Completeness of Hamming Distance for Pattern Matching and All-Pairs Problems

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Completeness of Hamming Distance for Pattern Matching and All-Pairs Problems

Master Thesis

Karim Labib

February 15, 2018

Advisors: Prof. Dr. P. Widmayer, Dr. P. Uznański, D. Graf
Department of Computer Science, ETH Zürich

Abstract

In pattern matching problems, we are given a text of length n and a pattern of length m where $m \leq n$. For each alignment of the pattern with the text, a score is calculated where different score functions can be used. What differentiates pattern matching problems is the score function used. Several of such problems share the same upper bound. For example, Hamming distance, Less-than, Threshold and ℓ_{2p+1} pattern matching problems all have their best algorithms with time complexity of $\mathcal{O}(n\sqrt{m \log m})$.

In all-pairs problems, we are given n vectors in $\mathbb{Z}^d$ and the goal is to compute some score function between each pair of vectors. Similar to pattern matching problems, all-pairs problems with the aforementioned score functions all share the same upper bound.

In this thesis, we survey different techniques used for such pattern matching and all-pairs problems and prove a new result connecting all those different score functions together by establishing a Hamming distance completeness result which states that all the previously mentioned score functions together with a wider range of score functions are actually equivalent to the Hamming distance score function up to a poly-logarithmic factor.

We also link the complexity of all-pairs Hamming distance to that of sparse matrix multiplication. By this result and the Hamming distance completeness result, we show that there is some inherent complexity barrier to a wide class of all-pairs problems.

Acknowledgements

It is customary to start the acknowledgments with thanking your supervisors. But in my case, it won't be just a customary thing because without the support of Przemek and Daniel, this thesis would have never come to a successful completion. I would like to express my gratitude to Daniel for first proposing that I work alongside him and Przemek and later giving me the chance (again after me doing more navigation) to actually write this thesis under their supervision. During the thesis period, I came to realize that without Przemek's wide knowledge and his ability to ask the right questions, the new ideas in this thesis would not have made it here. I am really grateful to being given the opportunity to work with Przemek and Daniel and learn a lot from their combined experience.

Rania, you are one of the main reasons I didn't give up during those months of the thesis. Your unparalleled love, encouragement and support are what kept me going at times when things seemed boring or meaningless. This phase in my life has come to a successful end thanks to you. This thesis is dedicated to you.

I would also like to thank all my friends in Zürich specially Hector Dejea, João Ribeiro, Erik Bakkeren, Sarah Daniel, Mariyana Koleva and Ahmed Fakhry who have made my stay in Zürich feel a bit like home. I am grateful for all the good times I spent with each and every one of you.

Last but not least, I would like to express my gratitude to my dad, mom and sister back home who supported me emotionally, handled all my mood swings wisely and tolerated all my annoying behavior whenever I was back home thinking of my studies. I am where I am today because of all what you've done. I am lucky to have been raised among such an amazing family.

Contents

Nomenclature

α	$\alpha = \sup\{0 \leq r \leq 1 \mid \omega(1,r,1) = 2 + o(1)\}$
ℓ_p	The ℓ_p norm of a vector $\mathbf{A}$ is equal to $\sqrt[p]{\sum_i \lvert\mathbf{A}[i]\rvert^p}$
$\mathbb{1}[\varphi]$	A conditional function that is equal to 1 iff φ is true and 0 otherwise
$\mathbf{A} \overset{g}{\cdot} \mathbf{B}$	$(+,g)$-vector product of two equally sized vectors. It is a generalization of the dot product. A general binary function g replaces the usual multiplication
$\mathbf{A}, \mathbf{B}, \ldots$	Vectors of integers
$\mathbf{P} \diamond \mathbf{T}$	Used in the context of pattern matching where $\mathbf{P}$ is the pattern and $\mathbf{T}$ is the text. The output of this operation is a vector $\mathbf{O}$ where $\mathbf{O}[i] = \sum_k \mathbf{P}[k] \cdot \mathbf{T}[i+k]$
$\mathcal{A}, \mathcal{B}, \ldots$	Matrices of integers
$\mathcal{A}_{*i}$	The i-th column of matrix $\mathcal{A}$
$\mathcal{B}_{i*}$	The i-th row of matrix $\mathcal{B}$
ω	Square matrix multiplication exponent, i.e. multiplying matrices of size $n \times n$ takes $\mathcal{O}(n^{\omega})$ time
$\omega(r,s,t)$	Rectangular matrix multiplication exponent, i.e. multiplying matrices of size $n^r \times n^s$ and $n^s \times n^t$ takes $\mathcal{O}(n^{\omega(r,s,t)})$ time
ρ	ρ is a solution to $\rho = \omega(1, 4-\rho, 1)$ where $\rho \leq 2.6834$

$\star$	Ignore (don't care) symbols. For any unary or binary function, $f(\star) = g(\star, y) = g(x, \star) = 0$
$\widetilde{\mathcal{O}}(\cdot)$	same as $\mathcal{O}$ notation but suppresses polylogarithmic factors
$\lvert \cdot \rvert$	the number of nonzero entries in a matrix/vector/set

Chapter 1

Introduction

1.1 Overview

Exact *pattern matching* is one of the oldest and well studied problems in the field of computer science. Given a pattern and a text, the solution to the problem is to answer whether the pattern exists as a substring of the text or not (recall the famous "KMP" algorithm [19]). Later, research in text processing took different lines and started studying variants of the problem. In particular, a line of research involved generalizing the exact pattern matching problem by defining a score function between pairs of characters of the underlying alphabet of the text and pattern and asking for the score of every alignment of the pattern with the text. Thus, instead of asking of the existence of an exact occurrence of the pattern in the text, the new variant asks for how close every substring of the text (of the same length as the pattern) is to the pattern.

A seemingly different domain is the *all-pairs* type of problems. Those problems are of interest in computational geometry where you are given a set of points in a high dimensional space and we are asked to measure a score function for each pair of points. A special case of such problems is finding the closest pair (for the given score function) in such set of points.

In this thesis, we survey different results in both domains. We focus on problems in each domain that share a similar structure and have the same time complexity upper bound. We describe the algorithms given in the literature for solving them and emphasize how similar the techniques used are. We then obtain new results demonstrating that this identical time complexity and similarity in the techniques is actually not a coincidence but that there are strong underlying theoretical reasons why these problems are computationally equivalent.

1.2 Our contribution

Here, we give a brief summary of the content of the thesis and in particular the new results.

- In Chapter 2, we present some preliminaries and notations to be used throughout the thesis.
- Chapters 3 and 4 present a survey of different problems in pattern matching and all-pairs, respectively.
- Chapter 5 includes the new results obtained in connecting the different problems together. Specifically, we prove Theorem 5.6 which establishes a Hamming distance completeness of a wide class of vector products. Proving this theorem involves showing reductions between a broad class of algebraic binary functions. The two directions of the equivalence required for proving the theorem are provided in Theorems 5.14 and 5.15.
- In Chapter 6, we make use of our Hamming distance completeness result to actually show that a wide range of all-pairs problems are linked to a very natural problem which is the multiplication of sparse matrices. We survey a simple yet efficient fast sparse matrix multiplication algorithm and use ideas from there to help us prove the equivalence. The new two main Theorems 6.8 and 6.11 proven in that section involves reductions in both directions.
- In Chapter 7, we present consequences of our reductions and we use our framework in relevant pattern matching problems.
- Finally, in Chapter 8, we give a summary of the obtained results and mention some related directions and open problems worth studying.

Chapter 2

Preliminaries

Throughout this thesis, we denote vectors of integers with block letters $\mathbf{A}, \mathbf{B}, \dots$ and we denote matrices of integers with calligraphic letters $\mathcal{A}, \mathcal{B}, \dots$. When talking about pattern matching, we will denote the text and pattern with $\mathbf{T}$ and $\mathbf{P}$ respectively. $\mathbf{T}$ is always assumed to be of size n and $\mathbf{P}$ of size m such that $m \leq n$.

2.1 Pattern matching problems

Different variants of pattern matching problems have been studied. The setting of a pattern matching problem is common in all of these problems. $\mathbf{P}$ is always slided along $\mathbf{T}$. The goal is to compute for each alignment of the pattern with the text a value that measures some relation between the corresponding pairs of entries of the pattern and text and then sum up these measures for all pairs. Therefore, the output of pattern matching problems is an integer vector $\mathbf{O}$ of size $(n - m + 1)$ where each output value $\mathbf{O}[i]$ corresponds to the relation between the m-substring of $\mathbf{T}$ starting at i, $\mathbf{T}[i, \dots, i + m - 1]$, and $\mathbf{P}$. Thus, the difference between different variants of the problem is how to relate entries of the alphabet of the text and pattern together. To this end, we need the following definition.

Definition 2.1 $(+, g)$***-vector product***. *Given two vectors* $\mathbf{A}, \mathbf{B}$ *of equal size* k *and a binary function* $g : \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}$, *their* $(+, g)$*-vector product is defined as:*

$$\mathbf{A} \overset{g}{\cdot} \mathbf{B} = \sum_{i=0}^{k-1} g(\mathbf{A}[i], \mathbf{B}[i])$$

Using that definition, the output entry $\mathbf{O}[i]$ of a pattern matching problem is equal to $(\mathbf{T}[\mathbf{i}, \dots, \mathbf{i} + \mathbf{m} - \mathbf{1}] \overset{g}{\cdot} \mathbf{P})$ and through using different binary functions g, we can specify different versions of pattern matching.

2.2 All-pairs problems

The setting of such problems is as follows: we are given two sets of vectors $\mathbb{A} = \{\mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_n\}$ and $\mathbb{B} = \{\mathbf{B}_1, \mathbf{B}_2, \ldots, \mathbf{B}_n\}$. All vectors are of the same dimension d. The problem asks to compute some binary function on every pair of vectors in $\mathbb{A} \times \mathbb{B}$ and output the result in a matrix of size $n \times n$. For all-pairs problems, we think of the problem as being given a matrix $\mathcal{A}$ of size $n \times d$ whose rows are the vectors $\mathbf{A}_1, \ldots, \mathbf{A}_n$ and a matrix $\mathcal{B}$ of size $d \times n$ whose columns are the vectors $\mathbf{B}_1, \ldots, \mathbf{B}_n$. We can look at problems in this realm as computing a special matrix product.

Definition 2.2 $(+, g)$***-matrix product***. *Given two matrices $\mathcal{A}, \mathcal{B}$ of size $n \times d$ and $d \times n$ respectively. We denote the rows of $\mathcal{A}$ by $\mathbf{A}_1, \ldots, \mathbf{A}_n$ and columns of $\mathcal{B}$ by $\mathbf{B}_1, \ldots, \mathbf{B}_n$. If we are given a binary function $g : \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}$, then the $(+, g)$-matrix product of $\mathcal{A}$ and $\mathcal{B}$ is defined as:*

$$\mathcal{A} \overset{g}{\otimes} \mathcal{B} = \mathcal{C} \ \textit{ such that } \ \mathcal{C}[i,j] = \mathbf{A}_i \overset{g}{\cdot} \mathbf{B}_j$$

2.3 Interesting $(+, g)$-vector products

We present here different variants of $(+, g)$-vector products. Obviously, for each $(+, g)$-vector product, we obtain a different variant of pattern matching or all-pairs problem.

Hamming Distance counts the number of mismatches between corresponding pairs of elements in equally sized vectors $\mathbf{A}, \mathbf{B}$. Formally,

$$\mathsf{Ham}[\mathbf{A}, \mathbf{B}] \overset{\text{def}}{=} \mathbf{A} \overset{\neq}{\cdot} \mathbf{B} = \left|\{\, k \ : \mathbf{A}[k] \neq \mathbf{B}[k]\}\right|.$$

Similarity Measure counts the number of matches between corresponding pairs of elements in equally sized vectors $\mathbf{A}, \mathbf{B}$. Formally,

$$\mathsf{Sim}[\mathbf{A}, \mathbf{B}] \overset{\text{def}}{=} \mathbf{A} \overset{=}{\cdot} \mathbf{B} = \left|\{\, k \ : \mathbf{A}[k] = \mathbf{B}[k]\}\right|.$$

L_p **Distance** computes the p-th power of the ℓ_p norm distance between vectors $\mathbf{A}, \mathbf{B}$

$$L_p[\mathbf{A}, \mathbf{B}] \overset{\text{def}}{=} \sum_i |\mathbf{A}[i] - \mathbf{B}[i]|^p.$$

Dominance Product counts the number of positions where elements of $\mathbf{A}$ are less than or equal their corresponding elements in $\mathbf{B}$

$$\mathsf{Dom}[\mathbf{A}, \mathbf{B}] \overset{\text{def}}{=} \mathbf{A} \overset{\leq}{\cdot} \mathbf{B} = \left|\{\, k \ : \mathbf{A}[k] \leq \mathbf{B}[k]\}\right|.$$

Threshold distance counts the number of positions where elements of $\mathbf{A}$ differ from their corresponding elements in $\mathbf{B}$ by more than a given parameter δ

$$\mathsf{Thr}_\delta[\mathbf{A}, \mathbf{B}] \stackrel{\text{def}}{=} |\{k : |\mathbf{A}[k] - \mathbf{B}[k]| > \delta\}|.$$

2.4 Convolution

The convolution of two vectors $\mathbf{A}$ and $\mathbf{B}$ of size n is defined to be a vector $\mathbf{C}$ of size $2n$ such that

$$\mathbf{C} = \mathbf{A} \odot \mathbf{B}, \quad \text{where } \mathbf{C}[k] = \sum_{k=i+j} \mathbf{A}[i] \cdot \mathbf{B}[j]$$

A motivating example of where convolution is used is polynomial multiplication. If we consider the elements of $\mathbf{A}$ and $\mathbf{B}$ to be the coefficients of two univariate polynomials over the same variable, then we can verify that the vector $\mathbf{C}$ holds the coefficients of the polynomial obtained by multiplying those two polynomials. The naive solution for obtaining that result works in time $\mathcal{O}(n^2)$ since it requires multiplying every element in $\mathbf{A}$ with every element in $\mathbf{B}$. However, the breakthrough came through the development of the *Fast Fourier Transform (FFT)* which computes the convolution of two vectors of size n in $\mathcal{O}(n \log n)$. For details about how *FFT* works, c.f. [11].

Relationship between Convolution and Pattern Matching Convolution is an important operation that has been shown to be essential in solving pattern matching problems. We show the connection between convolution and the pattern matching setting (of sliding a pattern over the text) through the use of a concrete simple pattern matching problem. Specifically, we show how convolution can be directly used to solve the *pattern matching with mismatches* problem over binary strings where the output is the number of mismatches between every alignment of the pattern with the text.

Theorem 2.3 (Folklore) *Given a binary pattern* $\mathbf{P}$ *of size* m *and a binary text* $\mathbf{T}$ *of size* $\mathcal{O}(m)$*, we can solve the pattern matching with mismatches problem in time* $\mathcal{O}(m \log m)$ *using FFT.*

Proof Let us define $\overline{\mathbf{P}}$ to be equal to $\mathbb{1} - \mathbf{P}$ which simply means that we convert every 1 in $\mathbf{P}$ to 0 and vice versa. Similarly, we define $\overline{\mathbf{T}} = \mathbb{1} - \mathbf{T}$. Thus for a given alignment at location i, it can be seen that due to the binary nature of the strings, the number of mismatches, $\mathbf{O}[i]$, is

$$\mathbf{O}[i] = \mathsf{Ham}[\mathbf{P}, \mathbf{T}[i, \ldots, i+m-1]] = \sum_{k=0}^{m-1} \overline{\mathbf{P}}[k] \cdot \mathbf{T}[i+k] + \sum_{k=0}^{m-1} \mathbf{P}[k] \cdot \overline{\mathbf{T}}[i+k]$$

We just focus on the first summand in the above equation and show how it can be computed via convolution. The same arguments follow for the second summand by symmetry.

In order to compute such summand for $0 \leq i \leq n - m$ using convolution, we need to tailor $\mathbf{T}$ and $\overline{\mathbf{P}}$ to the convolution format. To this end, it will be beneficial to observe that the convolution of two arrays $\mathbf{A}, \mathbf{B}$ is the sliding of the reverse of one vector along the other. Thus, as it is seen from below, at this particular alignment, you compute $\mathbf{C}[2]$

$$\begin{array}{ccccccccc} & & \mathbf{A}[0] & \mathbf{A}[1] & \mathbf{A}[2] & \mathbf{A}[3] & \mathbf{A}[4] & \mathbf{A}[5] & \dots \\ \dots & \mathbf{B}[3] & \mathbf{B}[2] & \mathbf{B}[1] & \mathbf{B}[0] & & & & \end{array}$$

This means that, for massaging $\mathbf{T}$ and $\overline{\mathbf{P}}$ into the convolution format, we create a new vector $\overline{\mathbf{P}}_{\text{rev}}$ by reversing $\overline{\mathbf{P}}$ and padding it with zeros so that both vectors $\overline{\mathbf{P}}_{\text{rev}}$, $\mathbf{T}$ input to the convolution function are of the same size. The convolution operation will reverse it one more time giving us our desired summand $\sum_{k=0}^{m-1} \overline{\mathbf{P}}[k] \cdot \mathbf{T}[i+k]$ with some extra un-needed values. Explicitly, $\sum_{k=0}^{m-1} \overline{\mathbf{P}}[k] \cdot \mathbf{T}[i+k] = (\overline{\mathbf{P}}_{\text{rev}} \odot \mathbf{T})[i+m-1]$. Thus, the problem can be solved using convolution in time $\mathcal{O}(m \log m)$ $\square$

In fact, the connection between convolution and the pattern matching setting is valid even for non-binary strings. This means that convolution can be used to compute $\mathbf{P} \diamond \mathbf{T}$ where

$$(\mathbf{P} \diamond \mathbf{T})[i] \stackrel{\text{def}}{=} \sum_{k=0}^{m-1} \mathbf{P}[k] \cdot \mathbf{T}[i+k]$$

for arbitrary $\mathbf{P}$ and $\mathbf{T}$. The use of binary strings was simply to motivate the connection between convolution and pattern matching using a concrete meaningful problem.

2.5 Matrix multiplication

Matrix multiplication is a binary operation that produces a matrix from two matrices. In this thesis, we consider the algebraic matrix product over a ring (such as integers). The product of two $n \times n$ matrices $\mathcal{A}$ and $\mathcal{B}$ is an $n \times n$ matrix $\mathcal{C}$ where

$$\mathcal{C}[i,j] = \sum_k \mathcal{A}[i,k] \cdot \mathcal{B}[k,j]$$

is the result of the dot product of the vectors formed by the i-th row of matrix $\mathcal{A}$ and the j-th column of matrix $\mathcal{B}$. For a long time it was believed that matrix multiplication requires cubic time until Strassen [29] gave an algorithm that computes it in $\mathcal{O}(n^{2.81})$. From there, a long line of research has been initiated trying to improve the exponent of the complexity of matrix multiplication. Precisely, this line of research cares about finding the optimal value of the exponent ω for which the matrix multiplication of $n \times n$ matrices takes $\mathcal{O}(n^{\omega})$. This culminated in the work of Coppersmith and Winograd [9] where they showed that $\omega < 2.376$. No progress has been made since then until recently culminating in the result by Stothers [28] and Vassilevska Williams [33] and then improved by Le Gall [20] where it was shown that $\omega < 2.3728639$.

Rectangular matrix multiplication is the operation defined to be the multiplication of two non-square matrices of dimensions $n \times d$ and $d \times n$ where $d \neq n$. We denote by $\omega(r,s,t)$ the exponent of multiplying an $\lceil n^r \rceil \times \lceil n^s \rceil$ matrix by a $\lceil n^s \rceil \times \lceil n^t \rceil$ one. The typical problem studied in this realm is understanding the value of the $\omega(1,k,1)$ for some $k \geq 0$. Note that for $k = 1$, $\omega(1,1,1) = \omega$. Coppersmith [8] showed that $\omega(1, 0.172, 1) = 2$. This means that the product of of two matrices of size $n \times \lceil n^{0.172} \rceil$ and $\lceil n^{0.172} \rceil \times n$ takes time $\mathcal{O}(n^2)$ which is almost linear in the size of the output (a matrix with n^2 entries). This led to the introduction of the parameter α defined as follows:

$$\alpha = \sup\{0 \leq r \leq 1 \mid \omega(1,r,1) = 2 + o(1)\}$$

Thus, from [8] we get that $\alpha \geq 0.172$. Later, Coppersmith [10] showed that $\alpha > 0.29462$. Recently, Le Gall in [21], [20] and then Le Gall and Urrutia [22] improved the bound further leading to $\alpha \geq 0.31389$.
Moreover, other work was done improving the upper bound for $\omega(1,k,1)$ for $k \neq 1$. More important were the bounds obtained by Huang and Pan [16] which was then further improved by Ke et al. [18]. Le Gall in [21] currently has the best upper bounds for $\omega(1,k,1)$ for $k \neq 1$ obtaining e.g. $\omega(1,2,1) < 3.251640$.

A result similar to Theorem 2.3 can be stated for establishing a connection between all-pairs problems and matrix multiplication. Explicitly, we can show that computing the Hamming distance between all pairs of two sets of binary vectors can be computed in the same time complexity required to multiply two matrices. This shows that algorithms solving pattern matching problems or all-pairs problems can make use of algebraic methods. In general, this sheds light on how useful these algebraic methods can be in the design of algorithms.

Chapter 3

Known Results in Pattern Matching

In this chapter, we survey some of the known pattern matching algorithms. We focus on problems whose time complexity is in $\widetilde{\mathcal{O}}(n\sqrt{m})$ (where $\widetilde{\mathcal{O}}$ is the same as $\mathcal{O}$ notation but suppresses polylogarithmic factors). This is because later on in the thesis, we show that these problems are in fact equivalent in terms of complexity (up to polylogarithmic factors). Problems surveyed in this chapter are Hamming distance pattern matching, L_1 pattern matching, less-than pattern matching and threshold pattern matching.

3.1 Hamming Distance Pattern Matching

HammingDistancePatternMatching (HamPM) as previously mentioned is the problem of counting the number of mismatches for each alignment of the pattern with the text. Written formally, the output is a vector $\mathbf{O}$ where for $0 \le i \le n-m$

$$\mathbf{O}[i] = \mathsf{Ham}[\mathbf{P}, \mathbf{T}[i, \ldots, i+m-1]]$$

This problem admits a trivial $\mathcal{O}(nm)$ brute force solution. All pattern matching problems, where pairwise comparisons between single characters take constant time, admits this naive time complexity. The challenge has always been to go beyond this quadratic time complexity. Fischer and Patterson [13] showed that this problem can in fact be solved faster than quadratic time.

Theorem 3.1 ([13]) *Given a pattern and a text of lengths m and n respectively of an arbitrary alphabet,* HamPM *can be solved in $\mathcal{O}(n\sqrt{m\log m})$ time.*

Before setting out to prove this result, let us first observe a useful trick that is considered a folklore result in pattern matching algorithms.

Observation 3.2 (Folklore) *For any pattern matching problem, we can always assume that $\mathbf{P}$ is of size m and $\mathbf{T}$ is of size $O(m)$. If the time complexity for the* *pattern matching problem is $\mathcal{O}(f(m))$ for some function f, then the time complexity for a text of size n is $\mathcal{O}(\frac{n}{m}f(m))$*

Proof Given a text of size n, we assume that $n \geq 2m$, otherwise there is nothing to prove. We divide the text into roughly $\lfloor n/m \rfloor$ overlapping parts of size $2m$. Thus the parts will be $\mathbf{T}[0,\ldots,2m-1], \mathbf{T}[m,\ldots,3m-1],\ldots$. We can then solve the pattern matching problem for each part separately and concatenate the resulting vectors at the end to obtain the final result for the whole text. The time complexity argument follows directly. □

Proof (Theorem 3.1) The proof works by computing the similarity measure instead of the Hamming distance in every alignment. The Hamming distance is obviously easily computed from the similarity measure by simply subtracting the similarity measure value from m. Let us denote by Σ the set of integers found in $\mathbf{P}$, namely the alphabet of the pattern. Also, let $\mathbb{F}$ be the set of the $|\mathbb{F}|$ most frequent elements in $\mathbf{P}$. The size of $\mathbb{F}$ will be determined later.
Let us also define the function $\chi_{=x} : \mathbb{Z}^n \to \mathbb{Z}^n$ which if given an integer vector $\mathbf{A}$, returns a vector $\mathbf{B}$ of the same size such that $\mathbf{B}[i] = 1$ iff $\mathbf{A}[i] = x$ and $\mathbf{B}[i] = 0$ otherwise. The algorithm has two parts:

1. **Convolution phase:** For each $f \in \mathbb{F}$, compute the similarity measure between $\chi_{=f}(\mathbf{P})$ and $\chi_{=f}(\mathbf{T})$ using convolution as shown in Theorem 2.3. Let $\mathbf{C}_f$ be the result of running the convolution for a particular f. Also let $\mathbf{C} = \sum_{f\in\mathbb{F}} \mathbf{C}_f$. This step captures all the contribution of the frequent elements in the similarity measure. This step takes $\mathcal{O}(|\mathbb{F}|n\log m)$

2. **Marking phase:** The missing part now is the contribution of the non-frequent elements in $\Sigma - \mathbb{F}$. The crucial observation is that each non-frequent element can not occur in the pattern more than $m/|\mathbb{F}|$ times. Otherwise, all frequent elements would occur more than $m/|\mathbb{F}|$ which is impossible. Following this observation, we can go over each entry in the text and *mark* the contribution of that entry, if it corresponds to an infrequent element, with the at most $m/|\mathbb{F}|$ locations in the pattern. This step takes $\mathcal{O}(n \cdot m/|\mathbb{F}|)$

Thus the total time complexity is $\mathcal{O}(|\mathbb{F}|n\log m + n\cdot m/|\mathbb{F}|)$. Choosing $|\mathbb{F}|$ to be equal to $\sqrt{\frac{m}{\log m}}$ gives us the desired time complexity of $\mathcal{O}(n\sqrt{m\log m})$.□

3.2 L_1 Pattern matching

$\mathrm{L_1}$PatternMatching ($\mathrm{L_1}$PM) is the problem of computing for each alignment of the pattern with the text, the sum of ℓ_1 distances between corresponding pairs of characters. Written formally, the output is a vector $\mathbf{O}$

where for $0 \leq i \leq n-m$

$$\mathbf{O}[i] = L_1[\mathbf{P}, \mathbf{T}[i, \ldots, i+m-1]]$$

This problem was introduced and solved independently by Amir et al. in [3] and by Clifford et al. in [6].

Theorem 3.3 ([3], [6]) *Given a pattern and a text of lengths m and n respectively of an arbitrary alphabet,* L_1PM *can be solved in* $\mathcal{O}(n\sqrt{m \log m})$ *time.*

The algorithm works as follows: The characters in the text are sorted according to their natural order and then divided into buckets. The ℓ_1 distance of characters from different buckets are obtained by computing convolution operations while the ℓ_1 distance of characters from the same bucket are obtained by a brute force operation. The algorithm is optimized by computing the best number of buckets which provides the best time complexity through a trade-off of the complexity of the convolution operations and the brute force.

Proof Let $\mathbb{S}$ be the lexicographically sorted list of the tuples $(\mathbf{P}[i], i)$ (Tuples are sorted by first element and by second element in case of a tie). Divide $\mathbb{S}$ into balanced buckets of size at most s, $\mathbb{B}_1, \mathbb{B}_2, \ldots, \mathbb{B}_{\lceil m/s \rceil}$.
For each element $\mathbf{T}[i]$, assign $\mathbf{T}[i]$ to bucket $\mathbb{B}_l$ such that $l = \max\{j : \mathbf{T}[i] \geq \min \mathbb{B}_j\}$ where $\min \mathbb{B}_j = \min\{a : (a,b) \in \mathbb{B}_j\}$ (In case $\mathbf{T}[i] < \min \mathbb{B}_1$, then assign $\mathbf{T}[i]$ to an additional empty bucket $\mathbb{B}_0$). The algorithm will have two parts similar to the proof of Theorem 3.1. A convolution step and a marking step.

1. **Convolution phase:** We create the following vectors out of **P** and **T** as follows:

$$\mathbf{A}_i[j] = \begin{cases} \mathbf{P}[j] & \text{if } (\mathbf{P}[j], j) \in \mathbb{B}_i \\ 0 & \text{otherwise} \end{cases}$$

$$\mathbf{B}_i[j] = \begin{cases} 1 & \text{if } \mathbf{T}[j] \text{ is assigned to } \mathbb{B}_k \wedge k < i \\ -1 & \text{if } \mathbf{T}[j] \text{ is assigned to } \mathbb{B}_k \wedge k > i \\ 0 & \text{otherwise} \end{cases}$$

$$\mathbf{C}_i[j] = \begin{cases} 1 & \text{if } (\mathbf{P}[j], j) \in \mathbb{B}_k \wedge k < i \\ -1 & \text{if } (\mathbf{P}[j], j) \in \mathbb{B}_k \wedge k > i \\ 0 & \text{otherwise} \end{cases}$$

$$\mathbf{D}_i[j] = \begin{cases} \mathbf{T}[j] & \text{if } \mathbf{T}[j] \text{ is assigned to } \mathbb{B}_i \\ 0 & \text{otherwise} \end{cases}$$

Thus, it can be verified that $\mathbf{X} = \sum_i (\mathbf{A}_i \diamond \mathbf{B}_i + \mathbf{C}_i \diamond \mathbf{D}_i)$ computes all the ℓ_1 distances between pattern elements and text elements that are assigned to different buckets. Explicitly, for any text element $\mathbf{T}[i]$ and pattern element $\mathbf{P}[j]$, if $\mathbf{T}[i]$ is assigned to bucket $\mathbb{B}_t$ and $\mathbf{P}[j] \in \mathbb{B}_p$. There are three different cases to consider:

a) $t = p$: In such case, for $p' \neq p : \mathbf{A}_{p'}[j] = 0$. As for $\mathbf{B}$, $\mathbf{B}_p[i] = 0$. Thus $(\sum_k \mathbf{A}_k \diamond \mathbf{B}_k)[i+j] = 0$. A similar argument can be made regarding $\mathbf{C}$ and $\mathbf{D}$.

b) $t > p$: In such case, $\mathbf{A}_p[j] = \mathbf{P}[j]$, $\mathbf{B}_p[i] = -1$, $\mathbf{C}_t[j] = 1$, $\mathbf{D}_t[i] = \mathbf{T}[i]$. thus $(\mathbf{A}_p \diamond \mathbf{B}_p)[i+j] = -\mathbf{P}[j]$ and $(\mathbf{C}_t \diamond \mathbf{D}_t)[i+j] = \mathbf{T}[j]$. However, for $r \notin \{p, t\}$, $(\mathbf{A}_r \diamond \mathbf{B}_r + \mathbf{C}_r \diamond \mathbf{D}_r)[i+j] = 0$. Thus, we get the correct value $\mathbf{X}[i+j] = T[i] - P[j]$.

c) $t < p$: An argument symmetric to that in case (b) can be made as well.

2. **Marking phase** In this phase, we want to capture the ℓ_1 distances between each text element and the pattern elements in the bucket the text element is associated with. In order to do this, a brute force procedure suffices where each text element $\mathbf{T}[i]$ associated with $\mathbb{B}_{i'}$ will be compared with every tuple $(\mathbf{P}[j], j) \in \mathbb{B}_{i'}$ and their ℓ_1 distance is aggregated to $\mathbf{Y}[i+j]$ where $\mathbf{Y}$ is the output vector of this phase.

The final output of the algorithm would then be $\mathbf{O} = \mathbf{X} + \mathbf{Y}$.
The running time of this algorithm is $\mathcal{O}(\frac{m}{s} \cdot n \log m + n \cdot s)$ where the first summand is the time complexity of running m/s convolution operations and the second summand is the time complexity of running the brute force algorithm in the marking phase. Setting s, the maximum size of each bucket, to be equal to $\sqrt{m \log m}$ gives the desired time complexity of $\mathcal{O}(n\sqrt{m \log m})$.$\square$

3.3 Less-than pattern matching

LessThanPatternMatching (LessThanPM) is the problem of computing the dominance product between a pattern and every equally-sized substring of the text. Written formally, the output is a vector $\mathbf{O}$ where for $0 \leq i \leq n - m$

$$\mathbf{O}[i] = \mathsf{Dom}[\mathbf{P}, \mathbf{T}[i, \ldots, i+m-1]] = \left|\{\, k \ : \mathbf{P}[k] \leq \mathbf{T}[i+k]\}\right|$$

The problem was introduced and an algorithm has been given for it by Amir and Farach in [2]. Our proof will be slightly different so that it follows the same structure as that in Theorem 3.3.

Theorem 3.4 ([2]) *Given a pattern and a text of lengths m and n respectively of an arbitrary alphabet,* LessThanPM *can be solved in* $\mathcal{O}(n\sqrt{m \log m})$.

Proof Similar to the proof of Theorem 3.3, we divide the pattern elements into buckets. We also assign each text element to some bucket with the same conditions.
The algorithm is also divided into two different phases:

1. **Convolution phase:** The difference, however, lies in how we construct the new vectors. In fact, we only need two sets of vectors $\mathbf{A}_i, \mathbf{B}_i$. These vectors will be constructed from $\mathbf{P}$ and $\mathbf{T}$ as follows:

$$\mathbf{A}_i[j] = \begin{cases} 1 & \text{if } (\mathbf{P}[j], j) \in \mathbb{B}_i \\ 0 & \text{otherwise} \end{cases}$$

$$\mathbf{B}_i[j] = \begin{cases} 1 & \text{if } \mathbf{T}[j] \text{ is assigned to } \mathbb{B}_k \wedge k > i \\ 0 & \text{otherwise} \end{cases}$$

   It can be verified that $\mathbf{X} = \sum_i \mathbf{A}_i \diamond \mathbf{B}_i$ computes all the number of pairs $(\mathbf{T}[i], \mathbf{P}[j])$ such that $\mathbf{T}[i] \geq \mathbf{P}[j]$, $\mathbf{T}[i]$ is assigned to bucket $\mathbf{B}_{i'}$, $(\mathbf{P}[j], j) \in \mathbf{B}_{j'}$ and $i' > j'$.

2. **Marking phase:** In this phase, we compute the number of pairs of text and pattern elements, where the text element is greater than or equal to a pattern element and both elements are assigned to the same bucket. In order to do this, a brute force procedure suffices where each text element $\mathbf{T}[i]$ associated with $\mathbb{B}_{i'}$ will be compared with every tuple $(\mathbf{P}[j], j) \in \mathbb{B}_{i'}$ and we increment $\mathbf{Y}[i + j]$ if $\mathbf{T}[i] \geq \mathbf{P}[j]$ where $\mathbf{Y}$ the output vector of this phase.

The final output of the algorithm would then be $\mathbf{O} = \mathbf{X} + \mathbf{Y}$. The running time analysis of such algorithm is exactly as shown in the proof of theorem 3.3 since we are following the same steps but with simpler construction of the auxiliary vectors. □

3.4 Threshold pattern matching

ThresholdPatternMatching (ThrPM) is the generalized version of HamPM. Given a threshold δ, the output is a vector $\mathbf{O}$ where for $0 \leq i \leq n - m$

$$\mathbf{O}[i] = \mathsf{Thr}_\delta[\mathbf{P}, \mathbf{T}[i, \ldots, i + m - 1]] = |\{k : |\mathbf{P}[k] - \mathbf{T}[i + k]| > \delta\}|$$

It can be easily seen that for $\delta = 0$, the problem becomes equivalent to the HamPM.

The problem was introduced and an algorithm has been given for it by Atallah and Duket in [4].

Theorem 3.5 ([4]) *Given a pattern and a text of lengths m and n respectively of an arbitrary alphabet and a threshold δ,* ThrPM *can be solved in $\mathcal{O}(n\sqrt{m \log m})$ time.*

The proof for this theorem is more convoluted than the previous problems. Thus, we provide a sketch for the proof omitting minor details where necessary. For full details, please refer to the original paper [4]. Throughout the proof, we assume that we are dealing with a text of length $\mathcal{O}(m)$ and using observation 3.2, we multiply the final complexity by n/m. Moreover, we provide an algorithm that actually computes the complement of the desired output. Precisely, it computes the number of pairs of symbols whose distance are at most δ from each other. However, similar to what was done for HamPM, we subtract the output of the algorithm from m to get the desired output.

Proof Let us first define n_a to be the number of occurrences of the symbol a in either the text or the pattern. The algorithm is divided into four stages:

1. We compute the contribution of all frequent symbols where a symbol a is said to be frequent if $n_a > \sqrt{m \log m}$. We have at most $\mathcal{O}(\sqrt{m/\log m})$ frequent symbols. For each frequent symbol, we can compute its contribution using a single convolution. Thus, the total time for computing the contribution of all frequent symbols is $\mathcal{O}(m\sqrt{m \log m})$. Thus, in all the remaining steps, we never consider frequent symbols.

2. Alphabet symbols that occur at least once (excluding the frequent symbols) are sorted increasingly. We then partition the symbols into maximal regions such that symbols in the same region are all within δ difference from each other. In other words, we start from the first symbol in the sorted list of alphabet symbols and place it in the first region and keep adding symbols in the sorted order and stop adding more symbols into the current region if the difference between that symbol and some symbol in the region exceeds δ. Since there is at most $\mathcal{O}(m)$ symbols in total, this is also a bound on the number of groups. We call a region whose sum of occurrences of its symbols is more than $\sqrt{m \log m}$ a *heavy* region, otherwise we call it a *light* region.

3. Intra-region contributions for each bucket can be easily computed with a single convolution operation in $\mathcal{O}(m \log m)$ time. There are at most $\mathcal{O}(\sqrt{m/\log m})$ heavy regions. Thus the intra-region contribution of heavy regions can be computed in a total time of $\mathcal{O}(m\sqrt{m \log m})$. As for light regions, a brute force approach is used for each region. Denoting by n_X the number of occurrences of symbols in region X, this is done in $\mathcal{O}(n_X^2)$. Summing up over all light regions we get the total time for all light regions. Using the fact that $n_X \leq \sqrt{m \log m}$ for light

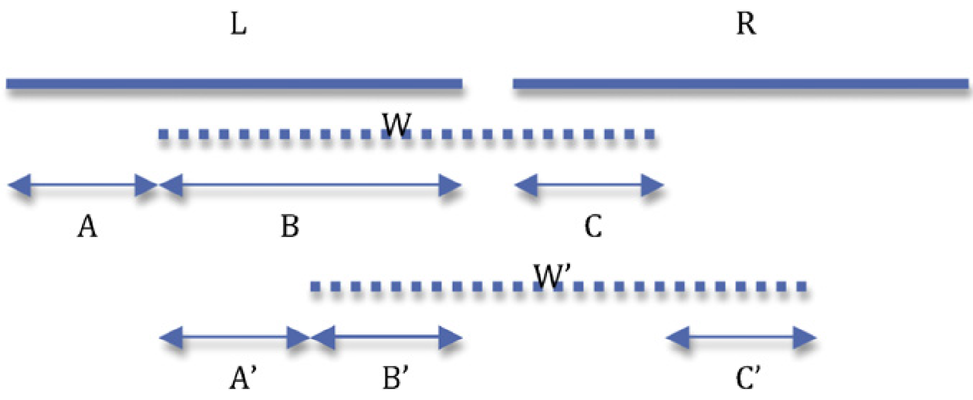


Figure 3.1: Figure courtesy of [4]. An illustration of the sweeping algorithm.

region X and that we have $\mathcal{O}(m)$ symbols in total so $\sum_X n_X = \mathcal{O}(m)$, we end up with:

$$\sum_X n_X^2 \leq \sqrt{m \log m} \sum_X n_X = \mathcal{O}(m\sqrt{m \log m})$$

4. For inter-region contribution, it is important to notice that contributions only occur between adjacent regions since for non adjacent regions, the difference between symbols would be greater than δ and so the inter region contribution of such regions would be 0. For any two adjacent regions X, Y: if one of the regions is light, we use brute force to compute the contribution between elements from these two regions in $\mathcal{O}(n_X \cdot n_Y)$. Thus, for each pair of adjacent regions with such property takes time $\sqrt{m \log m} \max(n_X, n_Y)$ and since each region is involved in at most two pairs (only with adjacent regions to the left and right), the total time becomes $\mathcal{O}(m\sqrt{m \log m})$

5. The only remaining tricky case is when the two adjacent regions are heavy. We compute the contribution of such pairs of regions through the use of a sweeping algorithm. Let us denote the two regions by L, R representing the left and right regions. We slide a window W of width δ over the two regions starting from the first symbol of L. We slide W to the right direction until the number of occurrences of symbols in R covered by W first exceeds $\sqrt{m \log m}$. At this point, W covers some symbols from L, denote them with B. It also covers some symbols from R, denote them with C and denote symbols from L not covered by W with A. Observe that without the last symbol in C, the number of occurrences of the remaining symbols is at most $\sqrt{m \log m} - 1$ and because of the first pre-processing step, the last symbol of C is not frequent. Thus the number of occurrenes of symbols in C is at most $2\sqrt{m \log m} - 1$. The contribution of pairs of symbols from $B \times C$ can be

computed in time $\mathcal{O}(m \log m)$ using two convolution operations. The contribution of pairs of symbols from $A \times C$ can be computed by brute force in time $\mathcal{O}(n_A \cdot n_C)$. Observe that the contribution of pairs of symbols from $A \times (R \backslash C)$ is zero since the difference between them exceeds δ. Starting from the first symbol of W, we slide another window, W' forgetting about symbols in A since they will not contribute with any other elements from R, and stopping next when the number of occurrences of symbols in $R \cap (W' \backslash W)$ exceeds $\sqrt{m \log m}$. We then define $A' = L \cap (W \backslash W')$, $B' = L \cap W'$ and $C' = R \cap (W' \backslash W)$ and perform the same operations as before but with A', B', C' instead. We repeat this process and stop when the sweeping operation reaches the last symbol of R. Refer to Figure 3.1

- The correctness of the algorithm follows from observing that pairs of symbols that provide any contribution are going to be in $A \times C, B \times C, A' \times C', B' \times C', \ldots$ and since the algorithm compute the contribution of all such pairs, the correctness follows. For details, refer to the original paper.

- In order to analyze the time complexity of the sweeping algorithm, we analyze the two computation steps, namely that of (a) $B \times C, B' \times C', \ldots$ and (b) $A \times C, A' \times C', \ldots$, separately.

  For part (a), observe that $\sqrt{m \log m} \leq |n_C|, |n'_C|, \cdots \leq 2\sqrt{m \log m} - 1$. It follows that there are at most $|n_R| / \sqrt{m \log m}$ of those partitions for region R. Thus, the number of such partitions for all regions is at most $\mathcal{O}(m) / \sqrt{m \log m}$. Thus, the computation of the contribution of pairs of symbols in $B \times C, B' \times C', \ldots$ takes time

  $$\frac{\mathcal{O}(m)}{\sqrt{m \log m}} \cdot \mathcal{O}(m \log m) = \mathcal{O}(m \sqrt{m \log m}).$$

  For part (b), the total time complexity is equal to $n_A \cdot n_C + n_{A'} \cdot n_{C'} + \ldots$. Using the observation that $|n_C|, |n_{C'}|, \cdots \leq 2\sqrt{m \log m}$ and that $A, A', \cdots$ form a partition of region L, we can see that $n_A \cdot n_C + n_{A'} \cdot n_{C'} + \cdots \leq 2\sqrt{m \log m}(n_A + n_{A'} + \cdots) = 2\sqrt{m \log m} \cdot \mathcal{O}(m) = \mathcal{O}(m\sqrt{m \log m})$. □

3.5 Related work

Another closely related problem to L_1PM, but complexity-wise easier, is L_2PatternMatching (L_2PM). For L_2PM, the output vector is $\mathbf{O}[i] = \sum_j (\mathbf{P}[j] - \mathbf{T}[i+j])^2$. As observed by Lipsky and Porat [24], this reduces to $\mathbf{O}[i] = \sum_j \mathbf{P}[j]^2 + \sum_j \mathbf{T}[i+j]^2 - 2\sum_j \mathbf{P}[j]\mathbf{T}[i+j]$. Thus, the dominating term in the

Name	Score function	Pattern Matching problem		Complexity	
Hamming	$\mathbb{1}[x \neq y]^*$	HAMPM	$\mathbf{O}[i] = \lvert\{j : \mathbf{P}[j] \neq \mathbf{T}[i+j]\}\rvert$	$\mathcal{O}(n\sqrt{m \log m})$	[1]
Dominance	$\mathbb{1}[x \leq y]$	LESSTHANPM	$\mathbf{O}[i] = \lvert\{j : \mathbf{P}[j] \leq \mathbf{T}[i+j]\}\rvert$	$\mathcal{O}(n\sqrt{m \log m})$	[2]
Threshold	$\mathbb{1}[\lvert x-y\rvert > \delta]$	THRPM	$\mathbf{O}[i] = \lvert\{j : \lvert\mathbf{P}[j] - \mathbf{T}[i+j]\rvert > \delta\}\rvert$	$\mathcal{O}(n\sqrt{m \log m})$	[4]
ℓ_1 distance	$\lvert x-y\rvert$	L_1PM	$\mathbf{O}[i] = \sum_j \lvert\mathbf{P}[j] - \mathbf{T}[i+j]\rvert$	$\mathcal{O}(n\sqrt{m \log m})$	[6, 3]
ℓ_2 distance	$(x-y)^2$	L_2PM	$\mathbf{O}[i] = \sum_j (\mathbf{P}[j] - \mathbf{T}[i+j])^2$	$\mathcal{O}(n \log m)$	[24]

Table 3.1: Summary of different pattern matching problems along with their score functions and runtime.

computation arises from computing a single convolution in time $\mathcal{O}(n \log m)$ via Fast Fourier Transform (FFT). This approach extends to L_{2p}PM, with runtime $\mathcal{O}(p^2 n \log m)$.

Weighted Pattern Matching in the most general setting asks for $\mathbf{O}[i] = \sum_j w(P[j], T[i+j])$ for some weight function w. Lipsky and Porat [24] presented a simple $\mathcal{O}(|\Sigma| n \log m)$ algorithm for solving this problem.

Eaxct Pattern Matching with Wildcards (wildcard characters are special characters that match any other character in the alphabet. We denote them later on by '$\star$'.) admits a simple deterministic $\mathcal{O}(n \log m)$ using a similar solution as weighted L_2PM, as shown by Clifford and Clifford [5].

Summary

In the chapter, we surveyed algorithms for problems whose time complexity is in $\mathcal{O}(n\sqrt{m \log m})$. They all relied on the technique of decomposing the problem into several simpler instances that can be solved by convolution together with a brute force part. Notice that L_2PM is an exception to those and can be solved much faster. Refer to Table 3.1 for a summary of the results.

* $\mathbb{1}[\varphi]$ is 1 iff φ and 0 otherwise.

Chapter 4

Known Results in All Pairs Problems

In this chapter, we survey some of the AllPairs problems. In particular, we survey two problems whose time complexity is nearly the same. Both problems admit a running time close to $\mathcal{O}(n^{(3+\omega)/2})$ if all vectors are of dimension $d = n$. We also explore different settings of the problem and some improvements in the mentioned complexity.

4.1 All-pairs dominance products

AllPairsDominanceProducts (APDom) is the all pairs problem corresponding to the less than pattern matching problem. The output is a matrix O where, for $1 \leq i, j \leq n$

$$O[i,j] = \mathsf{Dom}[\mathbf{A}_i, \mathbf{B}_j] = \left|\{\ k\ :\ \mathbf{A}_i[k] \leq \mathbf{B}_j[k]\}\right|$$

This problem admits a trivial $\mathcal{O}(n^2 d)$ brute force solution. Actually, every AllPairs problem, whose underlying score function between a pair of alphabet symbols takes constant time, admit this naive time complexity. The challenge has always been to go beyond this cubic, for $d = n$, time complexity. Matoušek [25] presented an elegant sub-cubic algorithm for this problem.

Theorem 4.1 ([25]) APDom *for vectors of dimension $d = n$ can be computed in time $\mathcal{O}(n^{(3+\omega)/2})$.*

Proof A crucial observation in any product between matrices $\mathcal{A}$ and $\mathcal{B}$ is that the elements of column i in matrix $\mathcal{A}$ are involved in the product calculation with elements in row i in matrix $\mathcal{B}$, i.e: elements of coordinate i interact, only, with each other. More formally,

$$\mathcal{A} \cdot \mathcal{B} = \sum_{i=1}^{d} \mathcal{A}_{*i} \cdot \mathcal{B}_{i*}$$

where $\mathcal{A}_{*i}$ is the i-th column vector of matrix $\mathcal{A}$ and $\mathcal{B}_{i*}$ is the i-th row of matrix $\mathcal{B}$. This means that a matrix product between matrices $\mathcal{A}$ and $\mathcal{B}$ is an operation that involves the calculations of d parallel outer products and summing the resulting matrices of all of them together.

We make use of this observation by treating the elements of each coordinate separately. On a high level, elements of each coordinate are to be partitioned into buckets according to their rank in the natural order. We then carry out computations between elements from different buckets and between elements from within the same bucket. Similar to what was done in Chapter 3, these two computations are performed by an algebraic computation and by brute force force respectively.

We start off by sorting the $2n$ vectors, $\mathbf{A}_1, \ldots, \mathbf{A}_n, \mathbf{B}_1, \ldots, \mathbf{B}_n$, once by each coordinate. Since we have n coordinates, this takes time $\mathcal{O}(n^2 \log n)$. Define the rank of a vector $\mathbf{X}$ according to coordinate c by $r_c(\mathbf{X})$. We partition the coordinates of vectors into equally sized buckets of size s (except the last bucket that might be smaller). This means, that for every coordinate, each vector lies in a specified bucket according to its rank. We, thus, have $\lceil n/s \rceil$ buckets for each coordinate. We create $\lceil n/s \rceil$ pairs of binary matrices. For $0 \leq \ell < \lceil n/s \rceil$, we define $\mathcal{A}_\ell$, $\mathcal{B}_\ell$ as follows:

$$\begin{aligned} \mathcal{A}_\ell[i,j] &= 1 \quad \text{if } r_j(\mathbf{A}_i) \in [s\ell, s\ell + s) \\ \mathcal{B}_\ell[j,k] &= 1 \quad \text{if } r_j(\mathbf{B}_k) \geq s\ell + s \end{aligned}$$

Missing entries for $\mathcal{A}_\ell$ and $\mathcal{B}_\ell$ are filled with 0. By regular matrix multiplication $\mathcal{A}_\ell \cdot \mathcal{B}_\ell$, in time $\mathcal{O}(n^\omega)$, we compute the contribution of elements of vectors in $\mathcal{A}$ that fall in bucket ℓ with corresponding elements in $\mathcal{B}$ that fall in buckets larger than ℓ. Since the value of elements in one bucket are assured to be at most the value of elements in later buckets, the matrix

$$\mathcal{C} = \sum_{\ell=0}^{\lceil n/s \rceil - 1} \mathcal{A}_\ell \cdot \mathcal{B}_\ell$$

computes the inter-bucket contributions.

However, we are still missing contributions between elements that fall within the same bucket. We compute such contribution in a brute force manner. Each element in matrix $\mathcal{A}$, say $\mathcal{A}[i,j]$, will only interact with elements in row j in matrix $\mathcal{B}$. Since we already know the rank of all vectors according to coordinate j, we can check, in brute force, whether $\mathcal{A}[i,j] \leq \mathcal{B}[j,k]$ only for vectors that fall in the same bucket, namely $\mathbf{B}_k$ where $\lfloor r_j(\mathbf{A}_i)/s \rfloor = \lfloor r_j(\mathbf{B}_k)/s \rfloor$. Since each bucket contains at most s elements, this takes time $\mathcal{O}(n^2 s)$. We can compute a result matrix $\mathcal{D}$ out of this operation. Thus, the final answer would be $\mathcal{C} + \mathcal{D}$ and the total time for both steps is $\mathcal{O}(\frac{n}{s} n^\omega + n^2 s)$. Choosing $s = n^{(\omega-1)/2}$ gives the best trade-off and the desired time bound. $\square$

Yuster [34] observed that this can be sped up. Instead of computing the intermediate square matrix multiplications $\mathcal{A}_\ell \cdot \mathcal{B}_\ell$ and summing them up, this can be done in one step through creating one pair of rectangular matrices and multiplying them instead. In that paper, Yuster first showed an equivalence between AllPairsDominanceProducts and another closely related problem named *permutation dominance* (cf. [34] for a definition of the problem).

Lemma 4.2 ([34]) APDom *can be computed in* $\mathcal{O}(n^\rho)$ *where* ρ *is a solution to* $\rho = \omega(1, 4-\rho, 1)$ *.

Proof For simplicity, we denote the bucket size by n^{1-s}. Thus, we have n^s buckets. The n^s pairs of matrices $\mathcal{A}_l, \mathcal{B}_l$ are created as before. The only difference is that we stack the square matrices into one pair of rectangular matrices of size $n \times (n \times n^s)$ and $(n \times n^s) \times n$ and use rectangular matrix multiplication as follows:

$$\mathcal{C} = \begin{bmatrix} \mathcal{A}_1 & \cdots & \mathcal{A}_{n^s} \end{bmatrix} \times \begin{bmatrix} \mathcal{B}_1 \\ \vdots \\ \mathcal{B}_{n^s} \end{bmatrix}$$

Using $r = 1 + s$, this rectangular matrix multiplication is computed in time $\mathcal{O}(n^{\omega(1,r,1)})$. Matrix $\mathcal{D}$ holding the intra-bucket contributions is computed by brute force exactly as before in time $\mathcal{O}(n^2 \cdot n^{1-s}) = \mathcal{O}(n^{4-r})$. Thus, the total time is $\mathcal{O}(n^{\omega(1,r,1)} + n^{4-r})$ and minimizing the runtime is done by solving $\omega(1, r, 1) = 4 - r$ for r. Substituting $\rho = 4 - r$ verifies the statement of the lemma. $\square$

4.1.1 Low dimensional case

If $d \ll n$, then Matousek's approach together with Yuster's approach can be generalized to the low dimensional case. Vassilevska and Williams [31] actually provided this generalization before Yuster provided his algorithm for the case $d = n$. Before stating their result, we state a lemma obtained by Huang and Pan [16]:

Lemma 4.3 *([16]). Let* $\alpha = \sup\{0 \le r \le 1 \mid \omega(1, r, 1) = 2 + o(1)\}$*. Then for all* $n^\alpha \le d \le n$*, one can multiply an* $n \times d$ *by a* $d \times n$ *matrix in time*

$$\mathcal{O}\left(d^{\frac{\omega-2}{1-\alpha}} \cdot n^{\frac{2-\omega\alpha}{1-\alpha}}\right)$$

Le Gall [21] and subsequently Le Gall and Urrutia [22] gave the best known bound for α, stating $\alpha > 0.31389$.

*$\omega(r, s, t)$ is the exponent of the time complexity of multiplying two matrices of size $\lceil n^r \rceil \times \lceil n^s \rceil$ and $\lceil n^s \rceil \times \lceil n^t \rceil$

Lemma 4.4 ([31])) *Given an $n \times d$ matrix $\mathcal{A}$ and a $d \times n$ matrix $\mathcal{B}$ where $d = \mathcal{O}(n^{(\omega-1)/2})$, we can compute their* APDom *in time $\mathcal{O}(n^{\frac{2\omega-\omega\alpha-2}{\omega-\alpha-1}} d^{\frac{2\omega-4}{\omega-\alpha-1}} + n^{2+o(1)})$*

Proof The proof follows similarly to that of Lemma 4.2. The matrix $\mathcal{C}$ is computed as before. However denoting the bucket's size by s, we get

$$\mathcal{C} = \begin{bmatrix} \mathcal{A}_1 & \cdots & \mathcal{A}_{n/s} \end{bmatrix} \times \begin{bmatrix} \mathcal{B}_1 \\ \vdots \\ \mathcal{B}_{n/s} \end{bmatrix}$$

Thus, $\mathcal{C}$ is computed via one rectangular matrix multiplication of matrices with size $n \times dn/s$ and $dn/s \times n$. Matrix $\mathcal{D}$ is computed in brute force manner. By Lemma 4.3, the total computation takes time

$$\mathcal{O}\Big((\frac{dn}{s})^{\frac{\omega-2}{1-\alpha}} \cdot n^{\frac{2-\omega\alpha}{1-\alpha}} + nds \Big)$$

Choosing s to be

$$s = d^{\frac{\omega+\alpha-3}{\omega-\alpha-1}} \cdot n^{\frac{\omega+\alpha-\omega\alpha-1}{\omega-\alpha-1}}$$

minimizes the total run time and gives the desired time bound. The reason behind the bound on the dimension d is that beyond that bound, the rectangular matrices created with sizes $n \times nd/s$ and $nd/s \times n$ will have their value $nd/s > n$ and thus Lemma 4.3 does not hold. □

Gold and Sharir [15] performed the analysis for the case $d > n^{(\omega-1)/2}$. They combined recent results in rectangular matrix multiplication together with an interpolation technique to obtain explicit expressions for values of $d \leq n^{1.056}$ which includes the case $d = n$. Using their analysis, they improved the exponent that was provided by Yuster for $d = n$.

4.1.2 Sparse dominance product

Vassilevska [30] and Vassilevska, Williams and Yuster [32] considered dominance product on sparse inputs where we denote by m_1 and m_2 the number of entries in $\mathcal{A}$ and $\mathcal{B}$, respectively that contribute to the score. Using refined bucketing and rectangular matrix multiplication, they obtain a bound of $\mathcal{O}(\min(n^\omega + \sqrt{m_1 m_2} \cdot n^{\frac{\omega-1}{2}}, n^2 + (m_1 m_2)^{\frac{\omega-2}{\omega-\alpha-1}} n^{\frac{2-\alpha\omega}{\omega-\alpha-1}}))$. However, Duan and Pettie [12] simplified this analysis and below we detail their contribution. They describe a simple yet novel procedure that helps them in achieving that, namely *row-balancing*.

Row-balancing and column-balancing In order to prove Theorem 4.5 below, they used a novel technique which they named row-balancing. The general idea is that given a single matrix with m relevant (finite i.e. $\notin \{\infty, -\infty\}$)

elements, row-balancing (column-balancing) outputs a new pair of matrices in such a way that each row (column) in each of the new matrices contains at most $\lceil m/n \rceil$ relevant entries. Below, we provide the steps for computing the output of the row-balancing procedure on a matrix $\mathcal{A}$ of size $n \times p$. Let $k = \lceil m/n \rceil$. The output of the row-balancing procedure on matrix $\mathcal{A}$ is denoted as $rb(\mathcal{A}) = (\mathcal{A}', \mathcal{A}'')$. $\mathcal{A}', \mathcal{A}''$ are to be of the same size as $\mathcal{A}$.

- Sort all of the finite elements in the i-th row of $\mathcal{A}$ in increasing order and divide the sorted list into parts $T_i^1, T_i^2, ...T_i^{a_i}$ such that all but the last part contains exactly k elements and the last part contains at most k elements.

- Matrix $\mathcal{A}'$ should contain all elements that exist in the last part of each sorted list for all rows. Formally,

$$\mathcal{A}'[i,j] = \begin{cases} \mathcal{A}[i,j] & if\ \mathcal{A}[i,j] \in T_i^{a_i} \\ \infty & \text{otherwise} \end{cases}$$

- All the remaining finite elements will go the second output matrix $\mathcal{A}''$. Since each part of the sorted lists is of size of exactly k and since $k = \lceil m/n \rceil$, then the number of parts is at most $m/k \leq n$. Thus each part can go to a separate row in matrix $\mathcal{A}''$. We choose an arbitrary injective mapping $\rho : [n] \times [m/k] \rightarrow [n]$ to map from a part of a certain sorted list of a certain row to a row in matrix $\mathcal{A}''$. Formally

$$\mathcal{A}''[i',j] = \begin{cases} \mathcal{A}[i,j] & if\ \mathcal{A}[i,j] \in T_i^l,\ l < a_i,\ \rho(i,l) = i' \\ \infty & otherwise \end{cases}$$

  Thus, $rb(\mathcal{A}) = (\mathcal{A}', \mathcal{A}'')$ decomposes a matrix $\mathcal{A}$ into two matrices $\mathcal{A}', \mathcal{A}''$ such that each row in the new matrices contains at most k elements and each finite element in $\mathcal{A}$ has only one corresponding element in either $\mathcal{A}'$ or $\mathcal{A}''$.

Using the above Row(Column)-Balancing technique, they were able to prove the following theorem:

Theorem 4.5 ([12]) *Let $\mathcal{A}$ and $\mathcal{B}$ be two $n \times n$ matrices where the number of non-(∞) values in $\mathcal{A}$ is m_1 and the number of non-($-\infty$) values in $\mathcal{B}$ is m_2. Then* APDom *of $\mathcal{A}$ and $\mathcal{B}$ can be computed in time $O(m_1 m_2/n + n^\omega)$.*

This means that if $m_1, m_2 \in O(n^{(1+\omega)/2})$, then the total time complexity of the algorithm is equal to $O(n^\omega)$

Proof As discussed before, in any matrix product between matrices $\mathcal{A}$ and $\mathcal{B}$, elements in column i of matrix $\mathcal{A}$ only interact with elements of row i in

$\mathcal{B}$. This observation is crucial for the proof. Let $(\mathcal{A}', \mathcal{A}'') = cb(\mathcal{A})$ be the column-balancing of matrix $\mathcal{A}$. We build two binary matrices $\hat{A}, \hat{B}$ such that

$$\begin{aligned} \hat{A}[i,k] = 1 &\quad \text{if } \mathcal{A}''[i,k] \neq \infty \\ \hat{B}[k,j] = 1 &\quad \text{if } \rho(k',\ell) = k \;\&\; \mathcal{B}[k',j] \geq \max_\ell T_{k'}^\ell \end{aligned}$$

In words, $\hat{A}$ will have a one for every finite element in $\mathcal{A}''$. Elements in column k of $\mathcal{A}''$ come from column k' of $\mathcal{A}$ if $\rho(k',\ell) = k$ where ℓ denotes the ℓ-th part of the sorted list of column k' of $\mathcal{A}$. Since, as previously mentioned, elements from column k' of matrix $\mathcal{A}$ interact only with elements in row k' of matrix $\mathcal{B}$ and since elements in part ℓ of column k' of matrix $\mathcal{A}$ ended up in column k in matrix $\mathcal{A}''$, we will put ones in row k of $\hat{B}$ for elements in row k' in matrix $\mathcal{B}$ that are at least in value the maximum element in part l of the sorted list of column k' of matrix $\mathcal{A}$. Thus, it can be verified that the matrix multiplication $\hat{C} = \hat{A} \cdot \hat{B}$ will count all the times where $\mathcal{A}[i,k] \leq \mathcal{B}[k,j]$ and $\mathcal{B}[k,j]$ is larger than the largest element of the part that element $\mathcal{A}[i,k]$ falls into. Two cases remain to be counted:

- Dominances that occur when $\mathcal{A}[i,k] \leq \mathcal{B}[k,j]$ where $\mathcal{A}[i,k] \in T_k^\ell$ but $\mathcal{B}[k,j]$ does not dominate all elements in T_k^ℓ
- Dominances that occur when $\mathcal{A}[i,k] \leq \mathcal{B}[k,j]$ where $\mathcal{A}[i,k] \in T_k^{a_k}$ (These are the elements that ended in $\mathcal{A}'$ in the column-balancing operation).

For each finite element in $\mathcal{B}$, $\mathcal{B}[k,j]$ will be compared with only one part of the sorted list of column k of $\mathcal{A}$, say part T_k^q where $q = \arg\max_z T_k^z : \mathcal{B}[k,j] \geq \min T_k^z$. This can be done in time $O(m_2 m_1/n)$ since each part of a sorted list contains at most $k = \lceil m_1/n \rceil$ elements from $\mathcal{A}$. $\square$

4.2 All-pairs L_1 distances

AllPairsL$_1$Distances (APL$_1$) is the all pairs problem corresponding to L$_1$PM. The output is a matrix O where, for $1 \leq i,j \leq n$

$$O[i][j] = L_1[\mathbf{A}_i, \mathbf{B}_j] = \sum_k |\mathbf{A}_i[k] - \mathbf{B}_j[k]|$$

This problem was presented and an algorithm was presented by Indyk et al. [17]. In that work, they solved the problem by tailoring the high/low frequency technique used for solving HamPM into the AllPairs- setting. However, we present a proof that is very similar to the one presented above for solving APDom. For brevity, we point out to the similar steps between the proofs and only write in detail where the proofs differ.

Theorem 4.6 APL$_1$ *for vectors of dimension $d = n$ can be computed in time $\mathcal{O}(n^{(3+\omega)/2})$.*

Proof The proof follows the same steps as that of Theorem 4.1. The sorting and bucketing is done exactly the same. The difference, however, lies in the construction of the auxiliary matrices for each bucket. Note that, unlike the dominance function, we are aiming to compute the L_1 distance between integers. Thus, for each bucket, we create two pairs of matrices instead of just one pair. This is needed since we want to compute the L_1 distance between elements in matrix $\mathcal{A}$ that are within a given bucket and elements in smaller and larger buckets. Observe, that for AllPairsDominanceProducts, we only cared for elements in larger buckets since they are the ones that give a contribution. Thus, we create $\lceil n/s \rceil$ pairs of binary matrices. For $0 \leq \ell < \lceil n/s \rceil$, we define $\mathcal{A}_\ell$, $\mathcal{B}_\ell$ and $\mathcal{C}_\ell$, $\mathcal{D}_\ell$ as follows:

$$\mathcal{A}_\ell[i,j] = \mathbf{A}_i[j] \quad \text{if } r_j(\mathbf{A}_i) \in [s\ell, s\ell + s)$$

$$\mathcal{B}_\ell[j,k] = \begin{cases} -1 & \text{if } r_j(\mathbf{B}_k) \geq s\ell + s \\ 0 & \text{if } r_j(\mathbf{B}_k) \in [s\ell, s\ell + s) \\ 1 & \text{if } r_j(\mathbf{B}_k) < s\ell \end{cases}$$

$$\mathcal{C}_\ell[i,j] = 1 \quad \text{if } r_j(\mathbf{A}_i) \in [s\ell, s\ell + s)$$

$$\mathcal{D}_\ell[j,k] = \begin{cases} \mathbf{B}_k[j] & \text{if } r_j(\mathbf{B}_k) \geq s\ell + s \\ 0 & \text{if } r_j(\mathbf{B}_k) \in [s\ell, s\ell + s) \\ -\mathbf{B}_k[j] & \text{if } r_j(\mathbf{B}_k) < s\ell \end{cases}$$

Using the constructed matrices, we compute the inter-bucket contributions as:

$$\mathcal{C} = \sum_{\ell=0}^{\lceil n/s \rceil - 1} \mathcal{A}_\ell \cdot \mathcal{B}_\ell + \mathcal{C}_\ell \cdot \mathcal{D}_\ell$$

To verify that this in fact measures the inter-bucket contributions, suppose for two vectors $\mathbf{A}_i$ and $\mathbf{B}_k$, elements of their coordinate j lies in buckets ℓ and ℓ' respectively such that $\ell' > \ell$. The contribution of such pair of coordinates is equal to $\mathbf{B}_k[j] - \mathbf{A}_i[j]$. This contribution will be obtained by the corresponding elements in the constructed matrices as $\mathcal{A}_\ell[i,j] \cdot \mathcal{B}_\ell[j,k] + \mathcal{C}_\ell[i,j] \cdot \mathcal{D}_\ell[j,k]$ which is equal to $\mathbf{A}_i[j] \cdot (-1) + 1 \cdot \mathbf{B}_k[j]$ which is the desired value. The opposite case when $\ell' < \ell$ is similar.

Computing the intra-bucket contributions is done in a brute force manner as before. Thus, the argument for the time complexity is essentially the same with the only difference that we are doing two matrix multiplications instead of one for each bucket which only gives an extra constant factor. □

4.3 Related work

AllPairsHammingDistances (APHam) was considered by Min, Kao and Zhu [26]. Inspired by the reduction from Hamming distance to L_1 in [23], they utilized the APL$_1$ algorithm from [17]. This resulted in a $\mathcal{O}(n^{(\omega+3)/2})$ time algorithm when $d = n$. They also utilized rectangular matrix multiplication bounds to provide a tradeoff in the complexity when $d \ll n$. Writing their upper bound in a general form, the complexity is $\mathcal{O}(n^{1+\omega(1,s,1)/2}\sqrt{d})$ where $d = n^s$. Given the improved bounds for rectangular matrix multiplication by Le Gall [21] and subsequently by Le Gall and Urrutia [22], the bounds from [26] are stronger.

Although not stated as such, one algorithm for computing the closest pair of points, in terms of L_∞ norm, in a high dimensional space presented in [17] can be adapted to computing AllPairsThresholdProducts (APThr) in time $\mathcal{O}(n^{(3+\omega)/2})$.

AllPairsL$_2$Distances (APL$_2$) as observed by Indyk et al. [17] reduces to a single matrix multiplication and thus admits a runtime of $\mathcal{O}(n^\omega)$. Similarly, APL$_{2p}$ admits a runtime of $\mathcal{O}(p^2 n^\omega)$. Again similar to what we saw for pattern matching problems, we see that L_2 is an "easy" score function. For every other score function mentioned, all solutions presented use a bucketing or a high/low frequency technique to decompose the problem into ones solvable by matrix multiplication plus a brute force computation. We refer to Tables 4.1 for a summary of AllPairs problems.

Other problems were considered in the literature that uses some of the problems mentioned above as subroutines. **Closest L_∞ Pair** was considered by Indyk et al. [17] where the presented algorithm uses binary search on top of APThr. The total runtime is $\mathcal{O}(n^{(\omega+3)/2} \log D)$, where D is the diameter of the input point set. **All Pairs Bottleneck Paths** in edge-capacitated graphs reduces to $\log n$ iterations of the $(\max, \min)$-MatrixProduct. [30], [32] and [12] used sparse APDom to obtain a sub-cubic time algorithm for the $(\max, \min)$-MatrixProduct. The complexity of the algorithm presented in [12] is $\mathcal{O}(n^{(\omega+3)/2})$.

Summary

In this chapter, we surveyed algorithms for AllPairs problems whose time complexity is close to $\mathcal{O}(n^{(\omega+3)/2})$ when the dimension of the involved vectors is $d = n$. They all relied on the technique of decomposing the problem into several simpler instances that can be solved by fast matrix multiplication together with a brute force part. We also surveyed work done for the cases of vectors with a low dimension $d < n$ and when the input vectors are sparse. Both these setting were surveyed in the context of APDom. On a dif-

Name	Score function		All Pairs problem	Complexity	
Hamming	$\mathbb{1}[x \neq y]^\dagger$	APHAM	$O[i][j] = \lvert\{k : \mathbf{A}_i[k] \neq \mathbf{B}_j[k]\}\rvert$	$\mathcal{O}(n^{(\omega+3)/2})$	[26]
Dominance	$\mathbb{1}[x \leq y]$	APDOM	$O[i][j] = \lvert\{k : \mathbf{A}_i[k] \leq \mathbf{B}_j[k]\}\rvert$	$\mathcal{O}(n^{\rho})^\ddagger$	[34]
Threshold	$\mathbb{1}[\lvert x-y\rvert > \delta]$	APTHR	$O[i][j] = \lvert\{k : \lvert\mathbf{A}_i[k] - \mathbf{B}_j[k]\rvert > \delta\}\rvert$	$\mathcal{O}(n^{(\omega+3)/2})$	[17]
ℓ_1 distance	$\lvert x-y\rvert$	APL_1	$O[i][j] = \sum_{k=1}^{n} \lvert\mathbf{A}_i[k] - \mathbf{B}_j[k]\rvert$	$\mathcal{O}(n^{(\omega+3)/2})$	[17]
ℓ_2 distance	$(x-y)^2$	APL_2	$O[i][j] = \sum_{k=1}^{n} (\mathbf{A}_i[k] - \mathbf{B}_j[k])^2$	$\mathcal{O}(n^{\omega})$	[17]

Table 4.1: Summary of different all-pair problems along with their score functions and runtime.

ferent note, notice that APL_2 is also an exception to those and can be solved much faster similar to the distinction made in pattern matching problems for L_2PM. Refer to Table 4.1 for a summary of the results.

$^\dagger\mathbb{1}[\varphi]$ is 1 iff φ and 0 otherwise.
$^\ddagger\rho \leq 2.6834$ is a solution to $\rho = \omega(1, 4-\rho, 1)$

Chapter 5

Hamming distance completeness

In the previous two chapters, we surveyed different problems in the worlds of generalized pattern matching and all-pairs computation. We have seen that many apparently different problems are using very similar techniques and thus having the same asymptotic time complexity. It is thus a natural question to ask:

Is there a shared source of hardness to those problems?

5.1 Known reductions

A partial answer was given to the above question by Lipsky and Porat [23], where they showed that both HAMPM and LESSTHANPM reduce to L_1PM showing that the latter problem is no easier than the former problems. This was shown by a straightforward reduction between the underlying g functions (refer to Definition 2.2) of the different problems. In [23], the authors gave the following two simple lemmas:

Lemma 5.1 ([23]) *For any pair of integers x, y*

$$|x-y+1| + |x-y-1| - 2|x-y| = \begin{cases} 0 & \text{if} \quad x \neq y \\ 2 & \text{if} \quad x = y \end{cases} .$$

Lemma 5.2 ([23]) *For any pair of integers x, y*

$$|x-y+1| - |x-y| = \begin{cases} +1 & \text{if} \quad x \geq y \\ -1 & \text{if} \quad x < y \end{cases} .$$

They showed that using these equalities between the underlying g functions of the different problems, we can establish linear reductions between the different pattern matching problems. Precisely, using Lemma 5.1, we establish

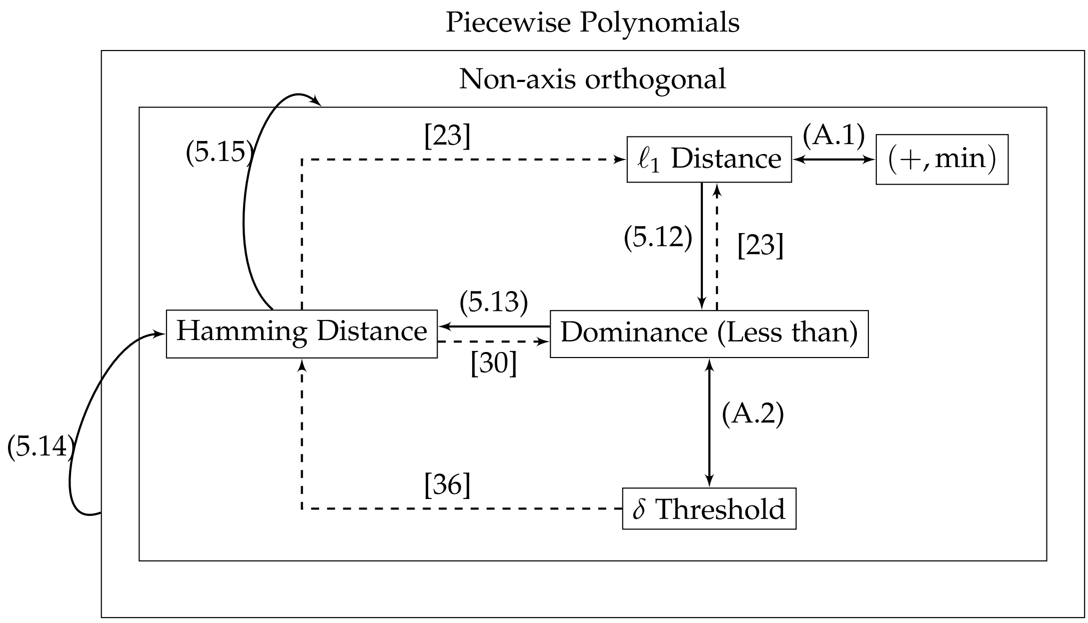


Figure 5.1: Existing and new reductions between problems, together with problem classes.

a reduction from HamPM to L_1PM and using Lemma 5.2, we establish a reduction from LessThanPM to L_1PM.

It has also been noted in [30] that APHam reduces to APDom through the use of the following lemma:

Lemma 5.3 *For any pair of integers x, y*

$$\mathsf{Ham}[x,y] = 2 - \mathsf{Dom}[x,y] - \mathsf{Dom}[y,x].$$

However, the question of whether e.g. HamPM could be substantially easier than L_1PM remained open. The first non-trivial reduction (although not stated as a lower-bound type result) was provided by Zhang and Atallah [36], where they showed that ThrPM with threshold δ reduces to $\mathcal{O}(\log \delta)$ instances of HamPM. See Figure 5.1 for a summary of the known reductions.

It is also worth noting that Lemma 5.1 was also used in [26] to reduce APHam to APL$_1$. This points us to the crucial observation that we can ignore the convolution or matrix-multiplication structure of the problems and just *focus on relations between the underlying binary functions*.

5.2 Reduction preserving linearity

Going into that direction, we give the following definition:

Definition 5.4 *We say that* $\diamond$ *reduces* preserving linearity *to instances of* $\square_1, \ldots, \square_K$, *if there are functions* $f_1, \ldots, f_K$ *and* $g_1, \ldots, g_K$ *and coefficients* $\alpha_1, \ldots, \alpha_K$, *such that for any two integers* x, y:*

$$x \diamond y = \sum_i \alpha_i \cdot \Big(f_i(x) \; \square_i \; g_i(y) \Big). \tag{5.1}$$

A one-to-many reduction from $\diamond$ to $\square$ is also a one-to-many reduction from $(+, \diamond)$ vector product/convolution/matrix multiplication to $(+, \square)$ vector product/convolution/matrix multiplication. Indeed, given (5.1), we have for any vectors $\mathbf{A}, \mathbf{B}$ and matrices $\mathcal{A}, \mathcal{B}$:

$$\begin{aligned} \mathbf{A} \overset{\diamond}{\cdot} \mathbf{B} &= \sum_i \alpha_i \cdot \Big(f_i(\mathbf{A}) \overset{\square_i}{\cdot} g_i(\mathbf{B}) \Big), \\ \mathbf{A} \overset{\diamond}{\odot} \mathbf{B} &= \sum_i \alpha_i \cdot \Big(f_i(\mathbf{A}) \overset{\square_i}{\odot} g_i(\mathbf{B}) \Big), \\ \mathcal{A} \overset{\diamond}{\otimes} \mathcal{B} &= \sum_i \alpha_i \cdot \Big(f_i(\mathcal{A}) \overset{\square_i}{\otimes} g_i(\mathcal{B}) \Big). \end{aligned}$$

where $f(\mathbf{A})$ and $f(\mathcal{A})$ denotes a coordinate-wise application of f to vector $\mathbf{A}$ and matrix $\mathcal{A}$, respectively. This is a generalization to the type of reductions given in [23] for example and helps in simply focusing on reductions between algebraic operators instead of caring about the structure of the encompassing problem whether it is a pattern matching or AllPairs- problem.

5.3 Main result

A natural question is to ask whether there are binary functions $\diamond$ for which the $(+, \diamond)$ product is *significantly harder to compute* than i.e. Hamming distance. We provide a partial negative answer to this question: we show that a broad class of *piecewise polynomial binary functions* is equivalent (up to polylogarithmic factors) in hardness as Hamming distance.

Definition 5.5 *For integers* A, B, C *and polynomial* $P(x, y)$ *we say that the function* $P(x, y) \cdot \mathbb{1}[Ax + By + C > 0]$ *is* halfplane polynomial. *We call a sum of halfplane polynomial functions* $\sum_i P_i(x, y) \cdot \mathbb{1}[A_i x + B_i y + C_i > 0]$ *a* piecewise polynomial. *We say that a function is* axis-orthogonal piecewise polynomial, *if it is piecewise polynomial and its summands satisfy that for every* i, $A_i = 0$ *or* $B_i = 0$.

*For the sake of simplicity, we are omitting in the definition the post-processing function necessary i.e. taking the p-th root for L_p norms.

Definition 5.5 captures a broad class of interesting binary functions on integer inputs, e.g.

$$
\begin{aligned}
\mathsf{Ham}(x,y) &= \mathbb{1}[x > y] + \mathbb{1}[x < y] \\
\max(x,y) &= x \cdot \mathbb{1}[x \geq y] + y \cdot \mathbb{1}[x < y] = x \cdot \mathbb{1}[x+1 > y] + y \cdot \mathbb{1}[x < y] \\
|x-y|^{2p+1} &= (x-y)^{2p+1} \cdot \mathbb{1}[x > y] + (y-x)^{2p+1} \cdot \mathbb{1}[x < y] \\
\mathsf{Thr}_\delta(x,y) &\stackrel{\text{def}}{=} \mathbb{1}[|x-y| \geq \delta] \\
&= \mathbb{1}[x \leq y - \delta] + \mathbb{1}[x \geq y + \delta] = \mathbb{1}[x < y + 1 - \delta] + \mathbb{1}[x + 1 > y + \delta]
\end{aligned}
$$

In this chapter, we prove the following main theorem:

Theorem 5.6 *Let $\diamond$ be a piecewise polynomial of constant degree and* $\text{poly}(\log n)$ *number of summands.*

- *If $\diamond$ is axis orthogonal, then $\diamond$ is "easy": $(+, \diamond)$ convolution takes $\widetilde{O}(n)$ time, $(+, \diamond)$ matrix multiplication takes $\widetilde{O}(n^\omega)$ time.*
- *Otherwise, $\diamond$ is* Hamming distance complete*: under one-to-polylog reductions, on inputs bounded in absolute value by* poly(n)*, $(+, \diamond)$ product is equivalent to Hamming distance, $(+, \diamond)$ convolution is equivalent to* HamPM *and $(+, \diamond)$ matrix multiplication is equivalent to* APHam.

By this theorem and the observation that we need only care about reductions between binary functions themselves, we obtain the following corollaries:

Corollary 5.7 *The following problems are equivalent under one-to-polylog reductions:* HamPM, LessThanPM, L_{2p+1}PM *for a constant integer $p \geq 0$,* ThrPM *and* $(+, \max)$-Convolution.

Corollary 5.8 *The following problems are equivalent under one-to-polylog reductions:* APHam, APDom, APL_{2p+1} *for a constant integer $p \geq 0$,* APThr *and* $(+, \max)$-MatrixProduct.

Observation 5.9 *The reductions in Theorem 5.6 preserve the dimension and potential sparsity in the inputs.*

One can read those results in two different ways: Positively, any improvement made to one problem translates to every other problem: i.e. [34] improved the exponent of APDom to less than $(3+\omega)/2$ and this improvement applies to all other AllPairs- problems considered here. Another example is the relation between the exponents of AllPairs- problems with n vectors each of dimension $d \ll n$. Here, the tradeoff achieved for one problem (i.e. Hamming distances) between d and the exponent (c.f. [26] and [15]) applies by our results to all the other AllPairs- problems considered here. Another example is the sparsity of the input where the tradeoff between the number of relevant entries in the input and the runtime (c.f. [30], [32] and [12]) applies to all of the mentioned problems.

Negatively, there is a shared barrier in a broad class of problems and one is unlikely to improve upon existing upper bounds without some significant breakthrough. For both pattern matching problems and geometric problems we consider here, existing runtimes come from a tradeoff between the number of buckets and the size of these buckets. Without a novel technique, this runtime cannot be improved. Similarly, any lower bound proof for one of the listed problems would immediately apply to every other problem.

Remark 5.10 *Throughout this chapter, we assume that all input values are integers, bounded in absolute value by $M \in \mathrm{poly}(n)$. The same assumption holds for coefficients of the considered binary functions (i.e. see Definition 5.5).*

5.3.1 Auxiliary notation

We need three operations for proving the main theorem. Namely,

Scaling: Observe that for many "natural" functions $\diamond$ and integers x, y, $x \diamond y$ is approximated by $\lfloor x/2 \rfloor \diamond \lfloor y/2 \rfloor$ (up to some fixed multiplicative factor). This allows us to unwind $x \diamond y$ into a weighted sum of $\mathcal{O}(\log(\max(|x|, |y|))$ corrective terms. For example, if for some constant C, integers $x, y \geq 0$ and some corrective function ξ:

$$x \diamond y = C \cdot (\lfloor x/2 \rfloor \diamond \lfloor y/2 \rfloor) + \xi(x, y)$$

then naturally

$$x \diamond y = 0 \diamond 0 + \sum_{i \geq 0} C^i \cdot \xi(\lfloor x/2^i \rfloor, \lfloor y/2^i \rfloor). \tag{5.2}$$

Sparsity: We consider a generalized version of the input with special "ignore" marks $\star$ as possible elements. Those elements of the input never contribute to the final score of the $(+, \diamond)$ product. Formally, we operate on $\mathbb{Z} + \{\star\}$, with special arithmetic rules (unless stated otherwise):

- for any *single argument* function: $f(\star) = \star$,
- for any *double argument* function: $g(\star, \star) = g(\star, y) = g(x, \star) = 0$.[†]

The goal of this formalism is twofold. The first one is to handle sparse inputs formally (i.e. vectors with $\mathcal{O}(n^{1-\varepsilon})$ relevant entries). The second one is that

[†]We have to keep in mind that whether a function has zero, one or two arguments is context dependent: i.e. writing: $\mathbb{1}[x \neq y] = 1 - \mathbb{1}[x = y]$, we have to treat 1 as a function of x and y as well.

such "ignore" marks coupled with filtering (defined below) allow us to split the input based on properties of its values.

We note that these "ignore" marks do not increase the computational complexity of Hamming distance. In the following lemma, we show that one can get rid of $\star$ when i.e. computing Hamming distance. We show this in the pattern matching setting for simplicity. However this can be easily extended for matrix multiplication problems as well.

Lemma 5.11 *Hamming distance in $\mathbb{N} + \{\star\}$ reduces preserving linearity to two instances of Hamming distance in $\mathbb{N}$.*

Proof Let $x, y \in \mathbb{N} + \{\star\}$. To compute $\mathsf{Ham}(x, y)$, we first use a mapping that puts $\star$ into a separate integer, and then apply a correction function that fixes distances between $\star$.

For the first instance:

$$f(t) = \begin{cases} 0 & \text{if } t = \star \\ t+1 & \text{otherwise} \end{cases}$$

As for the second instance:

$$g(t) = \begin{cases} 0 & \text{if } t = \star \\ 1 & \text{otherwise} \end{cases}$$

Observe that $\mathsf{Ham}(x, y) = \mathsf{Ham}(f(x), f(y)) - \mathsf{Ham}(g(x), g(y))$. $\square$

Filtering: We define the following functions:

$$\mathtt{odd}(x) \stackrel{\text{def}}{=} \begin{cases} x & \text{if } x \text{ is even} \\ \star & \text{otherwise} \end{cases} \qquad \mathtt{even}(x) \stackrel{\text{def}}{=} \begin{cases} x & \text{if } x \text{ is odd} \\ \star & \text{otherwise} \end{cases}$$

Those functions, when applied to a vector or a matrix, allow us to filter values according to parity, i.e. for $\mathbf{A} = [1, 2, 3, 4]$ one gets $\mathtt{even}(\mathbf{A}) = [\star, 2, \star, 4]$.

5.4 Warm-up reduction

We start by showing a reduction from L_1 distance to $\mathcal{O}(\log^2 n)$ instances of Hamming distance. This is a reduction that is fully contained in our general reduction, that is Theorem 5.6. However, since it uses simpler but similar techniques, and already has a nontrivial consequence (i.e. collapsing hardness of $\mathrm{L_1PM}$ and HamPM), we present it separately.

We now give two reductions that illustrate the usefulness of these techniques. Both reductions are illustrated in the Appendix A.2 (Figures A.1 and A.2).

Theorem 5.12 *The L_1 distance reduces to $\mathcal{O}(\log n)$ instances of dominance.*

Proof Since L_1 distance is shift-invariant, i.e. $|(x+\Delta)-(y+\Delta)| = |x-y|$ for any Δ, we can assume that $0 \leq x, y < M$ for some $M = \mathrm{poly}(n)$. Observe that for $x, y \geq 0$:

$$|x-y| = 2 \cdot \Big| \lfloor x/2 \rfloor - \lfloor y/2 \rfloor \Big| + \eta(x,y),$$

where, denoting $\mathsf{Dom}(x,y) \stackrel{\text{def}}{=} \mathbb{1}[x \leq y]$,

$$\begin{aligned}
\eta(x,y) &= \mathbb{1}[(x \text{ is odd}) \wedge (y \text{ is even}) \wedge (x \geq y)] - \mathbb{1}[(x \text{ is even}) \wedge (y \text{ is odd}) \wedge (x \geq y)] \\
&\quad + \mathbb{1}[(y \text{ is odd}) \wedge (x \text{ is even}) \wedge (y \geq x)] - \mathbb{1}[(y \text{ is even}) \wedge (x \text{ is odd}) \wedge (y \geq x)] \\
&= \mathsf{Dom}(\mathtt{odd}(-x), \mathtt{even}(-y)) - \mathsf{Dom}(\mathtt{even}(-x), \mathtt{odd}(-y)) \\
&\quad + \mathsf{Dom}(\mathtt{even}(x), \mathtt{odd}(y)) - \mathsf{Dom}(\mathtt{odd}(x), \mathtt{even}(y)).
\end{aligned}$$

By unwinding as in (5.2), we get $|x-y| = \sum_{i=0}^{\log M} 2^i \cdot \eta(\lfloor x/2^i \rfloor, \lfloor y/2^i \rfloor)$ which completes the reduction. □

Theorem 5.13 *Dominance reduces to $\mathcal{O}(\log n)$ instances of Hamming distance and multiplication.*

Proof Since dominance is shift-invariant, w.l.o.g. we assume that $0 \leq x, y < M$ for some $M = \mathrm{poly}(n)$.

Observe the following recurrence relation, for $x, y \geq 0$:

$$\begin{aligned}
\mathsf{Dom}(x,y) &= \mathsf{Dom}(\lfloor x/2 \rfloor, \lfloor y/2 \rfloor) - \mathbb{1}[(x \text{ is odd}) \wedge (x = y+1)] \\
&= \mathsf{Dom}(\lfloor x/2 \rfloor, \lfloor y/2 \rfloor) - \mathbb{1}[x \text{ is odd}] + \mathbb{1}[x \text{ is odd}] \cdot \mathsf{Ham}(x, y+1)
\end{aligned}$$

By unwinding as in (5.2), we get:

$$\begin{aligned}
\mathsf{Dom}(x,y) = 1 &- \sum_{i=0}^{\log M} \mathbb{1}\Big[\lfloor x/2^i \rfloor \text{ is odd}\Big] \\
&+ \sum_{i=0}^{\log M} \mathbb{1}\Big[\lfloor x/2^i \rfloor \text{ is odd}\Big] \cdot \mathsf{Ham}(\lfloor x/2^i \rfloor, \lfloor y/2^i \rfloor + 1).
\end{aligned}$$

Using filtering notation, this becomes

$$\begin{aligned}
\mathsf{Dom}(x,y) \quad &= \quad 1 - \sum_{i=0}^{\log M} \mathbb{1}\Big[\lfloor x/2^i \rfloor \text{ is odd}\Big] && \text{(*)} \\
&+ \quad \sum_{i=0}^{\log M} \mathsf{Ham}(\mathtt{odd}(\lfloor x/2^i \rfloor), \lfloor y/2^i \rfloor + 1) && \text{(**)}
\end{aligned}$$

Now observe, that (*) is purely a function of x. If x is guaranteed to be an integer, then evaluating it as part of an operator (i.e. inside convolution or matrix-multiplication) is trivial. As y is never mapped to $\star$ in (**), treating (*) as a single argument function suffices.

The second term (**) uses our filtering function and the convention that Ham evaluates to 0 if at least one of its inputs is $\star$. Thus (**) is a sum of $\mathcal{O}(\log n)$ Hamming distances on inputs from $\mathbb{Z} \cup \{\star\}$. By Lemma 5.11, each of those reduces to two instances of Hamming distance on inputs from $\mathbb{Z}$. $\square$

Remark: In general, we would have to take into account that both $x, y \in \mathbb{Z} \cup \{\star\}$. Thus, we would have to treat term (*) as a function of *both* x and y, that is evaluating to 0 if $x = \star$ or $y = \star$. In general, (*) reduces to evaluating, after the reduction step, some polynomial $Q(x', y') = f(x')$ (where y' might be $\star$) with $f(x') = 1 - \sum_{i=0}^{\log M} \mathbb{1}\left[\lfloor x'/2^i \rfloor \text{ is odd}\right]$. By Lemma 5.17, $f(x')$ can be done in the time of a regular convolution or matrix multiplication and thus the computation time for (*) is dominated by (**), that is HamPM and APHam, respectively.

5.5 Proof of Theorem 5.6

The goal of this section is to prove Theorem 5.6. We achieve this by showing two separate reductions, one from *all piecewise polynomial functions* to Hamming distance and one from Hamming distance to all non axis-orthogonal piecewise polynomials.

Our main technical contributions are the following results:

Theorem 5.14 *If $\diamond$ is a piecewise polynomial of degree d with c summands then it reduces to $\mathcal{O}(c \cdot d^2 \cdot \log^{d+1} n)$ instances of Hamming distance. The reduction works even if we allow "don't care" symbols.*

Theorem 5.15 *If $\diamond$ is a piecewise polynomial of degree d but is not an axis-orthogonal piecewise polynomial, then Hamming distance reduces to $\mathcal{O}(d^2)$ instances of $\diamond$ and multiplication.*

Proof of Theorem 5.14

The proof goes as follows. We consider every summand separately. We show that summands with "simple" conditions (that filter only one argument, i.e. x or y) are no harder than simple multiplication. Every other summand with conditional term $\mathbb{1}[A_i x + B_i y + C_i > 0]$ reduces under linear transformations of its arguments to $\mathbb{1}[x < y]$. It is thus enough to consider terms of the form $x^a y^b \mathbb{1}[x < y]$. We decompose such terms recursively into a sum of: terms with smaller values ($x/2, y/2$ instead of x, y), terms of smaller degree, and terms with a conditional term of a simpler form (i.e. $\mathbb{1}[x = y]$).

Exhaustively applying this decomposition leaves us with a polylog number of terms of the form $w(x) \cdot \mathbb{1}[x = y]$, for which we now describe how to deal with them (those decompose into a logarithmic number of regular Hamming distances).

Lemma 5.16 *For an integer weight function w, the character weighted matches, that is $w(x) \cdot \mathbb{1}[x = y]$, reduce to $\mathcal{O}(\log n)$ instances of Hamming distance and multiplication.*

Proof Let W be the upper bound on all values of w in the considered domain of inputs. Given two integers x, y, we observe the following equality:

$$w(x) \cdot [x = y] \quad = \quad \sum_{i=0}^{\log W} 2^i \cdot \mathbb{1}[w_i(x) = w_i(y)]$$

where the filtering function w_i is defined based on w:

$$w_i(x) = \begin{cases} x & i\text{-th bit of } w(x) \text{ is } 1 \\ \star & \text{otherwise.} \end{cases}$$

Observing that $\mathbb{1}[x = y] = 1 - \mathsf{Ham}(x, y)$ finishes the proof. □

Lemma 5.17 *An axis-orthogonal piecewise polynomial $\diamond$ of c summands of degree d reduces to $\mathcal{O}(d^2 c)$ multiplications.*

Lemma 5.18 *Given integers $a, b \geq 0$, the binary function $x^a y^b \cdot \mathbb{1}[x < y]$ reduces to $\mathcal{O}(\log^{a+b+1} n)$ instances of Hamming distance and multiplication.*

Proof Denote $\mathrm{MDom}_{a,b}(x, y) = x^a y^b \cdot \mathbb{1}[x < y]$, $\mathrm{MEq}_a(x, y) = x^a \cdot \mathbb{1}[x = y]$. First, we argue that w.l.o.g. $x, y \geq 0$. Indeed, observe that $\mathrm{MDom}_{a,b}(x - \Delta, y - \Delta) = (x - \Delta)^a (y - \Delta)^b \cdot \mathbb{1}[x < y]$, thus for large enough Δ, the computation of $\mathrm{MDom}_{a,b}$ on inputs of arbitrary sign reduces to $\mathcal{O}(ab)$ instances of MDom on non-negative inputs. Thus we assume that $0 \leq x, y \leq M$ for some $M = \mathrm{poly}(n)$.

We proceed with the following decomposition, where $u = \lfloor \frac{x}{2} \rfloor$ and $v = \lfloor \frac{y}{2} \rfloor$.

$$\begin{aligned} \mathrm{MDom}_{a,b}(x, y) \quad &= \quad (2u)^a (2v)^b \cdot \mathbb{1}[u < v] && (*) \\ &+ \quad (2u)^a (2v)^b \cdot \Big(\mathbb{1}[x < y] - \mathbb{1}[u < v] \Big) && (**) \\ &+ \quad \Big(x^a y^b - (2u)^a (2v)^b \Big) \cdot \mathbb{1}[x < y] && (***) \end{aligned}$$

Simplifying those terms separately, we have

$$\begin{aligned}
(*) &= 2^{a+b} \cdot \mathrm{MDom}_{a,b}(u,v),\\
(**) &= (2u)^a (2v)^b \cdot \mathbb{1}[\mathtt{even}(x) = \mathtt{odd}(y) - 1]\\
&= x^a (y-1)^b \cdot \mathbb{1}[\mathtt{even}(x) = \mathtt{odd}(y) - 1]\\
&= \mathrm{MEq}_{a+b}(\mathtt{even}(x), \mathtt{odd}(y) - 1),\\
(***) &= P_{a,b}(x,y) \cdot \mathbb{1}[\mathtt{odd}(x) < \mathtt{even}(y)]\\
&+ Q_{a,b}(x,y) \cdot \mathbb{1}[\mathtt{even}(x) < \mathtt{odd}(y)]\\
&+ R_{a,b}(x,y) \cdot \mathbb{1}[\mathtt{odd}(x) < \mathtt{odd}(y)]\,,
\end{aligned}$$

where

$$\begin{aligned}
P_{a,b}(x,y) &= (x^a y^b - (x-1)^a y^b)\\
Q_{a,b}(x,y) &= (x^a y^b - x^a (y-1)^b)\\
R_{a,b}(x,y) &= (x^a y^b - (x-1)^a (y-1)^b).
\end{aligned}$$

All in all, our recursion decomposes $\mathrm{MDom}_{a,b}(x,y)$ into several terms – either with the inputs reduced by a factor of 2, the test for dominance replaced with a test for equality, or to monomials of smaller degree (observe that each of $P_{a,b}(x,y)$, $Q_{a,b}(x,y)$ and $R_{a,b}(x,y)$ is of degree at most $a+b-1$). Let $T(a,b,m)$ denote the number of instances of Hamming distance that a single instance of $\mathrm{MDom}_{a,b}$, with inputs bounded in value by 2^m, is reduced to. Since by Lemma 5.16, MEq_{a+b} reduces to $\mathcal{O}(m \cdot (a+b))$ instances of Hamming distance, there is

$$T(a,b,m) \leq \mathcal{O}(m \cdot (a+b)) + T(a,b,m-1) + \sum_{\substack{0 \leq i \leq a \\ 0 \leq j \leq b \\ (i,j) \neq (a,b)}} 3T(i,j,m),$$

which is satisfied (for some constant C) by

$$T(a,b,m) \leq C \cdot m \cdot (a+b) \cdot \binom{a+b+m}{a,b,m} \cdot 4^a \cdot 4^b.$$

For fixed values a,b: $T(a,b,\log M) = \mathcal{O}(\log^{a+b+1} M)$. □

Consider an arbitrary piecewise-polynomial binary function $\diamond$. Consider its summand $P(x,y) \cdot \mathbb{1}[Ax + By + C > 0]$. If $A = 0$ or $B = 0$ then this reduces to a binary function of degenerate form, i.e. $P(x,y) \cdot \mathbb{1}[Ax + C > 0]$ which in turn reduces to $\mathcal{O}(d^2)$ multiplications by Lemma 5.17.

Otherwise, if $A \neq 0$ and $B \neq 0$, then there is a one-to-one linear input reduction, $u = -Ax$ and $v = By + C$, that reduces from $(-Ax)^i (By+C)^j \cdot \mathbb{1}[Ax + By + C > 0]$ to $u^i v^j \cdot \mathbb{1}[u < v]$. Note that any polynomial of degree a

and b over x and y is a linear combination of $(-Ax)^i(By+C)^j$ for $0 \leq i \leq a$ and $0 \leq j \leq b$.

By applying those reductions to each summand and applying Theorem 5.18 to each monomial of the summand, we reach the claimed bound.

Proof of Theorem 5.15

In order to prove Theorem 5.15, we will need the following lemma.

Lemma 5.19 *Consider a family of distinct lines $L = \{\lambda_i\}_{i=1}^{|L|}$, $\lambda_i = \{x, y : A_i x + B_i y + C_i = 0\}$ for integers A_i, B_i, C_i such that $|A_i|, |B_i|, |C_i| \leq M$. If there is at least one $\lambda \in L$ that is not axis-orthogonal, then there exists $\lambda_i \in L$ and $\alpha, \beta, \gamma, \delta$ such that:*

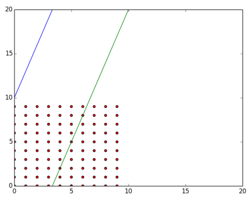
- *for any line λ_j that is not parallel to λ_i, the set $\{(\alpha x + \gamma, \beta y + \delta) : x, y \in [0 \ldots N]\}$ lies* on the same side *of λ_j.* **Condition (1)**
- *for any line λ_j that is parallel to λ_i, the sets $\{(\alpha x + \gamma, \beta y + \delta) : x > y\}$ and $\{(\alpha x + \gamma, \beta y + \delta) : x < y\}$ are separated by λ_j.* **Condition (2)**

Moreover, $|\alpha|, |\beta|, |\gamma|, |\delta| \leq \mathrm{poly}(M, N)$.

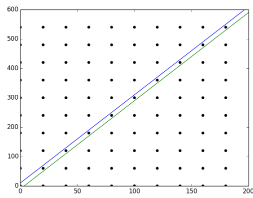
The idea of the proof is that we start off with a grid of points $\mathcal{G} = [N] \times [N]$ (where we denote by $[N]$ the set $\{0, 1, \ldots, N\}$) which is all the possible combinations of x and y values. We perform intermediate transformations to that grid of points, namely scaling and translation. At the conclusion of such transformations, the new grid satisfies the desired conditions.

Proof Without loss of generality, we assume that λ_1 has a positive slope. If for all i, slope of λ_i is negative, we can simply negate either x or y and work with this transformed input. We now carry out the following transformations:

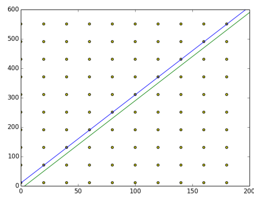
- Let $\Lambda = \{\lambda_i : A_i \cdot B_1 = A_1 \cdot B_i\}$ be the set of all lines parallel to λ_1. Due to the assumption that $A_i, B_i, C_i \in [0, \ldots, M]$, the absolute value of the maximum and minimum y-intercepts of all $\lambda_i \in \Lambda$ is M. Thus, in order to make sure that all lines in Λ separate, we scale x and y by $3M$. i.e. $\mathcal{G}' = ([N] \cdot 3M) \times ([N] \cdot 3M)$
- We scale the inputs according to the slope of λ_1. This means that we further scale x by B_1 and y by $-A_1$ (without loss of generality, we assume that $B_1 > 0$ and $A_1 < 0$). i.e. $\mathcal{G}'' = ([N] \cdot 3M \cdot B_1) \times ([N] \cdot 3M \cdot -A_1)$
- We translate the grid in the y direction such that all lines in Λ are separating the transformed grid according to the desired criteria. Let $s = \max_{\lambda_i \in \Lambda} -C_i/B_i$ be the maximum y-intercept of lines in Λ. Shifting the transformed grid by $\lceil s \rceil$ satisfies the criteria for parallel lines.



(a) Grid $\mathcal{G}$



(b) Grid $\mathcal{G}''$



(c) Grid $\mathcal{G}'''$

Figure 5.2: We have two parallel lines. We show the grids $\mathcal{G}$, $\mathcal{G}''$ and $\mathcal{G}'''$after the described scaling and translation operations.

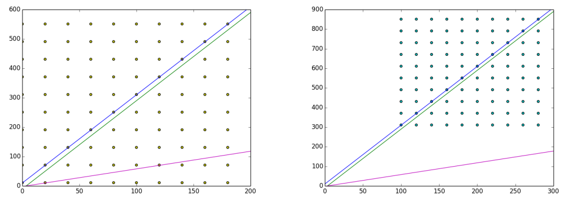
i.e. $\mathcal{G}''' = ([N] \cdot 3M \cdot B_1) \times ([N] \cdot 3M \ \cdot -A_1 + \lceil s \rceil)$. Refer to Figure 17 for a visualization of the applied transformations.

- What remains to do is to translate the transformed grid in the positive direction along the slope of lines in Λ (By moving along a slope m, we mean that if we translate a point by b in the x-direction, then we also need to translate the point by mb in the y-direction). Note that this translation will preserve the criteria for parallel lines (**Condition (1)**).

  However, if we go far enough along the slope, the grid will eventually lie on the same side of every line $\lambda_i \notin \Lambda$. How far we need to translate $\mathcal{G}'''$ depends on how close other lines are in terms of slope to lines in Λ. Thus, let us focus on how far we need to translate the x values. We denote by m the slope of lines in Λ. The maximum possible translation necessary in the x direction will be due to a line $\lambda_j \in \Lambda$ where $\lambda_j = \{(x,y) : y = mx - M\}$ and $\lambda_k \notin \Lambda$ where $\lambda_k = \{(x,y) : y = (m + 1/M)x + M\}$ (This is based on the assumption that the y- intercept can not exceed M in absolute value and the smallest possible difference in slope between any two lines is $1/M$). Notice that in our current grid $\mathcal{G}'''$, we scaled the x and y by at most $\mathcal{O}(M^2)$ meaning that $\mathcal{G}'''$ occupies a space of at most $\mathcal{O}(N \cdot M^2) \times \mathcal{O}(N \cdot M^2)$. Thus, we need to translate the grid up to the point that for an arbitrary y', the distance between points (x_j, y') and (x_k, y') lying on lines λ_j and λ_k respectively, is at least $N \cdot M^2$. Thus solving for

$$mx - M = (m - \frac{1}{M})(x + \Omega(N \cdot M^2)) + M$$

  gives $x = 2M^2 + m\Omega(N \cdot M^3) - \Omega(N \cdot M^2) = \Omega(N \cdot M^4)$. Thus, we need to translate $\mathcal{G}'''$ by at least $\Omega(N \cdot M^4)$ in the x-direction and $m\Omega(N \cdot M^4) = \Omega(N \cdot M^5)$ in the y-direction. Refer to Figure 17 for a visualization of the applied transformation.



(a) Grid $\mathcal{G}'''$ (b) Final Grid

Figure 5.3: Now we introduce a line not parallel to the previous two lines. We show the grids $\mathcal{G}'''$ and the final translation operation described to satisfy **Condition (2)**.

From all the above transformations, we can see that choosing the values

$$\begin{aligned} \alpha &= 3MB_1 \\ \gamma &= \Omega(N \cdot M^4) \\ \beta &= -3MA_1 \\ \delta &= \lceil s \rceil + m\gamma \end{aligned}$$

satisfies the desired properties and they are all bounded by $poly(M, N)$. $\square$

Now, with the help of the above lemma, we can proceed to proving Theorem 5.15

Proof (Theorem 5.15) Let us take the binary function $x \diamond y = \sum_i P_i(x, y) \cdot \mathbb{1}[A_i x + B_i y + C_i > 0]$ as in the theorem statement, assuming it is of the simplest form (no redundant terms and minimal number of summands possible). We construct a reduction from Hamming distance to $\diamond$ by a series of intermediate operators. We use Lemma 5.19 to transform our inputs x, y into x', y' such that the binary function $x' \diamond y'$ is of a simpler form. Applying finite discrete differentiation (which is described later) allows us to represent the "Dom" function as a summation of different applications of the binary operator $\diamond$.

Let d be the highest degree of any $P_1, P_2, \ldots$. Consider all the lines being borders of regions, that is $\lambda_i = \{(u, v) : A_i u + B_i v + C = 0\}$ (as elements of the continuous Euclidean plane).

We now apply Lemma 5.19, with $N = 3dM + 2d$.

Consider $F(x, y) \stackrel{\text{def}}{=} (\alpha x + \gamma) \diamond (\beta y + \delta)$. Limited to $x, y \in [0 \ldots N]$, $F(x, y)$ is piecewise linear of a much simpler form where the conditional function is based on the comparison operators $<, >, =$ applied on the inputs x, y:

$$F(x, y) = Q_>(x, y) \cdot \mathbb{1}[x > y] + Q_=(x, y) \cdot \mathbb{1}[x = y] + Q_<(x, y) \cdot \mathbb{1}[x < y]$$

for $Q_>, Q_=, Q_<$ being polynomials of degree at most d, and $Q_< \not\equiv Q_>$. Let D_x, D_y be the operators of discrete differentiation, that is

$$D_x F(x,y) \stackrel{\text{def}}{=} F(x+1,y) - F(x,y)$$

$$D_y F(x,y) \stackrel{\text{def}}{=} F(x,y+1) - F(x,y)$$

Since $Q_< \not\equiv Q_>$, then there are integers $0 \le a, b \le d$ such that $D_x^a D_y^b (Q_<(x,y) - Q_>(x,y)) \equiv c$ for some constant $c \neq 0$. Thus if we consider the function:

$$G(x,y) \stackrel{\text{def}}{=} \frac{1}{c} \cdot D_x^a D_y^b (F(x,y) - Q_>(x,y)) \tag{5.3}$$

$$= \frac{1}{c} \cdot D_x^a D_y^b (\underbrace{(Q_=(x,y) - Q_>(x,y))\mathbb{1}[x=y]}_{\text{Part 1}} + \underbrace{(Q_<(x,y) - Q_>(x,y))\mathbb{1}[x<y]}_{\text{Part 2}}) \tag{5.4}$$

Refer to A.3 for an illustration of the discrete differentiation operation. We notice that the function $G(x,y)$ has the following properties on $x, y \in [0 \ldots N - d]$: for $y - x > d$: $G(x,y) = 1$, and for $y - x < -d$: $G(x,y) = 0$. We observe that for $x, y \in [0 \ldots M]$, there is

$$\mathsf{Dom}(x,y) = G(3d \cdot x, 3d \cdot y + d).$$

All in all, Ham reduces to $\mathcal{O}(d^2)$ instances of $\diamond$ and a single evaluation of a fixed polynomial $Q_>(x,y)$, which reduces to $\mathcal{O}(d^2)$ multiplications. $\square$

Summary

In this chapter, we have shown the equivalence (up to polylogarithmic factors) of a broad class of binary functions. We have shown that all non-axis orthogonal piecewise polynomial functions are equivalent to Hamming distance. We have shown this by proving a reduction in both directions. This resolves the question about whether there is a shared source of hardness in all the problems surveyed in the previous chapters. This means that improving upon the exponent of one pattern matching or one AllPairs- problem will lead to an improvement in all the remaining related problems. This also helps us focus on improving the time complexity of one problem instead of working on improving several other problems simultaneously, which is similar to the advantages of the theory of NP-completeness [14] justifying the title of the chapter.

Chapter 6

Sparse Matrix Multiplication

In this chapter, we leverage the power of Hamming distance completeness discussed in the previous chapter. We do this by linking the complexity of APHam to another natural problem which is the multiplication of two sparse matrices. This in turn links the complexity of all other AllPairs-problems to that of sparse matrix multiplication. We state our contribution in the following theorem.

Theorem 6.1 *The time complexity of* APHam *on n vectors of dimension d is (under randomized Las Vegas reductions) within* $\mathrm{poly}(\log n)$ *from* $\mathsf{Sparse}(n, \min(d^2, nd), n; nd; nd)$ *where we denote* $\mathsf{Sparse}(a, b, c; m_1, m_2)$ *as the time complexity of the optimal algorithm for multiplying sparse matrices $a \times b$ and $b \times c$, with m_1 and m_2 nonzero entries respectively.*

6.1 Naive sparse matrix multiplication

There exists a straight forward naive algorithm for sparse matrix multiplication.

Lemma 6.2 (Naive Sparse Matrix Multiplication) *Sparse matrix multiplication of two matrices $\mathcal{A}$, $\mathcal{B}$ of size $n \times n$ each containing m non-zero elements can be computed in time $\mathcal{O}(m \cdot n)$.*

Proof We notice that the product $\mathcal{C} = \mathcal{A}\mathcal{B}$ can be computed as a sum of n vector outer products. Precisely, $\mathcal{C} = \sum_{i=0}^{n-1} \mathcal{A}_{*i}\mathcal{B}_{i*}$ where $\mathcal{A}_{*i}$ is the i-th column of $\mathcal{A}$ and $\mathcal{B}_{i*}$ is the i-th row of $\mathcal{B}$. Let $|\cdot|$ denote the number of nonzero entries in a matrix/vector/set. Considering that the matrices are sparse with m non-zero entries in each matrix, the naive algorithm requires $\sum_{i=0}^{n-1} |\mathcal{A}_{*i}| \cdot |\mathcal{B}_{i*}|$ multiplication operations. This gives an upper bound on the number of multiplication operations equal to $\sum_{i=0}^{n-1} |\mathcal{A}_{*i}| \cdot |\mathcal{B}_{i*}| = (\sum_{i=0}^{n-1} |\mathcal{A}_{*i}|) \cdot n = m \cdot n$. This is in fact tight. An instance that takes this exact time is where the m non-zero entries of $\mathcal{A}$ are present in the first m/n columns of $\mathcal{A}$ and the m non-zero entries of $\mathcal{B}$ are present in the first m/n rows of $\mathcal{B}$. Thus the number of multiplication operations required is equal to $(m/n) \cdot n^2 = m \cdot n$. $\square$

6.2 Fast sparse matrix multiplication

In [35], Yuster and Zwick presented a simple algorithm, combining rectangular matrix multiplication and a brute force technique to present an analysis of $\mathsf{Sparse}(n, n, n; m_1, m_2)$. They gave the following algorithm:

Algorithm 1 Fast sparse matrix multiplication $FSMM(\mathcal{A}, \mathcal{B})$

INPUT: matrices $\mathcal{A}, \mathcal{B}$ of dimension $n \times n$
OUTPUT: matrix $\mathcal{C} = \mathcal{A}\mathcal{B}$ of dimension $n \times n$

1: $\pi \leftarrow$ a permutation where $|\mathcal{A}_{*\pi(1)}| \cdot |\mathcal{B}_{\pi(1)*}| \geq |\mathcal{A}_{*\pi(2)}| \cdot |\mathcal{B}_{\pi(2)*}| \geq \cdots \geq |\mathcal{A}_{*\pi(n)}| \cdot |\mathcal{B}_{\pi(n)*}|$
2: $\ell \leftarrow \min_{0 \leq r \leq n} M(n, r, n) + \sum_{k=r+1}^{n} |\mathcal{A}_{*\pi(k)}| \cdot |\mathcal{B}_{\pi(k)*}|$ ▷ $M(n, r, n)$ denotes the time complexity of multiplying two matrices of dimensions $n \times r$ and $r \times n$
3: $I \leftarrow \{\pi(1), \ldots, \pi(\ell)\}$
4: $J \leftarrow \{\pi(\ell+1), \ldots, \pi(n)\}$
5: $\mathcal{C}_1 \leftarrow \mathcal{A}_{*I}\mathcal{B}_{I*}$ ▷ $\mathcal{A}_{*I}$ is the sub matrix composed of columns of $\mathcal{A}$ whose indices are in I. $\mathcal{B}_{I*}$ is the sub matrix composed of rows of $\mathcal{B}$ whose indices are in I
6: $\mathcal{C}_2 \leftarrow \mathcal{A}_{*J}\mathcal{B}_{J*}$ ▷ using the same approach as the naive sparse matrix multiplication algorithm. Refer to Lemma 6.2.
7: $\mathcal{C} \leftarrow \mathcal{C}_1 + \mathcal{C}_2$

The correctness of the algorithm is direct since the algorithm simply decomposes the matrices $\mathcal{A}, \mathcal{B}$ into two non-overlapping pairs of matrices. Elements of each pair of matrices only interact with each other. Thus, we can add the multiplication result of each pair together to obtain the final result.

Before stating their time complexity analysis of the above algorithm, we restate a Lemma mentioned before (Lemma 4.3) by Huang and Pan in a simpler fashion.

Lemma 6.3 ([16]) $\omega(1,r,1) \leq \begin{cases} 2 & \text{if } 0 \leq r \leq \alpha \\ 2+\beta(r-\alpha) & \text{if } \alpha < r \leq 1 \end{cases}$
where $\beta = \frac{\omega-2}{1-\alpha}$ and α is defined as in Lemma 4.3.
Thus, denoting by $M(n,\ell,n)$ the time complexity of multiplying two matrices of dimensions $n \times \ell$ and $\ell \times n$,

$$M(n,\ell,n) \leq n^{2-\alpha\beta} \cdot \ell^{\beta}$$

Yuster and Zwick [35] gave the following theorem that consequently gives an upper bound on $\mathsf{Sparse}(n,n,n;m_1,m_2)$. Refer to the paper for the proof.

Theorem 6.4 ([35]) *The time complexity of the algorithm $FSMM(\mathcal{A},\mathcal{B})$ multiplying two $n \times n$ matrices $\mathcal{A},\mathcal{B}$ with m_1,m_2 non-zero entries respectively is bounded by*

$$\mathcal{O}\left(\min\{(m_1m_2)^{\frac{\beta}{\beta+1}} n^{\frac{2-\alpha\beta}{\beta+1}} + n^{2+o(1)}, m_1n, m_2n, n^{\omega}\}\right)$$

For proving the above theorem, they gave the following simple lemma that we will use later on (Lemma 3.2 in [35]).

Lemma 6.5 ([35]) *For any $1 \leq \ell < n$ chosen in Algorithm 1, we have*

$$\sum_{k>\ell} |\mathcal{A}_{*\pi(k)}| \cdot |\mathcal{B}_{\pi(k)*}| \leq \frac{m_1m_2}{\ell}$$

6.3 APHam and sparse matrix multiplication

The interesting thing about the algorithm $FSMM(\mathcal{A},\mathcal{B})$ is that it is very similar in spirit to the algorithms presented before for APDom (see Theorem 4.1) and APL_1 (see Theorem 4.6). The algorithm divides the problem into two parts, one is solved algebraically through the use of algebraic matrix multiplication and another solved in a brute force manner. This leads us to question whether there is any connection between AllPairs- problems and sparse matrix multiplication. We answer this positively by giving this following theorem.

Theorem 6.6 *The time complexity of* APHam *on n vectors of dimension d is (under randomized Las Vegas reductions) within* $\text{poly}(\log n)$ *from* $\mathsf{Sparse}(n, \min(d^2, nd), n; nd; nd)$.

Thus, taking advantage of the completeness of APHam among all considered AllPairs- problems, we learn that the complexity of all those problems is closely linked to the very natural problem of multiplying of two rectangular, very sparse matrices.

We start with the following reduction, which we believe to be a folklore result. Here, by 0/1 matrices we mean matrices with integer entries being either 0 or 1 (but all arithmetic is performed in the ring $\mathbb{Z}$).

Lemma 6.7 (Folklore) *Multiplication of (sparse) integer matrices has the same complexity as multiplication of (sparse) 0/1 matrices (up to* $\mathrm{poly}(\log n)$ *factors).*

Proof Consider the multiplication of two integer matrices with nonnegative entries $\mathcal{A} \times \mathcal{B}$, bounded in value by M. For integer k we define $\mathrm{bit}_k(x)$ to be the value of k-th bit of x. Denote $\mathcal{A}_k = \mathrm{bit}_k(\mathcal{A})$ to be the 0/1 matrix selecting k-th bit of $\mathcal{A}$ entries, and $\mathcal{B}_k = \mathrm{bit}_k(\mathcal{B})$. Consider

$$\mathcal{A}' = \begin{bmatrix} \mathcal{A}_0 \\ \vdots \\ \mathcal{A}_{\log M} \end{bmatrix} \in \mathbb{Z}^{(n \log M) \times n} \text{ and } \mathcal{B}' = \begin{bmatrix} \mathcal{B}_0 & \cdots & \mathcal{B}_{\log M} \end{bmatrix} \in \mathbb{Z}^{n \times (n \log M)}$$

$$\mathcal{A}' \times \mathcal{B}' = \begin{bmatrix} \mathcal{A}_0 \times \mathcal{B}_0 & \cdots & \mathcal{A}_0 \times \mathcal{B}_{\log M} \\ \vdots & \ddots & \vdots \\ \mathcal{A}_{\log M} \times \mathcal{B}_0 & \cdots & \mathcal{A}_{\log M} \times \mathcal{B}_{\log M} \end{bmatrix} \in \mathbb{Z}^{(n \log M) \times (n \log M)}.$$

Since $\mathcal{A} = \sum_i 2^i \mathcal{A}_i$ and $\mathcal{B} = \sum_i 2^i \mathcal{B}_i$, there is $\mathcal{A} \times \mathcal{B} = \sum_i \sum_j 2^{i+j} \mathcal{A}_i \times \mathcal{B}_j$, meaning that $\mathcal{A} \times \mathcal{B}$ follows from the product of two 0/1 matrices of dimensions that are larger by a factor of $\mathcal{O}(\log n)$. To get rid of the nonnegativity assumption, we can represent any integer matrices $\mathcal{A}, \mathcal{B}$ as $\mathcal{A} = \mathcal{A}_1 - \mathcal{A}_2$ and $\mathcal{B} = \mathcal{B}_1 - \mathcal{B}_2$ where $\mathcal{A}_1, \mathcal{A}_2, \mathcal{B}_1, \mathcal{B}_2$ are nonnegative, and consider the product $\begin{bmatrix} \mathcal{A}_1 \\ \mathcal{A}_2 \end{bmatrix} \times \begin{bmatrix} \mathcal{B}_1 & \mathcal{B}_2 \end{bmatrix}$. This introduces a constant overhead of 2 to the analysis of the size of the matrices. $\square$

6.3.1 Reducing APHam to sparse matrix multiplication

An easy reduction shows that APHam reduces to the multiplication of sparse 0/1 rectangular matrices. This follows in spirit the ideas used in [34] (where it was done for dominance), but instead of packing only the "dense" part of the computation into a matrix multiplication problem, we put it all. K

Theorem 6.8 APHam *on vectors of dimension d reduces deterministically to* Sparse$(n, N, n; M; M)$ *for some $M, N = \mathcal{O}(nd)$. Furthermore, for $d < n$, N can be as small as $\mathcal{O}(d^2)$.*

Proof Let $\mathcal{U} = \begin{bmatrix} u_1 & \cdots & u_n \end{bmatrix}^T$ and $\mathcal{V} = \begin{bmatrix} v_1 & \cdots & v_n \end{bmatrix}^T$ for $u_1, \ldots, u_n, v_1, \ldots,$ $v_n \in \mathbb{Z}^d$. W.l.o.g. we assume that entries of those vectors are actually from

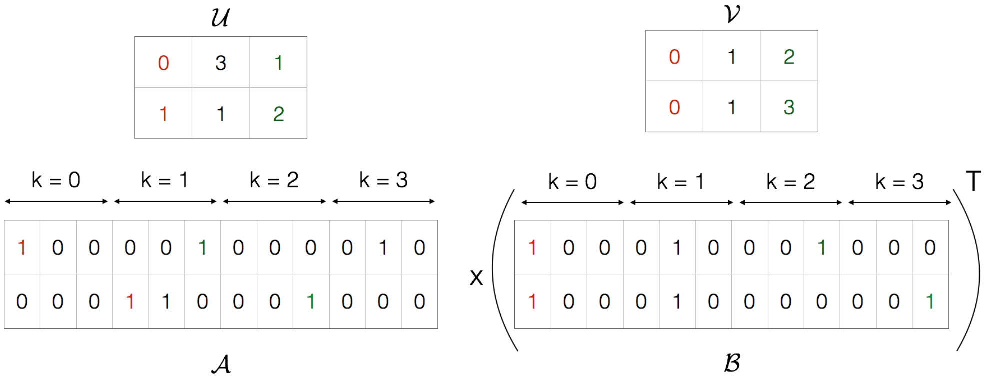


Figure 6.1: Example of the construction of the auxiliary matrices used in the reduction from APHAM to sparse matrix multiplication. In the example, we start off by two pair of matrices each containing $n = 2$ vectors where each vector is of dimension $d = 3$.

$[2n]$ (defined as before to be the set $\{0, 1, \ldots, 2n-1\}$) – if it is otherwise, we can scan each coordinate separately and rename the entries. Consider $\mathcal{A} \in \{0,1\}^{n \times N}$ for $N = 2nd$, defined as such: $\mathcal{A}[i, j + k \cdot d] = 1$ iff $U[i, j] = k$, for $0 \le i < n$, $0 \le j < d$ and $0 \le k < 2n$. Similarly, we construct $\mathcal{B} \in \{0,1\}^{n \times N}$ from $\mathcal{V}$. Refer to Figure 6.1 for an example construction of the matrices $\mathcal{A}, \mathcal{B}$.

We now observe that $\mathcal{C} = \mathcal{A} \times \mathcal{B}^T$ allows us to compute $\mathcal{U} \overset{\mathsf{Ham}}{\otimes} \mathcal{V}$, since for any $i, j \in [n]$, $\mathcal{C}[i, j] = d - \mathsf{Ham}(u_i, v_j)$.

Now to reduce the value of N, we use a technique from [35]. First, we observe that the columns of $\mathcal{A}$ and the rows of $\mathcal{B}^T$ can be simultaneously rearranged in order, so we assume that they are sorted according to $|\mathcal{A}_{*i}| \cdot |\mathcal{B}_{i*}|$. We then truncate $\mathcal{A}$ to keep only the first d^2 columns and truncate $\mathcal{B}^T$ to keep only the first d^2 rows. By Lemma 6.5, the time needed to compute the contribution of the truncated rows/columns is $\mathcal{O}(|\mathcal{A}| \cdot |\mathcal{B}|/d^2) = \mathcal{O}(n^2)$. This truncation is acceptable since $\mathsf{Sparse}(n, N, n; M; M) = \Omega(n^2)$ for all N, M because this is the time already needed to write down the output of matrix multiplication of two matrices of size $n \times N$ and $N \times n$. $\square$

Observe that regardless of whether one solves the APHAM instance by adapting [34], or by using Theorem 6.8 and [35], the resulting computation is roughly similar, thus it is no surprise that the resulting runtime is identical. To see this, let's analyze the case when $d = n$ to compare it with Yuster's approach. The matrices $\mathcal{A}$ and $\mathcal{B}$ constructed in Theorem 6.8 are of size $n \times 2n^2$ and $2n^2 \times n$. Also, $m_1 = m_2 = n^2$. Then, we compute a sparse matrix multiplication with time complexity $\mathsf{Sparse}(n, 2n^2, n; n^2, n^2)$. Following in the footsteps of Algorithm 1, we need to choose a value $l = n^r$ that minimizes $n^{\omega(1,r,1)} + (m_1 m_2)/l$ where the first part comes from the rectangular matrix multiplication and the second from the brute force and through

Lemma 3.2 from [35]. Thus, we are trying to minimize the run time equal to $n^{\omega(1,r,1)} + n^{4-r}$ which is the same expression obtained when analyzing Yuster's approach (see Lemma 4.2).

We can use the same techniques to derive a relation between APHam on *sparse inputs* with sparse matrix multiplication. We obtain the following:

Corollary 6.9 APHam *on inputs A,B of size n, with m_1 and m_2 relevant entries, respectively, takes $\mathcal{O}(\mathsf{Sparse}(n, \frac{m_1 m_2}{n^2}, n; m_1, m_2))$ time.*

Proof Following the reasoning from the proof of Theorem 6.8, we construct an instance of sparse matrix multiplication with the appropriate parameters such that its time complexity is $\mathsf{Sparse}(n, N, n; m_1, m_2)$ for some large integer N. However, to go from counting "matches" which this 0/1 matrix multiplication does, to counting mismatches, we need to count the number of aligned relevant entries between A and B. This is done with a single multiplication of sparse matrices in time $\mathsf{Sparse}(n, N, n; m_1, m_2)$. Now, once again using the Lemma 3.2 from [35], the second dimension on both of them is truncated to $m_1 m_2 / n^2$ in $\mathcal{O}(n^2)$ time. This means that we solve one part of the problem in time $\mathsf{Sparse}(n, m_1 m_2 / n^2, n; m_1, m_2)$ and another part in brute force in $\mathcal{O}(n^2)$ time. □

By this reduction, in order to improve the current upper bounds for APHam (n vectors with dimension $d = n$), one needs to improve the sparse matrix multiplication upper bound for almost square matrices. We, thus have the following corollary.

Corollary 6.10 *Any improvement to the exponent of the complexity of $\mathsf{Sparse}(n, n^{4-\rho+\varepsilon}, n; n^2, n^2)$, for $\varepsilon > 0$, beyond that obtained by techniques from [35] would improve the exponent of* APHam *to $\tilde{\mathcal{O}}(n^{\rho-\varepsilon})$ (instead of $\tilde{\mathcal{O}}(n^{\rho})$).*

Proof As said before, The matrices $\mathcal{A}$ and $\mathcal{B}$ constructed in Theorem 6.8 are of size $n \times 2n^2$ and $2n^2 \times n$. Also, $m_1 = m_2 = n^2$. Since we can afford $\tilde{\mathcal{O}}(n^{\rho-\varepsilon})$ time, we can truncate the matrices' second dimension such that the truncated part is computed by brute force in time at most $m_1 m_2 / \ell = n^4 / \ell = n^{\rho-\varepsilon}$ (using Theorem 3.2 from [35]) which gives $\ell = n^{4-\rho+\varepsilon}$. This means that if we can improve the complexity of $\mathsf{Sparse}(n, n^{4-\rho+\varepsilon}, n; n^2, n^2)$ to time $\tilde{\mathcal{O}}(n^{\rho-\varepsilon})$, then we improve the exponent of APHam. □

Current bounds imply that one needs to improve $\mathsf{Sparse}(n, n^{1.3167+\varepsilon}, n; n^2, n^2)$. ka

6.3.2 Reducing sparse matrix multiplication to APHam

We now present a converse argument, that the multiplication of *arbitrary* sparse matrices is no harder than the corresponding APHam. Here the reduction is a little bit more tricky, since e.g. the 0/1 matrices resulting from

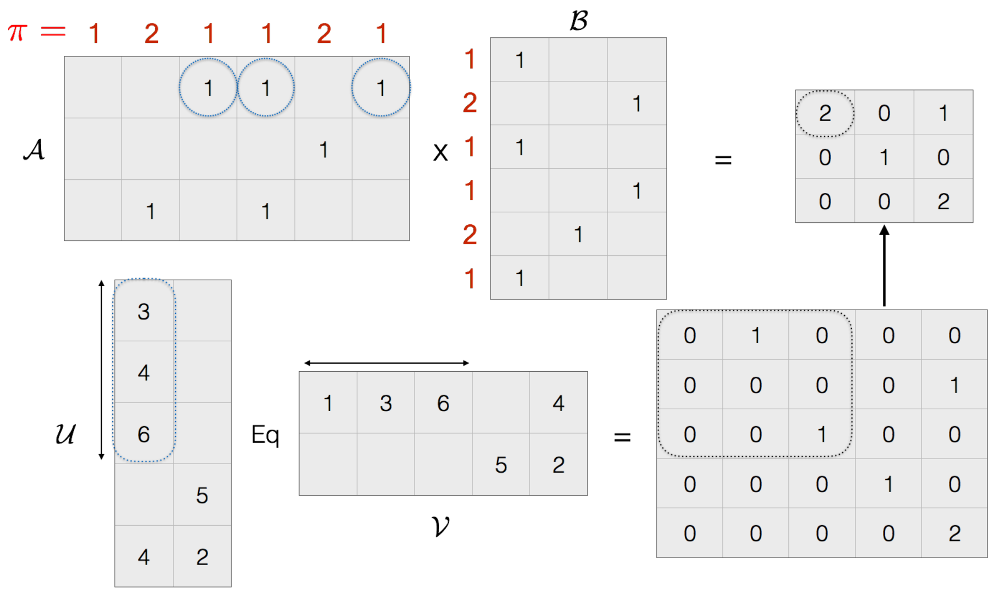


Figure 6.2: An example of how the reduction from sparse matrix multiplication to APHAM works. In the example, we have $n = 3$ and $N = 6$. The uniformly chosen vector π is indicated. The circled '1''s in the top matrix exemplify a conflict that requires row-splitting. The result of row splitting is indicated in matrix $\mathcal{U}$. The arrows are used to indicate the sub-matrices that correspond to the splitting of the first row and first column from matrices $\mathcal{A}$ and $\mathcal{B}$ respectively. The result of computing the $(+,=)$-matrix product (complement of APHAM) on $\mathcal{U}$ and $\mathcal{V}$ is shown in the right bottom where the circled sub-matrix is the output of the $(+,=)$-matrix product on the sub-matrices of the first row split and column split. The addition of elements in this sub-matrix gives the desired result of $\mathcal{A}_{1*} \cdot \mathcal{B}_{*1}$ in the original instance of sparse matrix multiplication.

Theorem 6.8 have a combinatorial inner structure that arbitrary instances of matrix multiplication might not have.

Theorem 6.11 *For $N \geq n$, the multiplication problem in Sparse$(n, N, n; N, N)$ reduces under a randomized (Las Vegas) reduction to an* APHAM *instance with $\widetilde{\mathcal{O}}(n)$ vectors of dimension $\widetilde{\mathcal{O}}(N/n)$.*

Proof Let $\mathcal{A}, \mathcal{B}$ be the input $0/1$ matrices. W.l.o.g. N is divisible by n, as if it is not the case, we round N up to the nearest multiplicity of n and pad $\mathcal{A}$ and $\mathcal{B}$ with zeroes accordingly. Denote $d = N/n$. As a first step, we pick uniformly $\pi \in [d]^{[N]}$, which we use to decide for columns of $\mathcal{A}$ (rows of $\mathcal{B}$) contribute to which columns of output $\mathcal{U}$ (rows of $\mathcal{V}$) they contribute.

We want to construct matrix $\mathcal{U}$, such that if $\mathcal{A}[i,j] = 1$ then $\mathcal{U}[i, \pi(j)] = j$. However, such mapping might not be well defined, as there might be conflicts of the form j_1, j_2 such that $\mathcal{A}[i, j_1] = \mathcal{A}[i, j_2] = 1$ and $\pi(j_1) = \pi(j_2)$. We deal with conflicts by row-splitting.

Let $r_{i,k} = \sum_{j:\pi(j)=k} \mathcal{A}[i,j]$ be the number of ones that are mapped to a given i,k cell. Denote $c_i = \max_j r_{i,j}$. Then i-th row of $\mathcal{A}$ is split into c_i rows in $\mathcal{U}$, i.e. rows $C_i + 1, \ldots, C_i + c_i$ where $C_i = c_1 + \ldots + c_{i-1}$. Then, for any i, j if $\mathcal{A}[i, j] = 1$, then $\mathcal{U}[C_i + t, \pi(j)] = j$, where $\mathcal{A}[i, j]$ was t-th value 1 cell among all $\mathcal{A}[i, x]$ such that $\pi(x) = \pi(j)$.

To complete the construction of $\mathcal{U}$, any value not yet set after processing all of $\mathcal{A}$ is assigned an unique value in its column.

Observe that since π was picked at random we have that $\mathbb{E}[r_{i,j}] = |\mathcal{A}_{i*}|/d$. By Chernoff bound, w.h.p. $|r_{i,j}| = \mathcal{O}(\log n / \log\log n) \cdot \lceil |\mathcal{A}_{i*}|/d \rceil$. Denote by $c_i = \max_j |r_{i,j}|$. As we split the i-th row of $\mathcal{A}$ into c_i rows in U, we bound the total number of rows in $\mathcal{U}$ as $\sum_i c_i = \mathcal{O}(\log n / \log\log n) \cdot \sum_i \lceil |\mathcal{A}_{i*}|/d \rceil = \mathcal{O}(\log n / \log\log n) \cdot (n + |\mathcal{A}|/d) = \mathcal{O}(n \log n / \log\log n)$.

The construction of $\mathcal{V}$ from $\mathcal{B}$ follows, switching row and column roles in the presented reduction.

Let the i-th row of $\mathcal{A}$ is mapped to rows $C_i + 1, \ldots, C_i + c_i$ in U, and the j-th column of B is mapped to columns $D_j + 1, \ldots, D_j + d_j$ in V. It follows that

$$(\mathcal{A} \times \mathcal{B})[i, j] = \sum_{a=1}^{c_i} \sum_{b=1}^{d_j} \left(d - (\mathcal{U} \overset{\mathsf{Ham}}{\otimes} \mathcal{V})[C_i + a, D_j + b] \right),$$

and so $\mathcal{U} \overset{\mathsf{Ham}}{\otimes} \mathcal{V}$ encodes $\mathcal{A} \times \mathcal{B}$.

To finish the argument, we observe that if $\mathcal{A}, \mathcal{B}$ are provided in a compressed form (which they need to, as an explicit representation is already too large), the $\mathcal{U}$ and $\mathcal{V}$ can be generated without any significant additional computational overhead, in time $\widetilde{\mathcal{O}}(|\mathcal{A}| + |\mathcal{B}| + N)$. $\square$

Chapter 7

Consequences of Our Reductions

There are several consequences of Theorem 5.14 and Theorem 5.15. The first one is that the improvement to APDom from [34] translates to other AllPairs problems:

Corollary 7.1 APDom, APL_1, APL_{2p+1}, APThr, APHam *and* $(+, \min)$-MatrixProduct *are solvable in time* $\widetilde{\mathcal{O}}(n^\rho)$, *where* $\rho \leq 2.6834$ *is a solution to* $\rho = \omega(1, 4 - \rho, 1)$.

Consequences on problems with sparse inputs Observe that the reductions we presented map $\star$ to $\star$. Thus, i.e. by [30],[32] and [12], we immediately get that all considered AllPairs problems are of the same complexity even on sparse inputs, up to a $\mathrm{poly}(\log n)$ multiplicative term and a $\mathsf{Sparse}(n, n, n; m_1, m_2)$ additive term.

Corollary 7.2 *Consider sparse inputs where we denote by* m_1 *and* m_2 *the number of entries in A and B that contribute to the score, where A and B are matrices of n vectors of dimension n.* APDom, APL_1, APL_{2p+1}, APThr, APHam *and* $(+, \min)$-MatrixProduct *are solvable in time* $\widetilde{\mathcal{O}}(\min(n^\omega + \sqrt{m_1 m_2} \cdot n^{\frac{\omega-1}{2}}, n^2 + (m_1 m_2)^{\frac{\omega-2}{\omega-\alpha-1}} n^{\frac{2-\alpha\omega}{\omega-\alpha-1}}))$.

Similarly, one can look into the relation between sparsity and runtime for pattern matching problems. Here, we obtain the following result

Theorem 7.3 *For a text of length n and a pattern of length m,* $n \geq m$, *with* s_t *and* s_p *relevant entries, respectively, the time complexity of* HamPM, LessThanPM, ThrPM *and* L_{2p+1}PM *is* $\widetilde{\mathcal{O}}(\sqrt{n s_t s_p} + n)$.

Proof Consider LessThanPM. The proof follows the non-sparse case. The s_p relevant entries of the pattern are sorted and partitioned into k buckets as in the non-sparse case. We compute inter-bucket contributions for each bucket in a total of k convolution steps in time $\mathcal{O}(kn \log m)$. Intra-bucket contributions are captured in a brute force manner in time $\mathcal{O}(s_t s_p / k)$ where each relevant text element is compared with at most s_p/k elements in its corresponding bucket. Choosing k to be $\max(1, \sqrt{(s_t s_p)/(n \log m)})$ gives the time bound of $\widetilde{\mathcal{O}}(n + \sqrt{n s_t s_p})$. $\square$

Consequences on AllPairs problems with dimension trade-off Since our reductions preserve the dimension of the problems, any tradeoff between $d \ll n$ and the runtime translates to all other problems as well, with a $\mathrm{poly}(\log n)$ multiplicative term and a $\widetilde{\mathcal{O}}(n^\omega)$ additive term. One can improve the runtime of the algorithm presented in [26] using the trick of batch-processing via rectangular matrix multiplication in [34], as done for Dominance Product in [15], to obtain the following time complexity:

Corollary 7.4 *For n vectors of dimension $d = n^s$ for $0 \leq s \leq 1$, APDom, APL_1, APL_{2p+1}, APThr, APHam and $(+, \min)$-MatrixProduct are solvable in time $\widetilde{\mathcal{O}}(n^{\rho(s)})$ where $\rho(s) = \inf\{x : 2 \leq x \leq 3 \text{ and } \omega(1, 2+2s-x, 1) \geq x\}$. In particular, for $d = \mathcal{O}(n^{\alpha/2}) \supseteq \mathcal{O}(n^{0.156945})$ all those problems are solvable in time $\widetilde{\mathcal{O}}(n^2)$.*

Weighted Pattern Matching We present the following application of the scaling/filtering framework: weighted mismatches. We distinguish between *position weighted mismatches* and *character weighted mismatches*. In the pattern matching setting, the former asks for $\mathbf{O}[i] = \sum_{j:\mathbf{P}[j] \neq \mathbf{T}[i+j]} w(j)$, whereas the latter asks for $\mathbf{O}[i] = \sum_{j:\mathbf{P}[j] \neq \mathbf{T}[i+j]} w(\mathbf{P}[j])$, for some given weight function $w : \mathbb{Z} \to \mathbb{Z}$. We see that character weighted mismatches are expressible by a function $w(x) \cdot \mathbb{1}[x \neq y]$ and get by Lemma 5.16 that Hamming Distance Pattern Matching with Character Weights is no harder than HamPM (up to a $\log n$ factor). For position weights, we present the following:

Theorem 7.5 *Hamming Distance Pattern Matching with Position Weights reduces to $\mathcal{O}(\log n)$ instances of* HamPM*.*

Proof We solve $\mathcal{O}(\log n)$ instances of HamPM with filtering involved. This is done by constructing different pattern strings where $\mathbf{P}_i$ is defined as follows:

$$\mathbf{P}_i[j] = \begin{cases} P[j] & i\text{-th bit of } w(j) \text{ is } 1 \\ \star & \text{otherwise.} \end{cases}$$

Let $\mathbf{O}_i$ be the result vector of HamPM between text $\mathbf{T}$ and pattern $\mathbf{P}_i$. The final result vector, $\mathbf{O}$, for the Hamming distance pattern matching with position weights can be computed such that $\mathbf{O}[k] = \sum_{i=0}^{\lceil \log W \rceil} 2^i \cdot \mathbf{O}_i[k]$ where W is the maximum position weight. Given our assumption that $W = \mathrm{poly}(n)$, the result follows. $\square$

All of the results presented in this chapter show the usefulness of our reductions. While it is no surprise that for example the technique of [34] can be applied to other AllPairs problems, it is a nice side effect of our reduction that it can be applied "automatically" without looking deeper into the structure of any of the different AllPairs problems involved. The reductions presented signify that regardless of whether we are looking for improved upper bounds, or new lower bounds, it is enough to concentrate on a single score function from the whole class of equivalent functions. In our opinion, Hamming distance is the "cleanest" score function, since it is the simplest – it assumes no arithmetic underlying structure of the alphabet (unlike e.g. L_1 distance) and not even an ordering of the alphabet.

Chapter 8

Conclusion

In this thesis, we presented a survey of pattern matching problems that are all of equivalent complexity. They used very similar techniques of combining algebraic computations together with a brute force computation to obtain their algorithms. We also surveyed corresponding all-pairs problems which used similar techniques as well. In this thesis, we asked the question of whether these problems are all related to each other. In the literature, there were simple results that linked some problems to each other. In this thesis, we extended this by showing that a broad class of problems are actually equivalent (up to poly-logarithmic factors). Since different problems differ only in their underlying binary function, all that was needed to link different problems together was to show reductions between the binary functions only. This resulted in us obtaining a Hamming distance completeness result where we showed that a broad class of binary functions are all equivalent to Hamming distance. Next, we made use of the Hamming distance completeness to link the complexity of the problem of all-pairs Hamming distance to that of multiplying very sparse rectangular matrices which consequently links all other problems to sparse matrix multiplication as well.

8.1 Open problems

We observe a class of equivalent PatternMatching problems and a class of equivalent AllPairs problems. For both classes, the current upper bound complexity comes from a tradeoff between the complexity of brute-force and of a fast algebraic solution to a simpler problem. In the former class, it is a tradeoff between $\mathcal{O}(n^2)$ of brute-force and $\mathcal{O}(n \log n)$ of FFT, resulting in $\widetilde{\mathcal{O}}(n^{3/2})$ complexity. In the latter class, it is a tradeoff between $\mathcal{O}(n^3)$ of brute-force and $\mathcal{O}(n^\omega)$ of fast matrix multiplication, resulting in a complexity of $\widetilde{\mathcal{O}}(n^\rho)$, where ρ is slightly below $(\omega + 3)/2$ (in fact, if $\omega \to 2$ then $\rho \to 5/2$). *Can we link those two phenomenons together?* That is, can we show that a significant improvement, let us say an $\mathcal{O}(n^{3/2-\varepsilon})$ algorithm for HamPM, would imply a significant improvement for APHam (i.e. an algorithm that works in time $\mathcal{O}(n^{5/2-\varepsilon})$ assuming $\omega \to 2$)? A very partial answer was provided by Clifford [7] where they showed a reduction from binary matrix multiplication to HamPM. We note here that the reduction presented there can actually be seen as one from APHam to HamPM.

Reduction from APHam to HamPM Given two matrices $\mathcal{A}, \mathcal{B}$ of sizes $n \times n$, we follow the same steps [7] in creating matrices $\mathcal{A}', \mathcal{B}'$ respectively. Note that in APHam we aim at computing the hamming distance between every row of $\mathcal{A}$ and column of $\mathcal{B}$. Thus $\mathcal{A}'$ and $\mathcal{B}'$ are created as follows:

$$\mathcal{A}'[i,j] = (\mathcal{A}[i,j], j)$$
$$\mathcal{B}'[j,k] = (\mathcal{B}[j,k], j)$$

Next, we create the pattern and text exactly as in [7]. The pattern **P** is created from $\mathcal{A}'$ by concatenating its rows into one string of length n^2. The text **T** is created from $\mathcal{B}'$ by concatenating the columns of $\mathcal{B}'$ with a special character '\$' in between each column and surrounding **T** with $n \cdot (n-1)$ '\$' characters in the beginning and end. Thus, **T** is of length $\mathcal{O}(n^2)$. The separating characters '\$' ensure that at every alignment at most one substring of **P** corresponding to a row from $\mathcal{A}'$ aligns with a substring of **T** corresponding to a column of $\mathcal{B}'$. Thus, the resulting vector from running HamPM on **P** and **T** contains the solution for APHam on the original matrices $\mathcal{A}$ and $\mathcal{B}$.

However, the reduction is unsatisfactory as it only shows that an $\mathcal{O}(n^x)$ upper bound to HamPM implies an $\mathcal{O}(n^{2x})$ upper bound to APHam. Thus, with the current upper bound of $\tilde{\mathcal{O}}(n^{1.5})$ for HamPM, this implies the trivial upper bound of $\mathcal{O}(n^3)$ for APHam. However, stated differently, we can say that the current upper bound for APHam of $\mathcal{O}(n^\rho)$ provides a conditional lower bound of $\Omega(n^{\rho/2})$. Thus we can see that for HamPM, there is still a gap between the exponent of $\rho/2 (\approx 1.3417)$ of the conditional lower bound and the best known exponent of the upper bound of 1.5.

Another interesting direction are the approximation versions of all our problems, where the final result **O** only needs to be computed up to a multiplicative factor. While fast approximation schemes exist for metric distances (Porat and Efremenko [27]), their existence is open for the approximation of non-metric distances like equality (counting matches), dominance and threshold.* *So is it that those problems are fundamentally harder to approximate or were we just not able to find better algorithms yet?* As our reductions use not exclusively positive coefficients, they do not preserve approximation guarantees and thus this question is left open.

*The approximation algorithm for equality used in Corollary 1 in [36] does not give a speedup for the approximation of the threshold score if the equality score is small.

Appendix A

Appendix

A.1 Supplementary reductions

Theorem A.1 *L_1 distance reduces to* min *and multiplications.* min *reduces to L_1 and multiplications.*

Proof Observe that: $\min(x, y) = x/2 + y/2 - |x - y|/2$ and $|x - y| = x + y - \min(x, y)$. $\square$

Lemma A.2 *Dominance and δ-threshold are equivalent.*

Proof Since both dominance and threshold are shift-invariant, we assume $0 \leq x, y \leq M$ for some M bounded by $\text{poly}(n)$. Dominance reduces to one instance of threshold as $\mathsf{Dom}(x, y) = \mathsf{Thr}_\delta(x + \delta, y)$ for any $\delta > M$. Threshold reduces to two instances of dominance as $\mathsf{Thr}_\delta(x, y) = \mathsf{Dom}(x, y + \delta) + \mathsf{Dom}(x + \delta, y) - 1$ for $\delta > 0$. $\square$

Remark: Thus, the main result from [36] where it was shown that δ-threshold can be reduced to $\mathcal{O}(\log \delta)$ instance of Hamming distance is implied by combining Theorem 5.13 with Lemma A.2.

A.2 Examples of our reductions

Figures A.1 and A.2 illustrate our reductions from Theorems 5.12 and 5.13, respectively.

A.3 Discrete differentiation

Below, we give an illustration for the discrete differentiation of equations of the same form as Part 1 and Part 2 in Equation (5.4). Consider the discrete

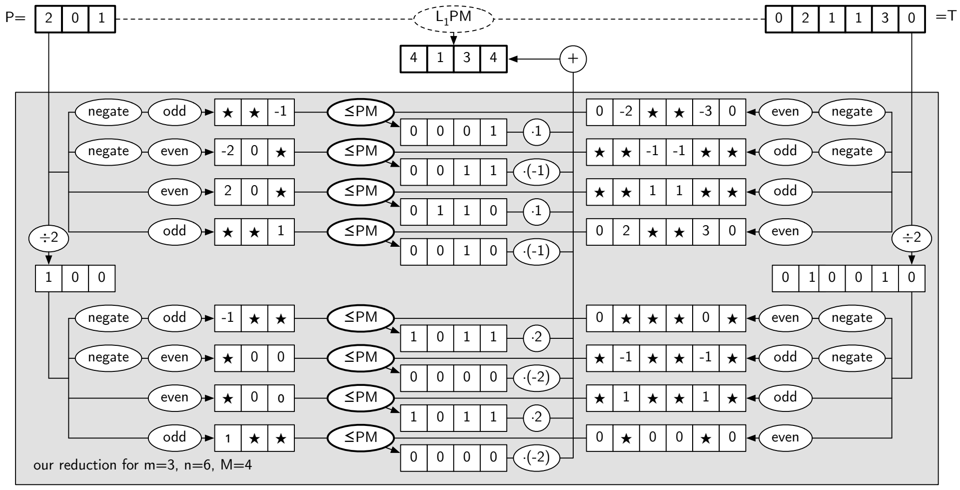


Figure A.1: Our reduction from $\mathrm{L_1}$PatternMatching to LessThanPatternMatching in Theorem 5.12 instantiated for a pattern P of length $m = 3$ and a text T of length $n = 6$ over an alphabet of integers $\{0, 1, 2, 3\}$, so $M = 2^2 = 4$.

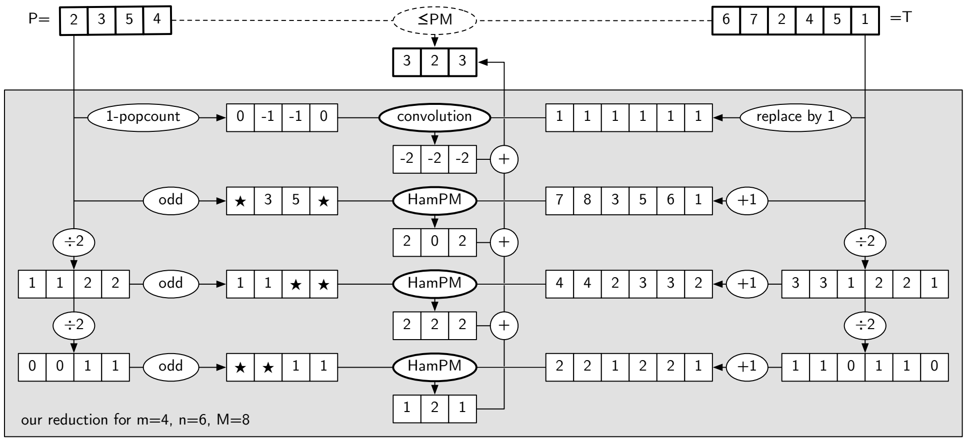


Figure A.2: Our reduction from LessThanPatternMatching to HammingDistancePatternMatching in Theorem 5.13 instantiated for a pattern P of length $m = 4$ and a text T of length $n = 6$ over an alphabet of integers $\{0, 1, 2, 3, 4, 5, 6, 7\}$, so $M = 2^3 = 8$.

function $H(x, y) = (xy)\mathbb{1}[x = y]$. We show the function and its derivative for $x, y \in [0, \ldots, 10]$. Refer to Table A.1.

We also consider the discrete function $G(x, y) = (xy^2)\mathbb{1}[x < y]$. We show the function and its derivative for $x, y \in [0, \ldots, 10]$. Refer to Table A.4.

Table A.1: Illustration of an equation of the form of Part 1 in Equation (5.4), together with its derivative.

Table A.2: $H(x, y) = (xy)\mathbb{1}[x = y]$

$x \backslash y$	0	1	2	3	4	5	6	7	8	9	10
0	(0)	0	0	0	0	0	0	0	0	0	0
1	0	(1)	0	0	0	0	0	0	0	0	0
2	0	0	(4)	0	0	0	0	0	0	0	0
3	0	0	0	(9)	0	0	0	0	0	0	0
4	0	0	0	0	(16)	0	0	0	0	0	0
5	0	0	0	0	0	(25)	0	0	0	0	0
6	0	0	0	0	0	0	(36)	0	0	0	0
7	0	0	0	0	0	0	0	(49)	0	0	0
8	0	0	0	0	0	0	0	0	(64)	0	0
9	0	0	0	0	0	0	0	0	0	(81)	0
10	0	0	0	0	0	0	0	0	0	0	(100)

Table A.3: $D_x D_y(H(x, y))$

$x \backslash y$	0	1	2	3	4	5	6	7	8	9
0	(1)	-1	0	0	0	0	0	0	0	0
1	-1	(5)	-4	0	0	0	0	0	0	0
2	0	-4	(13)	-9	0	0	0	0	0	0
3	0	0	-9	(25)	-16	0	0	0	0	0
4	0	0	0	-16	(41)	-25	0	0	0	0
5	0	0	0	0	-25	(61)	-36	0	0	0
6	0	0	0	0	0	-36	(85)	-49	0	0
7	0	0	0	0	0	0	-49	(113)	-64	0
8	0	0	0	0	0	0	0	-64	(145)	-81
9	0	0	0	0	0	0	0	0	-81	(181)

Table A.4: Illustration of an equation of the form of Part 2 in Equation (5.4), together with its derivative.

Table A.5: $G(x, y) = (xy^2)\mathbb{1}[x < y]$

$x \backslash y$	0	1	2	3	4	5	6	7	8	9	10
0	(0)	0	0	0	0	0	0	0	0	0	0
1	0	(0)	4	9	16	25	36	49	64	81	100
2	0	0	(0)	18	32	50	72	98	128	162	200
3	0	0	0	(0)	48	75	108	147	192	243	300
4	0	0	0	0	(0)	100	144	196	256	324	400
5	0	0	0	0	0	(0)	180	245	320	405	500
6	0	0	0	0	0	0	(0)	294	384	486	600
7	0	0	0	0	0	0	0	(0)	448	567	700
8	0	0	0	0	0	0	0	0	(0)	648	800
9	0	0	0	0	0	0	0	0	0	(0)	900
10	0	0	0	0	0	0	0	0	0	0	(0)

Table A.6: $D_x D_y^2(G(x, y))$

$x \backslash y$	0	1	2	3	4	5	6	7	8
0	(4)	1	2	2	2	2	2	2	2
1	-4	(17)	-6	2	2	2	2	2	2
2	0	-18	(52)	-25	2	2	2	2	2
3	0	0	-48	(121)	-62	2	2	2	2
4	0	0	0	-100	(236)	-123	2	2	2
5	0	0	0	0	-180	(409)	-214	2	2
6	0	0	0	0	0	-294	(652)	-341	2
7	0	0	0	0	0	0	-448	(977)	-510
8	0	0	0	0	0	0	0	-648	(1396)
9	0	0	0	0	0	0	0	0	-900

Bibliography

[1] Karl R. Abrahamson. Generalized string matching. *SIAM J. Comput.*, 16(6):1039–1051, 1987.

[2] A. Amir and M. Farach. Efficient 2-dimensional approximate matching of half-rectangular figures. *Inf. Comput.*, 118(1):1–11, 1995.

[3] Amihood Amir, Ohad Lipsky, Ely Porat, and Julia Umanski. Approximate matching in the L_1 metric. In *CPM*, pages 91–103, 2005.

[4] Mikhail J. Atallah and Timothy W. Duket. Pattern matching in the hamming distance with thresholds. *Inf. Process. Lett.*, 111(14):674–677, 2011.

[5] Peter Clifford and Raphaël Clifford. Simple deterministic wildcard matching. *Inf. Process. Lett.*, 101(2):53–54, 2007.

[6] Peter Clifford, Raphaël Clifford, and Costas S. Iliopoulos. Faster algorithms for δ,γ-matching and related problems. In *CPM*, pages 68–78, 2005.

[7] Raphaël Clifford. Matrix multiplication and pattern matching under Hamming norm. http://www.cs.bris.ac.uk/Research/Algorithms/events/BAD09/BAD09/Talks/BAD09-Hammingnotes.pdf, 2009. Retrieved March 2017.

[8] D. Coppersmith. Rapid multiplication of rectangular matrices. *SIAM Journal on Computing*, 11(3):467–471, 1982.

[9] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. In *STOC*, pages 1–6, 1987.

[10] Don Coppersmith. Rectangular matrix multiplication revisited. *J. Complex.*, 13(1):42–49, 1997.

[11] Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson. *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition, 2001.

[12] Ran Duan and Seth Pettie. Fast algorithms for (max, min)-matrix multiplication and bottleneck shortest paths. In *SODA*, pages 384–391, 2009.

[13] M. J. Fischer and M. S. Paterson. String-matching and other products. Technical report, Cambridge, MA, USA, 1974.

[14] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.

[15] Omer Gold and Micha Sharir. Dominance products and faster algorithms for high-dimensional closest pair under L_∞. In *ISAAC*, pages 39:1–39:12, 2017.

[16] Xiaohan Huang and Victor Y. Pan. Fast rectangular matrix multiplication and applications. *J. Complex.*, 14(2):257–299, June 1998.

[17] Piotr Indyk, Moshe Lewenstein, Ohad Lipsky, and Ely Porat. Closest pair problems in very high dimensions. In *ICALP*, pages 782–792, 2004.

[18] ShanXue Ke, BenSheng Zeng, WenBao Han, and Victor Y. Pan. Fast rectangular matrix multiplication and some applications. *Science in China Series A: Mathematics*, 51(3):389–406, Mar 2008.

[19] Donald E. Knuth, Jr. James H. Morris, and Vaughan R. Pratt. Fast pattern matching in strings. *SIAM Journal on Computing*, 6(2):323–350, 1977.

[20] François Le Gall. Powers of tensors and fast matrix multiplication. In *ISSAC*, pages 296–303, 2014.

[21] François Le Gall. Faster algorithms for rectangular matrix multiplication. In *FOCS*, pages 514–523, 2012.

[22] François Le Gall and Florent Urrutia. Improved rectangular matrix multiplication using powers of the coppersmith-winograd tensor. In *SODA*, pages 1029–1046, 2018.

[23] Ohad Lipsky and Ely Porat. L_1 pattern matching lower bound. *Inf. Process. Lett.*, 105(4):141–143, 2008.

[24] Ohad Lipsky and Ely Porat. Approximate pattern matching with the L_1, L_2 and L_∞ metrics. *Algorithmica*, 60(2):335–348, 2011.

[25] Jiří Matoušek. Computing dominances in E^n (short communication). *Inf. Process. Lett.*, 38(5):277–278, 1991.

[26] Kerui Min, Ming-Yang Kao, and Hong Zhu. The closest pair problem under the Hamming metric. In *COCOON*, pages 205–214, 2009.

[27] Ely Porat and Klim Efremenko. Approximating general metric distances between a pattern and a text. In *SODA*, pages 419–427, 2008.

[28] Andrew James Stothers. *On the complexity of matrix multiplication*. PhD thesis, 2010.

[29] Volker Strassen. Gaussian elimination is not optimal. *Numer. Math.*, 13(4):354–356, 1969.

[30] Virginia Vassilevska. *Efficient algorithms for path problems in weighted graphs*. PhD thesis, Carnegie Mellon University, 2008.

[31] Virginia Vassilevska and Ryan Williams. Finding a maximum weight triangle in $n^{3-\delta}$ time, with applications. In *STOC*, pages 225–231, 2006.

[32] Virginia Vassilevska, Ryan Williams, and Raphael Yuster. All pairs bottleneck paths and max-min matrix products in truly subcubic time. *Theory of Computing*, 5(1):173–189, 2009.

[33] Virginia Vassilevska Williams. Multiplying matrices faster than coppersmith-winograd. In *STOC*, pages 887–898, 2012.

[34] Raphael Yuster. Efficient algorithms on sets of permutations, dominance, and real-weighted APSP. In *SODA*, pages 950–957, 2009.

[35] Raphael Yuster and Uri Zwick. Fast sparse matrix multiplication. *ACM Trans. Algorithms*, 1(1):2–13, 2005.

[36] Peng Zhang and Mikhail J. Atallah. On approximate pattern matching with thresholds. *Inf. Process. Lett.*, 123:21–26, 2017.

ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

COMPLETENESS OF HAMMING DISTANCE FOR PATTERN MATCHING AND ALL-PAIRS PROBLEMS

Authored by (in block letters):
For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**
LABIB	KARIM

With my signature I confirm that
- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**
Zürich, 14/2/2018	Karim

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.