we were, in fact, able to obtain faster algorithms for "polynomially-decaying" kernels which include the Cauchy kernel. See Chapter 8.

[7]In particular, assuming a certain strengthening of SETH, known as the "non-deterministic SETH" [CGI+16], it is provably impossible to show SETH hardness for any of the linear variants of the studied ERM problems, at least via deterministic reductions. This is due to the fact that these problems have short certificates of optimality via duality arguments. Also, it should be noted that linear analogs of some of the problems considered in this section (e.g., linear ridge regression) can be solved in $O(nd^2)$ time using SVD methods.

Chapter 8

Kernel density evaluation

Kernel density evaluation is a basic computational task with many applications. Given a kernel function $k : \mathbb{R}^d \times \mathbb{R}^d \to [0, 1]$ and a dataset $P \subset \mathbb{R}^d$, we define the *kernel density function* of P at a point $q \in \mathbb{R}^d$ as:

$$\mathrm{KDF}_P(q) \triangleq \frac{1}{|P|} \sum_{p \in P} k(p, q).$$

The task of computing KDF can be formulated in multiple ways:

- As a data structure problem: given P, build a data structure supporting KDF evaluation queries for a given q,
- In the batch setting: given two sets P and Q, compute $\mathrm{KDF}_P(q)$ for all $q \in Q$, or
- In the "all pairs" setting: given two sets P and Q, compute

$$k(P, Q) \triangleq \sum_{q \in Q} \mathrm{KDF}_P(q). \tag{8.1}$$

The batch version of the problem has been studied extensively. In particular, the celebrated Fast Multipole Method gives an efficient approximate algorithm for this problem in low dimensions [GS91] and has been very influential in scientific computing. Unfortunately, the complexity of this approach scales exponentially with the dimension, while many applications require evaluating kernel densities for *high-dimensional* point-sets. This includes *kernel density estimation*, a classic tool in non-parametric statistics, where the kernel function is used to extend the empirical distribution function over a discrete set of points smoothly to the whole space. This in turn yields algorithms for *mode estimation* [GSM03], *outlier detection* [SZK14], *density based clustering* [RW10] and other problems. Another class of applications stems from applying kernel methods (e.g., regression [SSPG16]) to objects described by point-clouds or distributions, by extending kernel functions to pairs of sets [GFKS02]. Computing over such *set kernels* requires many "all-pairs" evaluations as defined in Equation 8.1.

Yet another application is kernel mean estimation using an empirical average (see, e.g., [MFS+17], section 3.4.2).

A popular method for evaluating kernel densities in high dimensions involves random sampling [IHG08]. Under the assumption that $\mathrm{KDF}_P(x) \geq \mu$ for some $\mu \in (0, 1]$, sampling $\Theta(\frac{1}{\mu\epsilon^2} \log(1/\delta))$ points in P suffices to estimate the desired value up to a factor of $1+\epsilon$ with probability $1-\delta$. This yields a data structure whose query time is equal to the number of samples times d. The query time can be improved further: in a very recent work [CS17] showed that for several kernels, one can reduce the query time to roughly $\frac{d}{\sqrt{\mu}\epsilon^2}$. Their approach applies to multiple popular kernels, including Gaussian, exponential and t-Student for constant t. However, their technique is tailored to the specific cases since it requires a hash function family with collision probability matching the kernel in a certain way. It is unclear how to generalize it to handle more general kernels. All of the aforementioned results translate to the batch setting, with the running time essentially equal to the query time bound multiplied by $|Q|$.

The main disadvantage of the above high-dimensional results is that the value of μ can be arbitrarily low, leading to high query time. Unfortunately, there is evidence that one cannot obtain fast query times (independent of μ) for arbitrary kernels. Unconditional lower bounds have already been shown for core-sets [PT18a, PT18b] (with almost matching upper bounds) and in a slightly more general computational model [CS17]. Moreover, in Section 8.6 we demonstrate that in the "all-pairs" setting, approximating KDF values for kernels with exponential tails (such as Gaussian) up to a constant factor requires $n^{2-o(1)}$ time (where $n = |Q| = |P|$) unless SETH fails. This holds even if the values are lower bounded by $\mu = \exp(-\log^{O(1)} n)$; the bound can be strengthened further to any $n^{-\omega(1)}$ by using the recent result of [Rub18]. This gives evidence that, for kernels that decay fast enough, there is no algorithm with a runtime of $n^{2-\Omega(1)}/\mu^{o(1)}$. This leads to the natural question: What class of kernels admits efficient kernel density evaluation?

8.1 Our results

We present a data structure with a small query time, independent of μ, for kernels whose value changes at most *polynomially* with the distance. Formally:

Definition 8.1.1 ((l,t)-smooth function)**.** *We call a function $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$* (l,t)-smooth *if for all $p, p', q \in \mathbb{R}^d$ with $p \neq q$ and $p' \neq q$ we have*

$$\max\left(\frac{k(p,q)}{k(p',q)}, \frac{k(p',q)}{k(p,q)}\right) \leq l \max\left(\frac{\|p-q\|}{\|p'-q\|}, \frac{\|p'-q\|}{\|p-q\|}\right)^t.$$

Note that kernel functions do not need to be non-increasing or non-decreasing in the distance to be smooth. This general class of kernels includes several well-studied functions [Gen01, Ras04]. For example:

- Rational Quadratic kernel $k(p,q) = \frac{1}{(1+\|p-q\|^2)^\beta}$ is $(1, 2\beta)$-smooth.[1]
- t-Student kernel $k(p,q) = \frac{1}{1+\|p-q\|^t}$ is $(1,t)$-smooth.

Smooth kernels are frequently used in machine learning and statistics, often yielding results similar to or better than the (more popular) Gaussian kernel (see, e.g., [SGPS15] (supplementary material section)). For such kernels, we give a data structure with the query time of

$$l2^{O(t)} d \log(n\Phi)/\varepsilon^2 \log(1/\delta),$$

where Φ is the aspect ratio of the data and query points (see Theorem 8.3.1). Furthermore, if we assume a natural kernel decay property (satisfied by the above kernels), the query time becomes $l2^{O(t)} d \log(n)/\varepsilon^2 \log(1/\delta)$, removing the dependence on the aspect ratio (see Section 8.4). This improves over the aforementioned algorithm of [CS17] for a wide class of kernel functions. In particular, we achieve the query time of

$$O(d \log(n)/\varepsilon^2 \log(1/\delta))$$

for the Cauchy kernel, and more generally for the Rational Quadratic kernel with $\beta = O(1)$. The above results assume that the distance function $\|p-q\|$ is induced by the Euclidean norm.

Other distance functions. We consider a more general version of our setting where the kernel function is smooth with respect to a distance function $s : \mathbb{R}^d \times \mathbb{R}^d \to [0, \infty)$. For example, we can consider the following generalization of the t-Student kernel $k(p,q) = \frac{1}{1+s(p,q)^t}$ that is $(1,t)$-smooth with respect to s. For distance functions that are constant power of ℓ_2, as powering changes the smoothness parameters of the kernel only by constant factors, we can immediately apply our algorithm. Using this fact, we can also apply the algorithm to ℓ_w norms with $w \in [1,2)$ by embedding them into $(\ell_2)^2$ [LN14]. More generally, it can be extended to other distance functions that can be embedded into $(\ell_2)^2$ with some low distortion D (see Section 8.5). Perhaps surprisingly, this only multiplies the *running time* of the algorithm (by a factor of $D^{O(t)}$), while the approximation factor remains equal to $1+\epsilon$. In particular, this yields an algorithm for the ℓ_∞ norm with the running time multiplied by a factor $d^{O(t)}$; in the specific case of the t-Student kernel, the query time becomes $d^{O(t)} \log(n)/\epsilon^2 \log(1/\delta)$. We complement the latter algorithm by showing that, assuming SETH, there is no kernel density evaluation algorithm for the t-Student kernel under ℓ_∞ norm with a query time of $d^{O(1)} 2^{t/10} n^{o(1)}$ and any constant approximation factor (see Section 8.6).

8.2 Preliminaries

For $w \geq 1$ and $p \in \mathbb{R}^d$, we refer to $\|p\|_w \triangleq \left(\sum_{i\in[d]} |p_i|^w\right)^{1/w}$ as the ℓ_w distance. For $w = \infty$ we define $\|p\|_\infty \triangleq \max_{i\in[d]} |p_i|$ and refer to it as ℓ_∞ distance. For $w = 2$ we

[1] For $\beta = 1$, it is often referred to as *Cauchy kernel*.

will sometimes omit the subscript and write $\|p\|$. For $z > 0$ we write $(\ell_w)^z$ to denote the distance function $\|p\|_w^z = (\|p\|_w)^z$.

For an event E we set $[E] = 1$ if the event E happens and $[E] = 0$ otherwise.

8.3 Algorithm for kernel density evaluation

We show the following theorem.

Theorem 8.3.1. *Let $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ be an (l,t)-smooth function that can be evaluated in time t_k and P be a point-set of n points from $\mathbb{R}^d$. For a query point $q \in \mathbb{R}^d$ let $\Phi_{\max}$ be the upper bound on distance between any two points from $P \cup \{q\}$ and let $\Phi_{\min}$ be the lower bound on the distince between any two distanct points from $P \cup \{q\}$. Define the "aspect ratio" $\Phi \triangleq \Phi_{\max}/\Phi_{\min}$.*

For any $\varepsilon > 0$, it is possible to preprocess the point-set P in $nl2^{O(t)}(d+\log(n\Phi))/\varepsilon^2$ time such that for any query point $q \in \mathbb{R}^d$ we can output a $1+\varepsilon$ approximation to $KDF_P(q)$ in time $l2^{O(t)}t_k \log(n\Phi)/\varepsilon^2$.

The success probability can be amplified to $1-\delta$ by increasing the preprocessing and the query time by a multiplicative factor of $\log(1/\delta)$.

We note that typically $t_k = \Theta(d)$, e.g., for the Rational Quadratic and the t-Student kernels. We note that the t-Student kernel $\frac{1}{1+\|p-1\|^t}$ is $(1,t)$-smooth and we get the preprocessing time $n2^{O(t)}(d+\log(n\Phi))/\varepsilon^2$ and $2^{O(t)}d\log(n\Phi)/\varepsilon^2$ query time. If we replace the ℓ_2 distance with $\ell_w, w \geq 1$ distance, we get the preprocessing time $nd^{O(t)}\log(n\Phi)/\varepsilon^2$ and $d^{O(t)}\log(n\Phi)/\varepsilon^2$ query time as the kernel function $\frac{1}{1+\|p-1\|_w^t}$ is $(d^{O(t)},t)$-smooth (with respect to ℓ_2). For $w \in [1,2)$ we get improved runtimes in Section 8.5.

Let $W \triangleq \frac{1}{n}\sum_{i=1}^n w_i$ for some real values w_i and our goal is to estimate W. As an example, think of $w_i = k(p_i, q)$, where points p_i come from the point-set $P = \{p_1, p_2, \ldots, p_n\}$. Then $\mathrm{KDF}_P(q) = W$ and we recover the kernel density evaluation problem.

Let H be a family of hash functions $h : [n] \to S$. Think of S as a fairly small set. For example, for the kernel density evaluation problem we will construct family H with the set S of size (roughly) $|S| \leq O(\log n)$. Fix any $h \in H$ and let i_s be a uniformly random sample from $h^{-1}(s)$ for every $s \in S$. That is, we pick a random element from every "bucket" $s \in S$. Then the random variable

$$Z \triangleq \frac{1}{n}\sum_{s \in S} |h^{-1}(s)| w_{i_s}$$

is an unbiased estimator of the quantity $W = \frac{1}{n}\sum_{i=1}^n w_i$, i.e., $\mathbb{E}[Z] = W$.

To be able to efficiently estimate W, we also need to bound the variance of the random variable Z. We note that the variable depends on the family H. What properties should the family H of hash functions satisfy so that the random variable has a small variance (over a uniformly random hash function $h \in H$ and random samples from the buckets)?

It turns out that the following property is sufficient. There exists a quantity T such that for any $i, i' \in [n]$ we have the following bound: $\Pr_{h \leftarrow H}[h(i) = h(i')] \leq T \frac{w_i}{w_{i'}}$. As we show below, if we have this property, then

$$\mathrm{Var}[Z] \leq (1+T)W^2 = (1+T)\,\mathbb{E}[Z]^2.$$

Thus, by taking an average of $O(T/\varepsilon^2)$ samples of Z we get a $1 \pm \varepsilon$ approximation of W. Think of T as a constant. For the kernel density evaluation problem we will show that T is a function only of l and t—the smoothness parameters of the kernel function. It remains to show the bound on the variance.

$$\begin{aligned}\mathrm{Var}[Z] \leq \mathbb{E}[Z^2] &= \frac{1}{n^2}\,\mathbb{E}\sum_{s \in S} |h^{-1}(s)|^2 w_{i_s}^2 \\ &\quad + \frac{1}{n^2}\,\mathbb{E}\sum_{s \neq s'} |h^{-1}(s)|\, w_{i_s}\, |h^{-1}(s')|\, w_{i_{s'}}.\end{aligned}$$

We bound the two terms separately. We start with the second term, which is equal to

$$\begin{aligned}&\frac{1}{n^2}\ \mathop{\mathbb{E}}_{h \leftarrow H}\ \mathop{\mathbb{E}}_{\substack{\forall s \in S: \\ i_s \leftarrow h^{-1}(s)}}\ \sum_{s \neq s'} |h^{-1}(s)|\, w_{i_s}\, |h^{-1}(s')|\, w_{i_{s'}} \\ &= \frac{1}{n^2}\,\mathop{\mathbb{E}}_{h} \sum_{s \neq s'} \left(|h^{-1}(s)| \mathop{\mathbb{E}}_{i_s} w_{i_s}\right)\left(|h^{-1}(s')| \mathop{\mathbb{E}}_{i_{s'}} w_{i_{s'}}\right) \\ &\leq \frac{1}{n^2}\,\mathop{\mathbb{E}}_{h} \left(\sum_{s} |h^{-1}(s)| \mathop{\mathbb{E}}_{i_s} w_{i_s}\right)\left(\sum_{s'} |h^{-1}(s')| \mathop{\mathbb{E}}_{i_{s'}} w_{i_{s'}}\right) \\ &= \frac{1}{n^2}\left(\sum_{i=1}^{n} w_i\right)^2 \\ &= W^2.\end{aligned}$$

Now we upper bound the first term:

$$\begin{aligned}\frac{1}{n^2}\,\mathbb{E}\sum_{s \in S} |h^{-1}(s)|^2 w_{i_s}^2 &= \frac{1}{n^2}\,\mathop{\mathbb{E}}_{h}\sum_{s} |h^{-1}(s)|^2 \mathop{\mathbb{E}}_{i_s} w_{i_s}^2 \\ &= \frac{1}{n^2}\,\mathop{\mathbb{E}}_{h}\sum_{s} |h^{-1}(s)| \sum_{i' \in h^{-1}(s)} w_{i'}^2 \\ &= \frac{1}{n^2}\,\mathop{\mathbb{E}}_{h}\sum_{s}\sum_{i \in [n]} [i \in h^{-1}(s)] \sum_{i' \in [n]} [i' \in h^{-1}(s)] w_{i'}^2 \\ &= \frac{1}{n^2}\sum_{i,i' \in [n]} \mathop{\mathbb{E}}_{h}\sum_{s} [i, i' \in h^{-1}(s)] w_{i'}^2\end{aligned}$$

$$
\begin{aligned}
&= \frac{1}{n^2} \sum_{i,i' \in [n]} \Pr_h[h(i) = h(i')] w_{i'}^2 \\
&\leq \frac{1}{n^2} T w_i w_{i'} \\
&= TW^2,
\end{aligned}
$$

where in the inequality we use the bound $\Pr_h[h(i) = h(i')] \leq T\frac{w_i}{w_{i'}}$. We get the required bound on the variance.

We are ready to describe our faster algorithm for the kernel density evaluation problem. Let k be an (l,t)-smooth kernel. Given a point-set $P = \{p_1, \ldots, p_n\} \subset \mathbb{R}^d$ consisting of n points, we preprocess the set in $O(ntd + nt\log(n\Phi))$ time so that the following holds. For a query point $q \in \mathbb{R}^d$, define $w_i \triangleq k(p_i, q)$. The preprocessing step is randomized and it induces a hash function $h : [n] \to S$ with $|S| \leq O(\log(n\Phi))$. For every $s \in S$, a uniformly random i_s from $h^{-1}(s)$ can be sampled in $2^{O(t)}$ time. Thus, the random variable Z can be computed in $\left(2^{O(t)} + t_k\right)\log(n\Phi) + O(td)$ time (the factor $O(td)$ comes from a projection step as described below). To get a $1+\varepsilon$ approximation, we build $O(T/\varepsilon^2)$ independent data structures and take average of the random variables produced by the data structures. We show that $T \leq l2^{O(t)}$. The final preprocessing time becomes $nl2^{O(t)}(d + \log(n\Phi))/\varepsilon^2$ and the final query time becomes $l2^{O(t)}t_k \log(n\Phi)/\varepsilon^2$. To simplify the expression for the query time, we used the fact that $t_k \geq \Omega(d)$.

Preprocessing step. Let d' be an integer (we will later set $d' = 10t$). Pick a random d'-dimensional subspace of the d-dimensional space and let $A' \in \mathbb{R}^{d' \times d}$ be the projection matrix to this subspace. Let $A \triangleq \sqrt{d/d'}A'$ be the scaled projection matrix. We map every point $p \in P$ to $Ap \in \mathbb{R}^{d'}$. In the rest of the section we will use the following concentration inequality multiple times.

Lemma 8.3.2 ([DG03])**.** *For all $K \geq 2$ and for all $x \in \mathbb{R}^d$ we have*

$$\Pr[\|x\|/K \leq \|Ax\| \leq K\|x\|] \geq 1 - K^{-d'/4}.$$

After mapping the point-set P to the d'-dimensional space, we build a d'-dimensional quadtree for the points $Ap, p \in P$ as follows. We choose a large enough d'-dimensional axis-parallel box with equal side-lengths so that it contains all points. We recursively split the box into smaller boxes as long as the box has more than one point in it. Let R be the side-length of a box. If the box has more than one point, we split the box into $2^{d'}$ equal and non-overlapping smaller boxes of side-length $R/2$. We bound the time spent on the preprocessing at the end of the analysis.

Query step. In the preprocessing step we described the construction of the quadtree such that the partitioning of the boxes stops when we reach a box with a single point. For the sake of the simplicity of the description of the query step it is useful think of the partitioning as infinite: we keep partitioning boxes with a single point. The

actual implementation does not need infinite partitioning, it is only for the easy of exposition.

Let $q \in \mathbb{R}^d$ be the query point and Aq be the query point mapped into the d'-dimensional space. Below we describe the hash function $h : [n] \to S$. For a point p and a real number r, let $\mathrm{Ball}(p, r)$ be the Euclidean ball centered at the point p and of radius r.

For $r = R, R/2, R/4, R/8, \ldots$ we do the following. Consider all boxes with the side-length r that do not intersect $\mathrm{Ball}(Aq, \sqrt{d'}r)$ and that are not covered by the boxes considered in the previous iterations. We add r to S and for every point p_i from any of the boxes considered in this iteration we assign $h(i) = r$.

We observe that this procedure partitions the point $Ap, p \in P$ into subsets such that the distances from q' to all points from the same subset are "roughly" equal.

In what follows we bound the time needed to compute the random variable

$$Z \triangleq \frac{1}{n} \sum_{s \in S} |h^{-1}(s)| k(p_{i_s}, q)$$

and variance of Z. We note that even though we sample i_s from the d'-dimensional space, we compute the kernel value $k(p_{i_s}, q)$ in the d-dimensional space.

Variance of Z. Our goal is to show that $\Pr_h[h(i) = h(i')] \leq T \frac{w_i}{w_{i'}}$ for $T \leq l 2^{O(d')}$. We will show that $\Pr_h[h(i) = h(i')] \leq T \min\left(\frac{w_i}{w_{i'}}, \frac{w_{i'}}{w_i}\right)$. We observe that it is sufficient to show

$$\Pr_h[h(i) = h(i')] \leq 2^{O(d')} \min\left(\frac{\|p_i - q\|}{\|p_{i'} - q\|}, \frac{\|p_{i'} - q\|}{\|p_i - q\|}\right)^t$$

as the rest follows from the smoothness property as stated in Definition 8.1.1. For the rest we set $d' \triangleq 10t$.

Without loss of generality $\|p_{i'} - q\| \geq \|p_i - q\|$. Let $K \triangleq \|p_{i'} - q\| / \|p_i - q\| \geq 1$. Our goal is to show $\Pr_h[h(i) = h(i')] \leq 2^{O(d')} K^{-t}$. Note that if $K \leq 1000$, then the inequality is immediate for large enough hidden constant in the $2^{O(d')}$ factor. Thus, we assume that $K > 1000$. We claim that $h(i) = h(i')$ implies

$$\|Ap_{i'} - Aq\| \leq 10 \|Ap_i - Aq\|. \tag{8.2}$$

Indeed, let $r \triangleq h(i) = h(i')$. By the construction of the partition, both points $Ap_{i'}$ and Ap_i, lie outside $\mathrm{Ball}(Aq, \sqrt{d'}r)$. That is,

$$Ap_{i'}, Ap_i \notin \mathrm{Ball}(Aq, \sqrt{d'}r). \tag{8.3}$$

Furthermore, they are not covered by boxes of side-length $2r$ that do not intersect $\mathrm{Ball}(Aq, \sqrt{d'}2r)$, which implies that

$$Ap_{i'}, Ap_i \in \mathrm{Ball}(Aq, \sqrt{d'}4r). \tag{8.4}$$

Eqs. (8.3) and (8.4) together imply Eq. (8.2). Eq. (8.2) and $K > 1000$ together imply that either the event $\|Ap_{i'} - Aq\| \leq (K/10)^{-1/2} \|p_{i'} - q\|$ or the event $\|Ap_i - Aq\| \geq$

$(K/10)^{1/2}\|p_i - q\|$ holds. We call the first event E_1 and the second event E_2. We use Lemma 8.3.2 and conclude

$$\begin{aligned}\Pr_h[h(i) = h(i')] &\leq \Pr[E_1 \text{ or } E_2] \\ &\leq \Pr[E_1] + \Pr[E_2] \\ &\leq 2 \cdot (K/10)^{-d'/8} \\ &\leq 2^{O(d')} K^{-t}\end{aligned}$$

as required.

Time needed to compute Z. For every r we sample a uniformly random point from the union of boxes. To bound the time needed to compute Z, we need to bound two quantities: the number of different boxes considered per side-length r and the number of different values r per query. We start by bounding the first quantity—the number of boxes. From the analysis of the variance we know that all boxes of side-length r are contained in $\text{Ball}(Aq, \sqrt{d'}4r)$. Thus, the number of boxes is upper bounded by

$$\text{Volume}(\text{Ball}(Aq, \sqrt{d'}4r))/\text{Volume}(\text{box with side-length } r) \leq 2^{O(d')},$$

where we use the fact that the volume of a d-dimensional ball of radius r is upper bounded by $\frac{2^{O(d)}}{\sqrt{d}^d} r^d$ [Bal97]. This implies that a uniformly random i_r from $h^{-1}(r)$ can be sampled in $2^{O(d')}$ time. To bound the number of different values r considered in a query step, we use the definition of $\Phi_{\max}$ and $\Phi_{\min}$. Using Lemma 8.3.2 we argue that the largest distance between any two points from $\{Ap \mid p \in P\} \cup \{Aq\}$ is upper bounded by $\Phi'_{\max} \leq n^{O(1)}\Phi_{\max}$ with all but arbitrarily small constant probability. Similarly, the smallest distance between any two distinct points is lower bounded by $\Phi'_{\min} \geq n^{-O(1)}\Phi_{\min}$ with all but arbitrary small constant probability. Let $\Phi' \triangleq \Phi'_{\max}/\Phi'_{\min}$. Since the side-length of the considered boxes decreases by a factor of 2 in every iteration, the number of different values r that are considered is upper bounded by $|S| \leq O(\log \Phi') \leq O(\log(n\Phi))$ as promised.

Finally, we account for the time needed to compute Aq, which is $O(d'd)$.

Preprocessing time. The time needed to compute $Ap, p \in P$ is $O(nd'd)$. Since the number of quantities r is $O(\log(n\Phi))$, the total time spend on building the multi-dimensional quadtree is $O(nd'd + nd' \log(n\Phi))$.

8.4 Removing the aspect ratio

In this section we introduce a natural decay property for kernels that is satisfied for many smooth kernels such as t-Student, rational quadratic and other non-smooth kernels such as Guassian or exponential. As we will see, this property allows us to get rid of the dependence on the aspect ratio Φ in Theorem 8.3.1.

Definition 8.4.1 ($(\alpha, \gamma, \varepsilon)$-decay)**.** *For $\alpha, \gamma \geq 1$ and $\varepsilon > 0$ we call a function $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ $(\alpha, \gamma, \varepsilon)$-decaying, if for all $n \geq 2$ the following holds.*

- *for all* p, p', q *with* $n^{-\gamma}\|p-q\| \geq \|p'-q\| \geq n^{-\alpha}$, $k(p,q) \leq \varepsilon k(p',q)/n$;

- *for all* p, q *with* $\|p-q\| < n^{-\alpha}$, $k(p,q) = (1 \pm \varepsilon)\, k(q,q)$.

That is, the function k decays sufficiently fast for all pairs of points that are sufficiently far, and at the same time does not change by much for points that are "close". This property makes it possible to consider only $O(\gamma \log n)$ different scales when estimating kernel density (although the actual scales could be quite different for different queries). We show that for $(O(1), O(1), \varepsilon)$-decaying function we can approximate $\mathrm{KDF}_P(q)$ within a factor of $1 \pm O(\varepsilon)$ and get rid of the aspect ratio. We note that the Rational Quadratic kernel and the t-Student kernel are $(O(1), O(1), 1/n^{10})$-decaying as long the parameters β and t (resp.) can be lower bounded by an arbitrary small constant.

In Theorem 8.3.1 we provided an algorithms whose preprocessing and the query runtime bounds depend on the aspect ratio Φ. In this section we show that essentially the same algorithm allows to get rid of the dependency on Φ if the (l,t)-smooth kernel function k is also $(\alpha, \gamma, \varepsilon)$-decaying.

We define $m \triangleq |\{p \in P \mid \|p-q\| \leq n^{-\alpha}\}|$ to be the number of points in P that are at distance less than $n^{-\alpha}$ from q. Let $z \triangleq \min_{p \in P} \max(\|p-q\|, n^{-\alpha})$ be the closest distance from q to $p, p \in P$ thresholded at $n^{-\alpha}$. The following expression follows immediately from Definition 8.4.1.

$$m\, k(q,q) + \sum_{\substack{p \in P: \\ z \leq \|p-q\| \leq z n^{\gamma}}} k(p,q) \;=\; (1 \pm O(\varepsilon))\mathrm{KDF}_P(q). \tag{8.5}$$

To estimate the left hand size of Eq. (8.5), we modify the algorithm from Theorem 8.3.1. The points $p \in P$ with $z \leq \|p-q\| \leq z n^{\gamma}$ are mapped to Ap and by Lemma 8.3.2 and the construction of the hash function, the points will be be assigned at most $O(\log n^{\gamma}) \leq O(\gamma \log n)$ different hash values. Thus, to estimate the contribution from these points we only need to examine $O(\gamma \log n)$ different hash values. To be able to examine these hash values, we must be able to approximate $\min_{p \in P} \|Ap - Aq\|$, which can be done in $O(d' \log n) \leq O(t \log n)$ time for a given Aq after $2^{O(t)} n \log n$ preprocessing of $Ap, p \in P$ (Theorems 2.10 and 3.16 in [HPIM12]). After examining these hash values, we can count the number of points that are left and are closer than the examined points in $2^{O(d')}$ time.

Altogether we arrive at the following theorem.

Theorem 8.4.2. *Let k be an (l,t)-smooth function that is also $(\alpha, \gamma, \varepsilon)$-decaying. After $nl2^{O(t)}(d + \gamma \log n)/\varepsilon^2$ time preprocessing, for a given q we can approximate $KDF_P(q)$ within the factor of $1 + O(\varepsilon)$ in $l2^{O(t)} t_k \gamma \log(n)/\varepsilon^2$ time.*

For the Rational Quadratic kernel and the t-Student kernel we get the runtime bounds with $\gamma = O(1)$ as long as the parameters β and t (resp.) can be lower bounded by an arbitrary small constant.

8.5 Algorithms via embeddings

8.5.1 A useful tool

We will use the following theorem which is a variant of Theorem 116 from [LN14].

Theorem 8.5.1. *Let $2 > w \geq 1$, $R > 0$ and $\Phi \geq 2$ be real numbers. There exists a mapping $f : \mathbb{R}^d \to \mathbb{R}^{O(d \log(\Phi d/\varepsilon)/\varepsilon)}$ such that for any $p, q \in \mathbb{R}^d$ with $R \leq \|p-q\|_w \leq \Phi R$, we have*

$$(1-\varepsilon)\|p-q\|_w^w \leq \|f(p)-f(q)\|_2^2 \leq (1+\varepsilon)\|p-q\|_w^w.$$

The embedding can be computed in $O(d \log(\Phi d/\varepsilon)/\varepsilon)$ time.

The proof in [LN14] of the theorem starts by noting that it is sufficient to construct an embedding $f_0 : \mathbb{R} \to \mathbb{R}^{O(\log(\phi/\varepsilon)/\varepsilon)}$ such that for any $x, y \in \mathbb{R}$ with $|x-y| \leq \phi R$ we have

$$(1-\varepsilon)|x-y|^w - \varepsilon R^w \leq \|f_0(x)-f_0(y)\|_2^2 \leq (1+\varepsilon)|x-y|_w^w + \varepsilon R^w.$$

The embedding for d dimensions follows by concatenating d one-dimensional embeddings, one for each dimension. The one-dimensional embedding is shown assuming $\phi = d^{O(1)}$, but in fact it works for arbitrary ϕ since the parameter d is not tied to the dimensionality of the original space for the map f_0. Furthermore, we note that the argument works for all $w \in [1, 2)$, even though it is stated that $w \in (1, 2)$. This is because the argument is a discretization of the mapping in Remark 5.10 from [MN04], which holds for $w = 1$ as well. Overall, we obtain the following:

Theorem 8.5.2 ([LN14])**.** *Let $2 > w \geq 1$, $R > 0$ and $\phi \geq 2$ be real numbers. There exists a mapping $f : \mathbb{R}^d \to \mathbb{R}^{O(d \log(\phi/\varepsilon)/\varepsilon)}$ such that for any $p, q \in \mathbb{R}^d$ with $\|p-q\|_w \leq \phi R$, we have*

$$(1-\varepsilon)\|p-q\|_w^w - \varepsilon R^w d \leq \|f(p)-f(q)\|_2^2 \leq (1+\varepsilon)\|p-q\|_w^w + \varepsilon R^w d.$$

Furthermore, in the statement of Theorem 8.5.2, if p and q additionally satisfy $dR \leq \|p-q\|_w$, then

$$(1-2\varepsilon)\|p-q\|_w^w \leq \|f(p)-f(q)\|_2^2 \leq (1+2\varepsilon)\|p-q\|_w^w.$$

This allows us to rewrite Theorem 8.5.2 in terms of the "aspect ratio" Φ and gives Theorem 8.5.1.

8.5.2 Algorithms

Suppose that a kernel function $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is (l, t)-smooth with respect to some distance function $s : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$. That is,

$$\max\left(\frac{k(p,q)}{k(p',q)}, \frac{k(p',q)}{k(p,q)}\right) \leq l \max\left(\frac{s(p,q)}{s(p',q)}, \frac{s(p',q)}{s(p,q)}\right)^t.$$

Furthermore, suppose that the distance function s can be embedded into $(\ell_w)^z$ with distortion D for some constants $w \in [1,2)$ and $z > 0$. That is, for some map $f : \mathbb{R}^d \to \mathbb{R}^{d'}$ and for all $p, q \in \mathbb{R}^d$ we have

$$\|f(p) - f(q)\|_w^z \leq s(p,q) \leq D\|f(p) - f(q)\|_w^z. \tag{8.6}$$

By Theorem 8.5.1 we know that there exists an embedding $f' : \mathbb{R}^{d'} \to \mathbb{R}^{d''}$ of $(\ell_w)^w$ into $(\ell_2)^2$. We use this theorem and combine it with Eq. (8.6). We conclude that for any $R > 0$ and $\Phi \geq 2$ and for $d'' = O(d' \log(\Phi D d'))$ we have the following. For all $p, q \in \mathbb{R}^d$ such that $R \leq s(p,q) \leq \Phi R$:

$$\|f'(f(p)) - f'(f(q))\|_2^{2z/w} \leq s(p,q) \leq D^{O(1)}\|f'(f(p)) - f'(f(q))\|_2^{2z/w}.$$

This allows us to check that the kernel k is $(lD^{O(t)}, O(t))$-smooth with respect to ℓ_2 for points p, p', q that satisfy $R \leq s(p,q), s(p',q) \leq \Phi R$. We use Theorem 8.3.1 and obtain the following result.

Theorem 8.5.3. *Let k be an (l,t)-smooth kernel with respect to a distance function s. Let t_k be the time needed to evaluate k. Suppose that s can be embedded into $(\ell_w)^z$ with distortion D for some constants $w \in [1,2)$ and $z > 0$. We assume that $D \geq 2$. Let $f : \mathbb{R}^d \to \mathbb{R}^{d'}$ be the embedding and let t_f be the time needed to perform the embedding. For a point-set $P \subset \mathbb{R}^d$ of size $|P| = n$ and a query $q \in \mathbb{R}^d$ let Φ be the upper bound on the aspect ratio on $P \cup \{q\}$ with respect to s (similarly as in Theorem 8.3.1). Let $d'' \triangleq O(d' \log(\Phi D d'))$.*

For any $\varepsilon > 0$, it is possible to preprocess the point-set P in $nlD^{O(t)}(t_f + d'' + \log(n\Phi D))/\varepsilon^2$ time such that for any query point $q \in \mathbb{R}^d$ we can output a $1+\varepsilon$ approximation to $KDF_P(q)$ in time $lD^{O(t)}(t_f + d'' + t_k \log(n\Phi D))/\varepsilon^2$.

We illustrate the use of Theorem 8.5.3 by obtaining an algorithm for the kernel function $k(p,q) = \frac{1}{1+s(p,q)^t}$, where $s(p,q) = \mathrm{EMD}(p,q)$ is the planar earth mover's distance between $p, q \in \mathbb{R}^d$. We interpret p and q as two measures over a two-dimensional grid of size $\sqrt{d} \times \sqrt{d}$. The distance between two points from the grid are measured in ℓ_1.

Theorem 8.5.4. *Let $k(p,q) = \frac{1}{1+EMD(p,q)^t}$ be a kernel function and $P \subset \mathbb{R}^{\sqrt{d}\times\sqrt{d}}$ be a set of n measures. Let Φ be an upper bound on the aspect ratio of $P \cup \{q\}$ for a query measure $q \in \mathbb{R}^{\sqrt{d}\times\sqrt{d}}$. It is possible to preprocess the set of measures P in time $n(\log d)^{O(t)}(d\log(\Phi) + \log n)/\varepsilon^2$ such that the following holds. Given a query measure $q \in \mathbb{R}^{\sqrt{d}\times\sqrt{d}}$, we can approximate $KDF_P(q)$ within time $1+\varepsilon$ factor in $(\log d)^{O(t)} d^3 \log(n\Phi)$.*

Proof. It is known that the earth mover's distance embeds into ℓ_1 with distortion $D \leq O(\log d)$ [Cha02, IT03, NS07]. The resulting dimensionality is $d' = O(d)$ and the embedding can be computed in time $t_f = O(d)$. It is known how to evaluate $\mathrm{EMD}(p,q)$ in time $t_k = O(d^3)$ for arbitrary two measures p and q (using the "Hungarian" method [Law76]). The guarantees follow from Theorem 8.5.3 by observing that the kernel function $k(p,q)$ is $(1,t)$-smooth with respect to $s(p,q) = \mathrm{EMD}(p,q)$. □

Using Theorem 8.5.3, for any $1 \leq w \leq 2$ and $t > 0$, we can answer queries for the kernel function $k(p,q) = \frac{1}{1+\|p-q\|_w^t}$ in time $2^{O(t)} d \log(n\Phi d)/\varepsilon^2$ after $n 2^{O(t)}(d \log(\Phi d) + \log(n\Phi))/\varepsilon^2$ time preprocessing.

8.6 Hardness for kernel density evaluation

Hardness for ℓ_∞ distance. Let $s(p,q) \triangleq \|p-q\|_\infty$ be the ℓ_∞ distance function. Consider the kernel function $k(p,q) \triangleq \frac{1}{1+s(p,q)^t}$ for some parameter $t > 0$. Let $P \subset \mathbb{R}^d$ be a set of size $|P| = n$ and let $s(P,q) \triangleq \min_{p \in P} s(p,q)$. Let $C > 1$ be a parameter and consider two cases.

- $s(P,q) \leq 1$. In this case we have $\mathrm{KDF}_P(q) = \frac{1}{|P|} \sum_{p \in P} k(p,q) \geq \frac{1}{2n}$.
- $s(P,q) \geq C$. In this case we have $\mathrm{KDF}_P(q) \leq \frac{1}{C^t}$.

Consider the setting $t \triangleq 2 \log_C n$. If we have $s(P,q) \leq 1$, then we get $\mathrm{KDF}_P(q) \geq \frac{1}{2n}$ and, if we have $s(P,q) \geq C$, then we get $\mathrm{KDF}_P(q) \leq \frac{1}{n^2}$. Thus, if we can approximate the value of $\mathrm{KDF}_P(q)$ within any constant factor, then we can distinguish $s(P,q) \leq 1$ from $s(P,q) \geq C$. In particular, we consider the case $C \triangleq 3$. It is known that, assuming SETH, deciding whether $s(P,q) \leq 1$ or $s(P,q) \geq 3$ for any $d \geq \omega(\log n)$ requires $\Omega(n^{1-o(1)})$ time even if we allow an arbitrary polynomial time $n^{O(1)}$ preprocessing of the point-set P. This follows from [Ind01, Wil05].

This implies that approximating $\mathrm{KDF}_P(q)$ for $s(p,q) = \|p-q\|_\infty$ and any $d \geq \omega(\log n)$ within any constant factor and with any polynomial time $n^{O(1)}$ preprocessing cannot be done in $d^{O(1)} 2^{t/10} n^{o(1)}$ query time. Indeed, otherwise for $t = \log_3 n$ we would get a better than $n^{1-o(1)}$ query time, which would contradict SETH.

Hardness for the Gaussian kernel. Similarly as in Chapter 7, we will perform a reduction from the bichromatic Hamming close pair (BHCP) problem. We will use the unbalanced version of the BHCP problem. Let $\alpha > 0$ be a real-valued constant and consider any $d \geq \omega(\log n)$. Given two sets $P, Q \subseteq \{0,1\}^d$ of sizes $|P| = n, |Q| = m$ for $m = n^\alpha$ and an integer $t \in \{0, \ldots, d\}$, our goal is to determine if there is a pair of vectors $p \in P$ and $q \in Q$ such that $\mathrm{Hamming}(p,q) < t$. Assuming SETH, we know that this requires $\Omega(nm)^{1-o(1)}$ time. See Chapter 2.

Consider the value of $k(P,Q)$ as defined in Eq. (8.1) for the Gaussian kernel $k(p,q) \triangleq \exp(-C\|p-q\|^2)$ with the parameter $C \triangleq 10\alpha \log n$. We observe that $\exp(-C\|p-q\|^2) = \exp(-C \cdot \mathrm{Hamming}(p,q))$ for any binary vectors p and q. We distinguish two cases.

- There exist $p' \in P$ and $q' \in Q$ such that $\mathrm{Hamming}(p',q') < t$. Then $\mathrm{KDF}_P(q) \geq \frac{1}{n} \exp(-C \cdot \mathrm{Hamming}(p',q')) \geq \frac{1}{n} \exp(-C(t-1))$.
- For all $p \in P$ we have $\mathrm{Hamming}(p,q) \geq t$. Then $\mathrm{KDF}_P(q) \leq m \exp(-Ct)$, which is much smaller than $\frac{1}{n} \exp(-C(t-1))$ by our choice of $C = 10\alpha \log n$.

Thus, since we can choose arbitrary $d \geq \omega(\log n)$, we cannot approximate $k(P, Q)$ within any additive factor $\mu \leq \exp(-\omega(\log^2 n))$ in less than $(nm)^{1-o(1)}$ time assuming SETH. Furthermore, the same reduction gives $\Omega(nm)^{1-o(1)}$ lower bound for approximating $k(P, Q)$ within any *multiplicative* constant factor.

The $\Omega(nm)^{1-o(1)}$ lower bound for additive approximation can be further strengthened to any $\mu \leq n^{-\omega(1)}$ by using [Rub18]. This conditionally rules out algorithms of the form $(nm)^{1-\Omega(1)}/\varepsilon^{o(1)}$.

Bibliography

[AB15] Alekh Agarwal and Léon Bottou. A lower bound for the optimization of finite sums. In *International Conference on Machine Learning (ICML)*, 2015.

[AB17] Amir Abboud and Arturs Backurs. Towards Hardness of Approximation for Polynomial Time Problems. *ITCS*, 2017.

[AB18] Amir Abboud and Karl Bringmann. Tighter Connections Between Formula-SAT and Shaving Logs. In *ICALP*, 2018.

[ABVW15] Amir Abboud, Arturs Backurs, and Virginia Vassilevska Williams. Tight hardness results for LCS and other sequence similarity measures. In *Foundations of Computer Science (FOCS), 2015 IEEE 56th Annual Symposium on*, pages 59–78. IEEE, 2015.

[AC75] Alfred V Aho and Margaret J Corasick. Efficient string matching: an aid to bibliographic search. *Communications of the ACM*, 18(6):333–340, 1975.

[AC01] J. Aach and G. Church. Aligning gene expression time series with time warping algorithms. *Bioinformatics*, 17(6):495–508, 2001.

[ACW16] Josh Alman, Timothy M Chan, and Ryan Williams. Polynomial Representations of Threshold Functions and Algorithmic Applications. 2016.

[AHVWW16] Amir Abboud, Thomas Dueholm Hansen, Virginia Vassilevska Williams, and Ryan Williams. Simulating branching programs with edit distance and friends: or: a polylog shaved is a lower bound made. In *Proceedings of the forty-eighth annual ACM symposium on Theory of Computing*, pages 375–388. ACM, 2016.

[AI06] Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *Symposium on Foundations of Computer Science (FOCS)*, 2006.

[AIL+15] Alexandr Andoni, Piotr Indyk, Thijs Laarhoven, Ilya Razenshteyn, and Ludwig Schmidt. Practical and Optimal LSH for Angular Distance. In *Advances in Neural Information Processing Systems (NIPS)*. 2015.

[AKL+16] Amihood Amir, Tsvi Kopelowitz, Avivit Levy, Seth Pettie, Ely Porat, and B Riva Shalom. Mind the gap: Essentially optimal algorithms for online dictionary matching with one gap. In *27th International Symposium on Algorithms and Computation*, page 1, 2016.

[AKO10] Alexandr Andoni, Robert Krauthgamer, and Krzysztof Onak. Polylogarithmic approximation for edit distance and the asymmetric query complexity. In *FOCS*, pages 377–386, 2010.

[AR15] Alexandr Andoni and Ilya Razenshteyn. Optimal data-dependent hashing for approximate near neighbors. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pages 793–801. ACM, 2015.

[AR18] Amir Abboud and Aviad Rubinstein. Fast and Deterministic Constant Factor Approximation Algorithms for LCS Imply New Circuit Lower Bounds. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 94. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

[ARW17] Amir Abboud, Aviad Rubinstein, and Ryan Williams. Distributed PCP Theorems for Hardness of Approximation in P. In *2017 IEEE 58th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 25–36. IEEE, 2017.

[AS16] Yossi Arjevani and Ohad Shamir. Dimension-free iteration complexity of finite sum optimization problems. In *Advances in Neural Information Processing Systems (NIPS)*. 2016.

[AS17] Yossi Arjevani and Ohad Shamir. Oracle complexity of second-order methods for finite-sum problems. In *International Conference on Machine Learning*, pages 205–213, 2017.

[AVW14] Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *Foundations of Computer Science (FOCS), 2014 IEEE 55th Annual Symposium on*, pages 434–443. IEEE, 2014.

[AVWW14] Amir Abboud, Virginia Vassilevska Williams, and Oren Weimann. Consequences of faster alignment of sequences. In *International Colloquium on Automata, Languages, and Programming*, pages 39–51. Springer, 2014.

[AW15] Josh Alman and Ryan Williams. Probabilistic polynomials and hamming nearest neighbors. In *Foundations of Computer Science (FOCS), 2015 IEEE 56th Annual Symposium on*, pages 136–150. IEEE, 2015.

[AWY15] Amir Abboud, Ryan Williams, and Huacheng Yu. More Applications of the Polynomial Method to Algorithm Design. In *Proceedings of the*

Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms, pages 218–230. Society for Industrial and Applied Mathematics, 2015.

[Bac13] Francis Bach. Sharp analysis of low-rank kernel matrix approximations. In *Conference on Learning Theory (COLT)*, 2013.

[Bal97] Keith Ball. An elementary introduction to modern convex geometry. *Flavors of geometry*, 31:1–58, 1997.

[BB07] Léonand Bottou and Olivier Bousquet. The tradeoffs of large scale learning. In *Advances in Neural Information Processing Systems (NIPS)*, 2007.

[BCIS18] Arturs Backurs, Moses Charikar, Piotr Indyk, and Paris Siminelakis. Efficient Density Evaluation for Smooth Kernels. *FOCS*, 2018.

[BFC08] Philip Bille and Martin Farach-Colton. Fast and compact regular expression matching. *Theoretical Computer Science*, 409(3):486 – 496, 2008.

[BGL17] Karl Bringmann, Allan Grønlund, and Kasper Green Larsen. A dichotomy for regular expression membership testing. In *Foundations of Computer Science (FOCS), 2017 IEEE 58th Annual Symposium on*, pages 307–318. IEEE, 2017.

[BGSW16] Karl Bringmann, Fabrizio Grandoni, Barna Saha, and Virginia Vassilevska Williams. Truly Sub-cubic Algorithms for Language Edit Distance and RNA Folding via Fast Bounded-Difference Min-Plus Product. In *Foundations of Computer Science (FOCS), 2016 IEEE 57th Annual Symposium on*, pages 375–384. IEEE, 2016.

[BHR00] Lasse Bergroth, Harri Hakonen, and Timo Raita. A Survey of Longest Common Subsequence Algorithms. In *String Processing and Information Retrieval, 2000. SPIRE 2000. Proceedings. Seventh International Symposium on*, pages 39–48. IEEE, 2000.

[BI15] Arturs Backurs and Piotr Indyk. Edit Distance Cannot Be Computed in Strongly Subquadratic Time (unless SETH is false). In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pages 51–58. ACM, 2015.

[BI16] Arturs Backurs and Piotr Indyk. Which Regular Expression Patterns are Hard to Match? In *Foundations of Computer Science (FOCS), 2016 IEEE 57th Annual Symposium on*, pages 457–466. IEEE, 2016.

[BIS17] Arturs Backurs, Piotr Indyk, and Ludwig Schmidt. On the Fine-Grained Complexity of Empirical Risk Minimization: Kernel Methods and Neural Networks. *NIPS*, 2017.

[BK15] 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.

[BK18] Karl Bringmann and Marvin Künnemann. Multivariate Fine-Grained Complexity of Longest Common Subsequence. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1216–1235. Society for Industrial and Applied Mathematics, 2018.

[Bri14] Karl Bringmann. Why walking the dog takes time: Frechet distance has no strongly subquadratic algorithms unless SETH fails. In *Foundations of Computer Science (FOCS), 2014 IEEE 55th Annual Symposium on*, pages 661–670. IEEE, 2014.

[BS16] Francis Bach and Suvrit Sra. Stochastic optimization: Beyond stochastic gradients and convexity. *NIPS Tutorial*, 2016. `http://suvrit.de/talks/vr_nips16_bach.pdf`.

[BSV14] Eli Ben-Sasson and Emanuele Viola. Short PCPs with Projection Queries. In *International Colloquium on Automata, Languages, and Programming*, pages 163–173. Springer, 2014.

[BT09] Philip Bille and Mikkel Thorup. Faster regular expression matching. In *Automata, Languages and Programming*, pages 171–182. Springer, 2009.

[Cal08] Chris Calabro. A lower bound on the size of series-parallel graphs dense in long paths. In *Electronic Colloquium on Computational Complexity (ECCC)*, volume 15, 2008.

[CBMS15] Nicolo Cesa-Bianchi, Yishay Mansour, and Ohad Shamir. On the complexity of learning with kernels. In *Conference On Learning Theory (COLT)*, 2015.

[CDG+18] Diptarka Chakraborty, Das Debarati, Elazar Goldenberg, Michal Koucký, and Michael Saks. Approximating Edit Distance Within Constant Factor in Truly Sub-Quadratic Time. In *FOCS*, 2018.

[CDL+12] Marek Cygan, Holger Dell, Daniel Lokshtanov, Dániel Marx, Jesper Nederlof, Yoshio Okamoto, Ramamohan Paturi, Saket Saurabh, and Magnus Wahlström. On Problems as Hard as CNF-SAT. In *Computational Complexity (CCC), 2012 IEEE 27th Annual Conference on*, pages 74–84. IEEE, 2012.

[CGI+16] Marco L Carmosino, Jiawei Gao, Russell Impagliazzo, Ivan Mihajlin, Ramamohan Paturi, and Stefan Schneider. Nondeterministic Extensions of the Strong Exponential Time Hypothesis and Consequences for Non-reducibility. In *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science*, pages 261–270. ACM, 2016.

[CGL+18] Lijie Chen, Shafi Goldwasser, Kaifeng Lyu, Guy N Rothblum, and Aviad Rubinstein. Fine-grained Complexity Meets IP = PSPACE. *arXiv preprint arXiv:1805.02351*, 2018.

[CH97] Richard Cole and Ramesh Hariharan. Tree pattern matching and subset matching in randomized $O(n \log^3 m)$ time. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 66–75. ACM, 1997.

[CH02] Richard Cole and Ramesh Hariharan. Verifying candidate matches in sparse and wildcard matching. In *Proceedings of the Thirty-fourth annual ACM symposium on Theory of computing*, pages 592–601. ACM, 2002.

[Cha02] Moses S Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, pages 380–388. ACM, 2002.

[CIP02] Moses Charikar, Piotr Indyk, and Rina Panigrahy. New algorithms for subset query, partial match, orthogonal range searching, and related problems. In *ICALP*, pages 451–462, 2002.

[CLR+14] Shiri Chechik, Daniel H Larkin, Liam Roditty, Grant Schoenebeck, Robert E Tarjan, and Virginia Vassilevska Williams. Better approximation algorithms for the graph diameter. In *Proceedings of the Twenty-Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1041–1052. SIAM, 2014.

[CPB+98] ENRICO GIANLUCA Caiani, A Porta, Giuseppe Baselli, M Turiel, S Muzzupappa, F Pieruzzi, C Crema, A Malliani, and Sergio Cerutti. Warped-average template technique to track on a cycle-by-cycle basis the cardiac filling phases on left ventricular volume. In *Computers in Cardiology 1998*, pages 73–76. IEEE, 1998.

[CS17] Moses Charikar and Paris Siminelakis. Hashing-based-estimators for kernel density in high dimensions. In *Foundations of Computer Science (FOCS), 2017 IEEE 58th Annual Symposium on*, pages 1032–1043. IEEE, 2017.

[CW16] Timothy M Chan and Ryan Williams. Deterministic APSP, Orthogonal Vectors, and More: Quickly Derandomizing Razborov-Smolensky.

In *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1246–1255. Society for Industrial and Applied Mathematics, 2016.

[DG03] Sanjoy Dasgupta and Anupam Gupta. An elementary proof of a theorem of Johnson and Lindenstrauss. *Random Structures & Algorithms*, 22(1):60–65, 2003.

[DH09] Evgeny Dantsin and Edward A. Hirsch. Worst-case upper bounds. In *Handbook of Satisfiability*, pages 403–424. 2009.

[DW06] Evgeny Dantsin and Alexander Wolpert. MAX-SAT for Formulas with Constant Clause Density Can Be Solved Faster Than in $\mathcal{O}(2^n)$ Time. In *International Conference on Theory and Applications of Satisfiability Testing*, pages 266–276. Springer, 2006.

[Edd04] Sean R Eddy. How do RNA folding algorithms work? *Nature biotechnology*, 22(11):1457, 2004.

[FP74] Michael J Fischer and Michael S Paterson. String-Matching and Other Products. Technical report, DTIC Document, 1974.

[Fri08] Martin C. Frith. Large-scale sequence comparison: Spaced seeds and suffix arrays. 2008. `http://last.cbrc.jp/mcf-kyoto08.pdf`.

[Gal85] Zvi Galil. Open problems in stringology. In *Combinatorial Algorithms on Words*, pages 1–8. Springer, 1985.

[Gen01] Marc G Genton. Classes of kernels for machine learning: a statistics perspective. *Journal of machine learning research*, 2(Dec):299–312, 2001.

[GFKS02] Thomas Gärtner, Peter A Flach, Adam Kowalczyk, and Alexander J Smola. Multi-instance kernels. In *ICML*, volume 2, pages 179–186, 2002.

[GM16] Ofer Grossman and Dana Moshkovitz. Amplification and derandomization without slowdown. In *Foundations of Computer Science (FOCS), 2016 IEEE 57th Annual Symposium on*, pages 770–779. IEEE, 2016.

[Gra16] Szymon Grabowski. New tabulation and sparse dynamic programming based techniques for sequence similarity problems. *Discrete Applied Mathematics*, 212:96–103, 2016.

[GRS99] Minos N Garofalakis, Rajeev Rastogi, and Kyuseok Shim. SPIRIT: Sequential pattern mining with regular expression constraints. In *VLDB*, volume 99, pages 7–10, 1999.

[GS91] Leslie Greengard and John Strain. The Fast Gauss Transform. *SIAM Journal on Scientific and Statistical Computing*, 12(1):79–94, 1991.

[GS17] Omer Gold and Micha Sharir. Dynamic Time Warping and Geometric Edit Distance: Breaking the Quadratic Barrier. ICALP, 2017.

[GSM03] Bogdan Georgescu, Ilan Shimshoni, and Peter Meer. Mean shift based clustering in high dimensions: A texture classification example. In *ICCV*, volume 3, page 456, 2003.

[Gus97] Dan Gusfield. *Algorithms on strings, trees and sequences: computer science and computational biology.* Cambridge university press, 1997.

[Hir75] D. S. Hirschberg. A linear space algorithm for computing maximal common subsequences. *Commun. ACM*, 18(6):341–343, June 1975.

[HPIM12] Sariel Har-Peled, Piotr Indyk, and Rajeev Motwani. Approximate Nearest Neighbor: Towards Removing the Curse of Dimensionality. *Theory of computing*, 8(1):321–350, 2012.

[HS77] James W. Hunt and Thomas G. Szymanski. A fast algorithm for computing longest subsequences. *Commun. ACM*, 20(5):350–353, 1977.

[IHG08] Charles L Isbell, Michael P Holmes, and Alexander G Gray. Ultra-fast Monte Carlo for Statistical Summations. In *Advances in Neural Information Processing Systems*, pages 673–680, 2008.

[IKW02] Russell Impagliazzo, Valentine Kabanets, and Avi Wigderson. In search of an easy witness: Exponential time vs. probabilistic polynomial time. *Journal of Computer and System Sciences*, 65(4):672–694, 2002.

[ILPS14] Russell Impagliazzo, Shachar Lovett, Ramamohan Paturi, and Stefan Schneider. 0-1 Integer Linear Programming with a Linear Number of Constraints. *CoRR*, abs/1401.5512, 2014.

[Ind98] Piotr Indyk. Faster algorithms for string matching problems: Matching the convolution bound. In *Foundations of Computer Science, 1998. Proceedings. 39th Annual Symposium on*, pages 166–173. IEEE, 1998.

[Ind01] Piotr Indyk. On approximate nearest neighbors under ℓ_∞ norm. *Journal of Computer and System Sciences*, 63(4):627–638, 2001.

[IP01] Russell Impagliazzo and Ramamohan Paturi. On the Complexity of k-SAT. *Journal of Computer and System Sciences*, 62(2):367–375, 2001.

[IPZ01] Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which Problems Have Strongly Exponential Complexity? *Journal of Computer and System Sciences*, 63:512–530, 2001.

[IT03] P. Indyk and N. Thaper. Fast color image retrieval via embeddings. Workshop on Statistical and Computational Theories of Vision (at ICCV), 2003.

[JP04] N.D. Jones and P. Pevzner. *An introduction to bioinformatics algorithms*. Cambridge, Mass: MIT Press., 2004.

[Kal02] Adam Kalai. Efficient pattern-matching with don't cares. In *Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 655–656. Society for Industrial and Applied Mathematics, 2002.

[KDY+06] Sailesh Kumar, Sarang Dharmapurikar, Fang Yu, Patrick Crowley, and Jonathan Turner. Algorithms to accelerate multiple regular expressions matching for deep packet inspection. *ACM SIGCOMM Computer Communication Review*, 36(4):339–350, 2006.

[KHDA12] Kenrick Kin, Björn Hartmann, Tony DeRose, and Maneesh Agrawala. Proton: Multitouch Gestures as Regular Expressions. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 2885–2894. ACM, 2012.

[KMP77] Donald E Knuth, James H Morris, Jr, and Vaughan R Pratt. Fast pattern matching in strings. *SIAM journal on computing*, 6(2):323–350, 1977.

[KR05] E. Keogh and C. A. Ratanamahatana. Exact indexing of dynamic time warping. *Knowl. Inf. Syst.*, 7(3):358–386, 2005.

[Law76] Eugene L Lawler. *Combinatorial optimization: networks and matroids*. Courier Corporation, 1976.

[Lee] LeetCode. Problem 139. Word Break. `https://leetcode.com/problems/word-break/`.

[LN14] Huy Lê Nguyên. *Algorithms for high dimensional data*. PhD thesis, Princeton University, 2014.

[LSSS14] Roi Livni, Shai Shalev-Shwartz, and Ohad Shamir. On the computational efficiency of training neural networks. In *Advances in Neural Information Processing Systems*, pages 855–863, 2014.

[MFS+17] Krikamol Muandet, Kenji Fukumizu, Bharath Sriperumbudur, Bernhard Schölkopf, et al. Kernel mean embedding of distributions: A review and beyond. *Foundations and Trends® in Machine Learning*, 10(1-2):1–141, 2017.

[Mie09] Thilo Mie. Short PCPPs verifiable in polylogarithmic time with $O(1)$ queries. *Annals of Mathematics and Artificial Intelligence*, 56(3-4):313–338, 2009.

[MM16] Cameron Musco and Christopher Musco. Recursive sampling for the Nyström method. *Advances in Neural Information Processing Systems (NIPS)*, 2016.

[MMR+01] Klaus-Robert Müller, Sebastian Mika, Gunnar Rätsch, Koji Tsuda, and Bernhard Schölkopf. An introduction to kernel-based learning algorithms. *IEEE transactions on neural networks*, 12(2):181–201, 2001.

[MN04] Manor Mendel and Assaf Naor. Euclidean quotients of finite metric spaces. *Advances in Mathematics*, 189(2):451–494, 2004.

[MP80] William J Masek and Michael S Paterson. A faster algorithm computing string edit distances. *Journal of Computer and System sciences*, 20(1):18–31, 1980.

[Mül07] M. Müller. *Information Retrieval for Music and Motion*. Springer Berlin Heidelberg, 2007.

[Mur12] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012.

[Mye92] Gene Myers. A Four Russians Algorithm for Regular Expression Pattern Matching. *Journal of the ACM (JACM)*, 39(2):432–448, 1992.

[Nav01] Gonzalo Navarro. A guided tour to approximate string matching. *ACM computing surveys (CSUR)*, 33(1):31–88, 2001.

[Nes04] Yurii Nesterov. *Introductory Lectures on Convex Optimization: A Basic Course*. Kluwer Academic Publishers, 2004.

[NR03] Gonzalo Navarro and Mathieu Raffinot. Fast and simple character classes and bounded gaps pattern matching, with applications to protein searching. *Journal of Computational Biology*, 10(6):903–923, 2003.

[NS07] Assaf Naor and Gideon Schechtman. Planar Earthmover is not in L_1. *SIAM Journal on Computing*, 37(3):804–826, 2007.

[NY83] Arkadii S. Nemirovski and David B. Yudin. *Problem Complexity and Method Efficiency in Optimization*. Wiley Interscience, 1983.

[PPSZ05] Ramamohan Paturi, Pavel Pudlák, Michael E Saks, and Francis Zane. An Improved Exponential-Time Algorithm for k-SAT. *Journal of the ACM (JACM)*, 52(3):337–364, 2005.

[PT18a] Jeff M Phillips and Wai Ming Tai. Improved coresets for kernel density estimates. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2718–2727. SIAM, 2018.

[PT18b] Jeff M. Phillips and Wai Ming Tai. Near-Optimal Coresets of Kernel Density Estimates. In Bettina Speckmann and Csaba D. Tóth, editors, *34th International Symposium on Computational Geometry (SoCG 2018)*, volume 99 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 66:1–66:13, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[Ras04] Carl Edward Rasmussen. Gaussian processes in machine learning. In *Advanced lectures on machine learning*, pages 63–71. Springer, 2004.

[RASC14] Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. CNN Features off-the-shelf: an Astounding Baseline for Recognition. In *Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2014.

[RJ93] L. Rabiner and B. Juang. *Fundamentals of Speech Recognition*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1993.

[RK05] Chotirat (Ann) Ratanamahatana and Eamonn J. Keogh. Three Myths about Dynamic Time Warping Data Mining. In *Proceedings of the 2005 SIAM International Conference on Data Mining, SDM 2005*, pages 506–510, 2005.

[RM03] Toni M. Rath and R. Manmatha. Word image matching using dynamic time warping. In *2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition CVPR 2003*, pages 521–527, 2003.

[RR08] Ali Rahimi and Benjamin Recht. Random features for large-scale kernel machines. In *Advances in Neural Information Processing Systems (NIPS)*. 2008.

[Rub18] Aviad Rubinstein. Hardness of approximate nearest neighbor search. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing*, pages 1260–1268. ACM, 2018.

[RVW13] Liam Roditty and Virginia Vassilevska Williams. Fast approximation algorithms for the diameter and radius of sparse graphs. In *Symposium on Theory of Computing (STOC)*, 2013.

[RW10] Alessandro Rinaldo and Larry Wasserman. Generalized density clustering. *The Annals of Statistics*, pages 2678–2722, 2010.

[SGPS15] Zoltán Szabó, Arthur Gretton, Barnabás Póczos, and Bharath Sriperumbudur. Two-stage sampled learning theory on distributions. In *Artificial Intelligence and Statistics*, pages 948–957, 2015.

[SS01] Bernhard Scholkopf and Alexander J Smola. *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2001.

[SSBD14] Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, 2014.

[SSPG16] Zoltán Szabó, Bharath K Sriperumbudur, Barnabás Póczos, and Arthur Gretton. Learning theory for distribution regression. *The Journal of Machine Learning Research*, 17(1):5272–5311, 2016.

[SSSS07] Shai Shalev-Shwartz, Yoram Singer, and Nathan Srebro. Pegasos: Primal estimated sub-gradient solver for SVM. In *International Conference on Machine Learning (ICML)*, 2007.

[SSTT17] Takayuki Sakai, Kazuhisa Seto, Suguru Tamaki, and Junichi Teruyama. Improved exact algorithms for mildly sparse instances of Max SAT. *Theoretical Computer Science*, 697:58–68, 2017.

[SW13] Rajesh Santhanam and Ross Williams. On medium-uniformity and circuit lower bounds. In *Computational Complexity (CCC), 2013 IEEE Conference on*, pages 15–23. IEEE, 2013.

[SZK14] Erich Schubert, Arthur Zimek, and Hans-Peter Kriegel. Generalized outlier detection with flexible kernel density estimates. In *Proceedings of the 2014 SIAM International Conference on Data Mining*, pages 542–550. SIAM, 2014.

[Tun11] Daniel Tunkelang. Retiring a great interview problem. 2011. `http://thenoisychannel.com/2011/08/08/retiring-a-great-interview-problem`.

[Val77] Leslie G Valiant. *Graph-theoretic arguments in low-level complexity*. Springer, 1977.

[Val12] Gregory Valiant. Finding correlations in subquadratic time, with applications to learning parities and juntas. In *Symposium on Foundations of Computer Science (FOCS)*, 2012.

[Vap98] Vlamimir Vapnik. *Statistical learning theory*. Wiley, 1998.

[Vio09] Emanuele Viola. *On the power of small-depth computation*. Now Publishers Inc, 2009.

[WDT+10] Xiaoyue Wang, Hui Ding, Goce Trajcevski, Peter Scheuermann, and Eamonn J. Keogh. Experimental comparison of representation methods and distance measures for time series data. *CoRR*, abs/1012.2789, 2010.

[Wil05] Ryan Williams. A new algorithm for optimal 2-constraint satisfaction and its implications. *Theoretical Computer Science*, 348(2):357–365, 2005.

[Wil13] Ryan Williams. Improving Exhaustive Search Implies Superpolynomial Lower Bounds. *SIAM Journal on Computing*, 42(3):1218–1244, 2013.

[Wil14] Ryan Williams. Algorithms for Circuits and Circuits for Algorithms: Connecting the Tractable and Intractable. In *Proceedings of the International Congress of Mathematicians*, 2014.

[WS01] Christopher K. Williams and Matthias Seeger. Using the Nyström Method to Speed Up Kernel Machines. In *Advances in Neural Information Processing Systems (NIPS)*. 2001.

[WS16] Blake E. Woodworth and Nathan Srebro. Tight complexity bounds for optimizing composite objectives. In *Advances in Neural Information Processing Systems (NIPS)*, 2016.

[WvdMBW16] Nicolaas Weideman, Brink van der Merwe, Martin Berglund, and Bruce Watson. Analyzing Matching Time Behavior of Backtracking Regular Expression Matchers by Using Ambiguity of NFA. In *International Conference on Implementation and Application of Automata*, pages 322–334. Springer, 2016.

[YPW+17] Yun Yang, Mert Pilanci, Martin J Wainwright, et al. Randomized sketches for kernels: Fast and optimal nonparametric regression. *The Annals of Statistics*, 45(3):991–1023, 2017.

[ZS03] Y. Zhu and D. Shasha. Warping indexes with envelope transforms for query by humming. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, SIGMOD '03, pages 181–192, 2003.