Tel Aviv University
Raymond and Beverly Sackler Faculty of Exact Sciences
Blavatnik School of Computer Science

New Algorithms for Some Classical Problems in P

A thesis submitted for the degree of
DOCTOR OF PHILOSOPHY
by
Omer Gold

Thesis supervisor: Prof. Micha Sharir

Submitted to the Senate of Tel Aviv University
August 2019

Abstract

The search for optimal algorithms is at the core of computer science since its emergence as a field. Yet for the majority of the studied problems we do not know whether state-of-the-art algorithms are the best possible. Among the most popular problems in P are those that have standard algorithms that run in $O(n^c)$ time for an input of size n, where $c = 2$ or 3. For $c = 3$ (cubic time), we can find combinatorial matrix multiplication problems, and computing all-pairs-shortest-paths (APSP) of a directed real-weighted graph. For $c = 2$ (quadratic time), we can find many fundamental problems, such as 3SUM, and many basic matching problems between strings, curves, and point-sequences, such as Edit Distance, Longest Common Subsequence, Discrete Fréchet Distance, Geometric Edit Distance, and Dynamic Time Warping. These problems are usually referred to as "quadratic problems".

In this thesis, we present improved algorithms and decision trees for the following core problems.

- We improve the $(2k-2)$-linear decision tree bound for k-SUM and the subquadratic algorithm for the famous 3SUM problem, both obtained by Grønlund and Pettie [104]. Follow-up to our work, Kane, Lovett, and Moran [114] obtained the breakthrough of showing a $2k$-linear decision tree for k-SUM with near-linear depth. Chan [55] improved our 3SUM algorithm and currently holds the record for the algorithm with the best runtime bound for the problem.

- We give the first subquadratic algorithms for computing Dynamic Time Warping (DTW) and Geometric Edit Distance (GED) between two point-sequences in $\mathbb{R}$ (and also in $\mathbb{R}^d$, for any constant d, when the underlying metric is polyhedral), breaking the nearly 50 years old quadratic time barrier for these problems. The DTW measure is an extremely popular matching method, being massively used in dozens of applications.

  For computing the related Discrete Fréchet Distance, we show linear decision trees with near-linear depth, for any fixed dimension, when the underlying metric is polyhedral.

- We give improved strongly-polynomial subcubic algorithms for solving the high-dimensional (e.g., $\mathbb{R}^n$) Closest Pair problem under the L_∞ metric, and give improved runtime analysis for computing the related *dominance product* matrix of n points in dimension polynomial in n. Computing the dominance product itself is an important task, since it is applied in many algorithms, in addition to our Closest Pair algorithm, as a major black-box ingredient.

- Another result, unrelated to the main motif of the thesis, shows the existence of sparse (subquadratic) *diameter spanners* with various size-stretch trade-offs, and gives efficient algorithms to construct them. That is, given a directed graph $G = (V, E)$ and a stretch factor $t > 0$, a subgraph $H = (V, E_H \subseteq E)$ is a *t-diameter spanner* iff $\text{diam}(H) \leqslant \lceil t \cdot \text{diam}(G) \rceil$, where $\text{diam}(H)$ and $\text{diam}(G)$ denote the diameter of H and G, respectively.

Acknowledgments

In the next several paragraphs I will thank the people who supported me during the time of pursuing my PhD. Then, I will share some of my personal experience from the last five years as a PhD student, so this will be a bit longer than a typical acknowledgments section.

First, I would like to deeply thank my advisor Micha Sharir, who took me under his wings and guided me on how to approach theoretical computer science. The most valuable thing for me was the numerous research discussions with Micha, having a peek at how he approaches research problems, the way he simplifies them and getting fast to their bottleneck, the core one should be focused on when trying to discover new results. I perceive these discussions as "private lessons" for a deep analytical thinking, and in particular, for research in theoretical computer science, regardless of the problems in hand. I feel it totally changed the way I approach analytical problems, and not only related to research, but also in other complex matters I encounter, such as in programming, and solving engineering problems. The value of these "lessons" is priceless.

I would like to thank Liam Roditty, who discussed with me about graph diameter algorithms, and connected me with Keerti Choudhary. This has led to the discovery of non-trivial "diameter spanners" in directed graphs, and to the joint paper with Keerti [65], which Chapter 6 in this thesis is based on. This is also the right place to thank my co-author Keerti Choudhary, for her dedication to our project and the fruitful joint work.

I would like to thank Reuven Cohen, my master's advisor from Bar-Ilan University, who encouraged me to pursue a PhD, and provided all the assistance needed to make it happen. I thank also Moshe Lewenstein who gave me a valuable feedback for my PhD research proposal, and invited me to a full week workshop on "Structure and Hardness in P" at the Dagstuhl castle in Germany.

I would like to thank the senior researchers who reviewed this thesis. I felt honored to receive your reviews. A special thanks goes to the reviewer who noticed that we can shave-off the $\log\log\log n$ factor from our Dynamic Time Warping and Geometric Edit Distance algorithms that appear in Chapter 4, using the SMAWK algorithm for totally monotone matrices [9].

I would also like to thank fellow PhD students with whom I had many interesting research conversation with: Sarel Cohen, who introduced to me some cool research questions in graph algorithms, and sparked my interest in the field. Dor Minzer, who I shared with some research problems I have been working on, and gave me valuable feedbacks. Orr Fischer, with whom I had many interesting conversations about theoretical computer science and academia in general.

Last but by no means the least, I would like to thank my parents, Michael and Neomi, who supported me throughout this long academic track, from starting as an undergraduate student in Ben-Gurion University more than 10 years ago, throughout pursuing my master's in Bar-Ilan University, and finally, throughout the extensive journey of doing a PhD in Tel Aviv University. Thank you for your endless support, encouragement, and being there for me whenever needed.

My Personal Experience. Computer science is a relatively new scientific field, and it is emerging with many important discoveries every year. The amount of new significant discoveries in theoretical computer science that were discovered only during the time I was a PhD student really amazed me. I feel lucky I had the opportunity to do research in theoretical computer science during this time in history, as I could witness plenty of new interesting results and breakthroughs being published by the community in "real time", and sometimes even to contribute a little bit. I do not know for how long this rate of new significant discoveries in computer science will continue, but thinking again about how young this scientific field is (which only 100 years ago nobody knew about), I guess that this rate will not decline anytime soon.

I remember that when I began my PhD studies in late 2014, my advisor Micha told me about a recent breakthrough on the 3SUM problem (determining whether there are three numbers that sum to zero in a given set of n real numbers). Allan Grønlund and Seth Pettie [104] showed that 3SUM can be solved in subquadratic time. Although only small polylogarithmic factors were improved over the well-known $\Theta(n^2)$ time bound, this result made huge strides, since the 3SUM problem is well-known for basing conditional lower bounds for many other problems, and therefore, it raised doubts on the optimality of many other algorithms, such as for determining whether n given points in the plane are located in a general position (i.e., no three points lie on a common line).

The same day, I started reading the paper of Grønlund and Pettie with enthusiasm, hoping that maybe a further improvement is possible. Since it was the first serious theory paper I read, it took a while until I controlled the details. It took months of thinking and many discussions with Micha until finally finding a way to improve their algorithm and decision tree bounds. Although the improvements were small (shaving polylogartihmic factors from both bounds), the exciting thing was that my first theory result was about a well-known problem.

Later, Timothy M. Chan [55] improved a bit further the algorithmic time bound for 3SUM (by another logarithmic factor). In 2017, a breakthrough on this problem came from Daniel Kane, Sachar Lovett, and Shay Moran [114], who showed that the decision tree complexity of 3SUM is near-linear, improving significantly our $O(n^{3/2})$ decision tree bound (and the $O(n^{3/2}\sqrt{\log n})$ bound of Grønlund and Pettie). I was very surprised that such a significant improvement is even possible. Their technique also gave near-linear decision tree complexity bounds for other core problems, such as "Sorting $X+Y$" and "All-Pairs-Shortest-Paths". What I described in this paragraph truly relates to what I mentioned in the first paragraph, about being lucky to witness breakthroughs during this time, especially when they are related to topics I have been working on.

Then, I have been working on extending the technique we used for the 3SUM problem, and looking for other fundamental problems to apply it, but did not find one. Until, one day Pankaj K. Agarwal gave a talk in our weekly computational geometry seminar about approximation algorithms for the Dynamic Time Warping and Geometric Edit Distance problems [8]. That was the first time I heard of these problems, and I discovered then that the best known algorithms

to solve them use a standard dynamic programming approach that takes quadratic time [150]. During his talk I started to think about whether we can break this quadratic-time barrier. It took a dramatically more sophisticated use of the techniques used in our 3SUM paper, in conjunction with other techniques, until we finally managed to break the 50 years old quadratic time barrier for both Dynamic Time Warping and Geometric Edit Distance by a $\log\log n$ factor (actually it was a $\log\log n/\log\log\log n$ factor, but thanks again to one of the reviewers of this thesis, who noticed that we can in fact shave-off the $\log\log\log n$ factor). Now, when I look at this improvement factor it seems funny, as its growth rate (in proportion to the input size n) is very slow, but this is the nice thing about theoretical computer science, our goal is to find the optimal algorithm, the one that its runtime cannot be improved by *any* asymptotic factor. Practically, our algorithm can perhaps improve the runtime for very large inputs (depends also on the constant of proportionality in our time bound) over the standard quadratic-time algorithm.

The next result was on the high-dimensional L_∞ Closest Pair problem. That is, finding the closest pair of points under the L_∞ metric in a given set of n points in $\mathbb{R}^d$, where $d = \text{poly}(n)$ (for example $d = n$). We gave a new algorithm for this problem, improving a previous algorithm of Piotr Indyk, Moshe Lewenstein, Ohad Lipsky, and Ely Porat [112]. The thing I remember the most from this paper is that it appeared in the ISAAC 2017 conference that was held in Phuket, Thailand in a very nice suites hotel on the beach. It was definitely my most unforgettable academic trip to date. This trip has led me to travel more in Thailand, learn more about Southeast Asia, and to visit the Philippines for a whole month a year later. I had a blast in both Thailand and the Philippines. I met in both countries super friendly people and liked the general relaxed vibe.

After I finished working on these three papers, I felt eager to diversify my research and looked for areas I have not worked on before. I started exploring more seriously about graph algorithms. This has led to some interesting discussions with Liam Roditty, who also connected me with Keerti Choudhary. The work with Keerti has led to our joint SODA paper [65], in which we proved the existence of various non-trivial "diameter spanners" for directed graphs. That is, that any sufficiently dense directed graph has a significantly sparser subgraph that preserves the diameter of the original graph up to a factor that is strictly less than 2 (called also "stretch factor"). We showed how to efficiently compute such subgraphs with various non-trivial size-stretch trade-offs. This opens a large room for future work, and it will be interesting to see what new results on this topic will be further discovered.

Omer Gold

December 28, 2019

Contents

Chapter 1

Introduction

The search for optimal algorithms is at the core of computer science since its emergence as a field. Yet for the majority of the studied problems we do not know whether state-of-the-art algorithms are the best possible. Among the most popular basic problems in P are those that have standard algorithms that run in $O(n^c)$ time, where $c = 2$ or 3. For $c = 3$ (cubic time), we can find many kinds of combinatorial matrix multiplication problems, and for $c = 2$ (quadratic time), we can find many fundamental problems, such as 3SUM, and many basic matching problems between strings, curves, and point-sequences, such as Edit Distance (ED), Longest Common Subsequence (LCS), Geometric Edit Distance (GED), Dynamic Time Warping (DTW), and Discrete Fréchet Distance. Since no $O(n^{2-\Omega(1)})$-time algorithm is known for any of these problems, they are usually referred to as "quadratic problems".

Motivated to find optimal algorithms for these basic problems, researchers have developed improved algorithms with time bounds of the form $O(n^c/\operatorname{polylog}(n))$, where $\operatorname{polylog}(n)$ stands for $\log^k n$, for some constant $k > 0$. Due to these works, the complexity of many classical quadratic problems has now upper bounds of the form $O\left(n^2/\operatorname{polylog}(n)\right)$. It was only recently that the well-known 3SUM problem (determining whether there are three numbers in a given set of n real numbers that sum to zero) was shown to have such a subquadratic bound by Grønlund and Pettie [104]. Since the complexity of the 3SUM problem serves as a lower bound for numerous other problems [95, 108], which are also called 3SUM-Hard problems (shown by reductions from 3SUM), any better understanding of its complexity is highly desired.

A complementary line of research to that of searching for optimal algorithms is to better understand our limits for faster algorithms by proving lower bounds. However, it seems that our current knowledge on "real" (unconditional) lower bounds is very limited. Nevertheless, in recent years, a significant progress has been made towards a better understanding of the hardness of basic problems in P, by proving "conditional lower bounds" via reductions from core problems, such as 3SUM, (min, +)-matrix multiplication (APSP), and CNF-SAT. For example, determining whether there are three collinear points in a set of n points in the plane is known to be at least as hard as 3SUM (this is one of the aforementioned 3SUM-Hard problems). It was recently shown that, assuming that CNF-SAT takes $\Omega\left(2^{(1-o(1))n}\right)$ time (which is implied by the so-called *Strong Exponential Time Hypothesis* (SETH)) implies that there is no $O\left(n^{2-\Omega(1)}\right)$-time algorithm for LCS, Discrete Fréchet Distance, Edit Distance, and DTW. See [3–6, 17, 23, 27, 38, 63, 95, 113, 122, 137, 138, 142, 153, 156], for examples of such conditional lower bounds.

Recent seminal works by Abboud *et al.* [3], and by Abboud and Bringmann [2] show that even an improvement by a sufficiently high polylogarithmic factor for any of these problems would lead to significant consequences, such as faster Formula-SAT algorithms, or new circuit complexity lower bounds. These works suggest that current state-of-the-art algorithms for these problems may be optimal, or near optimal, in the sense that polylogarithmic factor improvements in runtime may be the only way to push the efficiency of the solution "to the limit".

The review so far pertains to the standard *real-RAM model* (also referred to as the *uniform model*), which counts all the standard arithmetic and boolean operations performed by the algorithm. A degenerate, yet very popular model is the *decision tree model*, in which each branching is based on sign test of some (usually constrained) algebraic expression on the input values (note that this model does not count such a sign test on non-input values as a branching operation). In this model only branching operations are counted. The complexity of a decision tree is its depth (which bounds the number of branching operations in any execution). A very popular type of the decision tree model that we will often study in this work is the *r-linear decision tree* model, where all the algebraic expressions are restricted to be linear and with at most r terms. A formal definition of this model and its randomized variant are given in Section 2.1.

In this thesis, we give improved algorithms and decision trees, for some of the core problems in P, such as 3SUM, Dynamic Time Warping (DTW), Geometric Edit Distance (GED), Discrete Fréchet Distance, Dominance Product, and the High-Dimensional Closest Pair problem under L_∞.

In Chapter 3, we improve the 4-linear decision tree bound and the subquadratic algorithm of the famous 3SUM problem, and the $(2k-2)$-linear decision bound for k-SUM and Linear Degeneracy Testing, given by Grønlund and Pettie [104]. Follow-up to our work, Kane, Lovett, and Moran [114] have obtained a fascinating breakthrough on these problems, such as showing that the $2k$-linear decision tree complexity of k-SUM is only $O(kn \log^2 n)$.

In Chapter 4, we give the first subquadratic algorithms for computing Dynamic Time Warping (DTW) and Geometric Edit Distance (GED) between two point-sequences in $\mathbb{R}$, breaking the 50 years old quadratic barrier of these problems. The DTW and GED measures are extremely popular matching methods, being massively used in dozens of applications, such as speech recognition, geometric shape matching, DNA and protein sequences, matching of time series data, GPS, video and touch screen authentication trajectories, music signals, and countless data mining applications. see [48,71,77,118–120,132,140,151] for some examples. To date, searching "dynamic time warping" (with quotes) in Google Scholar yields approximately $40,000$ papers, and approximately $270,000$ results in the standard web search. This illustrates the tremendous popularity and importance of this measure.

In Chapter 5, we give improved algorithms for high-dimensional closest pair problems under the L_∞ metric. A standard (trivial) cubic time algorithm can find the closest pair in a set of n points in $\mathbb{R}^n$. We give the first strongly-polynomial subcubic algorithm for this problem, improving the previously known weakly-polynomial subcubic bound [112]. We also give improved runtime analysis for computing the related dominance product matrix of n points in dimension $\mathsf{poly}(n)$.

In the following sections, we give a short overview for each of the chapters of this thesis.

1.1 3SUM, k-SUM, and Linear Degeneracy

In Chapter 3 we study the 3SUM, k-SUM and Linear Degeneracy problems. This chapter is based on the article [101] by the author and his advisor.

Given a set of n real numbers, the general 3SUM problem is to decide whether there are three of them that sum to zero. Until a recent breakthrough by Grønlund and Pettie [104], a simple $\Theta(n^2)$-time deterministic algorithm for this problem was conjectured to be optimal. (In fact, an early study of 3SUM-Hard problems by Gajentaan and Overmars [95] denoted them as "n^2-Hard" problems.) Over the years many algorithmic problems have been shown to be reducible from the 3SUM problem or its variants, including the more generalized forms of the problem, such as k-SUM and k-variate linear degeneracy testing (k-LDT). Given a linear function $f(x_1, \ldots, x_k) = \alpha_0 + \sum_{1 \leqslant i \leqslant k} \alpha_i x_i$ and a finite set $A \subset \mathbb{R}$, the k-variate linear degeneracy testing problem (k–LDT) is to decide whether $0 \in f(A^k)$. When $f(x_1, \ldots, x_k) = \sum_{i=1}^{k} x_i$ the problem is called k-SUM. For a general k, it is an open question whether these problems can be solved in $o(n^{\lceil k/2 \rceil})$ time (as mentioned above, for $k = 3$ we can). The conjectured hardness of these problems have become extremely popular for basing conditional lower bounds for numerous algorithmic problems in P.

In Chapter 3, we show that the randomized 4-linear decision tree complexity of 3SUM is $O(n^{3/2})$, and that the randomized $(2k-2)$-linear decision tree complexity of k-SUM and k-LDT is $O(n^{k/2})$, for any odd $k \geqslant 3$. (See Section 2.1 for a formal definition of the randomized r-linear decision tree model.) These bounds (albeit being randomized) improve the corresponding $O(n^{3/2}\sqrt{\log n})$ and $O(n^{k/2}\sqrt{\log n})$ bounds obtained by Grønlund and Pettie [104] (they are the first who showed a linear decison tree for 3SUM with a subquadratic depth). Additionally, we give another deterministic algorithm for 3SUM that runs in $O(n^2 \log\log n / \log n)$ time, improving the corresponding $O\left(n^2(\log\log n/\log n)^{2/3}\right)$ time bound of Grønlund and Pettie [104]. The latter bound matches an independent bound by Freund [94], but our algorithm is somewhat simpler, due to a better use of the word-RAM model.

Following our work, there were many recent developments on these problems.

- A breakthrough by Kane, Lovett, and Moran [114] proved that the $2k$-linear decision tree complexity of k-SUM is only $O(kn \log^2 n)$, and gave near-optimal decision trees also for other fundamental problems such as "sorting $X + Y$", and APSP. [1]

- Chan [55] improved another logarithmic factor in the time complexity of 3SUM, showing a deterministic algorithm for 3SUM that runs in $O\left(n^2(\log\log n)^{O(1)}/\log^2 n\right)$ time.

- Lincoln *et al.* [127] proved that existing 3SUM algorithms can be implemented to use only $\widetilde{O}(\sqrt{n})$ read/write memory space-size.

[1] Their k-SUM decision tree complexity bound significantly improves our bound, albeit our bound is in the $(2k-2)$-linear decision tree model, which is slightly weaker than the $2k$-linear decision tree model.

- Positive 3SUM instances can be trivially confirmed in $O(1)$ *nondeterministic* time. Carmosino et al. [51] proved that *negative* 3SUM instances over integers can be confirmed in $\widetilde{O}(n^{3/2})$ nondeterministic time. They suggest that this result makes it "unlikely" that SETH implies an $\Omega(n^{2-o(1)})$-lower bound for 3SUM over integers.

- Barba *et al.* [26] studied a polynomial variant of 3SUM, in which the criterion $x + y + z = 0$ is replaced by $f(x, y, z) = 0$, for some constant-degree polynomial f. They showed an algebraic decision tree with depth $O(n^{12/7+\varepsilon})$, for any $\varepsilon > 0$, for this problem, and an actual subquadratic algorithm that runs in $O(n^2/\operatorname{polylog}(n))$ time. They also showed that general position testing in the plane (also known as 3-Collinearity: testing whether any three input points lie on a line) can be solved in subquadratic time, assuming the n input points lie on at most $(\log n)^{1/6-\varepsilon}$ constant-degree polynomial curves.

- Kopelowitz, Pettie, and Porat [122] showed that any $\Omega(n^{3/2+\varepsilon})$-lower bound on 3SUM over integers implies lower bounds on various problems such as triangle enumeration and offline set disjointness. Their results improve the conditional lower bounds of Pătraşcu [137].

1.2 Geometric Pattern Matching Algorithms

In Chapter 4 we study basic geometric pattern matching problems. This chapter is based on the articles [99, 102] by the author and his advisor.

Geometric pattern matching is the popular task of aligning or measuring similarity between curves or point-sequences in some metric space. Very popular similarity measures are Dynamic Time Warping, Geometric Edit Distance, and Discrete Fréchet Distance. In Chapter 4 we study the problems of computing these measures.

Dynamic Time Warping (DTW) and Geometric Edit Distance (GED) are basic similarity measures between curves or general temporal sequences (e.g., time series) that are represented as sequences of points in some metric space (X, dist); formal definitions of both measures are given in Chapter 4. The DTW measure is massively used in various fields of computer science and computational biology, such as speech recognition, geometric shape matching, comparing DNA and protein sequences, music signals, time series data, GPS, video and touch screen authentication trajectories, and countless data mining applications. See [48, 71, 77, 118–120, 132, 140, 151] for some examples.

Consequently, the task of computing the DTW (or GED) measure is among the core problems in P. Despite extensive efforts to find more efficient algorithms, the best-known algorithms for computing the DTW or GED between two n-point sequences in $X = \mathbb{R}^d$ are long-standing dynamic programming algorithms that require quadratic runtime, even for the one-dimensional case $d = 1$, which is perhaps one of the most used in practice.

First Subquadratic Algorithms for DTW and GED. In Section 4.1, we break the nearly 50 years old quadratic time bound for computing DTW or GED between two sequences of n points in $\mathbb{R}$, by presenting deterministic algorithms that run in $O\left(n^2/\log\log n\right)$ time. Our algorithms can be extended to work also in high-dimensional spaces $\mathbb{R}^d$, for any constant d, when the underlying distance-metric dist is polyhedral[2] (e.g., L_1, L_∞).

It is very plausible that these problems cannot be improved beyond this kind of bound (or at most a "small" polylogarithmic-factor improvement), as Bringmann and Künnemann [39] showed that even one-dimensional DTW has no $O(n^{2-\Omega(1)})$-time algorithm, unless SETH fails. Subsequently, Abboud *et al.* [3], and Abboud and Bringmann [2] showed that even a sufficiently large polylog(n)-factor improvement over the quadratic-time upper bound of similar matching problems, would lead to major consequences, such as faster Formula-SAT algorithms and new circuit lower bounds.

Discrete Fréchet Distance under Polyhedral Metrics. Another popular similarity measure between curves and point-sequences is the Discrete Fréchet Distance. Bringmann and Mulzer [40] showed that, assuming SETH, this problem cannot be solved in $O\left(n^{2-\Omega(1)}\right)$ time, even for the one-dimensional case (with the standard distance function $\mathsf{dist}(x,y) = |x-y|$). In Section 4.5 we show that there is a simple 2-linear decision tree for this problem with depth only $O(n\log^2 n)$, and in general, for two point-sequences in $\mathbb{R}^d$, we show that there is a $2d$-linear decision tree with depth $O(n\log^2 n)$, for any constant d, when the underlying distance metric is polyhedral[2] (e.g., L_1, L_∞).

1.3 High Dimensional Closest Pair under L_∞ and Dominance Product

In Chapter 5 we study the problem of computing high-dimensional closest pair under L_∞. This chapter is based on the article [100] by the author and his advisor.

Given n points in $\mathbb{R}^d$, the Closest Pair problem is to find a pair of distinct points at minimum distance under some specific metric. When d is constant, there are efficient algorithms that solve this problem, and fast approximate solutions for general d. However, obtaining an *exact* solution in very high dimensions (e.g., $d = n$) seems to be much less understood. We consider the high-dimensional L_∞ Closest Pair problem in $\mathbb{R}^d$, where $d = n^r$ for some $r > 0$, and the underlying metric is L_∞. Clearly, a naive algorithm can solve this problem in $O(dn^2)$ time. For $d = n$, Indyk *et al.* [112] give the first non-trivial subcubic-time algorithm for solving L_∞ Closest Pair, however, their result is weakly-polynomial since its runtime includes a factor that depends on the diameter of the input points.

[2]That is, the underlying metric is induced by a norm, whose unit ball is a symmetric convex polytope with a constant number of facets (this constant generally depends on the dimension d).

In Chapter 5, we improve and simplify the result of Indyk *et al.* [112] for L_∞ Closest Pair, by showing that this problem in $\mathbb{R}^d$ can be solved by a deterministic strongly-polynomial algorithm that runs in $O(DP(n,d)\log n)$ time, and by a randomized algorithm that runs in $O(DP(n,d))$ expected time, where $DP(n,d)$ is the time bound for computing the *dominance product* for n points in $\mathbb{R}^d$; that is, a matrix D, such that $D[i,j] = \left|\{k \mid p_i[k] \leqslant p_j[k]\}\right|$; this is the number of coordinates at which p_j dominates p_i.

For L_∞ Closest Pair in $\mathbb{R}^d$ where all the coordinates of the points are *integers* from some interval $[-M, M]$, we obtain an algorithm that runs in $\widetilde{O}\left(\min\{Mn^{\omega(1,r,1)},\ DP(n,d)\}\right)$ time, where $\omega(1,r,1)$ is the exponent of multiplying an $n \times n^r$ matrix by an $n^r \times n$ matrix. (The $\widetilde{O}(\cdot)$ notation hides poly-logarithmic factors.)

We also give slightly better bounds for $DP(n,d)$, over the known ones [129, 158]. by giving a more general analysis to the algorithm of Yuster [158] that uses rectangular matrix multiplications. By plugging into our analysis the best-known bounds for multiplying an $n \times d$ matrix by a $d \times n$ matrix (see [124, 126]), one can obtain improved bounds for computing $DP(n,d)$. Computing the dominance product itself is an important task, since it is applied also in many other algorithms as a major black-box ingredient (in addition to our L_∞ Closest Pair algorithm), such as algorithms for APBP (all pairs bottleneck paths) [74], and variants of APSP [158].

Following our work, Graf, Labib, and Uznański [123] showed that in $\mathbb{R}^d$, dominance product is computationally equivalent (up to polylogarithmic factors) to Closest Pair under any L_{2p+1} metrics (i.e., $L_3, L_5, L_7, \ldots$). (Note that for any *even* constant p the runtime bound for solving L_p Closest Pair is currently much smaller than $DP(n,d)$ [112].) In our result we actually show that d-dimensional dominance product is at least as hard as d-dimensional L_∞ Closest Pair. The result of Graf, Labib, and Uznański [123] together with our result show an interesting computational connection between dominance product and high-dimensional Closest Pair problems.

1.4 Diameter Spanners

In Chapter 6 we initiate the study of *Diameter Spanners*, described below. This chapter is based on the article [65] by the author and Keerti Choudhary on *Extremal Distance Spanners*. Since in [65] there are further results that we do not include in this thesis, we could simplify the presentation of the proofs of the theorems in Chapter 6 compared to their corresponding proofs in [65].

A *spanner* (also known as *distance spanner*) of an undirected graph $G = (V, E)$ is a subgraph $H = (V, E_H \subseteq E)$ that approximately preserves all the pairwise distances between vertices in the underlying graph G. Formally, H is a t-spanner of G iff for any pair of vertices $u, v \in V$, $d_H(u,v) \leqslant t \cdot d_G(u,v)$, where $d_H(u,v)$ and $d_G(u,v)$ are the distances between u and v in H and G, respectively. The parameter t is referred to as the *stretch factor* of H. Given an undirected graph G and a stretch factor t, a "good t-spanner" of G refers to a t-spanner that has a significantly

smaller (by a polynomial factor) set of edges than G has (i.e., significantly sparser than G).

Spanners were first introduced and studied in the 80s [22, 133, 134]. Althöfer *et al.* [15] showed that any undirected weighted graph with n vertices has a $(2k-1)$-spanner of with $O(n^{1+1/k})$ edges, for any integer $k > 0$. Assuming a widely-believed girth conjecture of Erdös [84], this stretch-size trade-off is essentially optimal.

Besides being theoretically interesting, spanners have numerous applications in different areas of computer science, such as distributed systems, communication networks and efficient routing schemes [16, 69, 70, 96, 97, 135, 141, 147], motion planning [68, 73], approximating shortest paths [66, 67, 79], and distance oracles [30, 148].

For directed graphs, the notion of spanners is far less understood. That is because we cannot have sparse spanners for general directed graphs. Even when the underlying graph is strongly connected, there exists graphs with $\Omega(n^2)$ edges such that excluding even a single edge from the graph results in a distance spanner with stretch as high as the diameter. In such a scenario, for directed graphs, a natural direction to study is construction of sparse subgraphs that approximately preserves the graph diameter. This property is captured by the notion of *t-diameter spanner*. Given a directed graph $G = (V, E)$, a subgraph $H = (V, E_H \subseteq E)$ is defined to be a t-diameter spanner iff the diameter $\mathrm{diam}(H)$ of H is at most t times the diameter $\mathrm{diam}(G)$ of G.

For $t = 2$ it is easy to construct such a subgraph H with $O(n)$ edges, as described in Chapter 6. This brings us to the following central question.

Question. Given a directed graph $G = (V, E)$, and a stretch factor $t < 2$, can we construct a t-diameter spanner $H = (V, E_H \subset E)$? If so, how small can $|E_H|$ be? and what is the trade-off between t and $|E_H|$?

In Chapter 6 we show the following.

1. For any unweighted directed graph G with n vertices, we can compute a subgraph H that is a $(3/2)$-diameter spanner of G, such that H has $O\left(n^{3/2}\sqrt{\log n}\right)$ edges.

2. For any unweighted directed graph G with n vertices and diameter D, we can compute a subgraph H that is a $(5/3)$-diameter spanner of G, such that H has $O\left(D^{1/3}n^{4/3}\log^{2/3} n\right)$ edges. This is sparser than the above $(3/2)$-diameter spanner for $D = o\left(\sqrt{n/\log n}\right)$.

3. Given an unweighted directed graph $G = (V, E)$ with n vertices, for any $\delta, \varepsilon \in [0, 1]$, we can compute a subgraph $H = (V, E_H \subseteq E)$ satisfying one of the following. Either

   (a) H is a $(1+\delta)$-diameter spanner of size $O(n^{2-\varepsilon}\log^{1-\varepsilon} n)$, or

   (b) H is a $(2-\delta)$-diameter spanner of size $O(n^{1+\varepsilon}\log^{\varepsilon} n)$.

For simplicity, the results given above are stated without runtime bounds (which are given in Chapter 6), and for unweighted directed graphs. Nevertheless, our diameter spanner constructions

support directed graphs with bounded positive edge-weights. If G is edge-weighted with weights taken from the interval $[1, W]$, then the stretch of H will only increase by an additional additive W term, on top of the multiplicative stretch factor t from the unweighted case.

In [65] we show that the bounds from 1 and 2 are tight, and we also study other types of extremal-distance spanners, such as *eccentricity-spanners* and *radius-spanners*. Additionally, we show in [65] how to maintain extremal-distance spanners in dynamic settings. We do not include these results in this thesis and refer the reader to [65] for further details.

We believe that extremal-distance spanners are interesting mathematical objects in their own right. Nevertheless, such a sparsification of graphs suffices for many of the original applications of the well-studied standard graph spanners mentioned above, such as in communication networks, facility location problem, routing, etc. In particular, diameter-spanners with a sparse set of edges are good candidates for backbone networks [96].

Chapter 2

Preliminaries and Techniques

2.1 Preliminaries and Notations

Linear Decision Trees. In Chapter 3 and Section 4.5 of Chapter 4 we study the complexity of problems in the *linear decision tree* model of computation, both deterministic and randomized variants. This model is often used to measure the number of *comparison* queries that an algorithm executes.

Consider an input $x = (x_1, \ldots, x_n)$ of n real numbers, for a problem, and consider a Boolean function f on x. An *r-linear decision tree* for f is a tree for which each node is labeled with a linear expression in x with at most r terms, and has two outgoing edges, labeled 0 and 1. The computation on input x for f proceeds at each node that it reaches by inspecting the sign of its corresponding r-linear expression. If the sign is positive the computation continues in the subtree reached by talking the 1-edge, otherwise it continues in the subtree reached by talking the 0-edge. Thus, the input x follows a path through the tree. Each leaf stores a 0/1 value. We say that a linear decision tree t *decides* f iff for every feasible input x, there is such a path in t, so that the value $f(x)$ is the one stored at the leaf that x reaches.

The complexity of the tree t is its depth $\mathsf{depth}(t)$, namely, the maximum length of a path in t from the root to a leaf. The *r-linear decision tree complexity* of the function f is

$$\mathcal{D}(f) = \min_{t \in T} \mathsf{depth}(t),$$

where T is the set of all r-linear decision trees that decide f.

The definition above refers to the deterministic setting. For the randomized setting, let $\mathcal{P}$ be a probability distribution over a set of r-linear decision trees $\mathcal{T}$ that decide a particular function f. $\mathcal{P}(t)$ is the probability that tree t is chosen from this distribution. For a particular input x, let $\mathrm{cost}(t, x)$ be the length of the path in t from the root to a leaf, following the branching operations on the input x. Denote the expected number of branching operations (sign tests) a tree chosen from $\mathcal{T}$ will make on input x by

$$\mathrm{cost}(\mathcal{P}, x) = \sum_{t \in \mathcal{T}} \mathcal{P}(t)\,\mathrm{cost}(t, x).$$

The *randomized r-linear decision tree complexity* of f is

$$\mathcal{R}(f) = \min_{\mathcal{P}} \max_{x} \mathrm{cost}(\mathcal{P}, x).$$

It is easy to observe that, for any function f, $\mathcal{R}(f) \leqslant \mathcal{D}(f)$. For more details on the decision tree model and its variants see Arora and Barak [21, Chapter 12].

Model of Computation. In Chapters 3 and 4, when we analyze the *time complexity* of our algorithms, we use a simplified Real RAM model of computation. In this simplified model, "truly real" numbers are subject only to the unit-time operations: addition/subtraction and comparison. In all other respects, the machine behaves like a $w = O(\log n)$-bit word-RAM with the standard repertoire of unit-time AC^0 operations, such as bitwise Boolean operations, and left and right shifts.

In Chapter 5, we use the standard Real RAM model, which includes also multiplication and subtraction of real numbers as unit-time operations. For more details on the Real RAM model (also known as the *uniform model*) see Preparata and Shamos [136, Page 28].

Notations. The following are useful notations we use throughout the thesis. Additional notations that are more problem-specific are defined within the corresponding chapters where they are applied.

- We denote by $[N] = \{1, \ldots, \lceil N \rceil\}$, the set of the first $\lceil N \rceil$ natural numbers, for any $N \in \mathbb{R}^+$.
- For a point $p \in \mathbb{R}^d$, we denote by $p[k]$ the k-th coordinate of p, for $k \in [d]$.
- The $\widetilde{O}(\cdot)$ notation is similar to the standard $O(\cdot)$ notation, but hides polylogarithmic factors.
- In the context of matrix multiplication, ω denotes the exponent of multiplying two square matrices of the same size. That is, two $n \times n$ matrices can be multiplied in $O(n^\omega)$ time. The current best known bound for ω is $\omega < 2.373$ [125]
- Given an algorithm A with runtime $T(n)$ on an input of size n, we say that
  - T (resp., A) is *strongly subquadratic* iff $T(n) = O(n^{2-\varepsilon})$, for some $\varepsilon > 0$.
  - T (resp., A) is *mildly subquadratic* iff $T(n) = o(n^2)$ and $T(n) = \omega(n^{2-\varepsilon})$, for every $\varepsilon > 0$. A typical mildly subquadratic bound is $O\left(n^2/\operatorname{polylog}(n)\right)$.
  - Analogously to the above, we use the terms *strongly subcubic* and *mildly subcubic*, with exponent 3 instead of 2.

2.2 Techniques

We present here techniques that will repeatedly appear in most of our results in Chapter 3 and Chapter 4. Other techniques that are more problem-specific will be described within the corresponding chapters where they are applied.

The Four Russians. In our algorithms in Chapters 3 and 4, we use variants of the so-called "Method of the Four Russians" [20] (named after the cardinality and nationality of its inventors), which sometimes can be exploited to improve algorithms that involve a matrix structure, such as dynamic programming algorithms. This method was originally used to improve the computation of the transitive closure of a graph. Since then it was adapted to improve many other algorithms, and in particular to improve running times by polylogarithmic factors. The basic idea is to decompose an $n \times n$ matrix into $(n/g)^2$ small sub-matrices (boxes), each of size $g \times g$. Then the hard challenge is to find a way to efficiently solve a corresponding sub-problem in each of these boxes, and obtain the solution for the original problem by combining the answers from the sub-problems, achieving an overall improved runtime. Usually a preprocessing is carried out to help us solve each of the small subproblems in a more efficient way. Since in many cases the preprocessing procedure takes exponential time, the improvement is typically conditioned to choosing the parameter g to be quite small, such as a suitable fractional power of $\log n$.

Fredman's Technique for All-Pairs-Shortest-Paths and Sorting $X + Y$. Computing $(\min, +)$-matrix multiplication (also known as $(\min, +)$-product) is one of the most studied problems in algorithm design, gaining its popularity by being closely related to computing all-pairs-shortest-paths (APSP) in real-weighted directed graphs. Formally, let A and B be $n \times d$ and $d \times n$ real-valued matrices, respectively. The $(\min, +)$-product of A by B is the $n \times n$ matrix C whose elements are given by

$$c_{i,j} = \min_{1 \leqslant k \leqslant d} \{a_{i,k} + b_{k,j}\}, \quad \text{for } i, j \in [n].$$

Naively, this can be computed in $O(dn^2)$ time. A weighted directed graph on n vertices can be encoded as an $n \times n$ matrix $W = (w_{i,j})$ in which $w_{i,j}$ is the weight of edge (i, j) if it exists, $w_{i,i} = 0$, and $w_{i,j} = +\infty$ otherwise. It is easy to see that the matrix W^n, the n-th power of W with respect to $(\min, +)$-product, contains the distances between all pairs of vertices in the graph (assuming there a no negative cycles). Clearly, W^n can be computed by executing $\lceil \log n \rceil$ $(\min, +)$-products. In fact, it is possible to compute W^n in essentially the same time required for just one $(\min, +)$-product (see Aho *et al.* [10, Section 5.9]). To this date it is a prominent open problem whether this matrix can be computed in $O(n^{3-\varepsilon})$ time, for some $\varepsilon > 0$.

Fredman showed, in a classical article from 1976 [93], that the number of comparisons needed to compute the $(\min, +)$-product of two $n \times n$ real-valued matrices is significantly smaller than n^3, and also showed the first algorithm for this problem that runs in mildly subcubic runtime. Specifically, Fredman showed that the 4-linear decision tree complexity of the $(\min, +)$-product problem is only $O(n^{2.5})$. He then used this decision tree coupled with the Four Russians technique to "shave" polylogarithmic factors off the naive cubic runtime bound. Since Fredman's pioneering technique is a basic component in our approach, we briefly overview his result for $(\min, +)$-product.

Given two $n \times n$ real-valued matrices A and B, our goal is to compute the $(\min, +)$-product matrix C of A by B. (We assume here that both matrices are of size $n \times n$, i.e., $d = n$, for simplicity.) First, we partition the $(\min, +)$-product problem into smaller subproblems. Let $g \leqslant n$ be some parameter that we will fix later. Assume for simplicity that n/g is an integer. We partition the matrix A into n/g smaller matrices $A_1, \ldots, A_{n/g}$, each of size $n \times g$ (i.e., n rows and g columns), and we partition the matrix B into n/g matrices $B_1, \ldots, B_{n/g}$, each of size $g \times n$ (i.e., g rows and n columns). For each $i, j \in [n/g]$, let $A_i[\ell, \cdot]$ be the set of elements in the ℓ-th row of A_i, and let $B_j[\cdot, m]$ be the set of elements in the m-th column of B_j.

We sort the pairwise difference set $A_i[\ell, \cdot] - A_i[\ell, \cdot] = \{a - a' \mid a, a' \in A_i[\ell, \cdot]\}$, for every $\ell \in [n]$. Then we merge these sorted sets, obtaining the sorted sets

$$S_i = \left\{a - a' \mid a, a' \in A_i[\ell, \cdot], \quad \text{for some } \ell \in [n]\right\},$$

for each $i \in [n/g]$. Similarly, we obtain the sorted sets

$$T_j = \left\{b - b' \mid b, b' \in B_j[\cdot, m], \quad \text{for some } m \in [n]\right\},$$

for each $j \in [n/g]$. In total, for all matrices A_i and B_j, this cost $O(n/g \cdot ng^2 \log n) = O(n^2 g \log n)$ comparisons. Fredman [93] showed that it actually costs only $O(n^2 g)$ comparisons, removing the $\log n$ factor, by using geometric arguments that we omit here, but explain in Chapter 3. Finally, we merge the sorted sets S_i and T_i, over all $i \in [n/g]$. Overall, this takes $O(n^2 g)$ comparisons.

Let C_ℓ be the $(\min, +)$-product of A_ℓ by B_ℓ, for each $\ell \in [n/g]$. In order to compute the entries of C_ℓ, we want to know the answers to comparisons of the form $a_{i,k} + b_{k,j} < a_{i,k'} + b_{k',j}$, where $k, k' \in [g]$, $i, j \in [n]$. Observe that

$$a_{i,k} + b_{k,j} < a_{i,k'} + b_{k',j} \iff a_{i,k} - a_{i,k'} < b_{k',j} - b_{k,j}. \tag{2.1}$$

The main point in "Fredman's trick" is that we have already computed the comparison on the right side of (2.1). Therefore, we can now compute the n/g $(\min, +)$-products matrices C_ℓ, for each pair A_ℓ, B_ℓ, $\ell \in [n/g]$, without using any further input comparisons (that is, comparisons that access the entries in A, B).

To compute the entry $c_{i,j}$ of the $(\min, +)$-product matrix C, we take the minimum value among the (i, j)-entries of the $(\min, +)$-product matrices $C_1, \ldots, C_{n/g}$. For each entry $(i, j) \in [n] \times [n]$ of C, this costs $O(n/g)$ comparisons. Thus, in total, this step costs $O(n^2 \cdot n/g) = O(n^3/g)$ comparisons. Hence, the overall number of comparisons needed is $O(n^2 g + n^3/g)$. This is minimized when $g = \sqrt{n}$, and we obtain that the number of comparisons is $O(n^{2.5})$. Since we compare linear expressions, each of at most 4 terms, the 4-linear decision tree complexity of this problem is $O(n^{2.5})$.

Fredman [93] also showed how to apply his $O(n^{2.5})$-depth decision tree in order to break the

cubic runtime bound of computing $(min,+)$-product of two $n \times n$ matrices (and thus of APSP). The idea behind his approach is to partition each of the matrices A and B into $(n/g)^2$ disjoint submatrices (boxes), each of size $g \times g$. It is easy to show that the $(min,+)$-product of A by B can be obtained by solving $(n/g)^3$ $(min,+)$-products between such boxes. This computation is done by implicitly constructing the the decision tree described above, but for matrices of size $g \times g$. Naively, constructing the tree cost $O\left(2^{g^{2.5}}\right)$ time. Using this tree we can solve each such $(min,+)$-product of two $g \times g$ matrices by running through the tree in $O(g^{2.5})$ time (the depth of the tree). Overall, we obtain an $O\left(2^{g^{2.5}} + n^3/\sqrt{g}\right)$ runtime. Choosing $g = \log^{2/5} n$ gives $O(n^3/\log^{1/5} n)$ time. Fredman actually gave the better runtime bound $O\left(n^2(\log\log n)^{1/3}/\log^{1/3} n\right)$, by showing a faster construction of the decision tree. However, we omit this construction here and refer the reader to [93] for details.

Another core problem that Fredman studied is "sorting $X + Y$". That is, given two sets of n real numbers X, Y, sort the set $X + Y = \{x + y \mid x \in X,\ y \in Y\}$. It is still a prominent open problem whether there exists a faster algorithm than the $\Theta(n^2 \log n)$-time naive algorithm. Using "Fredman's trick" combined with geometric arguments, Fredman [92] showed (also in 1976) that the 4-linear decision tree complexity of this problem is $O(n^2)$.

Remark. Recently, in a fascinating breakthrough by Kane, Lovett, and Moran [114], the above decision tree bounds were significantly improved, using different techniques. In particular, they showed that the 8-linear decision tree complexity of computing $(\min,+)$-product (or APSP) is only $O(n^2 \log^2 n)$, and this complexity is only $O(n \log^2 n)$ for sorting X+Y. They also showed (by using similar techniques) that the 6-linear decision tree complexity of 3SUM is only $O(n \log^2 n)$; this significantly improved (in a bit stronger model) our $O(n^{3/2})$ randomized 4-linear decision tree complexity bound from Chapter 3 (which improved the $O(n^{3/2}\sqrt{\log n})$ bound of Grønlund and Pettie [104]). See [114] and Chapter 3 for more details.

Generalized Fredman's Trick. In our improved DTW and GED algorithms in Chapter 4 we extend Fredman's trick to a more general setting of two linear expressions. The generalized trick is that

$$a_1 - b_1 + \cdots + a_r - b_r < a'_1 - b'_1 + \cdots + a'_t - b'_t \tag{2.2}$$

if and only if

$$a_1 + \cdots + a_r - a'_1 - \cdots - a'_t < b_1 + \cdots + b_r - b'_1 - \cdots - b'_t. \tag{2.3}$$

Here too, sorting separately the left-hand expressions and the right-hand expressions in inequalities of the form of (2.3) allows us to obtain, at no extra cost in the linear decision tree model, the results of the comparisons in the inequalities of the form of (2.2). See Chapter 4 for more details.

Chan's Dominance Reporting Technique. Chan [53] introduced a mechanism that combines Fredman's trick with a geometric domination technique (see below), which he initially used for improving the time complexity of APSP by an additional polylogarithmic factor. Later, Bremner *et al.* [37] used this Fredman-Chan mechanism to improve decision tree complexity bounds and polylogarithmic runtime factors, for restricted variants of sorting $X + Y$ and 3SUM, as well as other problems. Similarly, Grønlund and Pettie [104] used this combined mechanism to obtain an $O(n^2/\operatorname{polylog}(n))$-time 3SUM algorithm.

Given a finite set $P = \{p_1, \ldots, p_n\}$ of points in $\mathbb{R}^d$ such that each point is colored red or blue, the *bichromatic dominating pairs reporting* problem is to report all the pairs $(i, j) \in [n]^2$ such that p_i is red, p_j is blue, and p_i dominates p_j, i.e., p_i is greater than p_j at each of the d coordinates. A standard divide-and-conquer algorithm by Preparata and Shamos [136, p. 366] runs in $O(|P| \log^d |P| + K)$ time, where K is the output size. Chan [53] provided an improved runtime analysis for this algorithm that yields a *strongly* subquadratic time bound in the number of points (excluding the cost of reporting the output) when the dimension is $d = O(\log |P|)$, with a sufficiently small constant of proportionality.

Lemma 2.2.1 (Chan [53])**.** *Given a finite set $P = \{p_1, \ldots, p_n\}$ of points in $\mathbb{R}^d$ such that each point is colored red or blue, one can report all pairs $(i, j) \in [n]^2$, such that p_i is red, p_j is blue, and $p_i[k] > p_j[k]$ for every $k \in [d]$, in time $O(c_\varepsilon^d |P|^{1+\varepsilon} + K)$, where K is the output size, $\varepsilon \in (0, 1)$ is an arbitrary prespecified parameter, and $c_\varepsilon = 2^\varepsilon/(2^\varepsilon - 1)$.*

Throughout this thesis, we invoke Lemma 2.2.1 many times, with $\varepsilon = 1/2, c_\varepsilon \approx 3.42$, and $d = \delta \log n$, where $\delta > 0$ is a sufficiently small constant, chosen to make the overall running time of all the invocations dominated by the total output size. In other words, in this case the runtime of Chan's procedure is linear in the output size, with a strongly-subquadratic overhead. Another recent work by Chan [54] gives an efficient (linear in the output size, with a subquadratic overhead, but not strongly subquadratic as before) dominance reporting algorithm for dimensions slightly larger, up to $d = O(\log^2 n/(\log \log n)^3)$. Unfortunately, due to other bottlenecks in our algorithms, this improved algorithm does not improve the overall runtime of our results.

Since we invoke the algorithm from Lemma 2.2.1 many times throughout this thesis, we feel it is important to mention that the algorithm is quite simple, thereby its runtime complexity does not involve large constants nor subtle implementation. For the sake of completeness we provide here the algorithm and the proof for Lemma 2.2.1.

The Divide-and-Conquer Algorithm. Given a finite set $P = \{p_1, \ldots, p_n\}$ of red/blue points in $\mathbb{R}^d$, the divide-and-conquer algorithms works as follows. If $d = 0$, simply report every pair of red/blue points, so assume $d \geqslant 1$. Find the median h on the values of last coordinate (the d-th coordinate) of the n points in $O(n)$ time [36]. Partition P into two disjoint sets P_L, P_R, each

of size at most $\lceil n/2 \rceil$, where

$$P_L = \{p \in P \mid p[d] \leqslant h\}$$
$$P_R = \{p \in P \mid p[d] > h\}.$$

If (p_i, p_j) is a dominating pair, then either both p_i and p_j are in P_L, or both are in P_R, or one (the blue point) is in P_L and the other (the red point) is in P_R. By executing three corresponding recursive calls we find the dominating pairs of each of the three kinds. Two recursive calls, each on at most $\lceil n/2 \rceil$ points in $\mathbb{R}^d$, are on all points in P_L and all the points in P_R, respectively. The third recursive call is on all blue points in P_L and all red points in P_R, after stripping their last coordinate d; that is, at most n points in $\mathbb{R}^{d-1}$.

Excluding the cost of reporting the output, the runtime of this algorithm is bounded by $T_d(n)$, where

$$T_0(n) = T_d(1) = 0$$
$$T_d(n) \leqslant 2T_d(n/2) + T_{d-1}(n) + \gamma n,$$

for some constant $\gamma \in \mathbb{R}$. Put $\varepsilon \in (0, 1)$, and $c_\varepsilon = 2^\varepsilon/(2^\varepsilon - 1)$. We prove by induction that $T_d(n) \leqslant c_\varepsilon^d n^{1+\varepsilon} - \gamma n$. Clearly, this bound holds for $T_0(n)$ and $T_d(1)$. Assume that the bound holds for $T_{d'}(n)$, for all $d' < d$, and for $T_d(n')$, for all $n' < n$. Then

$$\begin{aligned} T_d(n) &\leqslant 2\left(c_\varepsilon^d (n/2)^{1+\varepsilon} - \gamma n/2\right) + \left(c_\varepsilon^{d-1} n^{1+\varepsilon} - \gamma n\right) + \gamma n \\ &= \left(c_\varepsilon^d/2^\varepsilon + c_\varepsilon^{d-1}\right) n^{1+\varepsilon} - \gamma n \\ &= \left(1/2^\varepsilon + 1/c_\varepsilon\right) c_\varepsilon^d n^{1+\varepsilon} - \gamma n \\ &= c_\varepsilon^d n^{1+\varepsilon} - \gamma n. \end{aligned}$$

This completes the proof of Lemma 2.2.1. □

Fractional Cascading. Fractional cascading was introduced by Chazelle and Guibas [60, 61], for solving the *iterative search problem*, defined as follows. Let U be an ordered universe of keys. Define a *catalog* as a finite ordered subset of U. Given a set of k catalogs $C_1, C_2, \ldots, C_k$ over U, such that $|C_i| = n_i$ for each $i \in [k]$, and $\sum_{i=1}^k n_i = n$, the iterative search problem is to provide a data structure that supports efficient execution of queries of the form: given a query $x \in U$, return the largest value less than or equal to x in each of the k catalogs.

Fractional cascading lets one preprocess the catalogs in $O(n)$ time, using $O(n)$ storage, and answer iterative search queries in $O(\log n + k)$ time per query, improving upon the naive time bound $O(k \log n)$. This is essentially optimal in terms of query time, storage size and preprocessing

time. The idea is to maintain a sufficient number of pointers across catalogs, so that, once we have the answer c_i to a query in a catalog C_i, we can follow a pointer to an element in C_{i+1}, which is only $O(1)$ indices away from the answer $c_{i+1} \in C_{i+1}$. These pointers are constructed by processing the catalogs in reverse order, starting from C_k and ending in C_1. For $i = k-1, k-2, \ldots, 1$, we copy every r-th element of C_{i+1} into C_i, where r is some small constant (catalog C_{i+1} is already augmented by copies of elements from higher-indexed catalogs, except catalog C_k). Then, knowing the location of the query x in some C_i, we can easily retrieve the two copied elements of C_{i+1} for which x lies in between, and use this data to limit the search in C_{i+1} to only r consecutive elements.

For our 3SUM decision tree, described in Section 3.6, we develop a specialized fractional cascading data structure, based on an unusual randomized variant of fractional cascading in a grid. We believe that our technique might be useful for other problems that involve iterative search in a grid/matrix structure. The detailed description of our technique will be given in Chapter 3.

Chapter 3

3SUM, k-SUM, and Linear Degeneracy

3.1 Background

The general 3SUM problem is formally defined as

3SUM: Given a finite set $A \subset \mathbb{R}$, determine whether there exist $a, b, c \in A$ such that $a+b+c = 0$.

An equivalent variant is that the input consists of three finite sets A, B, $C \subset \mathbb{R}$ of the same size, and the goal is to determine whether there are elements $a \in A$, $b \in B$, $c \in C$ such that $a+b+c = 0$. When the sets A, B, C are not of the same size, the problem is named unbalanced 3SUM.

The 3SUM problem and its variants are among the most fundamental problems in algorithm design. Although the 3SUM problem itself does not seem to have many compelling practical implications, it has been of wide interest due to numerous problems that can be reduced from it. The notion of 3SUM-Hardness is often used to describe such problems, namely, problems that are at least as hard as 3SUM. Thus, lower bounds on 3SUM imply lower bounds on dozens of other problems. Among them are fundamental problems in computational geometry [11, 27, 95, 145], dynamic graph algorithms [5,122,137], triangle enumeration [6,122], and pattern matching [17,18, 47,64,122,153].

By the time hierarchy theorem [105], there are problems in P with complexity $\Omega(n^k)$ for every fixed k. However, given a problem in P, proving an $\Omega(n^k)$ unconditional lower bound, for any specific $k > 1$, seems far beyond the state of the art in computational complexity theory. This has led researchers to settle on conditional lower bounds, based on the conjectured hardness of certain archetypal problems, such as 3SUM, $(\min, +)$-matrix multiplication, and CNF-SAT. See [4–6,17, 23,27,38,63,95,113,122,137,138,142,153,156] for many examples of such conditional lower bounds.

In the last decades, starting with a study of Gajentaan and Overmars [95], it was conjectured that any algorithm for 3SUM requires $\Omega(n^2)$ time. However, a fairly recent breakthrough by Grønlund and Pettie [104] showed that 3SUM can be solved in subquadratic time. Specifically, they gave a *deterministic* algorithm that runs in $O(n^2(\log\log n/\log n)^{2/3})$ time, and a randomized algorithm that runs in $O(n^2(\log\log n)^2/\log n)$ expected time and with high probability. Furthermore, they showed that there is a 4-linear decision tree for 3SUM with depth $O(n^{3/2}\sqrt{\log n})$ (i.e., the depth bounds the number of branching operations, each one is based on a sign test of a linear expression with at most 4 terms). These results raised serious doubts on the optimality of many algorithms for 3SUM-Hard problems. For example, the following problems are known to be 3SUM-Hard.

1. Given an n-point set in $\mathbb{R}^2$, determine whether it contains three collinear points (Gajentaan and Overmars [95]). See Chapter 7 for a discussion on this problem.

2. Given n triangles in $\mathbb{R}^2$, determine whether their union contains a hole, or compute the area of their union [95].

3. Given two n-point sets $X, Y \subset \mathbb{R}$, each of size n, determine whether all elements in $X + Y = \{x + y \mid x \in X,\ y \in Y\}$ are distinct (Barequet and Har-Peled [27]).

4. Given two n-edge convex polygons, determine whether one can be placed inside the other via translation and rotation [27].

Problems 1 and 2 are solvable in $O(n^2)$ time (see [95]). Problems 3 and 4 are solvable in $O(n^2 \log n)$ time (see [27]). In face of the new 3SUM result of Grønlund and Pettie [104], it is natural to ask whether these bounds are optimal. However, no better bounds are currently known (in spite of the improvement in [104]). Problem 3 (or its stronger variant of sorting $X+Y$) has special importance, as it is used for basing the conditional lower bounds for the problems in [27] and in [108]; these problems are therefore also classified as "(Sorting $X + Y$)-Hard". It is a prominent long-standing open problem whether Problem 3 can be solved in $o(n^2 \log n)$ time (see [72]). As mentioned Section 2.2, in a recent breakthrough by Kane, Lovett, and Moran [114] it was showed that the 8-linear decision tree complexity of Sorting $X + Y$ is only $O(n \log^2 n)$, significantly improving (albeit under a somewhat stronger model) Fredman's $O(n^2)$ 4-linear decision bound from 1976 [92]. Yet, it is still a prominent open question whether there exists an algorithm that runs faster than the naive $\Theta(n^2 \log n)$ algorithm. Unlike many other problems, even improvements by polylogarithmic factors are unknown for this problem.

In view of the results in [104], the 3SUM conjecture has been replaced by a relaxed, modern variant, asserting that 3SUM cannot be solved in *strongly subquadratic time* (even in expectation), i.e., in $O(n^{2-\epsilon})$ time, for any $\epsilon > 0$. This conjecture is widely accepted and believed by the computer science community, and so are its implications for deriving conditional lower bounds for other problems. Abboud and Vassilevska-Williams [6] argue, based on the collective computer science community efforts, that lower bounds that are based on the relaxed 3SUM conjecture should be at least as believable as any other known conditional lower bounds for a problem in P.

This relaxed conjecture is often applied to a more restricted variant, Integer3SUM, which is defined as follows.

Integer3SUM: Given a finite set $A \subseteq \{-U, \ldots, U\} \subset \mathbb{Z}$, determine whether there exist $a, b, c \in A$ such that $a + b + c = 0$.

Based on the conjecture that Integer3SUM requires $\Omega(n^{2-o(1)})$ time, Pătraşcu [137] proved lower bounds on triangle enumeration and on various problems in dynamic data structures. Recently, the reduction techniques of Pătraşcu were extended by Kopelowitz, Pettie, and Porat [122], implying improved lower bounds for this kind of problems. Examples for lower bounds based on this conjecture include the following:

— Given an undirected m-edge graph, enumerating up to m triangles (3-cycles) requires at least

$\Omega(m^{4/3-o(1)})$ time (Pătraşcu [137]).[1]

— Given a sequence of m updates to a directed graph (edge insertions and deletions) and two specified vertices s, t, determining whether t is reachable from s after each update, requires at least $\Omega(m^{4/3-o(1)})$ time (Abboud and Williams [5]).

— Given an edge-weighted undirected graph, deciding whether it contains a zero-weight triangle, requires at least $\Omega(n^{3-o(1)})$ time (Williams and Williams [157]).

The Integer3SUM problem is clearly not harder than 3SUM; however, any other relationship between them is unknown. Unlike 3SUM, Integer3SUM can be solved using fast Fourier transform in $O(n + U \log U)$ time, which is subquadratic even for a rather large universe size U.[2] Baran, Demaine, and Pătraşcu [25] showed that using randomized universe reductions, word packing, and table lookups, Integer3SUM can be solved in $O(n^2(\log\log n/\log n)^2)$ *expected* time and with high probability, on the word-RAM, where $U = 2^w$ and $w > \log n$ is the machine word size. Recently, Chan and Lewenstein [56] showed, based on results from additive combinatorics, strongly subquadratic time algorithms for special restricted cases of Integer3SUM.

The 3SUM problem was also extensively studied in its generalized forms, k-SUM and k-variate linear degeneracy testing (k-LDT), formally defined as

k-LDT and k-SUM: Given a k-variate linear function $f(x_1, \ldots, x_k) = \alpha_0 + \sum_{i=1}^{k} \alpha_i x_i$, where $\alpha_0, \ldots, \alpha_k \in \mathbb{R}$, and a finite set $A \subset \mathbb{R}$, determine whether there exists $(x_1, \ldots, x_k) \in A^k$ such that $f(x_1, \ldots, x_k) = 0$. When f is $\sum_{i=1}^{k} x_i$ the problem is called k-SUM(when $k = 3$ we get the 3SUM problem we started with).

There are simple algorithms that solve k-LDT in time $O(n^{(k+1)/2})$ when k is odd, or $O(n^{k/2}\log n)$ when k is even; see [12]. These algorithms are based on straightforward reductions to a 2SUM problem or to an unbalanced 3SUM problem, depending on whether k is even or odd, respectively. These are currently the best known upper bounds for the running time of solving k-LDT. Erickson [85] showed that, for an even k, there is a k-linear decision tree with depth $O(n^{k/2})$, removing an $O(\log n)$ factor when comparing to the uniform model. Erickson [85] showed that any k-linear decision tree for solving k-SUM must have depth $\Omega(n^{k/2})$ when k is even and $\Omega(n^{(k+1)/2})$ when k is odd. In particular, any 3-linear decision tree for 3SUM has depth $\Omega(n^2)$. Ailon and Chazelle [12] showed that any $(2k-1)$-linear decision tree for k-SUM must have depth $\Omega(n^{1+\Omega(1)})$.

Grønlund and Pettie [104] showed that using only one more variable per comparison leads to a dramatic improvement in the depth of the tree, which significantly beats the above lower bounds. Specifically, as will be reviewed below, they showed that there is a 4-linear decision tree for 3SUM with depth $O(n^{3/2}\sqrt{\log n})$, and by the reduction from k-LDT to unbalanced 3SUM, they concluded

[1] By Kopelowitz, Pettie, and Porat [122], the exponent 4/3 is optimal if the matrix multiplication exponent ω is 2 and if 3SUM requires $\Omega(n^{2-o(1)})$ time.

[2] Erickson [85] credits R. Seidel with this Integer3SUM algorithm (taken from [104]).

that there is a $(2k-2)$-linear decision tree for k-LDT with depth $O(n^{k/2}\sqrt{\log n})$, for any odd $k \geqslant 3$. Cardinal, Iacono, and Ooms [50] showed that if we allow arbitrarily many variables in a comparison (polynomial in n), then the linear decision tree complexity of k-SUM and k-LDT is $O(n^3 \log^3 n)$. This bound was improved by Ezra and Sharir [87] to $O(n^2 \log^2 n)$.

A recent breakthrough by Kane, Lovett, and Moran [114] significantly improves these results, giving near optimal bounds by showing a $2k$-linear decision tree with depth only $O(n \log^2 n)$. Their decision tree bound is one logarithmic factor away from the well-known $\Omega(n \log n)$ algebraic decision tree lower bound of Element-Uniqueness [31], which can be easily reduced to k-SUM. Their usage of $2k$ variables is optimal, since for $(2k-1)$-linear decision tree there is an $\Omega(n^{1+\Omega(1)})$ lower bound by Ailon and Chazelle [12].

Apart from the many lower bounds obtained from the conjectured hardness of 3SUM and its variants, in recent years, many lower bounds were obtained also from two other plausible conjectures. The first is that computing the $(\min, +)$–product of two $n \times n$ matrices takes $\Omega(n^{3-o(1)})$ time (aka APSP-Hardness); see for examples [5,6,156]. The second is that CNF-SAT takes $\Omega(2^{(1-o(1))n})$ time. The latter is often referred to as the *Strong Exponential Time Hypothesis* (SETH) [110,111]. A natural question is whether any of these conjectures (3SUM, SETH, APSP) are in fact equivalent, or whether they all derive from a basic unifying hypothesis. At the current state of knowledge, there is no strong relationship between any pair of these problems, so it may be possible that any one of them could be true or false, independently of the status of the others. A recent breakthrough by Carmosino *et al.* [51] provides evidence that such a relationship is *unlikely*, based on a nondeterministic variant of SETH; see [51] for details.

3.2 Summary of Our Results and Related Work

The following theorems capture our main results.

Theorem 3.2.1. *The randomized 4-linear decision tree complexity of* 3SUM *is* $O(n^{3/2})$.

Theorem 3.2.2. *The randomized* $(2k-2)$*-linear decision tree complexity of* k-SUM *and of* k-LDT *is* $O(n^{k/2})$*, for any odd* $k \geqslant 3$.

Theorem 3.2.3. 3SUM *can be solved deterministically in* $O(n^2 \log\log n / \log n)$ *time.*

Theorem 3.2.1 and Theorem 3.2.2 improve (albeit in a randomized setting) the respective $O(n^{3/2}\sqrt{\log n})$-depth and $O(n^{k/2}\sqrt{\log n})$-depth decision trees given by Grønlund and Pettie [104]. The aforementioned recent breakthrough by Kane, Lovett, and Moran [114] gives a 6-linear decision tree for 3SUM with depth $O(n \log^2 n)$, and in general, a $2k$-linear decision tree for k-SUM, and a $(2k+2)$-linear decision tree for k-LDT, both with depth $O(kn \log^2 n)$. Viewing our (and Grønlund and Pettie's) k-SUM results for $(2k-2)$-linear decision tree, with respect to Erickson's $\Omega(n^{\lceil k/2 \rceil})$

k-linear decision tree lower bound, and the $2k$-linear decision upper bound by Kane, Lovett, and Moran [114], shows that even adding only 1 or 2 terms to each linear comparison, can significantly improve the depth of the tree.[3]

Our technique for proving Theorems 3.2.1 and 3.2.2 includes a specialized data structure, based on an unusual randomized variant of fractional cascading in a grid.

In Theorem 3.2.3 we give an actual deterministic algorithm for 3SUM that runs (in the uniform model) in $O\left(n^2 \log\log n/\log n\right)$ time. The latter improves the $O(n^2(\log\log n/\log n)^{2/3})$-time bound of Grønlund and Pettie [104], and matches the bound given by a recent independent work of Freund [94]. Both algorithms, Freund's and ours, have common high-level ideas, but ours makes a better use of the word-RAM model, and is hence somewhat simpler. [4] Recently, Chan [55] further improved this bound by presenting a deterministic algorithm for 3SUM that runs in $O\left(n^2(\log\log n)^{O(1)}/(\log n)^2\right)$ time.

3.3 The Quadratic 3SUM Algorithm and Search-Contours

We give a brief overview of the quadratic-time algorithm. We follow the implementation given by Grønlund and Pettie [104], which is slightly different from the standard approach, but is useful for the explanation of the results of [104] and of this chapter. For later references, we present the algorithm for the more general three-set version of 3SUM, as defined in the first paragraph of Section 3.1.

The algorithm runs over each $c \in C$ and searches for $-c$ in the pairwise sum $A + B$. With a careful implementation, given below, each search takes $O(|A|+|B|)$ time, for a total of $O(|C|(|A|+|B|))$ time. We view $A + B$ as being a matrix whose rows correspond to the elements of A and columns to the elements of B, both listed in increasing order. To help visualizing some steps of the algorithms, we think of the rows arranged in increasing order from top to bottom, and of the columns from left to right.

1. Sort A and B in increasing order as $A(0), \ldots, A(|A|-1)$ and $B(0), \ldots, B(|B|-1)$.

2. For each $c \in C$,

2.1. Initialize lo $\leftarrow 0$ and hi $\leftarrow |B|-1$.

2.2. Repeat:

2.2.1. If $-c = A(\text{lo}) + B(\text{hi})$, report witness "$(A(\text{lo}), B(\text{hi}), c)$".

2.2.2. If $-c > A(\text{lo}) + B(\text{hi})$ then increment lo, otherwise decrement hi.

2.3. Until lo $= |A|$ or hi $= -1$.

3. If no witnesses were found report "no witness."

[3] We note that our results have been obtained and published long before the results of Kane, Lovett, and Moran [114]; see an initial arXiv version of our work in [101].

[4] The independent result of Freund [94] was brought to our attention after the completion of an initial version our work; see arXiv version of our work in [101].

372	389	407	439	454	480	534	609	635	655
397	414	432	464	479	505	559	634	660	680
420	437	455	487	502	528	582	657	683	703
442	459	477	509	524	550	604	679	705	725
478	495	513	545	560	586	640	715	741	761
500	517	535	567	582	608	662	737	763	783
523	540	**558**	590	605	631	685	760	786	806
548	565	583	615	630	656	**710**	785	811	831
594	611	629	661	676	702	756	831	857	877
627	644	662	694	709	735	789	864	890	910

Figure 3.3.1: The sky-blue colored entries form CONTOUR(710), and the purple colored ones form CONTOUR(558); A shared cell is shown in green. The lighter colors (light purple and light sky-blue) depict their *partial contour*, that is, the positions of the contours where we chose to increase the lo index ("go down") in the matrix while searching our element. All the elements in the matrix whose values are in $[558, 710)$ are enclosed between these two contours, excluding the partial contour of 558 and including the partial contour of 710.

The correctness easily follows from the fact that each row and column of $A + B$ is sorted in increasing order. Note that when a witness is discovered in Step 2.2.1, the algorithm can stop right there. However, in order to simplify future definitions and explanations, this implementation continues to search for more witnesses. After finding a witness we will always choose to decrement hi. This choice will be made throughout this chapter.

Define the *contour* of x, CONTOUR$(x, A+B)$, (CONTOUR(x), when the context is clear) to be the sequence of positions (lo, hi) encountered while searching for x in $A+B$ in the preceding algorithm. Lemma 3.3.1 is straightforward.

Lemma 3.3.1. *For $x < y \in \mathbb{R}$,* CONTOUR*(x) lies fully above* CONTOUR*(y); that is, for each $i, i', j \in \{0, \ldots, n-1\}$, if $(i, j) \in$* CONTOUR*(x) and $(i', j) \in$* CONTOUR*(y), then $i \leqslant i'$.*

By Lemma 3.3.1 a pair of contours can overlap, but never cross. Moreover, Lemma 3.3.1 implies a weak total order relation $\prec$ on the contours, which corresponds to the order between the searched elements, such that $x < y$ iff CONTOUR$(x) \prec$ CONTOUR(y), where the latter relation means that the two contours satisfy the properties stated in the lemma; see Figure 3.3.1.

3.4 Fredman's Trick, Pairwise Sums, and Fractional Cascading

We give an overview of the techniques we use in this chapter. This includes techniques from Fredman's prominent works from 1976 [92, 93], some of which were also discussed in Section 2.2.

For our result, we will develop a special randomized variant of *fractional cascading* (Chazelle and Guibas [60, 61]). In this section we also briefly review the standard fractional cascading method, to set the infrastructure upon which we will later develop our specialized variant.

Recall *Fredman's trick* described in Section 2.2: the trivial observation that $a + b < a' + b'$ iff $a - a' < b' - b$. We will liberally exploit this observation (trick) throughput this chapter.

Fredman showed that, given n numbers whose sorted order is one of $\Pi \leqslant n!$ realizable permutations, they can be sorted using a linear number of comparisons when Π is sufficiently small. More generally, we have:

Lemma 3.4.1 (Fredman 1976 [92]). *A list L of n numbers, whose sorted order is one of Π possible permutations, can be sorted with $2n + \log \Pi$ pairwise comparisons.*

Sorting Pairwise Sums and its Geometric Interpretation. Fredman describes the relation between the complexity of hyperplane arrangements and the decision tree complexity of sorting pairwise sums. Grønlund and Pettie [104] use similar arguments in their 3SUM decision tree, where they sort pairwise sums. Specifically, given two sets $A = \{a_1, \ldots, a_n\}$ and $B = \{b_1, \ldots, b_n\}$, each of n (distinct) real numbers, define the pairwise sum $A + B = \{a_i + b_j \mid i, j \in [n]\}$. The input A, B can be regarded as a point $p = (a_1, \ldots, a_n, b_1, \ldots, b_n) \in \mathbb{R}^{2n}$. The points in $\mathbb{R}^{2n}$ that agree with a fixed permutation of $A + B$ form a convex cone bounded by the set H of the $\binom{n^2}{2}$ hyperplanes $x_i + y_j - x_k - y_l = 0$, for $i, j, k, l \in [n]$, $(i, j) \neq (k, l)$. The number of possible sorted orders of $A + B$ is therefore bounded by the number of regions (of all dimensions) in the arrangement $\mathcal{A}(H)$ of H. As shown by Buck [44], the number of regions of dimension $k \leqslant d$ in an arrangement of m hyperplanes in $\mathbb{R}^d$ is at most

$$\binom{m}{d-k}\left(\binom{m-d+k}{0} + \binom{m-d+k}{1} + \cdots + \binom{m-d+k}{k}\right).$$

Thus, the number of regions of all dimensions is $O(m^d)$ (where the constant of proportionality is actually independent of d). Hence, the number of possible sorting permutations of $A + B$ is $O\left((n^4)^{2n}\right) = O(n^{8n})$. One can also construct the hyperplane arrangement explicitly in $O(m^d)$ time by a standard incremental algorithm [76]. The following lemma, taken from Grønlund and Pettie [104], extends this analysis by considering only a subset of these hyperplanes, and is an immediate consequence of these observations.

Lemma 3.4.2. *Let $A = \{a_1, \ldots, a_n\}$ and $B = \{b_1, \ldots, b_n\}$ be two sets, each of n real numbers, and let $F \subseteq [n]^2$ be a set of positions in the $n \times n$ grid. The number of realizable orders of $(A + B)_{|F} := \{a_i + b_j \mid (i, j) \in F\}$ is $O\left(\binom{|F|}{2}^{2n}\right)$, and therefore $(A + B)_{|F}$ can be sorted with at most $2|F| + 4n \log |F| + O(1)$ comparisons.*

In Lemma 3.4.2, the case $F = [n]^2$ goes back to Fredman [92], who showed that $O(n^2)$ comparisons suffice to sort $A + B$.

For some of the algorithms presented and reviewed in this chapter, it is important to assume that the elements of the pairwise sum are distinct, and therefore have a unique sorting permutation.

When numbers do appear multiple times, a unique sorting permutation can be obtained by breaking ties consistently (see [104] for details).

Iterative Search and Fractional Cascading. In our decision tree construction for 3SUM, we aim to speed-up binary searches of the same number, in many sorted sets. We will use for this task a special randomized variant of *fractional cascading*, which will be described in Section 3.6. First, we briefly recall the standard fractional cascading technique, which was introduced by Chazelle and Guibas [60, 61] and briefly reviewed in Section 2.2, for solving the *iterative search problem*, defined as follows. Let U be an ordered universe of keys. Define a *catalog* as a finite ordered subset of U. Given a set of k catalogs $C_1, C_2, \ldots, C_k$ over U, such that $|C_i| = n_i$ for each $i \in [k]$, and $\sum_{i=1}^{k} n_i = n$, the iterative search problem is to provide a data structure that supports efficient execution of queries of the form: given a query $x \in U$, return the largest value less than or equal to x in each of the k catalogs.

Fractional cascading lets one preprocess the catalogs in $O(n)$ time, using $O(n)$ storage, and answer iterative search queries in $O(\log n + k)$ time per query (as opposed to the trivial $O(k \log n)$ bound). This is essentially optimal in terms of query time, storage size and preprocessing time. The idea is to maintain a sufficient number of pointers across catalogs, so that, once we have the answer c_i to a query in a catalog C_i, we can follow a pointer to an element in C_{i+1}, which is only $O(1)$ indices away from the answer $c_{i+1} \in C_{i+1}$.

In order to construct these pointers query time, the fractional cascading method expands each catalog C_i to an augmented catalog L_i, starting with L_k and proceeding backwards down to L_1. L_k is the same as C_k, and for each $1 \leqslant i < k$, L_i is formed by merging C_i with every second element[5] of L_{i+1}. The items in C_i that were not originally in the catalog are marked as synthetic keys. From each synthetic key in C_i we add a bridge (pointer) to its copy in L_{i+1}. Using these bridges and additional pointers, from each real key to the two consecutive synthetic keys nearest to it, one can follow directly from each element of L_i (real or synthetic) to the elements in L_{i+1} nearest to it, and by construction, the gap between these elements is 2. Thus, given a query number x, after spending $O(\log n)$ time for searching it in L_1, it takes only $O(1)$ time to locate x in each subsequent catalog, for a total of $O(\log n + k)$ time, as desired. With some additional simple calculations, one can show that the total number of elements that are copied through the catalogs is only $O(n)$, and that the cost of doing it is also $O(n)$.

Fractional cascading can also be extended to support a collection of catalogs stored at the vertices of a directed acyclic graph (DAG), and each query searches with some specified element x through the catalogs stored at the nodes of some specified path in the DAG. In more detail, a *catalog graph* is a DAG in which each vertex stores a catalog (ordered list of keys). A query consists

[5] More generally, every r-th element, for a constant r; the choice of r provides a trade-off between the constants in the storage and query time.

of a key x and a path π in the graph, and the goal is to search with x in the catalog of each node of π. When the maximum in/out degree Δ of the catalog graph is constant, fractional cascading can be extended to this scenario, with the same bounds as before (albeit with larger constants of proportionality). Here too each catalog C_v at a node v, is expanded into an augmented catalog L_v, and each L_v passes to its predecessors every 2Δ-th element (instead of every second element in the earlier case, where Δ was 1). See [60, 61] for more details on the construction of the data structure, proof of correctness, and performance analysis.

In our algorithms we will present a special non-standard variant of this method, that lets us preserve the advantages of the other techniques (most notably, Fredman's trick) that we use.

3.5 Grønlund and Pettie's Subquadratic Decision Tree for 3SUM

In this section we give an overview of the subquadratic decision tree of Grønlund and Pettie [104]. In the following sections we show how their ideas can be extended and combined with additional techniques, to yield our improved results.

We give an overview of the subquadratic decision tree for 3SUM over a single input set A of size n, taken from [104], resulting in a 4-linear decision tree with depth $O(n^{3/2}\sqrt{\log n})$. This is shown by an algorithm that performs at most $O(n^{3/2}\sqrt{\log n})$ comparisons, where each comparison is a sign test of a linear expression with at most 4 terms.

1. Sort A in increasing order as $A(0), \ldots, A(n-1)$. Partition A into $\lceil n/g \rceil$ groups $A_1, \ldots, A_{\lceil n/g \rceil}$, each of at most g consecutive elements, where g is a parameter that we will fix later, by setting $A_i := \{A((i-1)g), \ldots, A(ig-1)\}$, for each $i = 1, \ldots, \lceil n/g \rceil - 1$, where $A_{\lceil n/g \rceil}$ may be smaller. The first and last elements of A_i are $\min(A_i) = A((i-1)g)$ and $\max(A_i) = A(ig-1)$.

2. Sort $D := \bigcup_{i \in [n/g]} (A_i - A_i) = \{a - a' \mid a, a' \in A_i \text{ for some } i\}$.

3. For all $i, j \in [n/g]$, sort $A_{i,j} := A_i + A_j = \{a + b \mid a \in A_i \text{ and } b \in A_j\}$.

4. For k from 1 to n,

4.1. Initialize lo $\leftarrow 1$ and hi $\leftarrow \lceil n/g \rceil$.

4.2. Repeat:

4.2.1. If $-A(k) \in A_{\text{lo,hi}}$, report "solution found" and halt.

4.2.2. If $\max(A_{\text{lo}}) + \min(A_{\text{hi}}) > -A(k)$ then decrement hi, otherwise increment lo.

4.3. Until lo $= \lceil n/g \rceil + 1$ or hi $= 0$.

5. Report "no solution" and halt.

This algorithm can be generalized in a straightforward way to solve the (unbalanced) three-set version of 3SUM. For the easy argument concerning the correctness of the algorithm, see [104].

With a proper choice of g, the decision tree complexity of the algorithm is $O(n^{3/2}\sqrt{\log n})$. Step 1 requires $O(n \log n)$ comparisons. By Lemma 3.4.2, Step 2 requires $O(n \log n + |D|) = O(n \log n + gn)$ comparisons to sort D. By Fredman's trick, if $a, a' \in A_i$ and $b, b' \in A_j$, $a + b < a' + b'$ holds iff $a - a' < b' - b$, and both sides of this inequality are elements of D. Thus, Step 3 does not require any real input comparisons, given the sorted order on D. For each iteration of the outer loop (in Step 4) there are at most $2\lceil n/g \rceil$ iterations of the inner loop (Step 4.2), since each iteration ends by either incrementing lo or decrementing hi. In Step 4.2.1 we can determine whether $-A(k)$ is in $A_{\text{lo,hi}}$ using binary search, in $\log |A_{\text{lo,hi}}| = O(\log g)$ comparisons. The total number of comparisons is thus $O(n \log n + gn + (n^2 \log g)/g)$, which becomes $O(n^{3/2}\sqrt{\log n})$ when $g = \sqrt{n \log n}$.

3.6 Improved Decision Trees for 3SUM, k-SUM, and k-LDT

In this section we show that the randomized decision tree complexity of 3SUM is $O(n^{3/2})$, and more generally, that the randomized decision tree complexity of k-LDT is $O(n^{k/2})$, for any odd $k \geqslant 3$. This bound removes the $O(\sqrt{\log n})$ factor in Grønlund and Pettie's decision tree bound (albeit under a randomized decision tree model). We show this result by giving a randomized algorithm that constructs a $(2k-2)$-linear decision tree whose expected depth is $O(n^{k/2})$.

To make the presentation more concise, we present it for the variant where we have three different sets A, B, C of n real numbers each, and we want to determine whether there exist $a \in A$, $b \in B$, $c \in C$, such that $a + b + c = 0$.

As in the previous section, we partition each of the sorted sets A and B into $\lceil n/g \rceil$ blocks, each consisting of g consecutive elements, denoted by $A_1, \ldots, A_{n/g}$, and $B_1, \ldots, B_{n/g}$, respectively. As above, but with a slightly different notation, we consider the $n \times n$ matrix $M = M^{AB}$, whose rows (resp., columns) are indexed by the (sorted) elements of A (resp., of B), so that $M(k, \ell) = a_k + b_\ell$, for $k, \ell \in [n]$. The partitions of A and of B induce, as before, a partition of M into n^2/g^2 boxes $M_{i,j}$, for $i, j \in [n/g]$, where $M_{i,j}$ is the portion of M with rows in A_i and columns in B_j.

Fredman's trick, combined with Lemma 3.4.2, allows us to sort all the boxes $M_{i,j}$ with $O(ng)$ comparisons. Since the problem is fully symmetric in A, B, C, we can also define analogous matrices M^{AC} and M^{BC}, constructed in the same manner for the pairs A, C and B, C, respectively, partition each of them into n^2/g^2 boxes, and obtain the sorted orders of all the corresponding boxes, with $O(ng)$ comparisons.

The crucial (costliest) step in Grønlund and Pettie's algorithm, which we are going to improve, is the searches of the elements of $-C$ in M^{AB}. For each $c \in C$, let $\sigma(c) = \text{CONTOUR}(-c)$ denote the staircase path contour of $-c$, as defined before Lemma 3.3.1. The length of $\sigma(c)$ is thus at most $2n$. Each of the paths $\sigma(c)$ visits some of the boxes $M_{i,j}$, and the index pairs (i, j) of these boxes also form a staircase pattern, as in the preceding sections. The number of boxes that a contour $\sigma(c)$ visits is at most $2\lceil n/g \rceil$. For each $c \in C$, the sequence of boxes that $\sigma(c)$ visits can be

obtained by invoking (an appropriate variant of) Step 4 of the algorithm in Section 3.5, excluding the binary search in Step 4.2.1. The total running time of this step, over all $c \in C$, is $O(n^2/g)$.

The paths $\sigma(c)$, being contours, have the structure given in Lemma 3.3.1, including the weak total order $\prec$ between them. Thus, we obtain the following.

Corollary 3.6.1. *For each box $M_{i,j}$, let $C_{i,j}$ denote the set of elements of C whose paths $\sigma(c)$ traverse $M_{i,j}$. Then $C_{i,j}$ is a contiguous subsequence of C.*

Put $\kappa_{i,j} := |C_{i,j}|$. Then we clearly have $\sum_{i,j\in[n/g]} \kappa_{i,j} = O(n^2/g)$. That is, the average number of elements of C that visit a box is $O(g)$, and, for each box, these elements form a contiguous subsequence of C, as just asserted in Corollary 3.6.1. Let $C^*_{i,j}$ denote the contiguous sequence of indices in C of the elements of $C_{i,j}$. That is, $C_{i,j} = \{c_\ell \mid \ell \in C^*_{i,j}\}$. With all these observations, we next proceed to derive the mechanism by which, for each box $M_{i,j}$, we can efficiently search in $M_{i,j}$ with the (negations of the) $\kappa_{i,j}$ corresponding elements of $C_{i,j}$.

We apply a special variant of fractional cascading. The twist is in the way in which we construct the augmented catalogs. Note that in each box $M_{i,j}$, we have g^2 elements of the form $a_k + b_\ell$, but only $2g$ indices k, ℓ. We want to sample elements from a box, and then copy and merge them into its right and top neighbor boxes. However, in order to be able to use Fredman's trick, we have to preserve the property that the number of element-indices (rows and columns) in each box stays $O(g)$ (unlike a naive implementation of fractional cascading, where it is enough that each augmented box be of size $O(g^2)$).

Thus, we sample elements from A (row elements) and elements from B (column elements) separately. We construct augmented sets $A'_1, \ldots, A'_{\lceil n/g \rceil}$. Starting with $A'_{\lceil n/g \rceil} = A_{\lceil n/g \rceil}$, we sample each element in $A'_{\lceil n/g \rceil}$ with probability $p = \frac{1}{4}$. Each sampled element is copied and merged with $A_{\lceil n/g \rceil - 1}$, and we denote by $A'_{\lceil n/g \rceil - 1}$ the new augmented set. Then we sample each element from $A'_{\lceil n/g \rceil - 1}$ with the same probability p, copy and merge the sampled elements with $A_{\lceil n/g \rceil - 2}$, obtaining $A'_{\lceil n/g \rceil - 2}$, and continue this process until the augmented set A'_1 is constructed. Similarly, we construct the augmented sets $B'_1, \ldots, B'_{\lceil n/g \rceil}$, but we do it in the opposite direction, starting from $B'_1 = B_1$ and ending with $B'_{\lceil n/g \rceil}$. Clearly, similar to standard fractional cascading, the expected size of each of the augmented sets is $O(g)$, as the expected numbers of additional elements placed in each box form a convergent geometric series. Now we sort

$$D_{A'} = \bigcup_{i\in[n/g]} \left(A'_i - A'_i\right) = \{a - a' \mid a, a' \in A'_i \text{ for some } i\}.$$

In each $A'_i - A'_i$, the expected number of elements $a_k - a_{k'}$ is $O(g^2)$, and the expected number of element indices k, k' is only $O(g)$. Thus, by Lemma 3.4.2, we can sort $D_{A'}$ with expected $O(ng)$ comparisons. Similarly, we sort $D_{B'} = \bigcup_{j\in[n/g]} \left(B'_j - B'_j\right)$ with the same expected number of comparisons. Then, we form the union $D' = D_{A'} \cup D_{B'}$ and obtain its sorted order by merging

$D_{A'}$ and $D_{B'}$. This costs additional expected $O(ng)$ comparisons. By Fredman's trick, from the sorted order of D', we can obtain the sorted order of the augmented boxes $A'_i + B'_j$, for each $i, j \in [n/g]$, without further comparisons (i.e., at no extra cost in our model).

With these augmentations of the row and column blocks, the matrix M^{AB} itself is now augmented, such that each modified box $M_{i,j} = A'_i + B'_j$ receives some fraction of the rows from the box $M_{i+1,j}$ below it, and a fraction of the columns from the box $M_{i,j-1}$ to its left. Each box $M_{i,j}$ corresponds to a vertex in the catalog graph, and it has (at most) two outgoing edges, one to the vertex that corresponds to $M_{i+1,j}$ and one to the vertex that corresponds to $M_{i,j-1}$ (it also has at most two incoming edges). Clearly this is a DAG with maximum in/out degree $\Delta = 2$, which is why we sampled $\frac{1}{2\Delta} = \frac{1}{4}$ of the rows/columns in each step. We complete the construction of this special fractional cascading data structure, by adding the appropriate pointers, similar to what is done in a standard implementation of fractional cascading (see Section 3.4). This does not require any further comparisons (and thus is free of cost in our model), since the pointers from synthetic keys (the sampled elements) to real keys, and pointers from real keys to synthetic keys, depend only on the sorted order of the augmented sets $M_{i,j}$, which we already computed. So the overall expected number of comparisons needed to construct this data structure is still $O(ng)$.

Consider now the search with $-c$, for some $c \in C$. Assume that the search has just visited some box $M_{i',j'}$, and now proceeds to search in box $M_{i,j}$. Thus, either $(i,j) = (i'+1, j')$ or $(i,j) = (i', j'-1)$. Assume, without loss of generality, that $(i,j) = (i'+1, j')$; a symmetric argument applies when $(i,j) = (i', j'-1)$, using columns instead of rows. In this case, the fractional cascading mechanism has sampled, in a random manner, an expected quarter of the rows of (the already augmented) $M_{i,j}$ and has sent them to $M_{i',j'} = M_{i-1,j}$. The output of the search at $M_{i-1,j}$, if $-c$ was not found there, includes two pointers to the largest element ξ^- of $M_{i,j}$ that is smaller than $-c$, and to the smallest element ξ^+ of $M_{i,j}$ that is larger than or equal to $-c$. We need to go over the elements in the sorted order of $M_{i,j}$ that lie between ξ^- and ξ^+, and locate $-c$ among them. If we do not find it, we get the two consecutive elements that enclose $-c$, retrieve from them two corresponding pointers to a pair of elements in the next box to be searched, that enclose $-c$ between them, and continue the fractional cascading search in the next box, in between these elements.

The main difficulty in this approach is that the number of elements of $M_{i,j}$ between ξ^- and ξ^+ might be large, because there might be many elements between ξ^- and ξ^+ in rows that we did not sample, and then we have to inspect them all, slowing down the search.

Concretely, in this case we sample, in expectation, a quarter of the rows of $M_{i,j}$ (recall that we actually sample the rows from an augmented box that has already received data from previous boxes, but let us ignore this issue for now). Collectively, these rows contain (in expectation) $\Theta(g^2)$ elements of $M_{i,j}$, but we have no good control over the size of the gaps of non-sampled elements between consecutive pairs of sampled ones. This is because there might be rows that we did not

60	70	80	90	100	110	120	130	140	150
160	170	180	190	200	210	220	230	240	250
260	270	280	290	300	310	320	330	340	350

Figure 3.6.1: An expensive step in the fractional cascading search: Assume that only the first and third rows (appearing in gray) are sent to the preceding box (above the current one), and that we search with $-c = 205$. The previous search locates $-c$ between $\xi^- = 150$ and $\xi^+ = 260$, say, and now we have to examine about half of the entire second row to locate $-c$ in the current box.

sample which contain many elements between ξ^- and ξ^+, and searching through such large gaps could slow down the procedure considerably. See Figure 3.6.1 for an illustration. (For a normal fractional cascading, this would not be an issue, but here the peculiar and implicit way in which we sample elements has the potential for creating this problem.)

We handle this problem as follows. Consider any gap of non-sampled elements of $M_{i,j}$ between a consecutive pair $\xi^- < \xi^+$ of sampled ones. We claim that the expected number of rows to which these elements belong is $O(1)$. Note that this is why we needed randomization; if we sampled every 4th element in A_i and B_j deterministically then the rows-gap between ξ^- to ξ^+ could be much larger, in all boxes $M_{i,j}$; see Figure 3.6.2 for an illustration.

Indeed, the probability to have k distinct rows in such a gap, conditioned on the choice of the row containing ξ^-, is $\frac{1}{4}\left(\frac{3}{4}\right)^k$, which follows since each row is sampled *independently* with probability 1/4. Hence, the (conditionally) expected row-size of a gap is

$$\sum_{k \geqslant 0} k \frac{1}{4} \left(\frac{3}{4}\right)^k = O(1),$$

as claimed. Denote this expected value as β. In other words, for each $c \in C_{i,j}$, let R_c be the set of rows that show up in the gap between the corresponding elements ξ^- and ξ^+ for c. The overall expected size $\sum_{c \in C_{i,j}} |R_c|$ is thus $\beta |C_{i,j}|$.

Fix a box $M_{i,j}$. For each $\ell \in C^*_{i,j}$ and for each $k \in R_{c_\ell}$, we need to locate $-c_\ell$ among the elements in row k of $M_{i,j}$. That is, we need to locate $-c_\ell$ among the elements of the set $a_k + B'_j$. This however is equivalent to locating $-a_k - c_\ell$ among the elements of B'_j.

We therefore collect the set S of all the sums $-a_k - c_\ell$, for $\ell \in C^*_{i,j}$ and $k \in R_{c_\ell}$, and recall that in expectation we have $|S| = O(|C_{i,j}|)$. The crucial observation is that we already (almost) know the order of these sums. To make this statement more precise, partition, in the usual manner, the sorted sequence C into $\lceil n/g \rceil$ blocks $C_1, C_2, \ldots, C_{\lceil n/g \rceil}$, each consisting of g consecutive elements in the sorted order. As mentioned earlier, a symmetric application of Fredman's trick allows us to obtain the sorted order of each box of the form $A'_i + C_j$, using a total of $O(ng)$ comparisons.

The number of (consecutive) blocks C_s of C that overlap $C_{i,j}$ is $t_{i,j} \leqslant \lceil \kappa_{i,j}/g \rceil + 2$. Moreover, each sum in S belongs to $-(A'_i + C_s)$ for one of these $t_{i,j}$ blocks. Since each of these sets is already sorted, we extract from them (with no extra comparisons) the elements of S as the union of $t_{i,j}$

-100	-99	$\cdots$	-91	67	68	$\cdots$	100
49	50	$\cdots$	58	216	217	$\cdots$	260
50	51	$\cdots$	69	217	218	$\cdots$	261
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
90	91	$\cdots$	99	257	258	$\cdots$	291
200	201	$\cdots$	209	367	368	$\cdots$	400

Figure 3.6.2: Example of a 44×44 box with sorted rows and columns, which is the sumset matrix of the ordered sets $\{0, 149, 150, \ldots, 190, 300\}$ and $\{-100, -99, \ldots, -91, 67, 68, \ldots, 100\}$. Say that we searched for element 150 in the previous box, which is augmented by every 4th row of this box. Since elements 100 and 200 are consecutive in this box, the fractional cascading mechanism locates element 150 between $\xi^- = 100$ and $\xi^+ = 200$, which sit in the first and last rows of this box (in grey color), respectively. A similar scenario can appear in all the $g \times g$ boxes, and for any choice of g, if we sample rows (resp., columns) deterministically. Hence, we needed randomization, in order to obtain that the rows-gap (resp., columns-gap) between ξ^- and ξ^+ is $O(1)$ in expectation.

sorted sequences $S_{i,s}$, where $S_{i,s} \subset -(A'_i + C_s)$ for each s. Arguing as above, the expected size of $S_{i,s}$ is $\beta|C_s| = O(g)$. We now merge each of the sorted sequences $S_{i,s}$ with B'_j, using an expected $O(g)$ comparisons for each merge. As a result, each sum $-a_i - c_\ell$ is located between two consecutive elements $b^-_{i,\ell} < b^+_{i,\ell}$ of B'_j. In other words, for each $c_\ell \in C_{i,j}$, we have at most $|R_{c_\ell}|$ candidates for being the largest element of $M_{i,j}$ that is smaller than $-c_\ell$ (these are the elements $a_i + b^-_{i,\ell}$, for $i \in R_{c_\ell}$), and we select the largest of them, requiring no comparisons, as these are all elements of the already sorted $A'_i + B'_j$. In the same manner, we find the smallest element of $M_{i,j}$ that is larger than $-c_\ell$. Having found these two elements, we can proceed to search $-c_\ell$ in the next box, using the appropriate pointers created by the fractional cascading mechanism (see Section 3.4).

The overall number of merges is

$$\sum_{i,j \in [n/g]} t_{i,j} \leqslant \sum_{i,j \in [n/g]} (\kappa_{i,j}/g + 2) = O(n^2/g^2),$$

and each of them costs $O(g)$ expected comparisons, for a total of $O(n^2/g)$ expected comparisons. Thus, the overall number of expected comparisons is $O(ng + n(\log g + n/g))$, which is $O(n^{3/2})$, when $g = \sqrt{n}$. This completes the proof of Theorem 3.2.1. □

k-SUM and k-LDT

The standard algorithm for k-variate linear degeneracy testing (k-LDT) for odd $k \geqslant 3$, is based on a straightforward reduction to an instance of unbalanced 3SUM, where $|A| = |B| = n^{(k-1)/2}$ and $|C| = n$; see [12] and [104]. The analysis of this section also applies for unbalanced 3SUM, and directly implies that it can be solved by using an expected number of

$$O\left(g\left(|A| + |B| + |C|\right) + |C|\left((|A| + |B|)/g + \log g\right)\right)$$

comparisons, where the first term is the cost of sorting the blocks of (the augmented) M^{AB}, M^{AC}, and M^{BC}, and where the second term is the cost of the fractional cascading searches. We have $|A| = |B| = n^{(k-1)/2}$, $|C| = n$, so by choosing $g = \sqrt{n}$, the bound becomes $O(n^{k/2})$. Thus, the randomized decision tree complexity of k-LDT (and thus of k-SUM) is $O(n^{k/2})$, for any odd $k \geqslant 3$, as stated in Theorem 3.2.2. □

3.7 Subquadratic Algorithms for 3SUM

In this section we use the technique of Chan [53] for dominance reporting, described in Section 2.2. We remind that Chan was the first to show the idea of combining dominance reporting with Fredman's trick, for improving polylogarithmic factors, where the problem he applied it on was the general version of the famous all-pairs-shortest-paths (APSP) problem. This mechanism was later extended by Bremner *et al.* [37] and used to show mildly subquadratic algorithms for various problems, such as (min, +)-Convolution, and restricted variants of sorting $X+Y$ and 3SUM. Later, Grønlund and Pettie [104] used similar techniques to give the first mildly subquadratic algorithm for the general 3SUM problem, as mentioned earlier.

Recall the following bichromatic dominance reporting result of Chan [53], described in Section 2.2.

Lemma 3.7.1 (Chan [53]). *Given a finite set $P = \{p_1, \ldots, p_n\}$ of points in $\mathbb{R}^d$, each is colored red or blue, one can report all pairs $(i, j) \in [n]^2$, such that p_i is red, p_j is blue, and $p_i[k] > p_j[k]$, for every $k \in [d]$, in time $O(c_\varepsilon^d |P|^{1+\varepsilon} + K)$, where K is the output size, $\varepsilon \in (0, 1)$ is an arbitrary prespecified parameter, and $c_\varepsilon = 2^\varepsilon/(2^\varepsilon - 1)$.*

Throughout this section, we invoke Lemma 3.7.1 a large number of times, with $\epsilon = 1/2, c_\epsilon \approx 3.42$, and $d = \delta \log n$, where $\delta > 0$ is sufficiently small to make the overall running time of all the invocations dominated by the total output size; see below for details.

Grønlund and Pettie [104] present two subquadratic algorithms for 3SUM, one is relatively simple, and the second one has slightly faster runtime but is more involved. Both algorithms are based on the decision tree algorithm described in Section 3.5, except that they use a much smaller value of g, in order to make the overall running time subquadratic. We give here a brief overview of the simpler algorithm. Their second algorithm has some common high-level features with our algorithm, presented in the next section, but our algorithm processes the data in a different, simpler, and more efficient manner.

Note that, sorting the set D in of Grønlund and Pettie's decision tree that is presented in Section 3.5 lets one obtain a comparison-efficient way to sort each of $A_{i,j}$. However, the actual running time is even more than quadratic, when all operations are considered. When the boxes $A_{i,j}$ are small enough, Grønlund and Pettie showed that it is possible to obtain the sorted orders

in each of the $(n/g)^2$ boxes, in (all inclusive) mildly subquadratic time.

Specifically, the algorithm enumerates *every* permutation $\pi : [g^2] \to [g]^2$, where $\pi = (\pi_r, \pi_c)$ is decomposed into row and column functions $\pi_r, \pi_c : [g^2] \to [g]$, so that $\pi(k) = (\pi_r(k), \pi_c(k))$, for each $k \in [g^2]$. By definition, π is the correct sorting permutation for the box $A_{i,j}$ iff $A_{i,j}(\pi(t)) < A_{i,j}(\pi(t+1))$ for all $t \in [g^2 - 1]$. Since $A_{i,j} = A_i + A_j$ this inequality can also be written

$$A_i(\pi_r(t)) + A_j(\pi_c(t)) < A_i(\pi_r(t+1)) + A_j(\pi_c(t+1)).$$

By Fredman's trick this is equivalent to saying that the (red) point p_j dominates the (blue) point q_i, where

$$\begin{aligned} p_j &= \big(A_j(\pi_c(2)) - A_j(\pi_c(1)), \ldots, A_j(\pi_c(g^2)) - A_j(\pi_c(g^2-1))\big) \\ q_i &= \big(A_i(\pi_r(1)) - A_i(\pi_r(2)), \ldots, A_i(\pi_r(g^2-1)) - A_i(\pi_r(g^2))\big)\,. \end{aligned}$$

Invoking $(g^2)!$ times the bichromatic dominance reporting algorithm from Lemma 3.7.1, we find, for each π, all such dominating pairs, that is, all boxes $A_{i,j}$ sorted by π. Note that, for each pair of indices j, i, there is exactly one invocation of the dominating pairs procedure in which the corresponding points p_j and q_i are such that p_j dominates q_i; this follows because we assume that all elements of $A_{i,j}$ are distinct (see a previous remark concerning this issue). This is important in order to keep the overall output size subquadratic.

By Lemma 3.7.1 and the remarks just made, the time to report all red/blue dominating pairs, over all $(g^2)!$ invocations of the procedure, is $O\Big((g^2)!c_\epsilon^{g^2-1}(2n/g)^{1+\epsilon} + (n/g)^2\Big)$, where the last term is the total size of the outputs (one for each box $A_{i,j}$). For $\epsilon = 1/2$ and $g = \frac{1}{2}\sqrt{\log n/\log\log n}$, the first term turns out to be negligible. The total running time is therefore $O((n/g)^2)$ for dominance reporting, and $O(n^2 \log g/g) = O\big(n^2(\log\log n)^{3/2}/(\log n)^{1/2}\big)$ for the binary searches in Steps 4.1–4.3. By Lemma 3.4.2 and Fredman [92], there are at most $O(g^{8g})$ realizable permutations of $A_{i,j}$ (which is much smaller than $(g^2)!$). Hence, this algorithm can be slightly improved to run in $O\big(n^2 \log\log n/\sqrt{\log n}\big)$ time, by constructing the arrangement of the hyperplanes (as defined in Section 3.4) explicitly, extracting from it the relevant permutations, and choosing $g = \Theta(\sqrt{\log n})$.

3.8 Improved Deterministic Subquadratic 3SUM Algorithm

In the algorithm of Grønlund and Pettie, described above, the boxes $A_{i,j}$ are sorted by using Fredman's trick to transform each permutation into a sequence of $g^2 - 1$ comparisons, which are then resolved by the bichromatic dominance reporting algorithm. Consequently, the space into which these sequences are encoded is of dimension $g^2 - 1$, thus having the $c_\epsilon^{g^2-1}$ factor in the running time of the bichromatic dominance reporting algorithm forced us to use $g = \Theta(\sqrt{\log n})$.

In order to use a larger value of g, we want to reduce the dimension of the points. Thus, we want to find a method to sort smaller sets, while still be able to do the binary searches in each box in $O(\log g)$ time.

Fix some $k \in [g^2]$, and let $(l, m) \in [g]^2$ be a point in the $g \times g$ grid, such that $A_{i,j}(l, m)$ is the k-th smallest element in the box $A_{i,j}$. Let $\tau = (\tau_{\mathrm{r}}, \tau_{\mathrm{c}})$ denote $\textsc{contour}(A_{i,j}(l, m))$, and enumerate its elements as $\tau(1), \tau(2), \ldots$. Recall that, if $\tau(t+1) = \tau(t) + (0, -1)$ then $A_{i,j}(l, m) \leqslant A_{i,j}(\tau(t))$, otherwise, if $\tau(t+1) = \tau(t) + (1, 0)$ then $A_{i,j}(l, m) > A_{i,j}(\tau(t))$. The contour starting position is $(\tau_{\mathrm{r}}(0), \tau_{\mathrm{c}}(0)) = (1, g)$, and it ends at the first t^* for which $\tau(t^*) = (g+1, \cdot)$ or $\tau(t^*) = (\cdot, 0)$. Recall that a pair of contours $\textsc{contour}(\mathrm{x})$ and $\textsc{contour}(\mathrm{y})$ in $A_{i,j}$ may overlap, but can never cross; see Lemma 3.3.1.

Let $\tau'(A_{i,j}(l, m)) = (\tau'(0), \tau'(1), \ldots, \tau'(t_{\tau'})) \subseteq \tau = \textsc{contour}(A_{i,j}(l, m))$ be the *partial contour* of τ, defined as the subsequence of positions of τ at which we chose to go down (i.e., increment lo in the quadratic algorithm). The sequence $\tau'(A_{i,j}(l, m))$ is of length at most g, since it contains at most one element of each row (at which we go down, by incrementing lo); see Figure 3.3.1.

Since the rows of $A_{i,j}$ are sorted, each position $(a, b) \in \tau'(A_{i,j}(l, m))$ satisfies

$$A_{i,j}(a, b') < A_{i,j}(l, m) \quad \text{for every } b' \leqslant b. \tag{3.1}$$

$$A_{i,j}(a, b'') \geqslant A_{i,j}(l, m) \quad \text{for every } b'' > b. \tag{3.2}$$

Thus, $\tau'(A_{i,j}(l, m))$ partitions $A_{i,j}$ into two sets: $A_{(i,j)L} \subset (-\infty, A_{i,j}(l, m))$ consists of the elements at positions in $\{(a, b') \mid (a, b) \in \tau'(A_{i,j}(l, m)) \text{ and } b' \leqslant b\}$, and $A_{(i,j)R} \subset [A_{i,j}(l, m), \infty)$ consists of the elements at positions in $\{(a, b'') \mid (a, b) \in \tau'(A_{i,j}(l, m)) \text{ and } b'' > b\}$. Rows succeeding the last row of $\tau'(A_{i,j}(l, m))$ are fully contained in $A_{(i,j)R}$. By construction, $A_{(i,j)L}$ is the set of all elements in $A_{i,j}$ that are smaller than the k–th smallest element $A_{i,j}(l, m)$, so the considerations just made, provide the structure of this set. See Figure 3.3.1 for an illustration.

Each partial contour τ' is thus a sequence of positions in $A_{i,j}$ such that (i) the rows containing these positions form a contiguous subsequence, starting from the first row of $A_{i,j}$, (ii) each row in this subsequence has exactly one entry of τ', and (iii) the sequence of columns of the entries of τ' is weakly monotone decreasing: if (a, b) and $(a+1, b')$ are in τ' then $b' \leqslant b$. Any sequence τ' that satisfies properties (i)–(iii) is called a *valid partial contour*. Note that a valid partial contour depends only on the positions of the contour in the box $A_{i,j}$, and not on the actual values of the entries of $A_{i,j}$.

Let μ' be some valid partial contour, as just defined, over a $[g] \times [g]$ position set, such that the sum of the column indices of positions in μ' is exactly k. Write $\mu' = (\mu'_{\mathrm{r}}, \mu'_{\mathrm{c}})$, as was done for permutations above, so that μ'_{r} gives the row indices of the elements of μ', and μ'_{c} gives their column indices. Denote by $t' \leqslant g$ the number of positions in μ'.

For given indices ℓ, m, we can determine, using (3.1) and (3.2), whether $\mu' = \tau'(A_{i,j}(l, m))$,

by testing, for each $t \in [t']$, whether $A_{i,j}(\mu'(t)) < A_{i,j}(l,m)$ and $A_{i,j}(\mu'(t)+(0,1)) > A_{i,j}(l,m)$, except for the t_0 for which $A_{i,j}(\mu(t_0)) = A_{i,j}(l,m)$, since then the second inequality becomes an equality; this takes at most $2t'-2$ comparisons. By Fredman's trick, and since $A_{i,j} = A_i + A_j$, this can be restated, that $\mu' = \tau'(A_{i,j}(l,m))$ iff the (red) point p_j dominates the (blue) point q_i, where

$$\begin{aligned} p_j &= \big(\ldots,\ A_j(m) - A_j(\mu'_{\mathrm{c}}(t)),\ A_j(\mu'_{\mathrm{c}}(t+(0,1)) - A_j(m)),\ \ldots\big) \\ q_i &= \big(\ldots,\ A_i(\mu'_{\mathrm{r}}(t)) - A_i(l),\quad A_i(l) - A_i(\mu'_{\mathrm{r}}(t)),\ \ldots\big)\,, \end{aligned} \tag{3.3}$$

where the $2t'-2$ coordinates are indexed in pairs by $t \in [t'] - \{t_0\}$.

We regard each box $A_{i,j}$ as being partitioned into $h = g \log g$ sets $A_{(i,j)1}, \ldots, A_{(i,j)h}$, each of size at most $s = g/\log g$, such that for $k \in [h]$, $A_{(i,j)k}$ is the set of all elements that are at least the $(k-1)s$–smallest element, and smaller than the ks–smallest element in $A_{i,j}$. Our goal is to compute, for each box $A_{i,j}$, the positions of the elements of the sets $A_{(i,j)1}, \ldots, A_{(i,j)h}$, and the correct sorting permutation of each of them, as well as to determine, for each $k \in [h]$, the position of the ks–smallest element in $A_{i,j}$.

Fix $k \in [h]$. We enumerate all the pairs of realizable valid partial contours $\mu'_{(k-1)s}$, μ'_{ks}, such that (i) $\mu'_{(k-1)s}$ lies to the left and above μ'_{ks}, and (ii) the sums of the column indices of their entries are $(k-1)s$ and ks, respectively. Let S_k be the set of positions enclosed between the two partial contours $\mu'_{(k-1)s}$ and μ'_{ks}, excluding $\mu'_{(k-1)s}$ and including μ'_{ks}. For each $A_{i,j}$, we want to identify the pair $(\mu'_{(k-1)s}, \mu'_{ks})$, for which $\mu'_{(k-1)s}$ and μ'_{ks} are the partial contours of the $(k-1)s$-smallest and the ks-smallest elements of $A_{i,j}$, respectively. Thus S_k is the set of the s positions of the elements of $A_{i,j}$ that are larger or equal to the $(k-1)s$-smallest element and smaller than the ks-smallest element. These are the positions of the set $A_{(i,j)k}$. See Figure 3.3.1 for an illustration.

It is easily seen that there are at most 2^{4g} pairs of sequences $(\mu'_{(k-1)s}, \mu'_{ks})$, and there is only one unique pair of valid partial contours $(\mu'_{(k-1)s}, \mu'_{ks})$ that satisfy all the above requirements for a specific box $A_{i,j}$, as there is only one $(k-1)s$–smallest element and only one ks–smallest element in $A_{i,j}$ (assuming, as above, that all the elements of $A_{i,j}$ are distinct). We enumerate all pairs of positions $P_1, P_2 \in [g]^2$, such that $P_1 \in \mu_{(k-1)s}$ and $P_2 \in \mu_{ks}$ (recall that μ' is a partial contour of some contour μ, where μ is uniquely determined from μ', see Figure 3.3.1). There are at most $(2g)^2 = 4g^2$ such positions. We also enumerate every realizable permutation $\pi : [s] \to S_k$ of the elements at positions in S_k (where, for each $A_{i,j}$, we want to identify the permutation that sorts its elements at the positions of S_k). The number of permutations is bounded trivially by $s! = (g/\log g)!$.

We now extend the points defined in (3.3), to make them encode additional information, as follows. For every tuple $(P_1, P_2, \mu'_{(k-1)s}, \mu'_{ks}, \pi)$, we create red points $\{p_j\}_{j\in[n/g]}$ and blue points $\{q_i\}_{i\in[n/g]}$ in $\mathbb{R}^{4(t'-1)+s-1}$, such that a red point p_j dominates a blue point q_i iff the following

conditions hold. (i) $\mu'_{(k-1)s} = \tau'(A_{i,j}(P_1))$, (ii) $\mu'_{ks} = \tau'(A_{i,j}(P_2))$, and (iii) π is the unique sorting permutation of the portion of $A_{i,j}$ with indices in S_k. The first $4t' - 4$ coordinates encode the correctness of $\mu'_{(k-1)s}$ and μ'_{ks} (as in (3.3), using the positions P_1, P_2 as those defining the respective contours), and the last $s - 1$ coordinates encode the correctness of π, as in Section 3.7 but for a permutation of size at most $s = g/\log g$. We do this $h = g \log g$ times, for each $k \in [h]$.

According to Lemma 3.7.1, the overall time to report all bichromatic dominating pairs is

$$O\left(h \cdot 2^{4g} g^2 s! c_\epsilon^{4(g-1)+s-1} (n/g)^{1+\epsilon} + h(n/g)^2\right).$$

The second term is the output size, because for each of the $(n/g)^2$ boxes $A_{i,j}$, there will be exactly h dominating pairs, one for each pair of consecutive partial contours, as above. By fixing $\epsilon = 1/2$ and $g = d \log n$ with a small enough d, the first term will be negligible and the runtime will be dominated by the output size $O(h(n/g)^2) = O(n^2 \log g/g) = O(n^2 \log\log n/\log n)$.

We can now search an element x in a box $A_{i,j}$, in $O(\log g)$ time. We first do a binary search, in $O(\log g)$ time, over the h positions storing the ks–smallest element of $A_{i,j}$, for $k \in [h]$ (we have already computed their positions, and, by definition, they are already sorted). This will give us a single set $A_{(i,j)k}$ that can possibly contain x. Then we do another binary search in $A_{(i,j)k}$, also in $O(\log g)$ time, as we already found its sorting permutation earlier. (Note that each such permutation π is of length at most $g/\log g$, and of values from $[g]^2$. Thus, by our earlier choice of g, π can be stored in a machine word of size $O(\log n)$, and be accessed in $O(1)$ time.) Each element $-A(k)$ is being searched in at most $2\lceil n/g \rceil$ boxes (as in Steps 4.1–4.3 of Grønlund and Pettie's decision tree, described in Section 3.5). Hence, the total running time of the algorithm is $O(n^2 \log g/g) = O(n^2 \log\log n/\log n)$ *deterministic* time. This proves Theorem 3.2.3. □

Chapter 4

Geometric Pattern Matching Algorithms

4.1 Dynamic Time Warping and Geometric Edit Distance

Dynamic Time Warping (DTW) and Geometric Edit Distance (GED) are basic similarity measures between curves or general temporal sequences (e.g., time series) that are represented as sequences of points in some metric space (X, dist). The DTW and GED measures are massively used in various fields of computer science and computational biology. Consequently, the tasks of computing these measures are among the core problems in P. Despite extensive efforts to find more efficient algorithms, the best-known algorithms for computing the DTW or GED between two sequences of points in $X = \mathbb{R}^d$ are long-standing dynamic programming algorithms that require quadratic runtime, even for the one-dimensional case $d = 1$, which is perhaps one of the most used in practice.

In this chapter, we present deterministic algorithms that run in $O\left(n^2/\log\log n\right)$ time, for computing DTW or GED between two sequences of n points in $\mathbb{R}$. This result breaks the nearly 50 years old quadratic time bound for this problems. Our algorithms can be extended to work also for higher dimensional spaces $\mathbb{R}^d$, for any constant d, when the underlying distance-metric dist is polyhedral (e.g., L_1, L_∞).

4.1.1 Problem Statements

Let $A = (p_1, \ldots, p_n)$ and $B = (q_1, \ldots, q_m)$ be two sequences of points (also referred to as curves) in some metric space (X, dist). A *coupling* $C = (c_1, \ldots, c_k)$ between A and B is an ordered sequence of distinct pairs of points from $A \times B$, such that $c_1 = (p_1, q_1)$, $c_k = (p_n, q_m)$, and

$$c_r = (p_i, q_j) \Rightarrow c_{r+1} \in \big\{(p_{i+1}, q_j),\ (p_i, q_{j+1}),\ (p_{i+1}, q_{j+1})\big\},$$

for $r < k$. The DTW-distance between A and B is

$$\mathsf{dtw}(A, B) = \min_{C:\, coupling} \left\{ \sum_{(p_i, q_j) \in C} \mathsf{dist}(p_i,\ q_j) \right\}. \tag{4.1}$$

A coupling C for which the above sum is minimized is called an *optimal coupling*. The DTW problem is to compute $\mathsf{dtw}(A, B)$, and sometimes also an optimal coupling C.

A *monotone matching* $\mathcal{M} = \{m_1, \ldots, m_k\}$ between A and B is a set of pairs of points from $A \times B$, such that any two pairs (p_i, q_j), $(p_{i'}, q_{j'}) \in \mathcal{M}$ satisfy that $i < i'$ iff $j < j'$. This also implies that each point in A is matched with at most one point in B and vice versa (possibly some points in $A \cup B$ do not appear in any pair of the matching); see Figure 4.1.1 for an illustration. Note the difference from coupling (defined above), which covers all points of $A \cup B$ and a point can appear in multiple pairs of the coupling. The cost of $\mathcal{M}$ is defined to be the sum of all the distances between the points of each pair in $\mathcal{M}$, plus a *gap* penalty parameter $\rho \in \mathbb{R}$, for each point in $A \cup B$ that does not appear in any pair of $\mathcal{M}$.

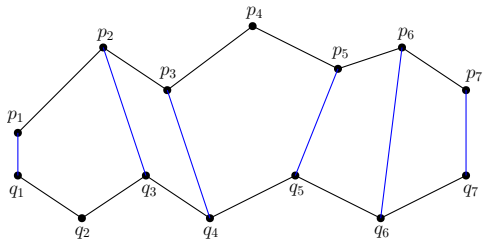

Figure 4.1.1: Example of a monotone matching (in blue) between two polygonal curves (represented by point-sequences) in the plane.

The Geometric Edit Distance (GED) between A and B is

$$\mathsf{ged}(A,B) = \min_{\mathcal{M}} \left\{ \Big(\sum_{(p_i,q_j)\in\mathcal{M}} \mathsf{dist}(p_i,\, q_j) \Big) + \rho\,(n+m-2|\mathcal{M}|) \right\}, \tag{4.2}$$

where the minimum is taken over all sets of monotone matchings $\mathcal{M}$ in the complete bipartite graph $A \times B$. A monotone matching $\mathcal{M}$ for which the above sum is minimized is called an *optimal matching*. The GED problem is to compute $\mathsf{ged}(A,B)$, and sometimes also an optimal matching. More sophisticated gap penalty functions have been proposed [75], but for this presentation, we focus on the standard linear gap penalty function, although our presented algorithm supports more complex gap penalty, such as taking ρ to be a linear function in the coordinates of the points of $A \cup B$. By tuning ρ correctly, meaningful matchings can be computed even when faced with outlier points that arise from measurement errors or short deviations in otherwise similar trajectories.

The DTW-distance and GED are massively used in dozens of applications, such as speech recognition, geometric shape matching, DNA and protein sequences, protein backbones, matching of time series data, GPS, video and touch screen authentication trajectories, music signals, and countless data mining applications; see [48, 71, 77, 118–120, 132, 140, 151] for some examples.

The best-known worst-case running times for solving DTW or GED are given by long-standing simple dynamic programming algorithms that require $\Theta(nm)$ time. We review the standard quadratic-time DTW and GED algorithms in Section 4.2 and 4.4, respectively.

DTW was perhaps first introduced as a speech discrimination method [150] back in the 1960's. GED is a natural extension of the well-known string version of Edit Distance, however, the subquadratic-time algorithms for the string version do not seem to extend to GED (see below).

A popular setting in both theory and practice is the one-dimensional case $X = \mathbb{R}$ (under the standard Euclidean distance $\mathsf{dist}(x,y) = |x-y|$). Even for this special case, no subquadratic-time algorithms have been known. We consider this case throughout most of the chapter.

4.1.2 Summary of Our Results and Related Works

Prior Results. Since no subquadratic-time algorithm is known for computing DTW, a number of heuristics were designed to speed up its exact computation in practice; see Wang *et al.* [152] for

a survey. Very recently, Agarwal *et al.* [8] gave a near-linear approximation scheme for computing DTW or GED for a restricted, although quite large, family of curves.

Recently, Bringmann and Künnemann [39] proved that DTW on one-dimensional point sequences whose elements are taken from $\{0, 1, 2, 4, 8\} \subset \mathbb{R}$ has no $O(n^{2-\Omega(1)})$-time algorithm, unless SETH fails. They proved a similar hardness result also for Edit Distance between two binary strings, improving the conditional lower bound of Backurs and Indyk [23]. This line of work was extended in a very recent work by Abboud *et al.* [3], and Abboud and Bringmann [2], where they show that even a sufficiently large polylog(n)-factor improvement over the quadratic-time upper bound of similar quadratic matching problems, may lead to major consequences, such as faster Formula-SAT algorithms, and new circuit complexity lower bounds.

Masek and Paterson [128] showed that Edit Distance between two strings of length at most n over an $O(1)$-size alphabet can be solved in $O(n^2/\log n)$ time. More recent works [34,103] managed to lift the demand for $O(1)$-size alphabet and retain a subquadratic-time bound by making a better use of the word-RAM model. However, these works do not seem to extend to GED, especially not when taking sequences of points with arbitrary real coordinates. In the string version, the cost of replacing a character is fixed (usually 1), hence, we only need to detect that two characters are not identical in order to compute the replacement cost, unlike in GED, where the analogous cost for two matched points is taken to be their distance, under some metric.

Our Results and Related Works. Efforts for breaking the quadratic time barrier for basic similarity measures between curves and point-sequences were recently stimulated by the result of Agarwal *et al.* [7] who showed that the discrete Fréchet distance can be computed in $O(n^2/\log n)$ time. Their algorithm for (discrete) Fréchet distance does not extend to DTW or GED, as the formula for the (discrete) Fréchet distance uses the max function over distances between pairs of points, while the formulas for DTW and GED involve their sum. As a result, the Fréchet distance is effectively determined by a single pair of sequence elements, which fits well into the use of the Four-Russians technique [20], while the DTW and GED are determined by many pairs of elements. This makes our algorithms much more subtle, involving a combination and extension of techniques from computational geometry and graph shortest paths.

To simplify the presentation, we present our results only for the "balanced" case $m = n$; extending them to the general case $m \leqslant n$ is easy. The standard $\Theta(mn)$-time algorithm is superior to our solution only when m is subpolynomial in n.

Our results are stated in the following theorems.

Theorem 4.1.1. *Given two sequences $A = (p_1, \ldots, p_n)$ and $B = (q_1, \ldots, q_n)$, each of n points in $\mathbb{R}$, the* DTW*-distance* $\mathsf{dtw}(A, B)$ *(and optimal coupling), or the* GED $\mathsf{ged}(A, B)$ *(and optimal matching) can be computed by a deterministic algorithm in $O(n^2/\log\log n)$ time.*

Theorem 4.1.1 gives the very first subquadratic-time algorithm for solving DTW, breaking the nearly 50 years old $\Theta(n^2)$ time bound [150]. We present the improved algorithm for DTW in Section 4.3. In Section 4.3.1 we extend our algorithm to give a more general result, which supports high-dimensional polyhedral metric spaces, as stated in Theorem 4.1.2 given below. In Section 4.4 we extend our algorithm to obtain a subquadratic solution for GED.

Theorem 4.1.2. *Let $A = (p_1, \ldots, p_n)$ and $B = (q_1, \ldots, q_n)$ be two sequences of n points in $\mathbb{R}^d$, where d is a constant and the underlying distance-metric is polyhedral*[1]*. Then $\mathsf{dtw}(A, B)$ (and optimal coupling), or $\mathsf{ged}(A, B)$ (and optimal matching) can be computed by a deterministic algorithm in $O(n^2/\log\log n)$ time.*

4.2 Preliminaries, Tools, and the Quadratic Time DTW Algorithm

Throughout this chapter, unlike Chapter 3, we view matrices with rows indexed in increasing order from bottom to top and columns indexed in increasing order from left to right, so for example, $M[0, 0]$ corresponds to the value of the leftmost-bottom cell of a matrix M.

In our algorithm, we will often use the following extension of Fredman's trick (see also Section 2.2).

$$\begin{gathered} a_1 - b_1 + \cdots + a_r - b_r < a'_1 - b'_1 + \cdots + a'_t - b'_t \\ \text{if and only if} \\ a_1 + \cdots + a_r - a'_1 - \cdots - a'_t < b_1 + \cdots + b_r - b'_1 - \cdots - b'_t. \end{gathered} \tag{4.3}$$

As in Chapter 3, our algorithm uses Chan's geometric domination technique, summarized in Section 2.2 and in Lemma 2.2.1. We repeat below the statement of the lemma, for the convenience of the reader.

Lemma 4.2.1 (Chan [53])**.** *Given a finite set $P = \{p_1, \ldots, p_n\}$ of points in $\mathbb{R}^d$ such that each point is colored red or blue, one can report all pairs $(i, j) \in [n]^2$, such that p_i is red, p_j is blue, and $p_i[k] > p_j[k]$ for every $k \in [d]$, in time $O(c_\varepsilon^d |P|^{1+\varepsilon} + K)$, where K is the output size, $\varepsilon \in (0, 1)$ is an arbitrary prespecified parameter, and $c_\varepsilon = 2^\varepsilon/(2^\varepsilon - 1)$.*

The Quadratic Time DTW Algorithm

We give an overview of the standard dynamic programming algorithm for computing the DTW-distance between two sequences of n points in $\mathbb{R}$, which requires quadratic time [150]. This algo-

[1] That is, the underlying metric is induced by a norm, whose unit ball is a symmetric convex polytope with $O(1)$ facets (e.g., L_1, L_∞).

rithm can be easily extended to return also the optimal coupling (see below). In Section 4.4 we overview a "similar in principle" algorithm for solving GED.

We are given as input two sequences $A = (p_1, \ldots, p_n)$ and $B = (q_1, \ldots, q_n)$ of n points in $\mathbb{R}$. (The algorithm below can be (trivially) modified to handle sequences of different lengths.)

1. Initialize an $(n+1) \times (n+1)$ matrix M and set $M[0,0] := 0$.
2. For each $\ell \in [n]$
2.1. $M[\ell, 0] := \infty$, $M[0, \ell] := \infty$.
3. For each $\ell \in [n]$,
3.1. For each $m \in [n]$,
3.1.1 $M[\ell, m] := |p_\ell - q_m| + \min\Big\{M[\ell-1, m],\, M[\ell, m-1],\, M[\ell-1, m-1]\Big\}$.
4. Return $M[n, n]$.

The optimal coupling itself can also be retrieved, at no extra asymptotic cost, by the standard technique of maintaining pointers from each (ℓ, m) to the preceding position

$$(\ell', m') \in \{(\ell-1, m),\, (\ell, m-1),\, (\ell-1, m-1)\}$$

through which $M[\ell, m]$ is minimized. Tracing these pointers backwards from (n, n) to $(0, 0)$ and reversing these links yields the desired optimal coupling.

4.3 Dynamic Time Warping in Subquadratic Time

As above, the input consists of two sequences $A = (p_1, \ldots, p_n)$ and $B = (q_1, \ldots, q_n)$ of n points in $\mathbb{R}$. Our algorithm can easily be modified to handle the case where A and B have different lengths.

Preparations

We fix some (small) parameter g, whose value will be specified later; for simplicity, we assume that $\frac{n}{g-1}$ is an integer. We decompose A and B into $s = \frac{n}{g-1}$ subsequences $A_1, \ldots, A_s$, and $B_1, \ldots, B_s$, such that for each $i, j \in \{2, \ldots, s\}$, each of A_i and B_j consists of $g-1$ consecutive elements of the corresponding sequence, prefixed by the last element of the preceding subsequence. We have that A_1 and B_1 are both of size $g-1$, each A_i and B_j is of size g, for $i, j \in \{2, \ldots, s\}$, and each consecutive pair A_i, A_{i+1} or B_j, B_{j+1} have one common element.

For each $i, j \in [s]$, denote by $D_{i,j}$ the *all-pairs-distances matrix* between points from A_i and points from B_j; specifically, $D_{i,j}$ is a $g \times g$ matrix (aka a *box*, see below for the cases $i = 1$ or $j = 1$) such that for every $\ell, m \in [g]$,

$$D_{i,j}[\ell, m] = \big|A_i(\ell) - B_j(m)\big|.$$

For all $i \in [s]$, we add a leftmost column with ∞ values to each box $D_{i,1}$, and similarly, we add a bottommost row with ∞ values to each box $D_{1,i}$. In particular, $D_{1,1}$ is augmented by both a new leftmost column and a new bottommost row. The common element $D_{1,1}[0,0]$ of this row and column is set to 0. Overall, we have $s^2 = \left(\frac{n}{g-1}\right)^2$ boxes $D_{i,j}$, all of size $g \times g$.

We define a *staircase path* P on a $g \times g$ matrix $D_{i,j}$ as a sequence of positions from $[g] \times [g]$ that form a monotone staircase structure, starting from a cell on the left or bottom boundary and ending at the right or top boundary, so that each subsequent position is immediately either to the right, above, or above-right of the previous one. Formally, by enumerating the path positions as $P(0), \ldots, P(t^*)$, we have $P(t+1) \in \{P(t)+(0,1), P(t)+(1,0), P(t)+(1,1)\}$, for each $t \in \{0, \ldots, t^*-1\}$. The path starts at some point $P(0) = (\cdot, 1)$ or $(1, \cdot)$, and ends at some t^* (not necessarily the first such index) for which $P(t^*) = (\cdot, g)$ or $(g, \cdot)$. Note that t^* can have any value in $[2g-2]$. The number of possible monotone staircase paths in a box $D_{i,j}$ is trivially bounded by $O(g^2 3^{2g-2})$, and the following more careful reasoning improves this bound to $O(3^{2g})$. Each staircase path can be encoded by its first position, followed by its sequence of moves, where each move is in one of the directions up/right/up-right. Thus, the number of staircase paths that start at some position $(r, 1)$ (resp. $(1, r)$) at the left (resp. bottom) boundary is bounded by 3^{2g-1-r}. Thus, the total number of staircase paths that start at the left or the bottom boundary is bounded by

$$2\sum_{r=1}^{g} 3^{2g-1-r} = O(3^{2g}).$$

We define the *cost* of a staircase path P in a box $D_{i,j}$ by

$$\mathrm{cost}_{i,j}(P) = \sum_{t=1}^{t^*} D_{i,j}(P(t)).$$

(For technical reasons, that will become clear in the sequel, we generally do not include the first position $P(0)$ of the path in evaluating its cost, except in the boxes $D_{i,1}$ and $D_{1,j}$ for all $i, j \in [s]$.) To ease the presentation, in the algorithm that follows, we assume (or ensure) that no two distinct paths in a box $D_{i,j}$ have the same cost. This will be the case if we assume that the input sequences are in sufficiently general position. In Section 4.3.2 we will show how this assumption can be completely removed, by adding a few additional steps to the preprocessing stage of the algorithm, without increasing its asymptotic time bound.

We denote by L the set of *positions* in the left and bottom boundaries of *any* box $D_{i,j}$, and by R the set of positions in the right and top boundaries (note that L and R have two common positions). Given a starting position $v \in L$, and an ending position $w \in R$, we denote by $S(v, w)$ the set of all staircase paths $P_{v,w}$ that start at v and end at w (if there is no staircase path between v and w, then $S(v, w) = \varnothing$). We say that $P^*_{v,w} \in S(v, w)$ is the *shortest path* between v and w in

$D_{i,j}$ iff

$$\text{cost}_{i,j}\left(P^*_{v,w}\right) = \min_{P_{v,w} \in S(v,w)} \left\{\text{cost}_{i,j}\left(P_{v,w}\right)\right\}.$$

Note that according to our general position assumption, the shortest path between v and w, within a given box, is *unique*.

First Stage: Preprocessing

The first stage of our algorithm is to construct a data structure in subquadratic time (and storage), such that for each box $D_{i,j}$, and for each pair of positions $(v, w) \in L \times R$, we can retrieve the shortest path $P^*_{v,w}$ in $D_{i,j}$ and $\text{cost}_{i,j}(P^*_{v,w})$ in $O(1)$ time, when such a path exists (i.e., when $S(v, w)$ is nonempty).

The algorithm enumerates all $(2g-1)^2$ pairs of positions (v, w) in a $g \times g$ matrix (box) such that $v \in L$ and $w \in R$, discarding pairs that cannot be connected by a monotone staircase path, and referring to the surviving pairs as *admissible*. Again, we simplify the notation by upper bounding this quantity by $4g^2$. For each such admissible pair $(v, w) \in L \times R$, we also enumerate *every* possible staircase path in $S(v, w)$ as $P_{v,w} : [t^*] \to [g] \times [g]$, where we write $P_{v,w} = \left(P^{\text{r}}_{v,w}, P^{\text{c}}_{v,w}\right)$ as a pair of row and column functions $P^{\text{r}}_{v,w}$, $P^{\text{c}}_{v,w} : [t^*] \to [g]$, so that $P_{v,w}(k) = \left(P^{\text{r}}_{v,w}(k), P^{\text{c}}_{v,w}(k)\right)$, for each $k \in [t^*]$. (Note that t^* is a path-dependent parameter, determined by v, w and the number of diagonal moves in the path.) In total, there are $O(3^{2g})$ possible staircase paths $P_{v,w}$ (for all admissible pairs $(v, w) \in L \times R$ combined), which we enumerate. The above enumerations are done using a natural lexicographic order, which induces a total order on the $< 4g^2$ admissible pairs of positions of $L \times R$, and for each such pair (v, w), a total order on all possible staircase paths $P_{v,w} \in S(v, w)$.

Given two staircase paths $P_{v,w}$ and $P'_{v,w}$ with the same starting and ending positions v, w in a box $D_{i,j}$, we want to use the extended Fredman trick (as in (4.3)) to compare $\text{cost}_{i,j}\left(P_{v,w}\right)$ with $\text{cost}_{i,j}\left(P'_{v,w}\right)$, by comparing two expressions such that one depends on points from A_i only and the other depends on points from B_j only. Suppose that $P_{v,w} = ((\ell_1, m_1), \ldots, (\ell_r, m_r))$ and $P'_{v,w} = ((\ell'_1, m'_1), \ldots, (\ell'_t, m'_t))$ (note that $(\ell_r, m_r) = (\ell'_t, m'_t) = w$, since both paths end at w, and that we ignore the common starting positions $(\ell_0, m_0) = (\ell'_0, m'_0) = v$). We have

$$\text{cost}_{i,j}\left(P_{v,w}\right) = \left|A_i(\ell_1) - B_j(m_1)\right| + \cdots + \left|A_i(\ell_r) - B_j(m_r)\right|,$$

and

$$\text{cost}_{i,j}\left(P'_{v,w}\right) = \left|A_i(\ell'_1) - B_j(m'_1)\right| + \cdots + \left|A_i(\ell'_t) - B_j(m'_t)\right|,$$

and we want to test whether, say, $\text{cost}_{i,j}\left(P_{v,w}\right) < \text{cost}_{i,j}\left(P'_{v,w}\right)$ (recall that we assume that equa-

lities do not arise), that is, testing whether

$$\big|A_i(\ell_1) - B_j(m_1)\big| + \cdots + \big|A_i(\ell_r) - B_j(m_r)\big| < \big|A_i(\ell'_1) - B_j(m'_1)\big| + \cdots + \big|A_i(\ell'_t) - B_j(m'_t)\big|. \quad (4.4)$$

The last term in each side of (4.4) is actually unnecessary, since they are equal. In order to transform this inequality into a form suitable for applying the extended Fredman trick (4.3), we need to replace each absolute value $|x|$ by either $+x$ or $-x$, as appropriate. To see what we are after, assume first that the expressions $A_i(\ell_k) - B_j(m_k)$ and $A_i(\ell'_k) - B_j(m'_k)$ are all positive, so that (4.4) becomes

$$A_i(\ell_1) - B_j(m_1) + \cdots + A_i(\ell_r) - B_j(m_r) < A_i(\ell'_1) - B_j(m'_1) + \cdots + A_i(\ell'_t) - B_j(m'_t).$$

By (4.3) we can rewrite this inequality as

$$A_i(\ell_1) + \cdots + A_i(\ell_r) - A_i(\ell'_1) - \cdots - A_i(\ell'_t) < B_j(m_1) + \cdots + B_j(m_r) - B_j(m'_1) - \cdots - B_j(m'_t),$$

which can be written as

$$A_i(P^{\mathrm{r}}_{v,w}(1)) + \cdots + A_i(P^{\mathrm{r}}_{v,w}(r)) - A_i(P'^{\,\mathrm{r}}_{v,w}(1)) - \cdots - A_i(P'^{\,\mathrm{r}}_{v,w}(t)) \quad (4.5)$$

$$< B_j(P^{\mathrm{c}}_{v,w}(1)) + \cdots + B_j(P^{\mathrm{c}}_{v,w}(r)) - B_j(P'^{\,\mathrm{c}}_{v,w}(1)) - \cdots - B_j(P'^{\,\mathrm{c}}_{v,w}(t)). \quad (4.6)$$

If $P_{v,w} = P^*_{v,w}$ (i.e., if $P_{v,w}$ is the shortest path from v to w) in $D_{i,j}$ then the inequality above holds for all pairs $(P_{v,w}, P'_{v,w})$, where $P'_{v,w} \in S(v,w)$ is any other staircase path between v and w.

For each admissible pair of positions $(v,w) \in L \times R$, we choose some staircase path $P_{v,w}$ as a candidate for being the shortest path from v to w. The overall number of sets of candidate paths is fewer than $(3^{2g})^{4g^2} = 3^{8g^3}$. For a fixed choice of such a set of paths (one path for each admissible pair $(v,w) \in L \times R$), we want to test, within some given box $D_{i,j}$, whether all the $< 4g^2$ chosen paths are the shortest paths between the corresponding pairs of positions. As unfolded next, we will apply this test for all boxes $D_{i,j}$, and output those boxes at which the outcome is positive (for the current chosen set of shortest paths). We will repeat the procedure for all $< 3^{8g^3}$ possible sets of candidate paths $P_{v,w}$. Since we enumerated the staircase paths in lexicographical order earlier, we can easily proceed through all sets of candidate paths, using this order.

Testing a Fixed Choice of Shortest Paths. For each subsequence A_i, we create a (blue) point α_i, and for each subsequence B_j we create a (red) point β_j, such that, for every admissible pair $(v,w) \in L \times R$, we have one coordinate for each path $P'_{v,w} \in S(v,w)$, different from the chosen path $P_{v,w}$. The value of α_i (resp., β_j) at that coordinate is the corresponding expression (4.5) (resp., (4.6)). The points α_i and β_j are embedded in $\mathbb{R}^{d_g}$, where $d_g = \sum_{(v,w)} \Gamma_{v,w}$ is the sum over

all admissible pairs $(v, w) \in L \times R$, and $\Gamma_{v,w}$ is the number of monotone staircase paths from v to w minus 1. As discussed earlier, $d_g = O(3^{2g})$.

We have that a (blue) point

$$\alpha_i = \left(\ldots, A_i(P^{\mathrm{r}}_{v,w}(1)) + \cdots + A_i(P^{\mathrm{r}}_{v,w}(r)) - A_i(P'^{\mathrm{r}}_{v,w}(1)) - \cdots - A_i(P'^{\mathrm{r}}_{v,w}(t)), \ldots\right)$$

is dominated by a (red) point

$$\beta_j = \left(\ldots, B_j(P^{\mathrm{c}}_{v,w}(1)) + \cdots + B_j(P^{\mathrm{c}}_{v,w}(r)) - B_j(P'^{\mathrm{c}}_{v,w}(1)) - \cdots - B_j(P'^{\mathrm{c}}_{v,w}(t)), \ldots\right),$$

if and only if each of the paths that we chose (a path for every admissible pair $(v, w) \in L \times R$) is the shortest path between the corresponding positions v, w in box $D_{i,j}$. The number of points is $2s = \Theta(n/g)$, and the time to prepare them, i.e., to compute all their coordinates, is $O(2s \cdot 3^{2g} \cdot g) = O(3^{2g} n)$.

By Lemma 4.2.1, we can report all pairs of points (α_i, β_j) such that α_i is dominated by β_j, in $O\left(c_\varepsilon^{O(3^{2g})}(n/g)^{1+\varepsilon} + K\right)$ time, where K is the number of boxes at which the test of our specific chosen paths comes out positive. As mentioned earlier, we use $\varepsilon = 1/2$, with $c_\varepsilon \approx 3.42$.

This runtime is for a specific choice of a set of shortest paths between all admissible pairs in $L \times R$. As already mentioned, we repeat this procedure at most 3^{8g^3} times. Overall, we will report exactly $s^2 = \Theta\left((n/g)^2\right)$ dominating pairs (red on blue), because the set of shortest paths between admissible pairs in $L \times R$ in each box $D_{i,j}$ is unique (recall that we assumed that any pair of distinct staircase paths in a box do not have the same cost). Since the overall number of sets of candidate paths is bounded by 3^{8g^3}, one path for each admissible pair, the overall runtime for all invocations of the bichromatic dominance reporting algorithm (including preparing the points) is

$$O\left(3^{8g^3}\left(3^{2g}n + c_\varepsilon^{O(3^{2g})}(n/g)^{1+\varepsilon}\right) + (n/g)^2\right).$$

Recall that, so far, we have assumed that all the differences within the absolute values $D_{i,j}[\ell, m] = \left|A_i(\ell) - B_j(m)\right|$ are positive, which allowed us to drop the absolute values, and write $D_{i,j}[\ell, m] = A_i(\ell) - B_j(m)$, for every $i, j \in [s]$, and $\ell, m \in [g]$, thereby facilitating the use of the extended Fredman trick (4.3). Of course, in general this will not be the case, so, in order to still be able to drop the absolute values, we also have to verify the signs of all these differences.

For each box $D_{i,j}$, there is a *unique* sign assignment $\sigma^* : [g] \times [g] \to \{-1, 1\}$ such that

$$D_{i,j}[\ell, m] = \left|A_i(\ell) - B_j(m)\right| = \sigma^*(\ell, m)(A_i(\ell) - B_j(m)),$$

for every $\ell, m \in [g]$ (our "general position" assumption implies that each difference is nonzero).

Thus for any staircase path $P = (P^{\mathrm{r}}, P^{\mathrm{c}})$ in $D_{i,j}$, of length t^*, we have

$$\mathrm{cost}_{i,j}(P) = \sum_{t=1}^{t^*} \sigma^*(P(t)) \left(A_i(P^{\mathrm{r}}(t)) - B_j(P^{\mathrm{c}}(t))\right).$$

Now we proceed as before, testing sets of paths, but now we also test sign assignments of the box, by trying *every* possible assignment $\sigma : [g] \times [g] \to \{-1, 1\}$, and modify the points α_i and β_j, defined earlier, by (i) adding sign factors to each term, and (ii) adding coordinates that enable us to test whether σ is the correct assignment σ^* for the corresponding boxes $D_{i,j}$.

Denote by P a candidate for the shortest path for some admissible pair of positions $(v, w) \in L \times R$, and let σ be a candidate sign assignment. Then, for every other path $P' \in S(v, w)$, we have the following modified coordinates for α_i and β_j respectively.

$$\left(\ldots, \sigma(P(1))A_i(P^{\mathrm{r}}(1)) + \cdots + \sigma(P(r))A_i(P^{\mathrm{r}}(r)) - \sigma(P'(1))A_i(P'^{\mathrm{r}}(1)) - \cdots - \sigma(P'(t))A_i(P'^{\mathrm{r}}(t)), \ldots\right),$$
$$\left(\ldots, \sigma(P(1))B_j(P^{\mathrm{c}}(1)) + \cdots + \sigma(P(r))B_j(P^{\mathrm{c}}(r)) - \sigma(P'(1))B_j(P'^{\mathrm{c}}(1)) - \cdots - \sigma(P'(t))B_j(P'^{\mathrm{c}}(t)), \ldots\right),$$

where we use the same notations as in (4.4), (4.5), and (4.6). In addition, to validate the correctness of σ, we extend α_i and β_j by adding the following g^2 coordinates to each of them. For every pair $(\ell, m) \in [g] \times [g]$, we add the following coordinates to α_i and β_j respectively.

$$(\ldots, -\sigma(\ell, m)A_i(\ell), \ldots),$$
$$(\ldots, -\sigma(\ell, m)B_j(m), \ldots).$$

This ensures that a point α_i is dominated by a point β_j iff $D_{i,j}[\ell, m] = \sigma(\ell, m)\left(A_i(\ell) - B_j(m)\right)$, for every $\ell, m \in [g]$, and all the $< 4g^2$ candidate paths that we test are indeed shortest paths in $D_{i,j}$.

The runtime analysis is similar to the preceding one, but now we increase the number of candidate choices by a factor of 2^{g^2} (this factor bounds the number of all possible sign assignments), and the dimension of the space where the points are embedded increases by g^2 additional coordinates. We now have $2s = \Theta(n/g)$ points in $\mathbb{R}^{d_g + g^2}$ ($d_g = O(3^{2g})$ is as defined earlier), and the time to prepare them (computing the value of each coordinate) is $O((n/g)(d_g + g^2)g) = O(3^{2g}n)$. There are at most 3^{8g^3} sets of candidate paths to test, and for each set, there are at most 2^{g^2} sign assignment to test, so in total, we invoke the bichromatic dominance reporting algorithm at most $2^{g^2}3^{8g^3} < 3^{8g^3+g^2}$ times, for an overall runtime (including preparing the points) of

$$O\left(3^{8g^3+g^2}\left(3^{2g}n + c_\varepsilon^{O(3^{2g})+g^2}(n/g)^{1+\varepsilon}\right) + (n/g)^2\right).$$

By setting $\varepsilon = 1/2$ and $g = \delta \log\log n$, for a suitable sufficiently small constant δ, the first two terms become negligible (strongly subquadratic), and the runtime is therefore dominated by

the output size, that is $O\left((n/g)^2\right) = O\left(n^2/(\log\log n)^2\right)$. Each reported pair (α_i, β_j) certifies that the current set of $< 4g^2$ chosen candidate paths are all shortest paths in box $D_{i,j}$. Each of the $s^2 = \Theta\left((n/g)^2\right)$ sets of shortest paths is represented by $O(g^3) = O((\log\log n)^3)$ bits (there are $< 4g^2$ shortest paths connecting admissible pairs, each of length at most $2g-1$, and each path can be encoded by its first position, followed by the sequence of its at most $2g-2$ moves, where each move is in one of the three directions up/right/up-right), and thus it can easily be stored in one machine word (for sufficiently small δ). Moreover, we have an order on the pairs (v, w) (induced by our earlier enumeration), so for each set, we can store its shortest paths in this order, and therefore, accessing a specific path (for some admissible pair) from the set takes $O(1)$ time (in the word-RAM model that we assume).

Note, however, that we obtain only the *positions* that the paths traverse and not their *cost*. In later stages of our algorithm we will also need to compute, on demand, the cost of certain paths, but doing this naively would take $O(g)$ time per path, which is too expensive for us. To handle this issue, when we choose a candidate sign assignment σ, and a set S of the $< 4g^2$ paths as candidates for the shortest paths, we also compute and store, for each path $P \in S$ that we have not yet encountered, the *rows-cost* of P in A_i,

$$V_i^{\mathrm{r}}(P, \sigma) = \sigma(P(1))A_i(P^{\mathrm{r}}(1)) + \cdots + \sigma(P(t^*))A_i(P^{\mathrm{r}}(t^*)),$$

for every $i \in [s]$, and the *columns-cost* of P in B_j,

$$V_j^{\mathrm{c}}(P, \sigma) = \sigma(P(1))B_j(P^{\mathrm{c}}(1)) + \cdots + \sigma(P(t^*))B_j(P^{\mathrm{c}}(t^*)),$$

for every $j \in [s]$, where t^* is the length of P. Observe that, for the correct sign assignment σ^* of box $D_{i,j}$,

$$\mathrm{cost}_{i,j}(P) = V_i^{\mathrm{r}}(P, \sigma^*) - V_j^{\mathrm{c}}(P, \sigma^*). \tag{4.7}$$

We do not compute $V_i^{\mathrm{r}}(P, \sigma) - V_j^{\mathrm{c}}(P, \sigma)$ yet, but only compute and store (if not already stored) the separate quantities $V_i^{\mathrm{r}}(P, \sigma)$ and $V_j^{\mathrm{c}}(P, \sigma)$, for each $P \in S$, for every chosen set S, and sign assignment σ. We store the values $V_i^{\mathrm{r}}(P, \sigma)$ and $V_j^{\mathrm{c}}(P, \sigma)$ in arrays, ordered by the earlier enumeration of all staircase paths, so that given a staircase path P, and indices $i, j \in \left[\frac{n}{g-1}\right]$, we can retrieve, upon demand, the values $V_i^{\mathrm{r}}(P, \sigma^*)$ and $V_j^{\mathrm{c}}(P, \sigma^*)$, and compute $\mathrm{cost}_{i,j}(P)$ by using (4.7), in $O(1)$ time. In total, over all possible candidate paths and sign assignments, this takes $O(2^{g^2}3^{2g} \cdot (n/g) \cdot g) = O(3^{g^2+2g}n)$ time and space, which is already subsumed by the time (and space) bound for reporting dominances from the previous stage.

To summarize this stage of the algorithm, we presented a subquadratic-time preprocessing procedure, which runs in $O\left((n/g)^2\right) = O\left(n^2/(\log\log n)^2\right)$ time, such that for any box $D_{i,j}$, and an admissible pair of positions $(v, w) \in L \times R$, we can retrieve the shortest path $P^*_{v,w}$ in $O(1)$

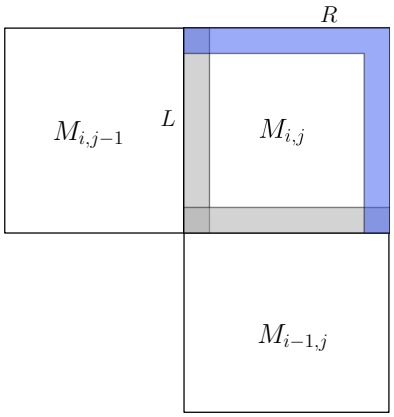

Figure 4.3.1: The L-boundary (shaded in gray) of box $M_{i,j}$ overlaps with the top boundary of $M_{i-1,j}$ and the right boundary of $M_{i,j-1}$. Once we have the values of M at the positions of the L-boundary of $M_{i,j}$, our algorithm computes the values of M at the positions of its R-boundary (shaded in blue).

time, and can also compute $\text{cost}_{i,j}(P^*_{v,w})$ in $O(1)$ time. This will be useful in the next stage of our algorithm.

Second Stage: Compact Dynamic Programming

Our approach is to view the $(n+1) \times (n+1)$ matrix M from the dynamic programming algorithm (see Section 4.2) as decomposed into $s^2 = \left(\frac{n}{g-1}\right)^2$ boxes $M_{i,j}$, each of size $g \times g$, so that each box $M_{i,j}$ occupies the same positions as does the corresponding box $D_{i,j}$. That is, the indices of the rows (resp., columns) of $M_{i,j}$ are those of A_i (resp., B_j). In particular, for each $i, j \in [s]$, the positions $(\cdot, g)$ on the right boundary of each box $M_{i,j}$ coincide with the corresponding positions $(\cdot, 1)$ on the left boundary of $M_{i,j+1}$, and the positions $(g, \cdot)$ on the top boundary of $M_{i,j}$ coincide with the corresponding positions $(1, \cdot)$ on the bottom boundary of $M_{i+1,j}$. Formally, $M_{i,j}[\ell, m] = M\left[(i-1)(g-1)+\ell, (j-1)(g-1)+m\right]$, for each position $(\ell, m) \in [g] \times [g]$. See Figure 4.3.1 for an illustration.

Our strategy is to traverse the boxes, starting from the leftmost-bottom one $M_{1,1}$, where we already have the values of M at the sequence L of positions of its left and bottom boundaries (initialized to the same values as in the algorithm in Section 4.2), and we compute the values of M on its top and right boundaries R. We then continue to the box on the right, $M_{1,2}$, now having the values on its L-boundary (where its left portion overlaps with the R-boundary of $M_{1,1}$ and its bottom portion is taken from the already preset bottom boundary), and we compute the values of M on its R-boundary. We continue in this way until we reach the rightmost-bottom box $M_{1,s}$. We then continue in the same manner in the next row of boxes, starting at $M_{2,1}$ and ending at $M_{2,s}$, and keep going through the rows of boxes in order. The process ends once we compute the values of M on the R-boundary of the rightmost-top box $M_{s,s}$, from which we obtain the desired entry $M[n, n]$.

For convenience, we enumerate the positions in L as $L(1), \ldots, L(2g-1)$ in "clockwise" order, so that $L(1)$ is the rightmost-bottom position $(1, g)$, and $L(2g-1)$ is the leftmost-top position $(g, 1)$. Similarly, we enumerate the positions of R by $R(1), \ldots, R(2g-1)$ in "counterclockwise" order, with the same starting and ending locations. Let $M_{i,j}(L) = \{M_{i,j}[L(1)], \ldots M_{i,j}[L(2g-1)]\}$ and $M_{i,j}(R) = \{M_{i,j}[R(1)], \ldots M_{i,j}[R(2g-1)]\}$, for $i, j \in [s]$.

By definition, for each position $(\ell, m) \in [n+1] \times [n+1]$, $M[\ell, m]$ is the minimal cost of a staircase path from $(0, 0)$ to (ℓ, m). It easily follows, by construction, that for each box $D_{i,j}$, and for each position $w \in R$, we have

$$M_{i,j}[w] = \min_{\substack{v \in L \\ (v,w) \text{ admissible}}} \Big\{ M_{i,j}[v] + \text{cost}_{i,j}(P^*_{v,w}) \Big\}. \tag{4.8}$$

(Note that, by definition, the term $D_{i,j}[v]$ is included in $M_{i,j}[v]$ and not in $\text{cost}_{i,j}(P^*_{v,w})$, so it is not doubly counted.) For each box $M_{i,j}$ and each position $w \in R$, our goal is thus to compute the position $u \in L$ that attains the minimum in (4.8), and the corresponding cost $M_{i,j}[w]$. We call such (u, w) the *minimal pair* for w in $M_{i,j}$.

For each box $D_{i,j}$, and each admissible pair $(v, w) \in L \times R$, we refer to the value $M_{i,j}[v] + \text{cost}_{i,j}(P^*_{v,w})$ as the *cumulative cost* of the pair (v, w), and denote it by $\text{c-cost}(v, w)$.

We can rewrite (4.8), for each position $w \in R$, as

$$M_{i,j}[w] = \min\big\{M^W_{i,j}[w],\, M^S_{i,j}[w]\big\},$$

where $M^S_{i,j}[w]$ is the minimum in (4.8) computed only over $v \in \{L(1), \ldots, L(g)\}$, which is the portion of L that overlaps the R-boundary of the bottom (south) neighbor $M_{i-1,j}$ (when $i > 1$), and $M^W_{i,j}[w]$ is computed over $v \in \{L(g), \ldots, L(2g-1)\}$, which overlaps the R-boundary of the left (west) neighbor $M_{i,j-1}$ (when $j > 1$). See Figure 4.3.1 for a schematic illustration. (Recall that the bottommost row and the leftmost column of M are initialized with ∞ values, except their shared cell $M[0, 0]$ that is initialized with 0.) The output of the algorithm is $M_{s,s}[R(g)] = M_{s,s}[g, g] = M[n, n]$. We can also return the optimal coupling, by using a simple backward pointer tracing procedure, similar in principle to the one mentioned for the quadratic algorithm in Section 4.2.

Computing Minimal Pairs. We still have to explain how to compute the minimal pairs (u, w) in each box $M_{i,j}$. Our preprocessing stage produces, for every box $D_{i,j}$, the set of all its shortest paths $S_{i,j} = \{P^*_{v,w} \mid (v, w) \in L \times R\}$ (ordered by the earlier enumeration of $L \times R$ and including only admissible pairs), and we can also retrieve the cost of each of these paths in $O(1)$ time (as explained earlier in the preprocessing stage). The cumulative cost (defined above) of each such pair (v, w) can also be computed in $O(1)$ time, assuming we have already computed $M_{i,j}[v]$. A naive, brute-force technique for computing the minimal pairs is to compute all the cumulative costs

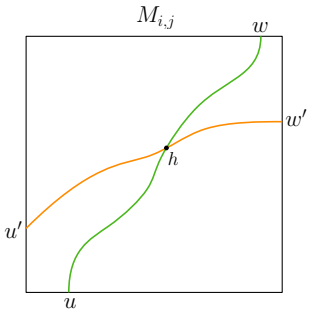

Figure 4.3.2: By Lemma 4.3.1, if (u, w) and (u', w') are minimal pairs in $M_{i,j}$, then the illustrated scenario is impossible, since the path $P^*_{u,w}$ (in green) is a portion of the shortest path from $M[0, 0]$ to $M_{i,j}[w]$, and the path $P^*_{u',w'}$ (in orange) is a portion of the shortest path from $M[0, 0]$ to $M_{i,j}[w']$. The illustrated intersection implies that one of the latter paths can decrease its cumulative cost by replacing its portion that ends at h by the respective portion that ends in h of the other path (recall that we assume that there are no two paths with the same cost), which contradicts the fact that both of these paths are shortest paths.

c-cost$_{i,j}(v, w)$, for all admissible pairs $(v, w) \in L \times R$, and select from them the minimal pairs. This however would take $O(g^2)$ time for each of the s^2 boxes, for a total of $\Theta(g^2 s^2) = \Theta(n^2)$ time, which is what we want to avoid.

Fortunately, we have the following important lemma, which lets us compute all the minimal pairs within a box, significantly faster than in $O(g^2)$ time.

Lemma 4.3.1. *For a fixed box $D_{i,j}$, and for any pair of distinct positions $w, w' \in R$, let $u, u' \in L$ be the positions for which (u, w) and (u', w') are minimal pairs in $M_{i,j}$. Then their corresponding shortest paths $P^*_{u,w}$ and $P^*_{u',w'}$ can partially overlap but can never cross each other. Formally, assuming that $w > w'$ (in the counterclockwise order along R), we have that for any $\ell, \ell', m \in [g]$, if $(\ell, m) \in P^*_{u,w}$ and $(\ell', m) \in P^*_{u',w'}$ then $\ell \geqslant \ell'$. That is, $P^*_{u,w}$ lies fully above $P^*_{u',w'}$ (partial overlapping is possible). In particular, we also have $u \geqslant u'$ (in the clockwise order along L).*

Lemma 4.3.1 asserts the so-called *Monge property* of shortest-path matrices (see, e.g., [45,116]). See Figure 4.3.2 for an illustration (of an impossible crossing) and a sketch of a proof.

Using Lemma 4.3.1, we first present a divide-and-conquer paradigm for computing the minimal pairs within a box $M_{i,j}$ in $O(g \log g)$ time, which is conceptually simple to perceive. However, this is not the best we can do. Afterwards, we present an even more efficient procedure that takes only $O(g)$ time in total.

We start by setting the median index $k = \lfloor |R|/2 \rfloor$ of $|R|$, and compute the minimal pair $(u, R(k))$ and its c-cost$(u, R(k))$, naively, in $O(g)$ time, as explained above. The path $P^*_{u,R(k)}$ decomposes the box $M_{i,j}$ into two parts, so that one part, X, consists of all the positions in $M_{i,j}$ that are (weakly) *above* $P^*_{u,R(k)}$, and the other part, Y, consists of all the positions in $M_{i,j}$ that are (weakly) *below* $P^*_{u,R(k)}$, so that X and Y are disjoint, except for the positions along the path $P^*_{u,R(k)}$ which they share. By Lemma 4.3.1, the shortest paths between any other minimal pair of positions in $L \times R$ can never cross $P^*_{u,R(k)}$. Thus, we can repeat this process separately in X and in Y. Note that

the input to each recursive step is just the sequences of positions of X and Y along L and R, respectively (and we encode each sequence simply by its first and last elements); there is no need to keep track of the corresponding portion of $M_{i,j}$ itself.

Denote by $T(a,b)$ the maximum runtime for computing all the minimal pairs (u,w), within any box $M_{i,j}$, for u in some contiguous portion L' of a entries of L, and w in some contiguous portion R' of b entries of R. Clearly, $T(1,b) = O(b)$, and $T(a,1) = O(a)$. In general, the runtime is bounded by the recurrence

$$T(a,b) = \max_{k\in[a]} \Big\{ T(k,\, \lfloor b/2 \rfloor) + T(a-k+1,\, \lfloor b/2 \rfloor) \Big\} + O(a).$$

It is an easy exercise to show, by induction, that the solution of this recurrence satisfies $T(a,b) = O\left((a+b)\log b\right)$. Thus, the runtime of the divide-and-conquer procedure described above, for a fixed box $M_{i,j}$, is $O\left((|R|+|L|)\log|R|\right) = O(g\log g)$.

The runtime of computing $M_{i,j}(R)$ for all $s^2 = \Theta\left((n/g)^2\right)$ boxes is thus $O\left((n/g)^2 g\log g\right) = O\left(n^2\log g/g\right)$. Overall, including the preprocessing stage, the total runtime of the algorithm is $O\left((n/g)^2 + n^2\log g/g\right) = O\left(n^2\log g/g\right)$. As dictated by the preprocessing stage, we need to choose $g = \Theta(\log\log n)$, so the overall runtime is $O\left(n^2\log\log\log n/\log\log n\right)$.

A Further Improvement: Removing the $\log\log\log n$ Factor. We can speed up the computation of minimal pairs even further, so that computing $M_{i,j}(R)$ for each box will take $O(g)$ time, improving the $O(g\log g)$ bound of the divide-and-conquer algorithm described above.

For each box $M_{i,j}$, denote by $M^{LR}_{i,j}$ the $(2g-1)\times(2g-1)$ matrix such that the L and R positions of $M_{i,j}$ correspond to the rows and columns of $M^{LR}_{i,j}$, respectively. Namely, each pair $(v,w)\in L\times R$ corresponds to a cell in $M^{LR}_{i,j}$ that its value is c-cost$_{i,j}(v,w)$ (that is, the cost of the shortest path from the origin of M that goes through v and ends at w). For convenience, we denote by $M^{LR}_{i,j}[\ell,m]$ the cell that corresponds to the pair $(L(\ell),R(m))\in L\times R$.

Lemma 4.3.1 implies the following observation.

Observation 4.3.2. *The matrix $M^{LR}_{i,j}$ is a Monge matrix. That is, for every $\ell<\ell'\in[2g-1]$ and every $m<m'\in[2g-1]$, we have that*

$$M^{LR}_{i,j}[\ell,m] + M^{LR}_{i,j}[\ell',m'] < M^{LR}_{i,j}[\ell,m'] + M^{LR}_{i,j}[\ell',m]. \tag{4.9}$$

Indeed, it is easy to check that if Equation (4.9) does not hold then we have a contradiction to Lemma 4.3.1. (See [46] for a survey on Monge matrices and their applications.)

Observation 4.3.2 immediately implies that the matrix $M^{LR}_{i,j}$ is *totally monotone*. That is, for

every $\ell < \ell' \in [2g-1]$ and every $m < m' \in [2g-1]$, we have that

$$M^{LR}_{i,j}[\ell, m] > M^{LR}_{i,j}[\ell, m'] \implies M^{LR}_{i,j}[\ell', m] > M^{LR}_{i,j}[\ell', m'].$$

To compute $M_{i,j}(R)$ we need to find the minimum of every column $w \in R$ (i.e., to find $\min_{v \in L} M^{LR}_{i,j}[v, w]$). Since $M^{LR}_{i,j}$ is totally monotone, we can use the SMAWK algorithm [9] to compute the minimum of each column of R in total $O(|L| + |R|) = O(g)$ time.

Thus, the runtime of computing $M_{i,j}(R)$ for all $s^2 = \Theta\left((n/g)^2\right)$ boxes becomes $O\left(n^2/g\right)$. This bound in fact dominates the total runtime of the algorithm, provided the we choose $g = \Theta(\log \log n)$, due to the preprocessing stage. Hence, we obtain that the total runtime of the algorithm is $O\left(n^2/\log \log n\right)$.

This completes the proof of Theorem 4.1.1 for DTW on a pair of point-sequences in $\mathbb{R}$. □

4.3.1 Extension to High-Dimensional Polyhedral Metric Spaces

The algorithm described above can be extended to work in higher dimensional spaces $\mathbb{R}^d$, for any constant d, when the underlying metric is polyhedral. That is, the underlying metric is induced by a norm, whose unit ball is a symmetric convex polytope with $O(1)$ facets. To illustrate this extension, consider the L_1-metric in $\mathbb{R}^d$, whose unit ball is the symmetric cross-polytope $|x_1| + \cdots + |x_d| \leqslant 1$, with 2^d facets. In this case, each entry in the blocks $D_{i,j}$ is a sum of d absolute values. By choosing a candidate sign assignment for all the absolute values, each comparison that the algorithm faces is a sign test of a $2d$-linear expression in the input (with coefficients $1, -1$), and the extended Fredman trick (4.3) can then be applied when comparing the costs of two staircase paths. Then, in much the same way as before, we can encode the inequalities into red and blue points α_i and β_j, and use a suitable modification of the preceding machinery to compare costs of staircase paths and validate sign assignments correctness. Omitting further details, we get a subquadratic algorithm for DTW in such a higher-dimensional setup under the L_1-metric, with the same asymptotic time bound as that of the algorithm described above, but with the constant of proportionality depending (exponentially) on d.

To handle general polyhedral metrics, let K denote the unit ball of the metric. For each pair of points $p_\ell \in A$, $q_m \in B$, we choose some facet of K as a candidate for the facet that is hit by the oriented ray that emanates from the origin in the direction of the vector $\overrightarrow{p_\ell q_m}$ (this replaces the sign assignments used in the one-dimensional case and for the L_1-metric). Given such a candidate facet, $\mathsf{dist}(p_\ell, q_m)$ is a linear expression, and the extended Fredman trick, with all the follow-up for comparing costs of staircase paths can be applied, except that we also need to validate the correctness of our chosen candidate facet of K. This is done as follows.

Assume, without loss of generality, that each facet of K is a $(d-1)$-simplex (this can be achieved by a suitable triangulation of the facets). Consider a simplex-facet f, and let F be the

cone spanned by f with apex at the origin. F is the intersection of d halfspaces, each of the form $\langle h_i, x\rangle \geqslant 0$, for suitable normal unit vectors $h_1, \ldots, h_d$. In order to verify that the direction $\overrightarrow{p_\ell q_m}$ hits f, we need to verify that $\langle h_i, q_m - p_\ell\rangle \geqslant 0$, or that $\langle h_i, q_m\rangle \geqslant \langle h_i, p_\ell\rangle$, for $i = 1, \ldots, d$. These are d linear tests, which fit well into the frame of the extended Fredman trick (they replace the sign test that are used in the one-dimensional case, and in the L_1-case).

Again, omitting the further, rather routine details, we obtain a subquadratic algorithm for DTW in any fixed dimension, under any polyhedral metric, with the same runtime as in Theorem 4.1.1 and as stated in Theorem 4.1.2. The constant of proportionality depends on the dimension d, and on the complexity of the unit ball K of the metric (i.e., the number and complexity of its facets).

4.3.2 Lifting the General Position Assumption

In the algorithm above, we assumed that in each box $D_{i,j}$ there are no two staircase paths with the same cost. This assumption was crucial for preserving the overall output size of the dominance reporting routines to be $O(n^2/g^2)$. Specifically, all we need to ensure is that for each admissible boundary pair from $L \times R$, there will be only one staircase path with minimum cost. Our goal is to be able to break ties consistently. However, this is not trivial, as we must find a way to do it while using the Fredman-Chan mechanism. We can do it as follows.

In the preprocessing stage, our algorithm enumerated all the $< 3^{2g}$ staircase paths in a $g \times g$ grid. These enumerations are done in a natural lexicographic order and thus induce a total order on the staircase paths. Denote this total order by $\mathcal{L}$. (Note that $\mathcal{L}$ is independent of the values of A and B.)

Let A, B be two given input sequences of points (numbers) in $\mathbb{R}$ (a similar solution works for the extension to $\mathbb{R}^d$ under polyhedral metrics described above). First, sort A and B in increasing order in $O(n \log n)$ time. Find a *positive closest pair* $(a, b) \in A \times B$, i.e., a pair satisfying

$$|a - b| = \min_{(a_i, b_j) \in A \times B:\, |a_i - b_j| > 0} \{|a_i - b_j|\}\,.$$

This can be done while merging the sorted A and B, in $O(n)$ time. (If we are in a polyhedral $\mathbb{R}^d$ metric space we use a straightforward modification of the standard $O(2^d n \log n)$ divide-and-conquer closest pair algorithm of Bentley and Shamos [32, 33] to find the positive closest pair in the set $A \cup B$.) Put $\varepsilon = |a - b|$. For every boundary pair in $L \times R$ there are strictly fewer than 3^{2g} staircase paths, denote this number by r. Set

$$\varepsilon_1 = \varepsilon/3^{2g} < \varepsilon_2 = 2\varepsilon/3^{2g} < \varepsilon_3 = 3\varepsilon/3^{2g} < \cdots < \varepsilon_r = r\varepsilon/3^{2g}.$$

For every boundary pair $(v, w) \in L \times R$, and every staircase path $P_{v,w}$ (recall that $P_{v,w}$ is a sequence of positions in the $g \times g$ grid, and is independent of the values of A and B), we check for

the index k of $P_{v,w}$ in the total order $\mathcal{L}$, and add ε_k to $\text{cost}_{i,j}(P_{v,w})$, for every $i,j \in \left[\frac{n}{g-1}\right]$.

For a boundary pair $(v,w) \in L \times R$, let $P_{v,w}$ and $P'_{v,w}$ be two distinct staircase paths, and let $k,k' \in [r]$ be their corresponding (distinct) indices in the total order $\mathcal{L}$. Assume, without loss of generality, that $k < k'$ (it must be that either $k < k'$ or $k' < k$, since the two paths are distinct, and $\mathcal{L}$ is a total order). Since $\varepsilon_k < \varepsilon_{k'} < |a-b|$, it holds that for every $i,j \in \left[\frac{n}{g-1}\right]$,

$$\text{cost}_{i,j}(P_{v,w}) \leqslant \text{cost}_{i,j}(P'_{v,w}) \text{ if and only if } \text{cost}_{i,j}(P_{v,w}) + \varepsilon_k < \text{cost}_{i,j}(P'_{v,w}) + \varepsilon_{k'}.$$

We now proceed with the same steps of the algorithm we described in Section 4.3 (and 4.3.1) but with the modified path costs. (Note that we used the same $\varepsilon_1,\ldots,\varepsilon_r$ for all boxes $D_{i,j}$, thus we can still use the extended Fredman trick for the new costs.) By the above, ties on the original costs of (any) two distinct staircase paths break on their new costs, according to their order in $\mathcal{L}$, while the other relations $(<,>)$ are preserved.

4.4 Geometric Edit Distance in Subquadratic Time

In this section, we show how our DTW algorithm from Section 4.3 can be modified to compute $\mathsf{ged}(A,B)$ (and optimal matching). Recall the definitions of monotone matching (see Figure 4.1.1), $\mathsf{ged}(A,B)$, and optimal matching from Section 4.1.1. First, we overview the standard dynamic programming algorithm for computing GED between two sequences $A = (p_1,\ldots,p_n)$ and $B = (q_1,\ldots,q_n)$, each of n points in $\mathbb{R}$.

The Quadratic Time GED Algorithm.

1. Initialize an $(n+1)\times(n+1)$ matrix M and set $M[0,0] := 0$.
2. For each $\ell \in [n]$
2.1. $M[\ell,0] := \ell\rho$, $M[0,\ell] := \ell\rho$.
3. For each $\ell \in [n]$,
3.1. For each $m \in [n]$,
3.1.1 $M[\ell,m] := \min\Big\{M[\ell-1,m]+\rho,\ M[\ell,m-1]+\rho,\ M[\ell-1,m-1]+|p_\ell - q_m|\Big\}$.
4. Return $M[n,n]$.

The optimal matching can be retrieved by maintaining pointers from each (ℓ,m) to the preceding position $(\ell',m') \in \{(\ell-1,m),(\ell,m-1),(\ell-1,m-1)\}$ through which $M[\ell,m]$ is minimized. By tracing these pointers backwards from (n,n) to $(0,0)$ and including in the matching only the positions that we reach "diagonally" (when going backwards), we obtain the optimal matching.

Subquadratic Time GED Algorithm. Recall the all-pairs-distances matrix D and its decomposition into boxes $D_{i,j}$, as defined in Section 4.3. For a monotone matching $\mathcal{M}$ between two point-subsequences A_i, B_j, let $\text{cost}_{i,j}(\mathcal{M})$ be the corresponding sum of distances in the definition of $\mathsf{ged}(A_i, B_j)$. To adapt our DTW algorithm for GED, we modify the way we evaluate the cost of a staircase path P in a box $D_{i,j}$, so that it equals the cost of its corresponding monotone matching $\mathcal{M}(P)$ (defined below).

We view each box $D_{i,j}$ as a weighted directed grid graph G, whose vertices are the pairs of $[g] \times [g]$, and its set of edges is

$$\begin{aligned}&\big\{\langle(\ell, m), (\ell+1, m)\rangle \mid \ell \in [g-1],\ m \in [g]\big\}\\ \bigcup&\big\{\langle(\ell, m), (\ell, m+1)\rangle \mid \ell \in [g],\ m \in [g-1]\big\}\\ \bigcup&\big\{\langle(\ell, m), (\ell+1, m+1)\rangle \mid \ell, m \in [g-1]\big\}.\end{aligned}$$

We refer to the edges in the first subset as vertical edges, the edges in the second subset as horizontal edges, and the ones in the third subset as diagonal edges. The weight of the vertical and horizontal edges is set to ρ, and the weight of each diagonal edge $\langle(\ell, m), (\ell+1, m+1)\rangle$ is $|A_i(\ell) - B_j(m)|$. Each staircase path P in $D_{i,j}$ is then a path in the graph G, whose corresponding monotone matching $\mathcal{M}(P)$ is defined to consist of exactly all the pairs of points $(A_i(\ell), B_j(m))$ that correspond to the positions (ℓ, m) from the diagonal edges $\langle(\ell, m), (\ell+1, m+1)\rangle$ of the path.

By defining $\text{cost}_{i,j}(P)$ to be the weight of its corresponding path in G, we obtain that $\text{cost}_{i,j}(P) = \text{cost}_{i,j}(\mathcal{M})$, and that the dynamic programming matrix M (given above) satisfies that for each position $(\ell, m) \in [n+1] \times [n+1]$, $M[\ell, m]$ is the minimal cost of a staircase path from $(0, 0)$ to (ℓ, m) in D. This implies that Lemma 4.3.1 can be used in this setup too, for computing the values on the R-boundaries of the boxes $M_{i,j}$, as done in the second stage of our DTW algorithm. Thus, once we have a corresponding data structure from the preprocessing procedure, we can apply the second stage of our DTW algorithm verbatim.

As for the preprocessing procedure, the cost of a staircase path in a box $D_{i,j}$ is now a sum of distances $|A_i(\ell) - B_j(m)|$, $\ell, m \in [g]$, plus a multiple of the parameter ρ. Since ρ is a fixed real number and the multiple of ρ in the cost of a staircase path in $D_{i,j}$ only depends on the positions of the path (and is independent of the actual values of A and B), we can execute a similar machinery as described in Section 4.3. That is, we can choose a candidate sign assignment as before, get a linear expression in $A_i(\ell)$ and $B_j(m)$ (which also involves a fixed multiple of ρ), then, the extended Fredman trick (4.3) can be applied when comparing the costs of two staircase paths and validating the correctness of candidate sign assignments. (Our algorithm works also for more general gap penalty functions, as long as they are linear in the coordinates of the points of $A \cup B$.) The rest of the preprocessing procedure and the extension to high-dimensional polyhedral metric spaces are similar to those we showed for DTW. In order to lift the general position assumption, a tiny

modification to what is described in Section 4.3.2 is required; to set ε from Section 4.3.2 as the minimum over the distance of the positive closest pair from $A \cup B$ and ρ, the rest is verbatim.

From the above, we obtain that $\mathsf{ged}(A, B)$ (and an optimal matching) can be computed in $O(n^2/\log\log n)$ time, as stated in Theorems 4.1.1 and 4.1.2 for GED. □

4.5 Near-Linear Depth Decision Trees for Discrete Fréchet Distance under Polyhedral Metrics

The Fréchet distance is a measure of similarity between curves that takes into account the location and ordering of the points along the curves. Therefore it is often better than the well-known Hausdorff distance as a metric for comparing parameterized shapes. This measure was introduced by Fréchet in 1906 [90].

Eiter and Mannila [78] introduced the *discrete Fréchet distance*, a variant also known as the *coupling distance*. They showed that this distance provides a good approximation for the Fréchet distance between curves, and provided a quadratic dynamic programming algorithm to compute it.

Since then many studies have been made about the *discrete* problem in the *Euclidean plane*. To name a few, Agarwal *et al.* [7] showed a subquadratic algorithm that runs in $O(n^2 \log\log n/\log n)$ time, Buchin *et al.* [42] showed an *algebraic computation tree* lower bound of $\Omega(n \log n)$, and Bringmann [38] recently showed that there is no algorithm with runtime $O(n^{2-\Omega(1)})$, assuming the *Strong Exponential Time Hypothesis* (SETH).

While much work has been made on the Euclidean discrete Fréchet distance, the problem in other metrics, such as L_1 and L_∞ has been much less investigated.

A related recent lower bound by Bringmann and Mulzer [40] shows that, assuming SETH, the discrete Fréchet distance cannot be solved in $O\left(n^{2-\Omega(1)}\right)$ time even for the one-dimensional case (with the standard distance function $\mathsf{dist}(x, y) = |x - y|$). Their result is relevant to the problems we investigate in this section, as the one-dimensional case lower bound fits to any L_p norm, $1 \leqslant p \leqslant \infty$. In other words, their conditional lower bound holds for the discrete Fréchet distance under any L_p norm, $1 \leqslant p \leqslant \infty$ (including L_∞).

From now on, The term M-Discrete Fréchet Distance Decision refers to the *decision problem* of determining whether the discrete Fréchet distance is at most some given parameter $\varepsilon > 0$, when the underlying norm is M. The term M-Discrete Fréchet Distance refers to the problem of computing the actual discrete Fréchet distance, when the underlying norm is M.

Our Results and Related Works. Given two n-point-sequences A, B in $\mathbb{R}^d$, our contribution is stated in the following theorems.

Theorem 4.5.1. *There is a 2-linear decision tree with depth $O(n \log^2 n)$ for the L_∞-*Discrete Fréchet Distance *between A and B, for any constant dimension d.*

Theorem 4.5.2. *There is a $2d$-linear decision tree with depth $O(n \log^2 n)$ for the L_1-*Discrete Fréchet Distance *between A and B, for any constant dimension d.*

Theorem 4.5.3. *Given a polyhedral metric[2] M, there is a $2d$-linear decision tree with depth $O(n \log^2 n)$ for the M-*Discrete Fréchet Distance *between A and B, for any constant dimension d.*

Theorem 4.5.3 is a generalization of Theorem 4.5.2. However, since the L_1 metric is a quite popular polyhedral metric, we feel it is worth stating the L_1 case separately, as given in Theorem 4.5.2.

As mentioned above, the $\Omega(n^{2-o(1)})$ conditional lower bound of Bringmann and Mulzer [40] holds for the problems that we study. We find the big gap between their near-quadratic conditional lower bound in the uniform model to our near-linear upper bound in the linear decision tree model particularity interesting.

In a related result, Buchin *et al.* [43] showed that the algebraic decision tree complexity of the Euclidean-Discrete Fréchet Distance problem in the plane is $\widetilde{O}(n^{4/3})$. This result is obtained by using a range searching technique of Katz and Sharir [117]. In Section 4.5.2 we will briefly review this result, and in Section 4.5.3 we argue that, for the problem under polyhedral metrics (e.g., L_1 and L_∞) in $\mathbb{R}^d$, the standard range searching approach does not seem capable of giving us the results we aim for, which we will establish using a different approach.

To prove the theorems above, we use a variant of the extended Fredman's trick (see Section 2.2).

4.5.1 Problem Statement and Quadratic Algorithm

The Fréchet distance is often illustrated by a man and a dog, each walking along a path (curve). The man has the dog on a leash. Each of them may choose their own speed and may stop but cannot walk backwards. Then the Fréchet distance is the length of the shortest leash that allows them to walk on their respective curves from beginning to end.

More formally, following [78] we define a curve as a continuous mapping $f : [0, 1] \to V$, where (V, ρ) is a metric space. Given two curves $f : [0, 1] \to V$ and $g : [0, 1] \to V$, their Fréchet distance is defined as

$$\delta_F(f, g) = \inf_{\alpha,\beta} \max_{t\in[0,1]} \rho(f(\alpha(t)), g(\beta(t))),$$

where α and β are arbitrary continuous nondecreasing functions from $[0, 1]$ onto $[0, 1]$.

When computing the Fréchet distance between arbitrary curves, one typically approximates the curves by polygonal curves. Eiter and Mannila [78] defined the *discrete Fréchet distance* between

[2] That is, the underlying metric is induced by a norm, whose unit ball is a symmetric convex polytope with a constant number of facets (this constant generally depends on the dimension d).

polygonal curves and showed that it gives a good approximation to the Fréchet distance between them.

A polygonal curve with n edges is a curve $P : [0,1] \to V$, such that for each $i \in \{0,1,\ldots,n-1\}$, the restriction of P to the interval $\left[\frac{i}{n}, \frac{i+1}{n}\right]$ is affine. Since the Fréchet distance is invariant under reparametrization, we can assume a polygonal curve P to be given by the ordered list of its vertices, i.e., a sequence $P = (p_0, \ldots, p_n)$.

Let $P = (p_0, \ldots, p_n)$ and $Q = (q_0, \ldots, q_m)$ be two polygonal curves given by their ordered lists of vertices. As in the study of the DTW distance in Section 4.1, a *coupling* $C = (c_0, \ldots, c_k)$ between P and Q is an ordered sequence of distinct pairs of vertices in P, Q, such that $c_0 = (p_0, q_0)$, $c_k = (p_n, q_m)$ and $c_r = (p_i, q_j) \Rightarrow c_{r+1} \in \{(p_{i+1}, q_j), (p_i, q_{j+1}), (p_{i+1}, q_{j+1})\}$. The *discrete Fréchet distance* between P and Q is

$$\delta_{dF}(P,Q) = \min_{C \text{ coupling}} \max_{(p_i,q_j)\in C} \rho(p_i, q_j).$$

Eiter and Mannila [78] showed that

$$\delta_F(P,Q) \leqslant \delta_{dF}(P,Q) \leqslant \delta_F(P,Q) + \max\{D(P), D(Q)\},$$

where $D(P)$ (resp., $D(Q)$) is the length of the longest edge in P (resp., Q). Thus, if we add vertices to the curves P, Q so that their edge lengths tend to zero, their discrete Fréchet distance will tend to their Fréchet distance.

Dynamic Programming Algorithm. Following [78], we quickly review the standard quadratic dynamic programming algorithm for the decision version of the discrete Fréchet distance, in a metric space (V, ρ).

Given two point sequences $A = (a_1, \ldots, a_n)$, $B = (b_1, \ldots, b_n)$, and a parameter $\varepsilon > 0$, the algorithm creates an $n \times n$ Boolean matrix M, whose rows and columns correspond to the points of A and B, respectively. The algorithm fills the matrix with values 0/1 row by row. Every cell $M_{i,j}$ in the matrix is filled by 1 iff both conditions hold:

1. At least one of the cells $M_{i-1,j}$, $M_{i,j-1}$, $M_{i-1,j-1}$ is filled with 1.
2. The distance $\rho(a_i, b_j)$ is at most ε.

Otherwise, $M_{i,j}$ is filled by 0. Intuitively, an entry $M_{i,j}$ is equal to 1 iff the pair (a_i, b_j) is reachable from the starting placement (a_1, b_1) of the trip with a "leash" of length ε. Otherwise, $M_{i,j}$ is equal to 0.

The runtime of the algorithm is quadratic and the number of input comparisons it makes is also quadratic, as there are potentially n^2 distinct pairs of points (a_i, b_j) to check whether $\rho(a_i, b_j) \leqslant \varepsilon$.

4.5.2 Decision Tree for the Euclidean Plane

Buchin et al. [43] showed a quadratic algebraic decision tree (where each branching is a sign test of a quadratic expression) with depth $O(n^{4/3}\log^3 n)$ for the Euclidean-Discrete Fréchet Distance in the plane.

First, they construct a decision tree for the Euclidean-Discrete Fréchet Distance Decision, as follows. The decision tree is based on invoking the quadratic dynamic programming algorithm following a preprocessing stage. All the input comparisons in the dynamic programming algorithm are made by checking if the distance of a point $a_i \in A$ from a point $b_j \in B$ is less than the fixed given parameter ε. The preprocessing stage will compute and store the answers for these pairwise distance queries in a Boolean matrix $T := (t_{ij})$, where $t_{ij} = 1$ if $\|a_i - b_j\|_2 \leqslant \varepsilon$, otherwise $t_{ij} = 0$.

Given two point sequences A, B, with $|A| = n$, $|B| = m$, and a parameter $\varepsilon > 0$, denote, for each point $a \in A$, the circle of radius ε centered at a as c_a. A point $b \in B$ lies inside a circle c_a iff $\|a - b\|_2 \leqslant \varepsilon$. We obtain a set C of n congruent circles (all of radius ε) and a set $P = B$ of m points.

Katz and Sharir [117] showed that one can compute a compact representation of the set of pairs of the form (c, p), where $p \in P$, $c \in C$, and p lies inside c, in $O\left((m^{2/3}n^{2/3} + m + n)\log n\right)$ time (and thus, this bound holds also for the number of input comparisons). This information suffices to construct T and invoke the dynamic programming algorithm in $O(mn)$ time, but without using any further input comparisons. Thus, when $|A| = |B| = n$, the number of input comparisons is $O(n^{4/3}\log n)$.

Agarwal *et al.* [7] showed that the optimization problem can be solved by using a *distance selection* algorithm in the plane (that returns the k-th smallest distance in $A \times B$, for any prespecified k) to guide a binary search, using the decision procedure. Overall, there are $O(\log n)$ calls to the decision procedure and to the distance selection algorithm. The distance selection algorithm of Katz and Sharir [117] runs in $O(n^{4/3}\log^2 n)$ time. Thus, in total, we obtain that the quadratic algebraic decision tree complexity of Euclidean-Discrete Fréchet Distance in the plane is $O\left(n^{4/3}\log^3 n\right)$.

4.5.3 Decision Trees for the Decision Problem under Polyhedral Metrics

Similar to the Euclidean case, range searching techniques can also be used for the Discrete Fréchet Distance Decision problem under other metrics, for computing the pairwise distance queries in the decision tree. However, as we now show, these techniques, when routinely implemented for the L_∞ or L_1 metrics in $\mathbb{R}^d$, will give weaker bounds than those we aim for in Theorem 4.5.2 and Theorem 4.5.1.

The simpler case is for the L_∞ metric, for which the unit ball in $\mathbb{R}^d$ is a d-dimensional hypercube. One can compute a d-dimensional range tree data structure for the points of A, in time $O(n\log^{d-1} n)$. For each point $b = (b_1, \ldots, b_d) \in B$, denote by c_b its corresponding d-sphere (under

L_∞) of radius ε, centered at b. Clearly, c_b is a d-dimensional hypercube of side-length 2ε.

For each $b \in B$, we query the range tree with its corresponding hypercube c_b. This will give us a compact representation of all the points of A that lie in c_b. The cost of a query in a d-dimensional range tree is $O(\log^d n)$. Using *fractional cascading*, this can be improved to $O(\log^{d-1} n)$ time. In total, this approach leads to a 2-linear decision tree of depth $O(n \log^{d-1} n)$.

For the L_1 metric, its unit ball is a d-dimensional cross-polytope with 2^d facets. Thus, the most naive querying such a ball will require 2^d queries, each performing $O(\log n)$ $2d$-linear comparisons, resulting in a $2d$-linear decision tree of depth $O(n \log^{2^d} n)$.

The range searching data structure is appropriate also when the queries are not known in advance. Using Fredman's trick, we leverage the fact that in our case all the queries are known in advance, to obtain better decision trees.

Decision Tree for L_1-Discrete Fréchet Distance Decision. We start by presenting a 4-linear decision tree with depth $O(n \log n)$ for the L_1-Discrete Fréchet Distance Decision in $\mathbb{R}^2$, and then we explain how to modify it to obtain a $2d$-linear decision tree with depth $O(n \log n)$ for the problem in $\mathbb{R}^d$. In Section 4.5.4 we will show how to solve the optimization problem by running through this decision tree $O(\log n)$ times. This will prove Theorem 4.5.2.

The following property allows us to apply *Fredman's trick* on pairwise distance queries under the L_1 norm.

For any real numbers $x, y, z \in \mathbb{R}$, with $z \geqslant 0$, $|x| + |y| \leqslant z$ *if and only if* all the following inequalities hold.

$$\begin{aligned} x + y &\leqslant z, & x - y &\leqslant z, \\ -x + y &\leqslant z, & -x - y &\leqslant z. \end{aligned}$$

Since the L_1 distance between a point $a_i = (x_i, y_i)$ and a point $b_j = (x_j, y_j)$ is defined by

$$\|a_i - b_j\|_1 = |x_i - x_j| + |y_i - y_j|,$$

the property above leads to the following observation.

Observation 4.5.4. *For $a_i = (x_i, y_i)$, $b_j = (x_j, y_j) \in \mathbb{R}^2$, $\|a_i - b_j\|_1 \leqslant \varepsilon$ if and only if all the following inequalities hold.*

$$\begin{aligned} x_i + y_i &\leqslant x_j + y_j + \varepsilon, \\ x_i - y_i &\leqslant x_j - y_j + \varepsilon, \\ -x_i + y_i &\leqslant -x_j + y_j + \varepsilon, \\ -x_i - y_i &\leqslant -x_j - y_j + \varepsilon. \end{aligned}$$

This observation is a sort of generalization of Fredman's trick for the L_1 distance between two points in the plane.

Recall that we are given two point sequences in the plane $A = (a_1, \ldots, a_n)$, $B = (b_1, \ldots, b_n)$, and a distance parameter ε. The following algorithm determines whether $\delta_{dF}(A, B) \leqslant \varepsilon$.

1. Sort $D_1 := \{x_i + y_i,\ x'_j + y'_j + \varepsilon \mid a_i = (x_i, y_i) \in A,\ b_j = (x'_j, y'_j) \in B\}$.
2. Sort $D_2 := \{x_i - y_i,\ x'_j - y'_j + \varepsilon \mid a_i = (x_i, y_i) \in A,\ b_j = (x'_j, y'_j) \in B\}$.
3. Sort $D_3 := \{-x_i + y_i,\ -x'_j + y'_j + \varepsilon \mid a_i = (x_i, y_i) \in A,\ b_j = (x'_j, y'_j) \in B\}$.
4. Sort $D_4 := \{-x_i - y_i,\ -x'_j - y'_j + \varepsilon \mid a_i = (x_i, y_i) \in A,\ b_j = (x'_j, y'_j) \in B\}$.
5. Using Observation 4.5.4, given the sorted orders on $D_1, \ldots, D_4$, construct the $n \times n$ Boolean matrix
$$T := (t_{ij}),\ \text{where } t_{ij} = \begin{cases} 1 & \text{if } \|a_i - b_j\|_1 \leqslant \varepsilon \\ 0 & \text{otherwise.} \end{cases}$$
6. Invoke the dynamic programming algorithm using T to settle all the distance queries.

Steps 1–4 require $O(n \log n)$ comparisons. Using Observation 4.5.4, Step 5 requires no comparisons (on the raw data) at all, given the sorted orders on $D_1, \ldots, D_4$. Specifically, to test whether $\|a_i - b_j\|_1 \leqslant \varepsilon$, we test the four corresponding inequalities from Observation 4.5.4. Each inequality test is resolved by the sorted orders on $D_1, \ldots, D_4$. Step 6 requires no comparisons on the input data, given the matrix T from Step 5. All comparisons are sign tests of 4-linear expressions. In total, the number of comparisons is $O(n \log n)$. The algorithm can be *implemented* to run in $O(n^2)$ time (in the uniform model), using only $O(n \log n)$ input comparisons.

The algorithm can easily be extended to $\mathbb{R}^d$, by using additional sorting steps (similar to steps 1–4), which lead to a $2d$-linear decision tree with depth $O(n \log n)$ (where the constant of proportionality depends exponentially on d). A generalization of Observation 4.5.4 to points $a_i = (x_{i,1}, \ldots, x_{i,d})$, $b_j = (x_{j,1}, \ldots, x_{j,d})$ in $\mathbb{R}^d$ leads to 2^d inequalities, each defined by a vector $\delta \in \{-1, 1\}^d$, and has the form

$$\sum_{k=1}^{d} \delta_k x_{i,k} \leqslant \sum_{k=1}^{d} \delta_k x_{j,k} + \varepsilon. \tag{4.10}$$

Each such inequality is a $2d$-linear expression. Thus, for the same problem in $\mathbb{R}^d$, the algorithm has 2^d sorting steps, and all comparisons are sign tests of $2d$-linear expressions.

Decision Tree for L_∞-Discrete Fréchet Distance Decision. The previous algorithm can easily be modified (and simplified) for the L_∞-Discrete Fréchet Distance Decision problem, but now each comparison will use only two of the input terms, unlike the previous algorithm, where each comparison used $2d$ input terms. As before, we first consider the problem in $\mathbb{R}^2$, and later extend it to

$\mathbb{R}^d$. The L_∞ distance between a point $a_i = (x_i, y_i) \in \mathbb{R}^2$ and a point $b_j = (x_j, y_j) \in \mathbb{R}^2$ is defined by $\|a_i - b_j\|_\infty = \max\{|x_i - x_j|, |y_i - y_j|\}$. Hence,

$$\|a_i - b_j\|_\infty \leqslant \varepsilon \Leftrightarrow (|x_i - x_j| \leqslant \varepsilon) \wedge (|y_i - y_j| \leqslant \varepsilon).$$

Thus we obtain the following observation.

Observation 4.5.5. *For $a_i = (x_i, y_i)$, $b_j = (x_j, y_j) \in \mathbb{R}^2$, $\|a_i - b_j\|_\infty \leqslant \varepsilon$ if and only if all the following inequalities hold.*

$$x_i \leqslant x_j + \varepsilon, \qquad x_j \leqslant x_i + \varepsilon,$$
$$y_i \leqslant y_j + \varepsilon, \qquad y_j \leqslant y_i + \varepsilon.$$

This leads to the following variant of the previous algorithm given above, where the sets to be sorted in Steps 1–4 are:

$$D_1 := \{x_i,\ x'_j + \varepsilon \mid a_i = (x_i, y_i) \in A,\ b_j = (x'_j, y'_j) \in B\},$$
$$D_2 := \{x'_j,\ x_i + \varepsilon \mid a_i = (x_i, y_i) \in A,\ b_j = (x'_j, y'_j) \in B\},$$
$$D_3 := \{y_i,\ y'_j + \varepsilon \mid a_i = (x_i, y_i) \in A,\ b_j = (x'_j, y'_j) \in B\},$$
$$D_4 := \{y'_j,\ y_i + \varepsilon \mid a_i = (x_i, y_i) \in A,\ b_j = (x'_j, y'_j) \in B\}.$$

Using Observation 4.5.5, given the sorted orders on $D_1, \ldots, D_4$, one can construct the Boolean matrix T from Step 5 with no further comparisons. Then, one can invoke the dynamic programming algorithm and use T for the distance queries, as in Step 6.

Similarly to the L_1 norm, the above algorithm uses $O(n \log n)$ input comparisons, each of which is a sign test of a 2-linear expression, and can be implemented to run in $O(n^2)$ time (in the uniform model).

Following a generalization of Observation 4.5.5 to points in $\mathbb{R}^d$, the algorithm can be extended to $\mathbb{R}^d$ by adding additional sorting steps. We have $2d$ sorting steps for the problem in $\mathbb{R}^d$, two for each coordinate. Each comparison will still be a 2-linear expression. Thus in total we obtain a 2-linear decision tree with depth $O(n \log n)$ for the problem in $\mathbb{R}^d$, for any constant d. Here the constant of proportionality depends only linearly on d.

Extension to General Polyhedral Metrics in $\mathbb{R}^d$. The decision tree described above can be extended to work under general polyhedral metrics in $\mathbb{R}^d$, for any constant d. That is, we assume that the underlying metric is induced by a norm, whose unit ball K is a symmetric convex polytope with $|K| = O(1)$ facets. Each facet of K corresponds to (at most) $2d$-linear expression, similar to (4.10) but possibly with coefficients that are different from 1. Thus, after $|K|$ sorting steps we

can use Fredman's trick to obtain the matrix T from Step 5 with no further comparisons on the input data. The rest of the algorithm proceeds more or less verbatim.

4.5.4 Solving the Optimization Problem

In Section 4.5.3 we gave $2d$-linear decision trees with depth $O(n \log n)$, for the Discrete Fréchet Distance Decision problem under general polyhedral metrics, and a 2-linear decision tree with depth $O(n \log n)$ for the L_∞-Discrete Fréchet Distance Decision problem. In order to prove our results, we next show how to solve the Discrete Fréchet Distance optimization problem using a corresponding linear decision tree with depth $O(n \log^2 n)$.

We start with the L_∞-Discrete Fréchet Distance problem. Then, we show how to extend our approach for L_1, and then for general polyhedral metrics.

L_∞-Discrete Fréchet Distance. Our procedure is similar to one we will give in Section 5.4, for finding the closest pair of points under the L_∞ metric in $\mathbb{R}^d$, using the decision procedure for this problem. Thus, here we will give only the general idea, and we refer the reader to Section 5.4 for more technical details, as the procedures are more or less verbatim.

We are given two point-sequences $A = \{a_1, \ldots, a_n\}$, $B = \{b_1, \ldots, b_n\}$ in $\mathbb{R}^d$. The solution δ_0, for the L_∞-Discrete Fréchet Distance in $\mathbb{R}^d$, is one of the $O(dn^2)$ values $a_i[k] - b_j[k]$, $i, j \in [n]$, $k \in [d]$. For each $k \in [d]$, we sort the points of A and B in increasing order of their k-th coordinate. This takes $O(dn \log n)$ comparisons in total. Let $\left(a_1^{(k)}, \ldots, a_n^{(k)}\right)$ and $\left(b_1^{(k)}, \ldots, b_n^{(k)}\right)$ denote the sequences of the points of A and B sorted in increasing order of their k-th coordinate, respectively. For each $k \in [d]$, let $M^{(k)}$ be an $n \times n$ matrix, so that for $i, j \in [n]$, we have

$$M^{(k)}[i, j] = a_i^{(k)}[k] - b_j^{(k)}[k].$$

We view the row indices from bottom to top, i.e., the first row is the bottommost one, and the column indices from left to right. We are in fact interested only in the upper triangular portion of $M^{(k)}$, where its elements are positive, but for simplicity of presentation, we can ignore this issue (since for negative values the decision procedure will always return false anyway).

Observe that each row of $M^{(k)}$ is sorted in decreasing order and each column is sorted in increasing order. Under these conditions, the selection algorithm of Frederickson and Johnson [91] can find the t-th-smallest element of $M^{(k)}$, for any $1 \leqslant t \leqslant n^2$, in $O(n)$ time.[3]

We use this method to conduct a simultaneous binary search, using the decision procedure from Section 4.5.3, over all d matrices $M^{(k)}$, to find δ_0. At each step of the search we maintain two counters $L_k \leqslant H_k$, for each k. Initially $L_k = 1$ and $H_k = n^2$. The invariant that we maintain

[3] Simpler algorithms can select the t-th-smallest element in such cases in $O(n \log n)$ time, which is also sufficient for our result.

is that, at each step, δ_0 lies in between the L_k-th and the H_k-th smallest elements of $M^{(k)}$, for each k.

For more technical details, we refer the reader to the procedure in Section 5.4, which is more or less verbatim. Overall, this procedure stops after $O(\log n + \log d)$ calls to the corresponding decision procedure from Section 4.5.3 and $O(d \log n)$ calls to the the selection algorithm of Frederickson and Johnson [91]. Thus, in total, we obtain that the number of input comparisons is bounded by $O(n \log^2 n)$, for any constant d. Note that each comparison we made involves only two input terms, thus we obtain a 2-linear decision tree with depth $O(n \log^2 n)$, for any constant d, where the constant of proportionality depends linearly on d. This completes the proof of Theorem 4.5.1.

L_1-Discrete Fréchet Distance. The technique described above for the L_∞ metric can easily be extended to the L_1 metric. The L_1 distance between a pair of points $a, b \in \mathbb{R}^d$ is $\sum_{k=1}^{d} |a[k] - b[k]|$, which can be written as $\sum_{k=1}^{d} \delta[k]\,(a[k] - b[k])$, for a suitable sign vector $\delta = (\delta[1], \ldots, \delta[d])$, which depends on a and b, where each of its entries $\delta[i]$, $i \in [d]$, is 1 or -1.

We iterate over all 2^d sign vectors. For each such vector δ, we form the following two sequences, each of which is sorted in increasing order, $A^{(\delta)} = (a_1^{(\delta)}, \ldots, a_n^{(\delta)})$ and $B^{(\delta)} = (b_1^{(\delta)}, \ldots, b_n^{(\delta)})$, where [4]

$$a_i^{(\delta)} = \sum_{k=1}^{d} \delta[k] a_i[k]$$
$$b_i^{(\delta)} = \sum_{k=1}^{d} \delta[k] b_i[k],$$

for $i \in [n]$. Then, for each pair $a_i \in A$, $b_j \in B$, there exists a sign vector δ such that the L_1 distance between a_i and b_j is $a_i^{(\delta)} - b_j^{(\delta)}$. In analogy with the L_∞ case, we define, for each δ, the matrix $M^{(\delta)}$ so that

$$M^{(\delta)}[i, j] = a_i^{(\delta)} - b_j^{(\delta)},$$

for $i, j \in [n]$. As before, each row (resp., column) of each of these matrices is sorted in increasing (resp., decreasing) order (since the sequences $A^{(\delta)}$ and $B^{(\delta)}$ are sorted in increasing order). We now have to search simultaneously through all these 2^d matrices for the entry that gives the discrete Fréchet distance between A and B, and we do it in full analogy to the way it was done in the L_∞ case, except that the number of matrices increases from d to 2^d. Thus, we pay $O(2^d n \log n)$ comparisons to sort the elements of $A^{(\delta)}$ and $B^{(\delta)}$, for each possible sign vector δ. We have $O(\log n + d)$ calls to the corresponding decision procedure from Section 4.5.3 and $O(2^d \log n)$ calls to the the selection algorithm of Frederickson and Johnson [91]. Thus, in total, we pay $O(n \log^2 n)$ comparisons, for any constant d, where the constant of proportionality depends exponentially on d.

[4] Note the slight abuse of notation, as the order of the indices in the sorted sequences $A^{(\delta)}$ and $B^{(\delta)}$ depends on the sign vector δ.

This completes the proof of Theorem 4.5.2.

Discrete Fréchet Distance under General Polyhedral Metrics. The case of a general polyhedral metric is handled similarly to the L_1 and L_∞ cases. For each facet f of the unit ball K of the metric, with normal vector $\mathbf{n}_f$, we form the following two sequences, each one is sorted in increasing order, $A^{(f)} = (a_1^{(f)}, \ldots, a_n^{(f)})$ and $B^{(f)} = (b_1^{(f)}, \ldots, b_n^{(f)})$, where [5]

$$a_i^{(f)} = \langle a_i, \mathbf{n}_f \rangle$$
$$b_i^{(f)} = \langle b_i, \mathbf{n}_f \rangle,$$

for $i \in [n]$. Then, for each pair $a_i \in A$, $b_j \in B$, there exists a facet f such that the polyhedral distance between a_i and b_j is $a_i^{(f)} - b_j^{(f)}$. This allows us to adapt the algorithm for the L_1 distance to this case too, where the number of matrices in which we search is the number of facets $|K|$ of the unit ball K of the metric.

In this case, we pay $O(|K| n \log n)$ comparisons to sort the elements of $A^{(f)}$ and $B^{(f)}$. We have $O(\log n + \log |K|)$ calls to the corresponding decision procedure from Section 4.5.3 and $O(|K| \log n)$ calls to the the selection algorithm of Frederickson and Johnson [91]. Thus, in total, we pay $O(n \log^2 n)$ comparisons, for any constants d and $|K|$, where the constant of proportionality depends linearly on the number of facets $|K|$. This completes the proof of Theorem 4.5.3.

[5] Here too, the order of the indices in the sorted sequences $A^{(\delta)}$ and $B^{(\delta)}$ depends on the facet f.

Chapter 5

High Dimensional Closest Pair under L_∞ and Dominance Product

5.1 Background

Finding the closest pair among a set of n points in $\mathbb{R}^d$ was among the first studied algorithmic geometric problems, considered at the origins of computational geometry; see [136, 144]. The distance between pairs of points is often measured by the L_τ metric, for some $1 \leqslant \tau \leqslant \infty$, under which the distance between the points $p_i = (p_i[1], \ldots, p_i[d])$ and $p_j = (p_j[1], \ldots, p_j[d])$ is

$$\mathsf{dist}_\tau(p_i, p_j) = \|p_i - p_j\|_\tau = \left(\sum_{k=1}^{d} \left| p_i[k] - p_j[k] \right|^\tau \right)^{1/\tau},$$

for $\tau < \infty$, and

$$\mathsf{dist}_\infty(p_i, p_j) = \|p_i - p_j\|_\infty = \max_k \left| p_i[k] - p_j[k] \right|,$$

for $\tau = \infty$. The Closest Pair problem and its corresponding *decision* variant, under the L_τ-metric, are defined as follows.

Closest Pair: Given a set S of n points in $\mathbb{R}^d$, find a pair of distinct points $p_i, p_j \in S$ such that $\mathsf{dist}_\tau(p_i, p_j) = \min_{\ell \neq m} \{\mathsf{dist}_\tau(p_\ell, p_m) \mid p_\ell, p_m \in S\}$.

Closest Pair Decision: Given a set S of n points in $\mathbb{R}^d$, and a parameter $\delta > 0$, determine whether there is a pair of distinct points $p_i, p_j \in S$ such that $\mathsf{dist}_\tau(p_i, p_j) \leqslant \delta$.

Throughout this chapter, the notation L_τ Closest Pair refers to the Closest Pair problem under some *specific* metric L_τ, for $1 \leqslant \tau \leqslant \infty$ (and we will mostly consider the case $\tau = \infty$).

In the algebraic computation tree model, the Closest Pair problem has a complexity lower bound of $\Omega(n \log n)$ (for any L_τ metric), even for the one-dimensional case $d = 1$, as implied from a lower bound for the Element-Uniqueness problem [31].

As for upper bounds, Bentley and Shamos [32, 33] were the first who gave a deterministic algorithm for finding the closest pair under the L_2 metric that runs in $O(n \log n)$ time for any *constant* dimension $d \geqslant 1$, which is optimal in the algebraic computation tree model, for any fixed d. Their algorithm uses the *divide-and-conquer* paradigm, and became since, a classical textbook example for this technique. In 1976 Rabin presented, in a seminal paper [139], a *randomized* algorithm that finds the closest pair in $O(n)$ expected time, using the *floor* function (which is not included in the algebraic computation tree model). His algorithm uses random sampling to decompose the problem into smaller subproblems, and uses the floor function in solving them, for a total of $O(n)$ expected time. Later, in 1979, Fortune and Hopcroft [89] gave a deterministic algorithm that uses the floor function, and runs in $O(n \log \log n)$ time.

The bounds above hold as long as the dimension d is constant, as they involve factors that are exponential in d. Thus, when d is large (e.g., $d = n$), the problem seems to be much less understood. Shamos and Bentley [33] conjectured in 1976 that, for $d = n$, and under the L_2 metric, the problem

can be solved in $O(n^2 \log n)$ time; so far, their conjectured bound is considerably far from the $O(n^\omega)$ state-of-the-art time bound for this case [112], where $\omega < 2.373$ denotes the exponent for matrix multiplication (see below). If one settles for approximate solutions, many efficient algorithms were developed in the last two decades, mostly based on LSH (locality sensitive hashing) schemes, and dimensionality reduction via the Johnson-Lindenstrauss transform; see [13, 19] for examples of such algorithms. These algorithms are often used for finding *approximate nearest neighbors*, which itself is of major importance and in massive use in many practical fields of computer science. Nevertheless, finding an *exact* solution seems to be a much harder task.

We consider the case where d depends on n, assuming specifically that $d = n^r$ for some $r > 0$. Note that a naive brute-force algorithm runs in $O(n^2 d)$ time and works for any metric L_τ. For some L_τ metrics, a much faster solution is known; see [112]. Specifically, the L_2 Closest Pair problem can be solved by one algebraic matrix multiplication, so for example when $d = n$, it can be solved in $O(n^\omega)$ time (as already mentioned above). If $\tau \geqslant 2$ is an *even* integer, then L_τ Closest Pair can be solved in $O(\tau n^\omega)$ time. However, for other L_τ metrics, such as when τ is *odd* (or fractional), or the L_∞ metric, the known solutions are significantly inferior.

For the L_1 and L_∞ metrics, Indyk *et al.* [112] obtained the first (and best known until now) non-naive algorithms for the case $d = n$. For L_1, they gave an algorithm that runs in $O\left(n^{\frac{\omega+3}{2}}\right) = O(n^{2.687})$ time, and for L_∞, one that runs in $O\left(n^{\frac{\omega+3}{2}} \log \mathcal{D}\right) = O(n^{2.687} \log \mathcal{D})$ time, where $\mathcal{D}$ is the diameter of the given point-set. The bound for L_∞ is *weakly polynomial*, due to the dependence on $\mathcal{D}$, and, for real data, only yields an approximation. Their paper is perhaps the most related previous work to our study.

Our new approach is based on two main observations. The first is showing a reduction from L_∞ Closest Pair Decision to another well-studied problem, *dominance product*. The second is by showing we can solve the optimization problem deterministically by executing the decision procedure only $O(\log n)$ times.

We also give improved runtime analysis for the dominance product problem, defined as follows.

Dominance Product: given a set S of n points $p_1, \ldots, p_n$ in $\mathbb{R}^d$, compute a matrix D such that for each $i, j \in [n]$, $D[i, j] = \left|\{k \mid p_i[k] \leqslant p_j[k]\}\right|$.

This matrix is called the *dominance product* or *dominance matrix* for S. For $d = n$, there is a non-trivial strongly subcubic algorithm by Matoušek [129] (see Section 5.2), and a slightly improved one by Yuster [158]. For $d \leqslant n$, there are extensions of Matoušek's algorithm by Vassilevska-Williams, Williams, and Yuster [149]. All of them use fast matrix multiplications.

Dominance product computations were liberally used to improve some fundamental algorithmic problems. For example, Vassilevska-Williams, Williams, and Yuster [149] gave the first strongly subcubic algorithm for the *all pairs bottleneck paths* (APBP) problem, using dominance product

computations. Duan and Pettie [74] later improved their algorithm, also by using dominance product computations. Yuster [158] showed that APSP can be solved in strongly subcubic time if the number of distinct weights of edges emanating from any fixed vertex is $O(n^{0.338})$. In his algorithm, he uses dominance product computation as a black box.

5.1.1 Summary of Our Results

Let $DP(n, d)$ denote an upper bound on the runtime of computing the dominance product (defined above) of n points in $\mathbb{R}^d$. We obtain the following results for the L_∞ Closest Pair problem in $\mathbb{R}^d$, where $d = n^r$, for some $r > 0$.

Theorem 5.1.1. L_∞ Closest Pair *can be solved by a deterministic algorithm that runs in* $O(DP(n, d) \log n)$ *time.*

Theorem 5.1.1 improves the $O(n^{2.687} \log \mathcal{D})$ time bound of Indyk *et al.* [112] (see above) in two aspects. First, the polynomial factor $n^{2.687}$ goes slightly down to $DP(n, n) = n^{2.684}$, which we then improve further to $n^{2.6598}$ in Theorem 5.1.4; this holds also for Theorem 5.1.2, stated below. The second aspect is that the $\log \mathcal{D}$ factor is replaced by a $\log n$ factor, which makes our algorithm strongly-polynomial, independent of the diameter of the given point-set, and yields an exact result also for points with real coordinates.

For the proof of Theorem 5.1.1, we first show a reduction from L_∞ Closest Pair Decision to dominance product computation, then we show that the optimization problem can be solved deterministically by executing the decision procedure only $O(\log n)$ times.

Theorem 5.1.2. L_∞ Closest Pair *can be solved by a randomized algorithm that runs in* $O(DP(n, d))$ *expected time.*

Theorem 5.1.3. *For points with integer coordinates from* $[-M, M]$, L_∞ Closest Pair *can be solved by a deterministic algorithm that runs in* $\widetilde{O}\left(\min\{Mn^{\omega(1,r,1)}, DP(n, d)\}\right)$ *time.*

From Theorem 5.1.3 we obtain that for n points in $\mathbb{R}^n$ with small integer coordinates we can solve the *optimization* problem in $O(n^\omega)$ time, which is a significant improvement compared to the general case from Theorems 5.1.1 and 5.1.2.

Additionally, we give a coherent spelled-out runtime analysis for obtaining the best bounds for $DP(n, d)$, for the entire range $d = n^r$, where $0 \leqslant r \leqslant 1.056$, using rectangular matrix multiplications. We demonstrate the use of our analysis by plugging into it the improved bounds for rectangular matrix multiplication by Le Gall [124], resulting in the bounds given below. Recently, Le Gall and Urrutia [126] reported further improvements on the bounds given in [124]. Their new bounds can be plugged into our analysis to give approximately 0.01 improvements on the exponents given below.

Theorem 5.1.4. *Given a set S of n points $p_1, \ldots, p_n$ in $\mathbb{R}^d$, the dominance product of S can be computed in $O(DP(n,d))$ time, where*

$$DP(n,d) \leqslant \begin{cases} d^{0.697}n^{1.896} + n^{2+o(1)} & \text{if } d \leqslant n^{\frac{\omega-1}{2}} \leqslant n^{0.687} \\ d^{0.909}n^{1.75} & \text{if } n^{0.687} \leqslant d \leqslant n^{0.87} \\ d^{0.921}n^{1.739} & \text{if } n^{0.87} \leqslant d \leqslant n^{0.963} \\ d^{0.931}n^{1.73} & \text{if } n^{0.963} \leqslant d \leqslant n^{1.056} \end{cases}.$$

In particular, we obtain that $DP(n,n) = n^{2.6598}$ (using a more precise calculation), which improves Yuster's $O(n^{2.684})$ time bound. As mentioned above, this bound can be slightly improved, using the new rectangular matrix multiplication bounds of Le Gall and Urrutia [126].

5.2 Dominance Product

Recall the dominance product problem: given n points $p_1, \ldots, p_n$ in $\mathbb{R}^d$, we want to compute a matrix D such that for each $i, j \in [n]$,

$$D[i,j] = \Big|\{k \mid p_i[k] \leqslant p_j[k]\}\Big|.$$

It is easy to see that the matrix D can be computed naively in $O(dn^2)$ time. Note that, in terms of decision tree complexity, it is straightforward to show that $O(dn \log n)$ pairwise comparisons suffice for computing the dominance product of n points in $\mathbb{R}^d$. However, the actual best known time bound to solve this problem is significantly larger than its decision tree complexity bound.

The first who gave a truly subcubic algorithm to compute the dominance product of n points in $\mathbb{R}^n$ is Matoušek [129]. We first outline his algorithm, and then present our extension and improved runtime analysis.

Theorem 5.2.1 (Matoušek [129])**.** *Given a set S of n points in $\mathbb{R}^n$, the dominance matrix for S can be computed in $O(n^{\frac{3+\omega}{2}}) = O(n^{2.687})$ time.*

Proof. For each $j \in [n]$, sort the n points by their j-th coordinate. This takes a total of $O(n^2 \log n)$ time. Define the *j-th rank of point p_i*, denoted as $r_j(p_i)$, to be the position of p_i in the sorted list for coordinate j. Let $s \in [\log n, n]$ be a parameter to be determined later. Define n/s pairs (assuming for simplicity that n/s is an integer) of $n \times n$ Boolean matrices $(A_1, B_1), \ldots, (A_{n/s}, B_{n/s})$ as follows:

$$A_k[i,j] = \begin{cases} 1 & \text{if } r_j(p_i) \in [ks, ks+s) \\ 0 & \text{otherwise,} \end{cases} \qquad B_k[i,j] = \begin{cases} 1 & \text{if } r_j(p_i) \geqslant ks+s \\ 0 & \text{otherwise,} \end{cases}$$

for $i, j \in [n]$. Put $C_k = A_k \cdot B_k^T$. Then $C_k[i,j]$ equals the number of coordinates t such that $r_t(p_i) \in [ks, ks+s)$, and $r_t(p_j) \geqslant ks+s$.

Thus, by letting $C = \sum_{k=1}^{n/s} C_k$, we have that $C[i,j]$ is the number of coordinates t such that $p_i[t] \leqslant p_j[t]$ and $\lfloor r_t(p_i)/s \rfloor < \lfloor r_t(p_j)/s \rfloor$.

Next, we compute a matrix E such that $E[i,j]$ is the number of coordinates t such that $p_i[t] \leqslant p_j[t]$ and $\lfloor r_t(p_i)/s \rfloor = \lfloor r_t(p_j)/s \rfloor$. Then $D := C + E$ is the desired dominance matrix.

To compute E, we use the n sorted lists we computed earlier. For each pair $(i,j) \in [n] \times [n]$, we retrieve $q := r_j(p_i)$. By reading off the adjacent points that precede p_i in the j-th sorted list in reverse order (i.e., the points at positions $q-1$, $q-2$, etc.), and stopping as soon as we reach a point p_k such that $\lfloor r_j(p_k)/s \rfloor < \lfloor r_j(p_i)/s \rfloor$, we obtain the list $p_{i_1}, \ldots, p_{i_l}$ of $l \leqslant s$ points such that $p_{i_x}[j] \leqslant p_i[j]$ and $\lfloor r_j(p_i)/s \rfloor = \lfloor r_j(p_{i_x})/s \rfloor$. For each $x = 1, \ldots, l$, we add a 1 to $E[i_x, i]$. Assuming constant time lookups and constant time probes into a matrix (as is standard in the Real RAM model), this entire process takes only $O(n^2 s)$ time. The runtime of the above procedure is therefore $O(n^2 s + \frac{n}{s} \cdot n^\omega)$. Choosing $s = n^{\frac{\omega-1}{2}}$, the time bound becomes $O(n^{\frac{3+\omega}{2}})$. □

Yuster [158] has slightly improved this algorithm to run in $O(n^{2.684})$ time, by using rectangular matrix multiplication.

5.2.1 Generalized and Improved Bounds

We extend Yuster's idea to obtain bounds for dimension $d = n^r$, for the entire range $r > 0$, and, at the same time, give an improved time analysis, using the recent bounds for rectangular matrix multiplications of Le Gall [124] coupled with an interpolation technique. This analysis is not trivial, as Le Gall's bounds for $\omega(1, r, 1)$ are obtained by solving a nonlinear optimization problem, and are only provided for a few selected values of r (see Table 1 in [124]). Combining Le Gall's exponents with an interpolation technique, similar to the one used by Huang and Pan [109], we obtain improved bounds for all values $d = n^r$, for any $r > 0$.

Note that the matrices A_k and B_k, defined above, are now $n \times d$ matrices. Thus, the sum C defined earlier, can be viewed as a product of block matrices

$$C = \begin{bmatrix} A_1 & A_2 & \cdots & A_{n/s} \end{bmatrix} \cdot \begin{bmatrix} B_1^T \\ B_2^T \\ \vdots \\ B_{n/s}^T \end{bmatrix}.$$

Thus, to compute C we need to multiply an $n \times (dn/s)$ matrix by a $(dn/s) \times n$ matrix. Computing E in this case can be done exactly as in Matoušek's algorithm, in $O(nds)$ time.

Consider first the case where d is small; concretely, $d \leqslant n^{\frac{\omega-1}{2}}$. In this case we compute C using

r	ω	ζ
$r_0 = 1.0$	$\omega_0 = 2.372864$	$\zeta_0 = 0.6865$
$r_1 = 1.1$	$\omega_1 = 2.456151$	$\zeta_1 = 0.7781$
$r_2 = 1.2$	$\omega_2 = 2.539392$	$\zeta_2 = 0.8697$
$r_3 = 1.3$	$\omega_3 = 2.624703$	$\zeta_3 = 0.9624$
$r_4 = 1.4$	$\omega_4 = 2.711707$	$\zeta_4 = 1.0559$

Table 5.1: The relevant entries from Le Gall's table (Table 1 in [124]), the value for ω_0 is taken from [125]. The dominance product can be computed in $O(n^{\omega_i})$ time, for dimension $d_i = n^{\zeta_i}$.

the following result by Huang and Pan.

Lemma 5.2.2 (Huang and Pan [109]). *Let $\alpha = \sup\{0 \leqslant r \leqslant 1 \mid w(1,r,1) = 2 + o(1)\}$. Then for all $n^\alpha \leqslant m \leqslant n$, one can multiply an $n \times m$ matrix with an $m \times n$ matrix in time $O\left(m^{\frac{\omega-2}{1-\alpha}} n^{\frac{2-\omega\alpha}{1-\alpha}}\right)$.*

Huang and Pan [109] showed that $\alpha > 0.294$. Recently, Le Gall [124] improved the bound on α to $\alpha > 0.302$. By plugging this into Lemma 5.2.2, we obtain that multiplying an $n \times m$ matrix with an $m \times n$ matrix, where $n^\alpha \leqslant m \leqslant n$, can be done in time $O(m^{0.535} n^{1.839})$.

From the above, computing C and E can be done in $O\left((dn/s)^{0.535} n^{1.839} + dns\right)$ time. By choosing $s = n^{0.896}/d^{0.303}$, the runtime is asymptotically minimized, and we obtain the time bound $O(d^{0.697} n^{1.896})$. This time bound holds only when $n^\alpha < n^{0.302} \leqslant dn/s \leqslant n$, which yields the time bound

$$O(d^{0.697} n^{1.896} + n^{2+o(1)}), \text{ for } d \leqslant n^{(\omega-1)/2} \leqslant n^{0.687}.$$

We now handle the case $d > n^{(\omega-1)/2}$. Note that in this case, $dn/s > n$ (for s as above), thus, we cannot use the bound from Lemma 5.2.2. Le Gall [124] gives a table (Table 1 in [124]) of values r (he refers to them as k), including values of $r > 1$ (which are those we need), with various respective exponents $\omega(1,r,1)$. We will confine ourselves to the given bounds for the values $r_1 = 1.1$, $r_2 = 1.2$, $r_3 = 1.3$, and $r_4 = 1.4$. We denote their corresponding exponents $\omega(1,r_i,1)$ by

$$\omega_1 \leqslant 2.456151, \quad \omega_2 \leqslant 2.539392, \quad \omega_3 \leqslant 2.624703, \quad \omega_4 \leqslant 2.711707$$

respectively. The exponent for $r_0 = 1$ is $\omega_0 = \omega \leqslant 2.372864$ (see [125, 155]).

The algorithm consists of two parts. For a parameter s, that we will fix shortly, the cost of computing $C = A \cdot B^T$ is $O\left(n^{\omega_r}\right)$, where ω_r is a shorthand notation for $\omega(1,r,1)$, and where $n^r = dn/s$, and the cost of computing E is $O(nds) = O\left(s^2 n^r\right)$. Dropping the constants of proportionality, and equating the two expressions, we choose

$$s = n^{(\omega_r - r)/2}, \qquad \text{that is,} \qquad d = sn^{r-1} = n^{(\omega_r + r)/2 - 1} = n^{\zeta_r},$$

for $\zeta_r = (\omega_r + r)/2 - 1$. Put $\zeta_i = \zeta_{r_i}$, for the values $r_0, \ldots, r_4$ mentioned earlier; see Table 5.1.

Now if we are lucky and $d = n^{\zeta_i}$, for $i = 0, 1, 2, 3, 4$, then the overall cost of the algorithm is

$\zeta_{\min}$	$\zeta_{\max}$	u	v
0.687	0.87	0.909	1.75
0.87	0.963	0.921	1.739
0.963	1.056	0.931	1.73

Table 5.2: The time bound for computing dominance product for n points in dimension $n^{\zeta_{\min}} \leqslant d \leqslant n^{\zeta_{\max}}$ is $O\left(d^u n^v\right)$.

$O(n^{\omega_i})$. For in-between values of d, we need to interpolate, using the following bound, which is derived in the earlier studies (see, e.g., Huang and Pan [109]), and which asserts that, for $a \leqslant r \leqslant b$, we have

$$\omega_r \leqslant \frac{(b-r)\omega_a + (r-a)\omega_b}{b-a}. \tag{5.1}$$

That is, given $d = n^{\zeta}$, where $\zeta_i \leqslant \zeta \leqslant \zeta_{i+1}$, for some $i \in \{0, 1, 2, 3\}$, the cost of the algorithm will be $O\left(n^{\omega_r}\right)$, where r satisfies

$$\zeta = \zeta_r = \frac{\omega_r + r}{2} - 1.$$

Substituting the bound for ω_r from (5.1), with $a = r_i$ and $b = r_{i+1}$, we have

$$\frac{(r_{i+1}-r)\omega_i + (r-r_i)\omega_{i+1}}{r_{i+1}-r_i} + r = 2(\zeta+1).$$

Eliminating r, we get

$$r = \frac{2(\zeta+1)(r_{i+1}-r_i) - r_{i+1}w_i + r_i w_{i+1}}{w_{i+1} + r_{i+1} - w_i - r_i}, \tag{5.2}$$

and the cost of the algorithm will be $O\left(n^{\omega_r}\right)$, where

$$\omega_r \leqslant \frac{(r_{i+1}-r)\omega_i + (r-r_i)\omega_{i+1}}{r_{i+1}-r_i}. \tag{5.3}$$

Note that r is a linear function of ζ, and so is ω_r. Writing $\omega_r = u\zeta + v$, the cost is

$$O\left(n^{\omega_r}\right) = O\left(n^{u\zeta+v}\right) = O\left(d^u n^v\right).$$

The values of u and v for each of our intervals are given in Table 5.2. (The first row covers the two intervals $1.0 \leqslant r \leqslant 1.1$ and $1.1 \leqslant r \leqslant 1.2$, as the bounds happen to coincide there.) See also Theorem 5.1.4 in Section 5.1.1. We have provided explicit expressions for $DP(n, d)$ only for $d \leqslant n^{\zeta_4} = n^{1.056}$, which includes the range $d \leqslant n$, which is the range one expects in practice. Nevertheless, the recipe that we provide can also be applied to larger values of d, using larger entries from Le Gall's table [124]. As mentioned earlier, the exponents we obtained for $DP(n, d)$ can be even slightly further improved by approximately 0.01, by plugging into our analysis the recent new bounds for rectangular matrix multiplication of Le Gall and Urrutia [126] (see Table 3 in [126]).

5.3 Reducing L_∞ Closest Pair Decision to Dominance Product

Recall that, given a set S of n points $p_1, \ldots, p_n$ in $\mathbb{R}^d$, the L_∞ Closest Pair problem is to find a pair of points (p_i, p_j), such that $i \neq j$ and $\|p_i - p_j\|_\infty = \min_{\ell \neq m \in [n]} \|p_\ell - p_m\|_\infty$. The corresponding decision version of this problem is to determine whether there is a pair of distinct points (p_i, p_j) such that $\|p_i - p_j\|_\infty \leqslant \delta$, for a given $\delta > 0$.

Naively, we can compute all the distances between every pair of points in $O(n^2 d)$ time, and choose the smallest one. However, as we see next, a significant improvement can be achieved, for any $d = n^r$, for any $r > 0$.

Specifically, we first obtain the following theorem.

Theorem 5.3.1. *Given a parameter $\delta > 0$, and a set S of n points $p_1, \ldots, p_n$ in $\mathbb{R}^d$, the set of all pairs (p_i, p_j) with $\|p_i - p_j\|_\infty \leqslant \delta$, can be computed in $O(DP(n, d))$ time.*

Proof. First, we note the following trivial but useful observation (also noted in Section 4.5).

Observation 5.3.2. *For a pair of points $p_i, p_j \in \mathbb{R}^d$, $\|p_i - p_j\|_\infty \leqslant \delta \iff p_i[k] \leqslant p_j[k] + \delta$ and $p_j[k] \leqslant p_i[k] + \delta$, for every coordinate $k \in [d]$.*

Indeed, a pair of points (p_i, p_j) satisfies $\|p_i - p_j\|_\infty = \max_{k \in [d]} |p_i[k] - p_j[k]| \leqslant \delta \iff$ for every coordinate $k \in [d]$, $|p_i[k] - p_j[k]| \leqslant \delta$. The last inequalities hold iff $p_i[k] - p_j[k] \leqslant \delta$ and $p_j[k] - p_i[k] \leqslant \delta$, or, equivalently, iff $p_i[k] \leqslant p_j[k] + \delta$ and $p_j[k] \leqslant p_i[k] + \delta$, for each $k \in [d]$. Although the rephrasing in the observation is trivial, it is crucial for our next step. It can be regarded as a (simple) variant of Fredman's trick (see Section 2.2 and [92]).

For every $i \in [n]$ we create a new point $p_{n+i} = p_i + (\delta, \delta, \ldots, \delta)$. Thus in total, we now have $2n$ points. Concretely, for every $i \in [n]$, we have the points

$$\begin{aligned} p_i &= \big(\quad p_i[1], \quad p_i[2], \quad \ldots, \quad p_i[d] \quad \big), \\ p_{n+i} &= \big(\quad p_i[1] + \delta, \quad p_i[2] + \delta, \quad \ldots, \quad p_i[d] + \delta \quad \big). \end{aligned}$$

We compute the dominance matrix D_δ for these $2n$ points, using the algorithm from Section 5.2.1. By Observation 5.3.2, a pair of points (p_i, p_j) satisfies

$$\|p_i - p_j\|_\infty \leqslant \delta \iff (D_\delta[i, n+j] = d) \ \wedge \ (D_\delta[j, n+i] = d)\,,$$

so we can find all these pairs in $O(n^2)$ additional time. Clearly, the runtime is determined by the time bound of computing the dominance matrix D_δ (which is at least quadratic), that is, $O(DP(n, d))$. □

The proof of Theorem 5.3.1 shows that solving the L_∞ Closest Pair Decision is not harder than computing the dominance matrix for n points in $\mathbb{R}^d$.

5.4 Solving the Optimization Problem

The algorithm from Theorem 5.3.1 solves the L_∞ Closest Pair Decision problem. It actually gives a stronger result, as it finds *all* pairs of points (p_i, p_j) such that $\|p_i - p_j\|_\infty \leqslant \delta$. We use this algorithm in order to solve the optimization problem L_∞ Closest Pair.

As a "quick and dirty" solution, one can solve the optimization problem by using the algorithm from Theorem 5.3.1 to guide a binary search over the diameter $\mathcal{D}$ of the input point set, which is at most twice the largest absolute value of the coordinates of the input points. If the coordinates are integers then we need to invoke the algorithm from Theorem 5.3.1 $O(\log \mathcal{D})$ times. If the coordinates are reals, we invoke it $O(B)$ times for B bits of precision. However, the dependence on $\mathcal{D}$ makes this method weakly polynomial, and, for real coordinates, only yields an approximation. As we show next, this naive approach can be replaced by strongly-polynomial algorithms, a deterministic one that runs in $O(DP(n,d)\log n)$ time, and a randomized one that runs in $O(DP(n,d))$ expected time.

5.4.1 Strongly-Polynomial Subcubic Algorithms

Theorem 5.4.1. *Given a set S of n points $p_1, \ldots, p_n$ in $\mathbb{R}^d$, the L_∞ Closest Pair problem can be solved for S in $O(DP(n,d)\log n)$ time.*

Proof. Since the distance between the closest pair of points, say p_i, p_j, is

$$\delta_0 = \|p_i - p_j\|_\infty = \max_{k\in[d]} \left|p_i[k] - p_j[k]\right|,$$

it is one of the $O(n^2 d)$ values $p_\ell[k] - p_m[k]$, $\ell, m \in [n]$, $k \in [d]$. Our goal is to somehow search through these values, using the decision procedure (i.e., the algorithm from Theorem 5.3.1). However, enumerating all these values takes $\Omega(n^2 d)$ time, which is too expensive, and pointless anyway, since by having them, the closest pair can be found immediately. Instead, we proceed in the following more efficient manner.

For each $k \in [d]$, we sort the points of S in increasing order of their k-th coordinate. This takes $O(dn \log n)$ time in total. Let $\left(p_1^{(k)}, \ldots, p_n^{(k)}\right)$ denote the sequence of the points of S sorted in increasing order of their k-th coordinate.[1] For each k, let $M^{(k)}$ be an $n \times n$ matrix, so that for $i, j \in [n]$, we have

$$M^{(k)}[i,j] = p_i^{(k)}[k] - p_j^{(k)}[k].$$

We view the row indices from bottom to top, i.e., the first row is the bottommost one, and the column indices from left to right. We are in fact interested only in the upper triangular portion of $M^{(k)}$, where its elements are positive, but for simplicity of presentation, we ignore this issue.

[1] Note the slight abuse of notation, as the order of the indices in the sorted sequence $\left(p_1^{(k)}, \ldots, p_n^{(k)}\right)$ depends on k.

Observe that each row of $M^{(k)}$ is sorted in decreasing order and each column is sorted in increasing order. Under these conditions, the selection algorithm of Frederickson and Johnson [91] can find the t-th-smallest element of $M^{(k)}$, for any $1 \leqslant t \leqslant n^2$, in $O(n)$ time.[2] (Simpler algorithms for this selection problem that achieve the same runtime were given later by Mirzaian and Arjomandi [131], and very recently also by Kaplan, Kozma, Zamir, and Zwick [115], using a more elegant and efficient technique.)

Note that we do not need to explicitly construct the matrices $M^{(k)}$, this will be too expensive. The bound of Frederickson-Johnson's algorithm holds as long as each entry of $M^{(k)}$ is accessible in $O(1)$ time, like in our case.

We use this method to conduct a simultaneous binary search over all d matrices $M^{(k)}$ to find δ_0. At each step of the search we maintain two counters $L_k \leqslant H_k$, for each k. Initially $L_k = 1$ and $H_k = n^2$. The invariant that we maintain is that, at each step, δ_0 lies in between the L_k-th and the H_k-th smallest elements of $M^{(k)}$, for each k.

Each binary search step is performed as follows. We compute $r_k = \lfloor (L_k + H_k)/2 \rfloor$, for each k, and apply the Frederickson-Johnson algorithm to retrieve the r_k-th smallest element of $M^{(k)}$, which we denote as δ_k, in total time $O(nd)$. We give δ_k the weight $H_k - L_k + 1$, and compute the weighted median δ_{med} of $\{\delta_1, \ldots, \delta_d\}$. We run the L_∞ Closest Pair Decision procedure of Theorem 5.3.1 on δ_{med}. Suppose that it determines that $\delta_0 \leqslant \delta_{\text{med}}$. Then for each k for which $\delta_k \geqslant \delta_{\text{med}}$ we know that $\delta_0 \leqslant \delta_k$, so we set $H_k := r_k$ and leave L_k unchanged. Symmetric actions are taken if $\delta_0 > \delta_{\text{med}}$. In either case, we remove roughly one quarter of the candidate differences; that is, the sum $\sum_{k \in [d]} (H_k - L_k + 1)$ decreases by roughly a factor of 3/4. Hence, after $O(\log n)$ steps, the sum becomes $O(d)$, and a straightforward binary search through the remaining values finds δ_0. The overall running time is

$$O(dn \log n + DP(n, d)(\log n + \log d)).$$

Since in our setting d is polynomial in n, and $nd \ll DP(n, d)$, we obtain that the overall runtime is $O(DP(n, d) \log n)$. This completes the proof of Theorem 5.1.1. □

Randomized Algorithm. Using randomization, we can improve the time bound of the preceding deterministic algorithm to equal the time bound of computing the dominance product $O(DP(n, d))$ in expectation, by using a randomized optimization technique of Chan [52]. Among the problems for which this technique can be applied, Chan specifically addresses the Closest Pair problem.

Theorem 5.4.2 (Chan [52])**.** *Let U be a collection of objects. If the* Closest Pair Decision *problem can be solved in $O(T(n))$ time, for an arbitrary distance function $d : U \times U \to \mathbb{R}$, then the*

[2] Simpler algorithms can select the t-th-smallest element in such cases in $O(n \log n)$ time, which is also sufficient for our approach.

Closest Pair *problem can be solved in* $O(T(n))$ *expected time, assuming that* $T(n)/n$ *is monotone increasing.*

We refer the reader to [52], for the proof of Theorem 5.4.2. By Theorem 5.3.1, L_∞ Closest Pair Decision can be solved in $O(DP(n,d))$ time. Clearly, $DP(n,d)/n$ is monotone increasing in n. Hence, by Theorem 5.4.2, we obtain a randomized algorithm for L_∞ Closest Pair that runs in $O(DP(n,d))$ expected time, as stated in Theorem 5.1.2.

5.5 A Faster Algorithm for L_∞ Closest Pair with Bounded Integer Coordinates

A considerable part of the algorithm from the previous section is the reduction to computing a suitable dominance matrix. The algorithms for computing dominance matrices given in Section 5.2 do not make any assumptions on the coordinates of the points, and support real numbers. When the coordinates are bounded integers, we can improve the algorithms. In particular, for n points in $\mathbb{R}^n$ with small integer coordinates we can solve the *optimization* problem in $O(n^\omega)$ time, which is a significant improvement compared to the $O(n^{2.6598})$ time bound of our previous algorithm for this case. As for integer coordinates that are bounded by a constant, the L_∞-diameter of the points is also a constant (bounded by twice the largest coordinate), it follows that one can use the decision procedure to (naively) guide a binary search over the diameter in constant time. Our improvement is based on techniques for computing $(\min,+)$-matrix multiplication over integer-valued matrices.

Theorem 5.5.1. *Let* S *be a set of* n *points* $p_1,\ldots,p_n$ *in* $\mathbb{R}^d$ *such that* $d = n^r$ *for some* $r > 0$*, and for all* $i \in [n]$*,* $k \in [d]$*,* $p_i[k]$ *is an integer in* $[-M, M]$*. Then the* L_∞ *closest pair can be computed in*

$$\widetilde{O}\left(\min\left\{Mn^{\omega(1,r,1)},\, DP(n,d)\right\}\right) \text{ time.}$$

We first define $(\max,+)$-product and $(\min,+)$-product over matrices.

Definition 5.5.2 (Distance products of matrices)**.** Let A be an $n \times m$ matrix and B be an $m \times n$ matrix. The $(\max,+)$-product of A and B, denoted by $A \star B$, is the $n \times n$ matrix C whose elements are given by

$$c_{ij} = \max_{1 \leqslant k \leqslant m} \{a_{ik} + b_{kj}\}, \quad \text{for } i,j \in [n].$$

Similarly, the $(\min,+)$-product of A and B denoted by $A * B$ is the $n \times n$ matrix C' whose elements are given by

$$c'_{ij} = \min_{1 \leqslant k \leqslant m} \{a_{ik} + b_{kj}\}, \quad \text{for } i,j \in [n].$$

We refer to either of the $(\min,+)$-product or the $(\max,+)$-product as a *distance product.*

The distance product of an $n \times m$ matrix by an $m \times n$ matrix can be computed naively in $O(n^2 m)$ time. When $m = n$, the problem is equivalent to the APSP (all pairs shortest paths) problem in a directed graph with real edge weights, and the fastest algorithm known is a recent one by Chan and Williams [57] that runs in $O\left(n^3/2^{\sqrt{\Omega(\log n)}}\right)$ time. It is a prominent long-standing open problem whether a truly subcubic algorithm for this problem exists. However, when the entries of the matrices are integers, we can convert distance products of matrices into standard algebraic products. We use a technique by Zwick [159].

Lemma 5.5.3 (Zwick [159])**.** *Given an $n \times m$ matrix $A = \{a_{ij}\}$ and an $m \times n$ matrix $B = \{b_{ij}\}$ such that $m = n^r$ for some $r > 0$, and all the elements of both matrices are integers from $[-M, M]$, their $(\min, +)$-product $C = A * B$ can be computed in $\widetilde{O}(Mn^{\omega(1,r,1)})$ time.*

With minor appropriate modifications, the $(\max, +)$-product of matrices A and B can be computed within the same time as in Lemma 5.5.3.

We now give an algorithm for computing all-pairs L_∞ distances, by using the fast algorithm for computing $(\max, +)$-product over bounded integers.

Lemma 5.5.4. *Let S be a set of n points $p_1, \ldots, p_n$ in $\mathbb{R}^d$, such that $d = n^r$ for some $r > 0$, and for all $i \in [n]$, $k \in [d]$, the coordinate $p_i[k]$ is an integer from the interval $[-M, M]$. Then the L_∞-distances between all pairs of points (p_i, p_j) from S can be computed in $\widetilde{O}(Mn^{\omega(1,r,1)})$ time.*

Proof. We create the $n \times d$ matrix $A = \{a_{ik}\}$ and the $d \times n$ matrix $B = (-A)^T = \{b_{ki}\}$, where

$$a_{ik} = p_i[k], \quad \text{for } i \in [n],\ k \in [d],$$
$$b_{ki} = -p_i[k], \quad \text{for } i \in [n],\ k \in [d].$$

Now we compute the $(\max, +)$-product $C = A \star B$. The matrix L of all-pairs L_∞-distances is then easily seen to be

$$L[i, j] = \max\{C[i, j], C[j, i]\} = \|p_i - p_j\|_\infty ,$$

for every pair $i, j \in [n]$.

Clearly, the runtime is determined by computing the $(\max, +)$-product $C = A \star B$. This is done as explained earlier, and achieves the asserted running time. □

Consequently, by taking the minimum time bound from the algorithm above, and the (deterministic) algorithm from Section 5.3, we obtain that for points in $\mathbb{R}^d$ with integer coordinates from $[-M, M]$, where $d = n^r$ for some $r > 0$, we can solve the L_∞ Closest Pair in the time stated in Theorem 5.1.3,

$$\widetilde{O}\left(\min\left\{Mn^{\omega(1,r,1)},\ DP(n, d)\right\}\right).$$

Chapter 6

Diameter Spanners

6.1 Background

In the area of graph sparsification, the notion of a *spanner* (also known as *distance spanner*) refers to a subgraph that approximately preserves all the pairwise distances between the vertices of the original graph. Formally, given an undirected (possibly weighted) graph $G = (V, E)$, the subgraph $H = (V, E_H \subseteq E)$ is a t-spanner of G iff for every pair of vertices $u, v \in V$, $d_H(u, v) \leqslant t \cdot d_G(u, v)$, where $d_H(u, v)$ and $d_G(u, v)$ are the distances between u and v in H and G, respectively ('distance' means the length of the shortest path). The parameter t is referred to as the *stretch factor* of H. Given an undirected graph G and a stretch factor t, a "good t-spanner" of G refers to a t-spanner that has a significantly smaller (by a polynomial factor) set of edges than G has (i.e., significantly sparser than G).

Spanners were first introduced and studied in the 80s [22,133,134]. Althöfer *et al.* [15] showed that any undirected weighted graph with n vertices has a $(2k-1)$-spanner of with $O(n^{1+1/k})$ edges, for any integer $k > 0$. Assuming a widely-believed girth conjecture of Erdős [84], this stretch-size trade-off is essentially optimal.

Besides being theoretically interesting, "good spanners" are known to have numerous applications in different areas of computer science, such as distributed systems, communication networks and efficient routing schemes [16,69,70,96,97,135,141,147], motion planning [68,73], approximating shortest paths [66,67,79], and distance oracles [30,148].

For directed graphs, the notion of spanners is far less understood. This is because we cannot have sparse spanners for general directed graphs. Even when the underlying graph is strongly-connected, there exists graphs with $\Omega(n^2)$ edges such that excluding even a single edge from the graph results in a spanner with a stretch as high as the diameter of G, i.e., as $\max_{u,v \in V} d_G(u, v)$ (if the graph is not fully connected then the diameter is ∞). In such a scenario, for directed graphs, a natural direction to study is the construction of sparse subgraphs that approximately preserves the graph diameter. This property is captured by the notion of a *t-diameter spanner*.

Diameter Spanner: Given a directed graph $G = (V, E)$ and a stretch factor $t > 0$, a subgraph $H = (V, E_H \subseteq E)$ is defined to be a t-diameter spanner iff $\text{diam}(H) \leqslant \lceil t \cdot \text{diam}(G) \rceil$, where $\text{diam}(H)$ and $\text{diam}(G)$ denote the diameter of H and G, respectively.

For $t = 2$ it is easy to construct such H with $O(n)$ edges, for the unweighted case, by computing the union of two BFS trees from some vertex v of the graph, one BFS tree is computed by taking only the outgoing edges from the nodes it reaches, and the other BFS tree is computed by taking only the ingoing edges. If G has edge-weights from the interval $[1, W]$, we can replace every BFS computation with Dijkstra's algorithm to compute 'forward' and 'backward' shortest-path trees from v. It is an easy exercise to show that this construction results in a 2-diameter spanner. This brings us to the following central question.

Question. Given a directed graph $G = (V, E)$, and a stretch factor $t < 2$, can we construct a

t-diameter spanner $H = (V, E_H \subset E)$? If so, how small can we make $|E_H|$? and what is the trade-off between t and $|E_H|$?

In this chapter we tackle the question above, by showing several constructions of t-diameter spanners for various $t < 2$. We believe that extremal-distance spanners are interesting mathematical objects in their own right. Nevertheless, such a sparsification of graphs suffices for many of the original applications of the well-studied standard graph spanners, such as in communication networks, facility location problem, routing, etc. In particular, diameter spanners with a sparse set of edges are good candidates for backbone networks [96].

6.2 Our Results and Related Works

The girth conjecture of Erdős [84] implies that there are undirected graphs on n vertices, for which any $(2k-1)$-spanner will require $\Omega(n^{1+1/k})$ edges. This conjecture has been proved for $k = 1, 2, 3, 5$ [154], and is widely believed to be true for any integer k. Thus, assuming the girth conjecture, one cannot expect better size-stretch trade-offs.

Althöfer *et al.* [15] were the first to show that any undirected weighted graph with n vertices has a $(2k-1)$-spanner of size $O(n^{1+1/k})$. The lower bound mentioned above implies that the $O(n^{1+1/k})$ size-bound of this spanner is essentially optimal. Althöfer *et al.* gave an algorithm to compute such a spanner, and subsequently, a long line of works have studied the question of how fast can we compute such a spanner, until Baswana and Sen [29] gave a linear-time algorithm.

A c-additive spanner of an undirected unweighted graph G is a subgraph H that preserves distances up to an additive constant c. That is, for any pair of nodes u, v in G it holds that $d_H(v, u) \leqslant d_G(v, u) + c$. This type of spanners were also extensively studied [14, 28, 62, 82]. For example, Baswana, Kavitha, Mehlhorn, and Pettie [28] showed how to construct a 6-additive spanner of size $O(n^{4/3})$. It was only recently that Abboud and Bodwin [1] proved that the 4/3 constant in the exponent of the $O(n^{4/3})$-size bound is tight, for any additive constant c.

Since for directed graphs distance spanners are impossible, the *roundtrip distance* metric was proposed. The roundtrip-distance between two vertices u and v is the distance from v to u plus the distance from u to v. Roditty, Thorup, and Zwick [141] presented the notion of *roundtrip spanners* for directed graphs. A roundtrip spanner of a directed graph G is a sparse subgraph H that approximately preserves the roundtrip distance between every pair of nodes v and u.

The question of finding the sparsest spanner of a given graph was shown to be NP-Hard by Peleg and Schäffer [133], in the same work that the graph spanner notion was introduced.

Diameter spanners were mentioned by Elkin and Peleg [80, 81], but in the context of approximation algorithms for finding the sparsest diameter spanner (a problem known to be NP-Hard). To the best of our knowledge, our work is the first to focus on the *existence* of various sparse (i.e., with $\ll n^2$ edges) diameter spanners, for directed graphs.

Our Results

Theorem 6.2.1. *Given an unweighted directed graph $G = (V, E)$ with n vertices, there exists a $(3/2)$-diameter spanner $H = (V, E_H \subseteq E)$ with at most $O(n^{3/2}\sqrt{\log n})$ edges. If G is edge-weighted from the interval $[1, W]$, then H satisfies $\operatorname{diam}(H) \leqslant \lceil (3/2)\operatorname{diam}(G) \rceil + W$. Such a subgraph H can be computed in expected $\widetilde{O}(m\sqrt{n})$ time with high probability.*

In Section 6.4.1 we construct an undirected unweighted graph with $\Theta(n^2)$ edges, such that even removing a single edge will increase the diameter by the factor $3/2$. Thus, in general, the $(3/2)$-factor cannot be improved (even for undirected graphs). However, this example uses a graph with diameter 2. Nevertheless, in [65] we show that the size-stretch trade-off from Theorem 6.2.1 is tight (up to polylogarithmic factors) even for directed unweighted graphs with diameter that is polynomial in n.

For directed graphs with diameter $o(\sqrt{n/\log n})$, we give a construction of a $(5/3)$-diameter spanner of size smaller than that in Theorem 6.2.1, as given in the following theorem.

Theorem 6.2.2. *Given an unweighted directed graph $G = (V, E)$ with n vertices and diameter D, There exists a $(5/3)$-diameter spanner $H = (V, E_H \subseteq E)$ with at most $O\left(D^{1/3}n^{4/3}\log^{2/3} n\right)$ edges. If G is edge-weighted from the interval $[1, W]$, then H satisfies $\operatorname{diam}(H) \leqslant \lceil (5/3)D \rceil + W$. Such a subgraph H can be computed in expected $\widetilde{O}(mn^{1/3}(D^{1/3} + (n/D)^{1/3}))$ time with high probability.* [1]

In [65] we show an $\Omega\left(n^{4/3}D^{1/3}\right)$ lower bound for the number of edges of a diameter spanner $H = (V, E_H \subseteq E)$ such that $\operatorname{diam}(H) < \lfloor 5/3 \rfloor D - 1$, for directed unweighted graphs, even with diameter that is polynomial in n.

Additionally, we give a generalized diameter spanner construction, which can be used to obtain either a diameter spanner with arbitrarily low stretch or a diameter spanner with arbitrarily small size, as described in the following theorem.

Theorem 6.2.3. *Given an unweighted directed graph $G = (V, E)$ with n vertices, for any $\delta, \varepsilon \in [0, 1]$, we can compute a subgraph $H = (V, E_H \subseteq E)$ satisfying one of the following. Either*

1. *H is a $(1 + \delta)$-diameter spanner of size $O(n^{2-\varepsilon}\log^{1-\varepsilon} n)$, or*
2. *H is a $(2 - \delta)$-diameter spanner of size $O(n^{1+\varepsilon}\log^{\varepsilon} n)$.*

If G is edge-weighted, with weights taken from the interval $[1, W]$, then the stretch of H will increase by an additive W term, in addition to the multiplicative stretch factor t, i.e., $\operatorname{diam}(H) \leqslant \lceil (1 + \delta)\operatorname{diam}(G) \rceil + W$ or $\operatorname{diam}(H) \leqslant \lceil (2 - \delta)\operatorname{diam}(G) \rceil + W$. Such a subgraph H can be computed in expected $\widetilde{O}\left(m(n^{\varepsilon} + n^{1+\varepsilon})\right)$ time with high probability.

Note that in Theorem 6.2.3 we require only one of the properties (1) or (2), we do not require both properties. As will be clear from the proof of Theorem 6.2.3, prior of having the input graph

[1] Although D appears in the bound on the computation time of H, we do not assume apriori knowledge of D.

G, we have no knowledge of, and no control over which of the two above properties will be the one that is satisfied by H. A particular interesting corollary of Theorem 6.2.3 arises if we set $\delta = \varepsilon = 1/3$. Then we can compute either a (4/3)-diameter spanner with $\widetilde{O}(n^{5/3})$ edges, or a (5/3)-diameter spanner with $\widetilde{O}(n^{4/3})$ edges.

In [65] we also study other types of extremal-distance spanners, such as *eccentricity spanners* and *radius spanners*. Given a graph $G = (V, E)$, the eccentricity of a vertex $v \in V$ is the maximum distance from v to any other vertex in the graph; formally, the eccentricity of v is $\text{ecc}(v) = \max_{u \in V} d_G(v, u)$. The radius of G is the minimum eccentricity of a vertex in the graph; formally, the radius of G is $\min_{v \in V} \text{ecc}(v)$. (Note that the diameter of G is $\max_{v \in V} \text{ecc}(v)$.) An eccentricity spanner is a subgraph of G that approximately preserves all the eccentricities in G. Similarly, a radius spanner is a subgraph of G that approximately preserves the radius of G. Additionally, we show in [65] how to maintain extremal-distance spanners in dynamic settings. We do not include these results in this thesis and refer the reader to [65] for further details.

6.3 Preliminaries and Techniques

Given a directed graph $G = (V, E)$, let $u, v \in V$ and $S \subseteq V$. We use the following notations.

- $d_G(u, v)$: the *length* of the shortest path from vertex u to vertex v in graph G. We sometimes denote it by $d(u, v)$, when the context is clear.
- $\pi_G(u, v)$: the shortest path from vertex u to vertex v in graph G. (We assume that the vertices are indexed from 1 to n. We break ties by always preferring the vertex with the smaller index as the next vertex in the path, starting from u and ending in v.)
- $\text{diam}(G)$: the diameter of graph G, that is, $\max_{p,q \in V} d_G(p, q)$.
- OUT-BFS(u): an outgoing breadth-first-search (BFS) tree rooted at vertex u, computed by taking only outgoing edges.
- IN-BFS(u): an incoming breadth-first-search (BFS) tree rooted at u, computed by taking only incoming edges (can be computed by applying OUT-BFS(u) on G with the edges reversed).
- OUT-BFS(u, d): the tree obtained from OUT-BFS(u) by truncating it at depth d (i.e., containing only the vertices at the first d levels).
- IN-BFS(u, d): the tree obtained from IN-BFS(u) by truncating it at depth d.
- OUT-BFS(S) (resp., IN-BFS(S)): the tree obtained from OUT-BFS(S) (resp., IN-BFS(S)), when the set $S \subseteq V$ is a super-node, i.e., the vertices of S are treated as one node, thus they are all at the first level of OUT-BFS(S) (resp., IN-BFS(S)) (this can be computed by adding a dummy vertex r and adding edges from (resp., to) r to (resp., from) all vertices of S,

compute OUT-BFS(r) (resp., IN-BFS(S)), and delete the vertex r (and its edges) from the resulting tree).

- $d_G(S, v)$ (rep., $d_G(v, S)$): the length of the shortest path from (resp., to) the set S to (resp., from) vertex v in G, when the set $S \subseteq V$ is a super-node, i.e., the vertices of S are treated as one node.
- $N_s^{out}(u)$: the s closest vertices of u in OUT-BFS(u), where ties are broken arbitrarily.
- $N_s^{in}(u)$: the s closest vertices of u in IN-BFS(u), where ties are broken arbitrarily.
- DEPTH(T): the depth of tree T.
- $P(V)$: the power-set of V.

In our results, we use an extension of the techniques of Aingworth *et al.* [14] and of Roditty and Williams [142]. Both, in their diameter approximation algorithms, first find a *hitting-set* $S \subseteq V$ of size $O((n \log n)/s)$ that *hits* $N_s^{out}(u)$ and $N_s^{in}(u)$ (i.e., $S \cap N_s^{out}(u) \neq \varnothing$ and $S \cap N_s^{in}(u) \neq \varnothing$), for every $u \in V$. We can find such a hitting set deterministically in $O(sn)$ time, using a greedy approach (for example, using the algorithm in [14]). There is also an easy *Monte Carlo* algorithm that runs in $O(n)$ time (independent of s) that finds such a hitting-set with high probability (at least $1 - \frac{1}{n^c}$, for some constant $c > 0$). This algorithm just samples a subset $S \subseteq V$ of size $O((n \log n)/s)$ uniformly at random, see Lemma 6.3.1 below. The advantage of the Monte Carlo algorithm is that we do not have to know or compute the sets $N_s^{out}(u)$ and $N_s^{in}(u)$ over $u \in V$ in advance. This was a crucial idea of Roditty and Williams [142] in improving the runtime (albeit randomized) of the diameter approximation algorithm of Aingworth *et al.* [14].

Inspired by recent works of Cairo, Grossi and Rizzi [49] and Backurs *et al.* [24], we use an extension of the technique of Roditty and Williams [142]. Instead of finding a hitting-set we find a *dominating set-pair*, defined below.

First, we need the following folklore hitting-set lemma.

Lemma 6.3.1. *Let $S_1, \ldots, S_n \subseteq V = \{1, \ldots, n\}$, such that $|S_i| \geqslant L$, for each $i \in [n]$. Let $c > 0$ be a constant, and put $r = (n(c+1)/L) \ln n$. Let $S \subseteq V$ be a random subset of size r (that is, sample r elements without replacement). Then S is a hitting-set for $S_1, \ldots, S_n$ with probability at least $1 - n^{-c}$.*

Proof. The probability for S to miss a particular set S_i, for some $i \in [n]$, is at most

$$\prod_{j=1}^{r} \frac{n - L - (j-1)}{n - (j-1)} \leqslant (1 - L/n)^r \leqslant n^{-1-c},$$

where the j-th factor in the product is the probability that the j-th element added to S misses S_i. From the union bound we have that the probability for S to miss at least one of the sets $S_1, \ldots, S_n$

is at most n^{-c}. □

The following lemma is an immediate corollary of Lemma 6.3.1.

Lemma 6.3.2. *Let $G = (V, E)$ be an n-vertex directed graph. Let n_1, n_2 be integers satisfying $n_1 n_2 = \gamma n \log n$, for some constant $\gamma > 1$. Let $S \subseteq V$ be a random subset of size n_1. Then, with high probability (that increases with γ), S has non-empty intersections with $N^{in}_{n_2}(v)$ and with $N^{out}_{n_2}(v)$, for each $v \in V$.*

We introduce the notion of $\langle h_1, h_2 \rangle$-dominating set-pair, which is a generalization of the standard definition of h-dominating set [106, 107].

Definition 6.3.3 (Dominating set-pair)**.** For a directed graph $G = (V, E)$, and a set-pair $(S_1, S_2) \in P(V) \times P(V)$, we say that (S_1, S_2) is $\langle h_1, h_2 \rangle$-dominating of size-bound $\langle n_1, n_2 \rangle$, if $|S_1| = O(n_1)$, $|S_2| = O(n_2)$, and one of the following conditions holds. Either

1. For each $x \in V$, $d_G(S_1, x) \leqslant h_1$, or
2. For each $x \in V$, $d_G(x, S_2) \leqslant h_2$.

S_1 is said to be h_1-*out-dominating* if it satisfies condition 1, and S_2 is said to be h_2-*in-dominating* if it satisfies condition 2.

We show that a dominating set-pair can be efficiently computed in directed graphs.

Lemma 6.3.4. *Let $G = (V, E)$ be a directed unweighted graph G, such that $|V| = n$ and $|E| = m$. Let $\delta \in [0, 1]$, and n_1, n_2 be integers satisfying $n_1 n_2 = \gamma n \log n$, for some constant $\gamma > 1$. We can compute, in time $O(m)$ with high probability, a $\langle \lfloor \delta D \rfloor, \lceil (1-\delta) D \rceil \rangle$-dominating set-pair $(S_1, S_2) \in P(V) \times P(V)$ such that $|S_1| \leqslant n_1$, $S_2 \leqslant n_2$.*

Proof. Let $S_1 \subseteq V$ be a uniformly random subset of V of size n_1. Let $w \in V$ be a vertex of the maximum depth in OUT-BFS(S_1) (ties are broken arbitrarily), i.e., w is the furthest vertex from S_1. Set $S_2 := N^{in}_{n_2}(w)$, which is computable in $O(m)$ time. By Lemma 6.3.2, with high probability, the set $N^{in}_{n_2}(w)$ contains a vertex of S_1. If not, then we re-sample S_1 and compute w and S_2 again. The number of times we do re-sampling is $O(1)$ with high probability, thus the runtime of computing (S_1, S_2) is $O(m)$ with high probability.

Now, if the depth of OUT-BFS(S_1) is bounded by δD, then S_1 is $\lfloor \delta D \rfloor$-out-dominating, since for each $x \in V$, $d_G(S_1, x) \leqslant \lfloor \delta D \rfloor$. Suppose S_1 is not $\lfloor \delta D \rfloor$-out-dominating, then in particular $d_G(S_1, w) > \delta D$ (since w is the furthest vertex from S_1), and thus IN-BFS$(w, \delta D)$ must have empty intersection with S_1. This is possible only when the vertices of IN-BFS$(w, \delta D)$ are fully contained in $N^{in}_{n_2}(w)$, since otherwise it must be that $N^{in}_{n_2}(w)$ is fully contained in the set of vertices of IN-BFS$(w, \delta D)$, but this contradicts the fact that $N^{in}_{n_2}(w) = S_2$ intersects with S_1. Thus, for each $x \in V$, $d_G(x, S_2)$ is bounded by DEPTH(IN-BFS(S_2)) $\leqslant \lceil$DEPTH(IN-BFS$(w)) - \delta D \rceil \leqslant \lceil D - \delta D \rceil = \lceil (1-\delta) D \rceil$. □

6.4 Construction of Diameter Spanners

Let $G = (V, E)$ be a directed graph with n vertices, m edges, and diameter D. From now on, we assume that the graph G is strongly connected (and thus $m \geqslant n$), as otherwise its diameter is ∞, for which finding a t-diameter-spanner is not interesting.

For simplicity, in the following subsections we assume that G is unweighted, but our constructions support positive (at least 1) edge-weights. If G has edge-weights from the interval $[1, W]$, we replace every application of BFS in the proof of Lemma 6.3.4 and in the procedures below with Dijkstra's algorithm to compute a shortest-paths tree. In this case, the stretch of the spanner will only increase by an additive W term (due to the rounding function used in the algorithms), and the running time will increase by a $\log n$ factor. The proofs are analogous to the proofs of the unweighted case.

In the following subsections, we provide several constructions of diameter spanners for G with various size-stretch trade-offs.

6.4.1 $(3/2)$-Diameter Spanner

A construction of a $(3/2)$-diameter-spanner can be quite easily obtained using a similar technique to the one used in the $(3/2)$-approximation algorithm for graph diameter by Roditty and Williams [142]. As a warm-up for our next diameter spanner constructions, we give here another construction for a $(3/2)$-diameter spanner, using the dominating set-pair definition (given above) and Lemma 6.3.4.

Let $(S_1, S_2) \in P(V) \times P(V)$ be a $\langle \lfloor \frac{1}{2}D \rfloor, \lceil \frac{1}{2}D \rceil \rangle$-dominating set-pair obtained by Lemma 6.3.4 by setting $\delta = 1/2$ and $n_1 = n_2 = \sqrt{cn \log n}$, for some constant $c > 0$.

We set H to be the union of the trees IN-BFS(s) and OUT-BFS(s), over all $s \in S_1 \cup S_2$. Formally,

$$H := \bigcup_{s \in S_1 \cup S_2} (\text{IN-BFS}(s) \cup \text{OUT-BFS}(s)).$$

We claim the H is a $(3/2)$-diameter spanner. To see this, consider two distinct vertices $x, y \in V$. If S_1 is $\lfloor D/2 \rfloor$-out-dominating, then there exists $s \in S_1$ such that $d_G(s, y) \leqslant D/2$. Since $d_H(x, s) = d_G(x, s) \leqslant D$, and $d_H(s, y) = d_G(s, y) \leqslant D/2$, we have $d_H(x, y) \leqslant 3D/2$. Similarly, if S_2 is $\lceil D/2 \rceil$-in-dominating, then we have $d_H(x, y) \leqslant \lceil 3D/2 \rceil$. Thus, the diameter of H is at most $\lceil 3D/2 \rceil$.

H is constructed by computing $|S_1 \cup S_2| = O(\sqrt{n \log n})$ BFS trees, each of size $O(n)$. Thus, H contains $O(n^{3/2}\sqrt{\log n})$ edges. The runtime for computing H is derived from $|S_1 \cup S_2|$ BFS computations, plus the runtime for finding the dominating set-pair (S_1, S_2), which by Lemma 6.3.4 is $O(m)$ with high probability. Thus in total, the runtime for computing H is $O(m|S_1 \cup S_2|) = O(m\sqrt{n \log n})$ with high probability. This completes the proof of Theorem 6.2.1. □

Algorithm 1: (5/3)-Diameter Spanner Construction

Input: $G = (V, E)$;
1. $H \leftarrow (V, \varnothing)$;
2. $(A_1, A_2) \leftarrow \langle \lfloor 2D/3 \rfloor, \lceil D/3 \rceil \rangle$-dominating-set-pair of size-bound $\langle \alpha \log n, n/\alpha \rangle$;
3. $(B_1, B_2) \leftarrow \langle \lfloor D/3 \rfloor, \lceil 2D/3 \rceil \rangle$-dominating-set-pair of size-bound $\langle n/\alpha, \alpha \log n \rangle$;
4. Add to H the edges of IN-BFS(A_2) and OUT-BFS(B_1);
5. **foreach** $s \in A_1 \cup B_2$ **do** add to H the edges of IN-BFS$(s) \cup$ OUT-BFS(s);
6. **foreach** $(u, v) \in A_2 \times B_1$ **do** add the edges of the shortest path $\pi_G(u, v)$ to H;
7. **return** H;

In general, the (3/2)-stretch factor cannot be improved (even for undirected graphs), as we can construct the following undirected graph $G = (V, E)$. Let $A = \{a_1, \ldots, a_n\}, B = \{b_1, \ldots, b_n\}, C = \{c_1, \ldots, c_n\}$ be sets of distinct vertices such that $V = A \cup B \cup C$. Let each of the sets A, B, C be an n-clique (i.e., in each set there is an edge between every pair of vertices). Connect an edge between a_i and b_i, for each $i \in [n]$. Finally, connect an edge between b_i and c_j, for each $i, j \in [n]$ (i.e., a bi-clique between B and C). Clearly, the number of edges of this graph is $\Theta(n^2)$ and its diameter is 2 (the longest path is from a vertex in A to a vertex in C). Now, if we remove an edge (b_i, c_j), for any $i, j \in [n]$, the diameter increases to 3 due to the shortest-path from a_i to c_j.

Note that the graph above has diameter 2. Nevertheless, as mentioned earlier, in [65] we show that the size-stretch trade-off from Theorem 6.2.1 is tight (up to polylogarithmic factors) even for directed unweighted graphs with diameter that is polynomial in n.

6.4.2 (5/3)-Diameter Spanner

Here we present a construction of a (5/3)-diameter spanner H that is sparser than the (3/2)-diameter spanner from Theorem 6.2.1 whenever $D = o(\sqrt{n/\log n})$. This will prove Theorem 6.2.2.

Let $\alpha > 0$ be a parameter that we will fix later. The construction of H is presented in Algorithm 1. We will now prove its correctness.

Consider two distinct vertices $x, y \in V$. If A_1 is a $\lfloor 2D/3 \rfloor$-out-dominating set, then $d_G(s, y) \leqslant \lfloor 2D/3 \rfloor$ for some $s \in A_1$. Thus

$$d_H(x, y) \leqslant d_H(x, s) + d_H(s, y) = d_G(x, s) + d_G(s, y) \leqslant D + \lfloor 2D/3 \rfloor = \lfloor 5D/3 \rfloor.$$

Similarly, if B_2 is a $\lceil 2D/3 \rceil$-in-dominating set, it can be shown that $d_H(x, y) \leqslant \lceil 5D/3 \rceil$.

Suppose that neither A_1 is $\lfloor 2D/3 \rfloor$-out-dominating nor B_2 is $\lceil 2D/3 \rceil$-in-dominating. Then, A_2 is $\lceil D/3 \rceil$-in-dominating and B_1 is $\lfloor D/3 \rfloor$-out-dominating (by definition of dominating set-pair). So $d_G(x, A_2), d_G(B_1, y) \leqslant \lceil D/3 \rceil$. Since H contains IN-BFS(A_2) and OUT-BFS(B_1), there must be $s_x \in A_2$ and $s_y \in B_1$ such that $d_H(x, s_x) = d_G(x, s_x) = d_G(x, A_2) \leqslant \lceil D/3 \rceil$ and $d_H(s_y, y) = d_G(s_y, y) = d_G(B_1, y) \leqslant \lceil D/3 \rceil$. Since H contains the shortest path between each pair of vertices

in $A_2 \times B_1$, we obtain that $d_H(s_x, s_y) = d_G(s_x, s_y) \leqslant D$. Therefore,

$$d_H(x,y) \leqslant d_H(x,s_x) + d_H(s_x,s_y) + d_H(s_y,y) = d_G(x,s_x) + d_G(s_x,s_y) + d_G(s_y,y) \leqslant \lceil 5D/3 \rceil.$$

We now analyze the size of H. We added to H the edges of the $O(\alpha \log n)$ BFS trees from Steps 5 and 4, which consist of $O(n\alpha \log n)$ edges in total. In Step 6 we added the shortest paths between all pairs in $A_2 \times B_1$, which use in total $O(n^2 D/\alpha^2)$ edges. Thus, the total number of edges in H is $O(n\alpha \log n + n^2 D/\alpha^2)$. This is minimized when $\alpha = \Theta\big((nD/\log n)^{1/3}\big)$. Therefore, the total number of edges in H is $O(D^{1/3} n^{4/3} \log^{2/3} n)$.

Observe that in order to compute α up to a suitable constant factor, it suffices to have an estimate of D. We can easily compute a 2-approximation for the diameter D in $O(m)$ time, since for any arbitrary vertex $w \in V$, $D \leqslant \textsc{depth}(\textsc{in-bfs}(w)) + \textsc{depth}(\textsc{out-bfs}(w)) \leqslant 2D$, and the depths of $\textsc{in-bfs}(w)$ and $\textsc{out-bfs}(w)$ are computable in $O(m)$ time.

We now analyze the running time of each step in Algorithm 1. By Lemma 6.3.4, the time to compute the set-pairs (A_1, A_2) and (B_1, B_2) from Steps 2 and 3 is $O(m)$ with high probability. Step 4 takes $O(m)$ time. Step 5 and 6 are done by computing $\textsc{in-bfs}(s)$ and $\textsc{out-bfs}(s)$, for each vertex $s \in A_1 \cup B_2 \cup A_2 \cup B_1$. Thus, Steps 5 and 6 takes $O\left(m \cdot |A_1 \cup B_2 \cup A_2 \cup B_1|\right)$ time. Overall, the total expected runtime of the algorithm is $O\left(m(|A_1 \cup A_2 \cup B_1 \cup B_2|)\right) = O\left(m(\alpha \log n + n/\alpha)\right) = \widetilde{O}\left(mn^{1/3}D^{1/3} + mn^{2/3}/D^{1/3}\right) = \widetilde{O}\left(mn^{1/3}(D^{1/3} + (n/D)^{1/3})\right)$. This completes the proof of Theorem 6.2.2. □

As mentioned earlier, in [65] we show an $\Omega\left(n^{4/3}D^{1/3}\right)$ lower bound for the size of a diameter spanner $H = (V, E_H \subseteq E)$ that satisfies $\mathrm{diam}(H) < \lfloor 5/3 \rfloor D - 1$, for directed unweighted graphs (even for such graphs with a diameter that is polynomial in n).

6.4.3 General (low-stretch or small-size)-Diameter Spanner

Here we generalize the result from Theorem 6.2.1. For any $\delta, \varepsilon \in [0,1]$, we can compute in $O\left(m(n^{\varepsilon} + n^{1+\varepsilon})\right)$ time with high probability, a subgraph $H = (V, E_H \subseteq E)$ satisfying *one* of the following. Either

1. H is a $(1+\delta)$-diameter spanner of size $O(n^{2-\varepsilon} \log^{1-\varepsilon} n)$, or
2. H is a $(2-\delta)$-diameter spanner of size $O(n^{1+\varepsilon} \log^{\varepsilon} n)$.

Let $\delta, \varepsilon \in [0,1]$, and use Lemma 6.3.4 to obtain a $\langle \lfloor \delta D \rfloor, \lceil (1-\delta)D \rceil \rangle$-dominating set-pair (S_1, S_2), such that $|S_1| \leqslant (n \log n)^{1-\varepsilon}$, and $|S_2| \leqslant \gamma (n \log n)^{\varepsilon}$, for some constant $\gamma > 1$. (Note that indeed $(n \log n)^{1-\varepsilon}(n \log n)^{\varepsilon} = \gamma n \log n$.)

Let H_1 (resp., H_2) be the union of the trees $\textsc{in-bfs}(s)$ and $\textsc{out-bfs}(s)$, for each $s \in S_1$ (resp., $s \in S_2$). The time for computing H_1 and H_2 is derived from $|S_1 \cup S_2|$ BFS computations, plus the time for finding the dominating set-pair (S_1, S_2), which is in total $O(m(n_1 + n_2))$ time with

high probability, where the constant of proportionality depends on γ. Note that H_1 contains $O(n^{2-\varepsilon}\log^{1-\varepsilon} n)$ edges, and H_2 contains $O(n^{1+\varepsilon}\log^{\varepsilon} n)$ edges.

Consider any two distinct vertices $x, y \in V$. If S_1 is $\lfloor \delta D \rfloor$-out-dominating, then there exists $s_1 \in S_1$ such that $d_G(s_1, y) \leqslant \lfloor \delta D \rfloor$. Since $d_{H_1}(x, s_1) = d_G(x, s_1) \leqslant D$, and $d_{H_1}(s_1, y) = d_G(s_1, y) \leqslant \lfloor \delta D \rfloor$, we have $d_{H_1}(x, y) \leqslant \lfloor (1+\delta)D \rfloor$. Similarly, if S_2 is $\lceil (1-\delta)D \rceil$-in-dominating, then there exists $s_2 \in S_2$ such that $d_G(x, s_2) \leqslant \lceil (1-\delta)D \rceil$. Since $d_{H_2}(x, s_2) = d_G(x, s_2) \leqslant \lceil (1-\delta)D \rceil$ and $d_{H_2}(s_2, y) = d_G(s_2, y) \leqslant D$, we have $d_{H_2}(x, y) \leqslant \lceil (2-\delta)D \rceil$. This completes the proof of Theorem 6.2.3. □

Chapter 7

Conclusions and Open Questions

The main results of this thesis are

1. Improved decision tree for k-SUM and improved algorithm for 3SUM. Following our work, our decision tree was significantly improved by Kane, Lovett, and Moran [114], and our 3SUM algorithm was improved by Chan [55] by an additional $\log n$ factor.

2. The first subquadratic algorithms for computing Dynamic Time Warping (DTW) and Geometric Edit Distance (GED) between two point-sequences in $\mathbb{R}$ (and also in $\mathbb{R}^d$, for any constant d, when the underlying metric is polyhedral), breaking the nearly 50 years old quadratic time barrier of these problems. These are currently the best-known algorithms for these problems.

3. Linear decision trees with near-linear depth for Discrete Fréchet Distance under polyhedral metrics in $\mathbb{R}^d$, for any constant d.

4. The first strongly-polynomial strongly subcubic algorithm for computing L_∞ Closest Pair for n points in $\mathbb{R}^n$, and showing the relation of this problem to computing dominance product.

5. Showing the existence of various sparse diameter spanners with stretch smaller than 2 for directed graphs, and giving efficient algorithms to construct them.

We conclude this thesis with several open problems, which may be interesting for future work.

7.1 Bringing the Four Russians to Geometry: Can we test general position in subquadratic time?

In light of the recent 3SUM results stated in Chapter 3, the results of Kane, Lovett and Moran [114], and of Chan [55], it is natural to ask whether similar improvements can be made for some well studied 3SUM-Hard problems. Perhaps the most famous one is the 3-Collinearity Testing problem (3-Collinearity) (also known as general position testing). That is, determining whether there are three collinear points in a set of n points in the plane. Chan [55] recently showed subquadratic algorithms for some 3SUM-Hard problems, however, not for 3-Collinearity.

In our algorithms from the results described in Chapters 3 and 4, we start by using the so-called "Method of the Four Russians" [20], described in Section 2.2. This method can be exploited to improve algorithms that involve a matrix structure. The basic idea is to decompose an $n \times n$ matrix into $(n/g)^2$ small sub-matrices (boxes), each of size $g \times g$. Then the hard challenge is to find a way to efficiently solve a corresponding sub-problem in each of these boxes, and obtain the solution for the original problem by combining the answers from the sub-problems, maintaining an overall improved runtime. A natural questions is:

Can we extend this technique for problems that do not involve a matrix structure, such as various geometric 3SUM-Hard problems in the plane?

Assuming that the input consists of n curves in the plane, our idea is to construct a *g-cutting* that decomposes $\mathbb{R}^2$ into $(n/g)^2$ disjoint cells, so that each cell is intersected by at most g of the input curves. This decomposes the original problem into $(n/g)^2$ smaller subproblems, each of size at most g. This is analogous to decomposing a matrix into small boxes in the standard Four Russians method. Then, if we can solve each subproblem in $o(g^2)$ time, we can solve all the subproblems in subquadratic time. The challenge left is to solve the original problem in subquadratic time, by using the solutions of the aforementioned subproblems.

3-Collinearity is a particularly interesting problem to tackle, as it is one of the more famous 3SUM-Hard problems, and can be solved in $O(n^2)$ time. We recall the problem:

3-Collinearity: Given a set S of n points in $\mathbb{R}^2$, decide whether S contains three collinear points.

Often this problem is stated in the following equivalent dual form.

3-Collinearity (dual): Given a set L of n lines in $\mathbb{R}^2$, decide whether L contains three concurrent lines (i.e., three lines that intersect at a common point).

Towards a subquadratic algorithm for 3-Collinearity. We look at the dual problem. Let L denote the set of lines dual to the n input points. We fix some small parameter g, and construct a *g-cutting* of the plane for L in $O(ng)$ time, using the standard techniques of Chazelle and Friedman [59] and Chazelle [58]. Specifically, we partition the plane into $O(n/g)^2$ triangles, where each triangle is intersected by at most g lines of L. Ignoring concurrencies at points on the boundaries of the triangles of the cutting (which are much easier to detect), each possible concurrency occurs inside one of these triangles, and therefore this cutting technique gives a partition of the original problem into small subproblems.

Any tuple of g lines in the plane can be represented by a point in $\mathbb{R}^{2g}$. We thus get a collection of $O(n/g)^2$ points in $\mathbb{R}^{2g}$, one for each triangle of the cutting, and our goal is to determine whether one of these points represents a g-tuple with three concurrent lines. Each such concurrency can be expressed by a quadratic polynomial equation in the coefficients of the three relevant lines. We thus get a collection $\mathcal{F}$ of $\binom{g}{3}$ quadratic surfaces in $\mathbb{R}^{2g}$, and the goal is to determine whether any of the $O(n/g)^2$ points lie on one of these surfaces. In other words, our goal is to construct a point location data structure such that a query to this data structure (by a point in $\mathbb{R}^{2g}$) will confirm/negate whether there are three lines among the g lines that intersect at a common point.

The data structure is for point location in the arrangement of the $O(g^3)$ surfaces in $\mathbb{R}^{2g}$ mentioned above, whose complexity is easily seen to be bounded by $O(2^{2g}g^{6g})$. Now we wish to query this data structure $O(n/g)^2$ times, one query for each cell of the cutting. However, in order to

obtain an overall subquadratic runtime, each query should take $o(g^2)$ time, which appears to be a rather serious bottleneck.

The best known query time for point location among *hyperplanes* in $\mathbb{R}^d$ is given in a recent algorithm of Ezra, Har-Peled, Kaplan, and Sharir [86], where a query takes $O(d^3 \log n)$ time, which improves an algorithm of Meiser [130] with $O(d^5 \log n)$ query time. If we restrict ourselves to the linear decision tree model, a recent work of Ezra and Sharir [88] gives an (unconstrained) linear decision tree with depth $O(n^2 \log^2 n)$, for point location among n hyperplanes in $\mathbb{R}^d$. In our case we have quadratic surfaces rather than hyperplanes, but even if the state-of-the-art techniques mentioned above could be adapted to this case, the query cost for $d = 2g$ and $n = \Theta(g^3)$ would be $O(g^3 \log g)$ in the uniform model, way too much for our needs. Even in the linear decision tree model it would be too expensive, as we obtain a bound of $O(g^2 \log g)$.

Nevertheless, in our case, since we can take g to be a very small quantity, we can afford to use a data structure with a huge amount of preprocessing time and storage, much more than the standard approaches. So our problem now is to construct a point location data structure for these surfaces, possibly with a very large storage and preprocessing time (even super-exponential in g), as long as the query time for a point (representing a set of g lines) is only $o(g^2)$.

7.2 Sorting $X + Y$

Recall the Sorting $X + Y$ problem discussed in Section 2.2 and in Chapter 3.

Sorting $X + Y$: Given two sets X and Y, each of n real numbers, sort

$$X + Y = \{x + y \mid x \in X,\, y \in Y\}\,.$$

A somewhat simpler variant of this problem is

Element Uniqueness in $X + Y$: Given two sets X and Y, each of n real numbers, determine whether all the elements of $X + Y$ are distinct.

Both problems are known to be 3SUM-Hard, and are also used for basing conditional lower bounds for other problems (see [27] and [108]), which are therefore classified as "(Sorting $X + Y$)-Hard". As mentioned in Section 2.2, the linear decision tree complexity of Sorting $X + Y$ (and Element Uniqueness in $X + Y$) was shown to be $O(n^2)$ by Fredman [92] in 1976, and in a recent breakthrough by Kane, Lovett, and Moran [114] this complexity was shown to be only $O(n \log^2 n)$. It is still a prominent long-standing open problem whether these problems can be solved in $o(n^2 \log n)$ time (see [72]), even for the case $X = Y$.

Our 3SUM algorithm and Sorting $X + Y$. In our 3SUM algorithm we showed that we can obtain the sorting permutations of boxes of size $\log n \times \log n$, that comprise $X + Y$, in

$O(n^2 \log\log n/\log n)$ overall time. Is it possible to pay an additional $o((\log n)^2/\log\log n)$ factor and determine whether all elements in $X + Y$ are distinct? Note that this is a *global* question: resolving element uniqueness in each box separately is not enough.

An additive combinatorics direction. Recently, Chan and Lewenstein [56] presented new algorithms for solving certain non-trivial restricted cases of 3SUM on integers in strongly subquadratic time, by using results from additive combinatorics. Specifically, they developed an algorithmic framework from a version of the Ballog-Szémeredi-Gowers (BSG) theorem [146], which states:

Theorem 7.2.1 (Ballog-Szémeredi-Gowers [146])**.** *Given sets A, B, S, each of size N, in any Abelian group, and some parameter $\alpha > 0$, such that $\left|\{(a,b) \in A \times B \mid a + b \in S\}\right| \geqslant \alpha N^2$, there exist subsets $A' \subseteq A$ and $B' \subseteq B$, satisfying $|A'|, |B'| = \Omega(\alpha N)$, such that $|A' + B'| = O((1/\alpha)^5 N)$.*

It may be interesting to investigate whether additive combinatorics techniques can be used for developing an algorithm for deciding whether the elements of $X + X$ are all distinct, for certain non-trivial cases of integer inputs. A relevant theorem that might be useful for this purpose is the following theorem by Brown and Buhler [41], and Roth [143].

Theorem 7.2.2. *For every $\varepsilon > 0$, there exists $n_0 = n_0(\varepsilon)$ with the following property. If A is an Abelian group of odd order and $|A| > n_0$, then every subset $B \subset A$ with $|B| > \varepsilon|A|$ contains a three-term arithmetic progression, i.e., distinct elements x, y, z such that $x + y = 2z$.*

Very recently, Bloom [35] established the following bound.

Theorem 7.2.3 (Bloom [35])**.** *If $A \subset \{1, \ldots, N\}$ contains no non-trivial*[1] *three-term arithmetic progressions then*

$$|A| \ll \frac{(\log\log N)^4}{\log N} N.$$

Connection to the problem. Observe that if A contains a three-term arithmetic progression (x, y, z) then not all elements of $A + A$ are distinct, since $x + z = y + y \in A + A$. Can one use this fact in conjunction with Theorem 7.2.3 to develop a faster algorithm for deciding whether all elements of $A + A$ are distinct, for certain non-trivial cases of integer inputs?

7.3 Additional Classical Quadratic Problems

In addition to the problems mentioned above, there are some fundamental quadratic problems for which even a polylogarithmic factor improvements are unknown. Since these problems are fundamental, it is worth trying to break their quadratic time (and sometimes also space) bounds. We discuss about two of them below.

[1] A trivial three-term arithmetic progression is one in which all three elements are the same.

Deterministic optimal binary search tree. We are given a set $X = \{a_1, \ldots, a_n\}$ of n ordered elements, and we wish to perform binary searches among them. We are given a set $\{A_1, \ldots, A_n\} \cup \{B_0, \ldots, B_n\}$ of $2n+1$ probabilities, where A_i is the probability that the search is with element a_i, for each $i \in [n]$, and B_i is the probability that the search is with an element between a_i and a_{i+1}, for each $i \in [n-1]$; B_0 is the probability of searching with an element strictly less than a_0, and B_n is the probability of searching with an element strictly greater than a_n. These $2n+1$ probabilities cover all possible searches, and therefore add up to 1.

The well known *optimal binary search tree* problem is the optimization problem of finding the binary search tree that minimizes the expected search time. Gilbert and Moore [98] showed in 1959 a dynamic programming algorithm for this problems that runs in $O(n^3)$ time. Knuth [121] showed in 1971 that their algorithm can be speeded up to run in $\Theta(n^2)$ time, and this currently the best-known time bound for this problem. Knuth's primary insight was that the optimality problem is hereditary, in the sense that if a certain tree is optimal for a given probability distribution, then its left and right subtrees must also be optimal for their (suitably scaled) appropriate subsets of the distribution.

Although this problem was not proven to be 3SUM-Hard, there is a reason to believe that this problem is harder than 3SUM, as Knuth's algorithm also uses $\Theta(n^2)$ space (no optimum binary search tree algorithm has been found that uses $o(n^2)$ space and polynomial time), whereas a standard quadratic-time algorithm for 3SUM uses only $O(n)$ space.

In view of the recent "quadratic-time breaking algorithms" discussed in this thesis, we propose two main open questions concerning the optimal binary search tree problem.

1. Can it be solved in polynomial time and $o(n^2)$ space? is it possible to break the quadratic space barrier?

2. Does Knuth's algorithm runs in optimal time? namely, is it possible to break the quadratic time barrier?

Minimum area triangle. Given a set P of n points in the plane, the *minimum-area triangle* problem is to find a triangle T of minimum are, whose vertices are in P (i.e., there is no other such a triangle whose area is smaller than the area of T). The corresponding decision version of this problem is to determine whether there exists a triangle of area not larger than a given parameter $K \geqslant 0$, whose vertices are in P .

Eppstein, Overmars, Rote, and Woeginger [83] presented a geometric algorithm that uses a dynamic programming approach and runs in $O(n^2)$ time and uses $O(n)$ space, for finding a minimum-area triangle. Our question is whether the decision version of this problem can be solved in subquadratic time. Note however, that this problem is 3SUM-Hard, moreover, it is "3-Collinearity-Hard", since deciding whether there are three collinear points in P can be done by deciding whether there

is a triangle of area 0, whose vertices are from P. Thus, a subquadratic-time solution for this problem immediately implies a subquadratic-time solution for 3-Collinearity. Hence, this question is better be tackled only if one can first solve 3-Collinearity in subquadratic time, which is the question given in Section 7.1.

Bibliography

[1] A. Abboud and G. Bodwin. The 4/3 additive spanner exponent is tight. In *Proc. 48th Annu. ACM Sympos. on Theory of Computing (STOC)*, pages 351–361, 2016.

[2] A. Abboud and K. Bringmann. Tighter connections between formula-sat and shaving logs. In *45th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 8:1–8:18, 2018.

[3] A. Abboud, T. D. Hansen, V. V. Williams, and R. Williams. Simulating branching programs with edit distance and friends: Or: A polylog shaved is a lower bound made. In *Proc. 48th Annu. ACM Sympos. on Theory of Computing (STOC)*, pages 375–388, 2016.

[4] A. Abboud and K. Lewi. Exact weight subgraphs and the k-SUM conjecture. In *Proc. 40th Int'l Colloq. on Automata, Languages and Programming (ICALP)*, pages 1–12, 2013.

[5] A. Abboud and V. V. Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *Proc. 55th Annu. Sympos. on Foundations of Computer Science (FOCS)*, pages 434–443, 2014.

[6] A. Abboud, V. V. Williams, and H. Yu. Matching triangles and basing hardness on an extremely popular conjecture. In *Proc. 47th Annu. ACM on Sympos. on Theory of Computing (STOC)*, pages 41–50, 2015.

[7] P. K. Agarwal, R. Ben Avraham, H. Kaplan, and M. Sharir. Computing the discrete Fréchet distance in subquadratic time. *SIAM J. Comput.*, 43(2):429–449, 2014.

[8] P. K. Agarwal, K. Fox, J. Pan, and R. Ying. Approximating dynamic time warping and edit distance for a pair of point sequences. In *Proc. 32nd International Sympos. on Computational Geometry (SoCG)*, pages 6:1–6:16, 2016.

[9] A. Aggarwal, M. M. Klawe, S. Moran, P. Shor, and R. Wilber. Geometric applications of a matrix-searching algorithm. *Algorithmica*, 2(1):195–208, 1987.

[10] A. V. Aho and J. E. Hopcroft. *The Design and Analysis of Computer Algorithms*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1974.

[11] O. Aichholzer, F. Aurenhammer, E. D. Demaine, F. Hurtado, P. Ramos, and J. Urrutia. On k-convex polygons. *Comput. Geom.*, 45(3):73–87, 2012.

[12] N. Ailon and B. Chazelle. Lower bounds for linear degeneracy testing. *J. ACM*, 52(2):157–171, 2005.

[13] N. Ailon and B. Chazelle. The fast Johnson-Lindenstrauss transform and approximate nearest neighbors. *SIAM J. Comput.*, 39(1):302–322, 2009.

[14] D. Aingworth, C. Chekuri, P. Indyk, and R. Motwani. Fast estimation of diameter and shortest paths (without matrix multiplication). *SIAM Journal on Computing*, 28(4):1167–1181, 1999.

[15] I. Althöfer, G. Das, D. Dobkin, D. Joseph, and J. Soares. On sparse spanners of weighted graphs. *Discrete Comput. Geom.*, 9(1), 1993.

[16] K. Alzoubi, X. Y. Li, Y. Wang, P. J. Wan, and O. Frieder. Geometric spanners for wireless ad hoc networks. *IEEE Transactions on Parallel and Distributed Systems*, 14(4):408–421, 2003.

[17] A. Amir, T. M. Chan, M. Lewenstein, and N. Lewenstein. On hardness of jumbled indexing. In *Proc. 41st Int'l Colloq. on Automata, Languages, and Programming (ICALP)*, pages 114–125, 2014.

[18] A. Amir, T. Kopelowitz, A. Levy, S. Pettie, E. Porat, and B. R. Shalom. Mind the gap: Essentially optimal algorithms for online dictionary matching with one gap. In *Proc. 27th Int'l Sympos. on Algorithms and Computation (ISAAC)*, pages 12:1–12:12, 2016.

[19] A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Commun. ACM*, 51(1):117–122, 2008.

[20] V. Arlazarov, E. Dinic, M. Kronrod, and I. Faradzev. On economical construction of the transitive closure of a directed graph. *Dokl. Akad. Nauk.*, 194(11), 1970.

[21] S. Arora and B. Barak. *Computational Complexity: A Modern Approach.* Cambridge University Press, 1st edition, 2009.

[22] B. Awerbuch. Communication-time trade-offs in network synchronization. In *Proc. 4th Annu. ACM Sympos. on Principles of Distributed Computing (PODC)*, pages 272–276. ACM, 1985.

[23] A. Backurs and P. Indyk. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). In *Proc. 47th Annu. ACM Sympos. on Theory of Computing (STOC)*, pages 51–58, 2015.

[24] A. Backurs, L. Roditty, G. Segal, V. V. Williams, and N. Wein. Towards tight approximation bounds for graph diameter and eccentricities. In *Proc. 50th Annu. ACM Sympos. on Theory of Computing (STOC)*, pages 267–280, 2018.

[25] I. Baran, E. D. Demaine, and M. Pătraşcu. Subquadratic algorithms for 3SUM. *Algorithmica*, 50(4):584–596, 2008.

[26] L. Barba, J. Cardinal, J. Iacono, S. Langerman, A. Ooms, and N. Solomon. Subquadratic Algorithms for Algebraic Generalizations of 3SUM. In *Proc. 33rd International Symposium on Computational Geometry (SoCG)*, pages 13:1–13:15, 2017.

[27] G. Barequet and S. Har-Peled. Polygon containment and translational min-Hausdorff-distance between segment sets are 3SUM-hard. *Int. J. Comput. Geometry Appl.*, 11(4):465–474, 2001.

[28] S. Baswana, T. Kavitha, K. Mehlhorn, and S. Pettie. Additive spanners and (α, β)-spanners. *ACM Trans. Algorithms*, 7(1):5:1–5:26, 2010.

[29] S. Baswana and S. Sen. A simple linear time algorithm for computing a $(2k-1)$-spanner of $O(n^{1+1/k})$ size in weighted graphs. In *Proc. 30th International Conference on Automata, Languages and Programming (ICALP)*, pages 384–396, 2003.

[30] S. Baswana and S. Sen. Approximate distance oracles for unweighted graphs in expected $O(n^2)$ time. *ACM Trans. Algorithms*, 2(4):557–577, 2006.

[31] M. Ben-Or. Lower bounds for algebraic computation trees. In *Proc. 15th Annu. ACM Sympos. on Theory of Computing (STOC)*, pages 80–86, 1983.

[32] J. L. Bentley. Multidimensional divide-and-conquer. *Commun. ACM*, 23(4):214–229, 1980.

[33] J. L. Bentley and M. I. Shamos. Divide-and-conquer in multidimensional space. In *Proc. of the 8th Annu. ACM Sympos. on Theory of Computing (STOC)*, pages 220–230, 1976.

[34] P. Bille and M. Farach-Colton. Fast and compact regular expression matching. *Theoretical Computer Science*, 409(3):486–496, 2008.

[35] T. F. Bloom. A quantitative improvement for Roth's theorem on arithmetic progressions. *Journal of the London Mathematical Society*, 2016.

[36] M. Blum, R. W. Floyd, V. R. Pratt, R. L. Rivest, and R. E. Tarjan. Time bounds for selection. *J. Comput. Syst. Sci.*, 7:448–461, 1973.

[37] D. Bremner, T. M. Chan, E. D. Demaine, J. Erickson, F. Hurtado, J. Iacono, S. Langerman, M. Pătraşcu, and P. Taslakian. Necklaces, convolutions, and $X + Y$. *Algorithmica*, 69:294–314, 2014.

[38] K. Bringmann. Why walking the dog takes time: Fréchet distance has no strongly subquadratic algorithms unless SETH fails. In *Proc. 55th IEEE Annu. Sympos. on Foundations of Computer Science (FOCS)*, pages 661–670, 2014.

[39] K. Bringmann and M. Künnemann. Quadratic conditional lower bounds for string problems and dynamic time warping. In *Proc. 56th Annu. IEEE Sympos. on Foundations of Computer Science (FOCS)*, pages 79–97, 2015.

[40] K. Bringmann and W. Mulzer. Approximability of the discrete Fréchet distance. *J. Comput. Geom.*, 7(2):46–76, 2016.

[41] T. C. Brown and J. P. Buhler. A density version of a geometric Ramsey theorem. *J. Combinatorial Theory, Series A*, 32(1):20–34, 1982.

[42] K. Buchin, M. Buchin, C. Knauer, G. Rote, and C. Wenk. How difficult is it to walk the dog? *Proc. 23rd Euro. Workshop Comput. Geom.*, pages 170–173, 2007.

[43] K. Buchin, M. Buchin, W. Meulemans, and W. Mulzer. Four soviets walk the dog - with an application to Alt's conjecture. pages 1399–1413, 2014.

[44] R. C. Buck. Partition of space. *Amer. Math. Monthly*, 50:541–544, 1943.

[45] R. E. Burkard, B. Klinz, and R. Rudolf. Perspectives of Monge properties in optimization. *Discrete Applied Mathematics*, 70(2):95–161, 1996.

[46] R. E. Burkard, B. Klinz, and R. Rudolf. Perspectives of Monge properties in optimization. *Discrete Applied Mathematics*, 70(2):95 – 161, 1996.

[47] A. Butman, P. Clifford, R. Clifford, M. Jalsenius, N. Lewenstein, B. Porat, E. Porat, and B. Sach. Pattern matching under polynomial transformation. *SIAM J. Comput.*, 42(2):611–633, 2013.

[48] E. G. Caiani, A. Porta, G. Baselli, M. Turiel, S. Muzzupappa, F. Pieruzzi, C. Crema, A. Malliani, and S. Cerutti. Warped-average template technique to track on a cycle-by-cycle basis the cardiac filling phases on left ventricular volume. In *Computers in Cardiology*, pages 73–76, 1998.

[49] M. Cairo, R. Grossi, and R. Rizzi. New bounds for approximating extremal distances in undirected graphs. In *Proc. 27th Annu. ACM-SIAM Sympos. on Discrete Algorithms (SODA)*, pages 363–376, 2016.

[50] J. Cardinal, J. Iacono, and A. Ooms. Solving k-SUM using few linear queries. In *Proc. 24th Annu. European Sympos. on Algorithms (ESA)*, pages 25:1–25:17, 2016.

[51] M. L. Carmosino, J. Gao, R. Impagliazzo, I. Mihajlin, R. Paturi, and S. Schneider. Non-deterministic extensions of the strong exponential time hypothesis and consequences for non-reducibility. In *Proc. ACM Conference on Innovations in Theoretical Computer Science (ITCS)*, pages 261–270, 2016.

[52] T. M. Chan. Geometric applications of a randomized optimization technique. *Discrete Comput. Geom.*, 22(4):547–567, 1999.

[53] T. M. Chan. All-pairs shortest paths with real weights in $O(n^3/\log n)$ time. *Algorithmica*, 50(2):236–243, 2008.

[54] T. M. Chan. Speeding up the four russians algorithm by about one more logarithmic factor. In *Proc. of the 26th Annu. ACM-SIAM Sympos. on Discrete Algorithms (SODA)*, pages 212–217, 2015.

[55] T. M. Chan. More logarithmic-factor speedups for 3SUM, (median,+)-convolution, and some geometric 3SUM-hard problems. In *Proc. 29th Annu. ACM-SIAM Sympos. on Discrete Algorithms (SODA)*, pages 881–897, 2018.

[56] T. M. Chan and M. Lewenstein. Clustered integer 3SUM via additive combinatorics. In *Proc. 47th Annu. ACM Sympos. on Theory of Computing (STOC)*, pages 31–40, 2015.

[57] T. M. Chan and R. Williams. Deterministic APSP, orthogonal vectors, and more: Quickly derandomizing Razborov-Smolensky. In *Proc. of the 27th Annu. ACM-SIAM Sympos. on Discrete Algorithms (SODA)*, pages 1246–1255, 2016.

[58] B. Chazelle. Cutting hyperplanes for divide-and-conquer. *Discrete Comput. Geom.*, 9(2):145–158, 1993.

[59] B. Chazelle and J. Friedman. A deterministic view of random sampling and its use in geometry. *Combinatorica*, 10(3):229–249, 1990.

[60] B. Chazelle and L. J. Guibas. Fractional cascading: I. A data structuring technique. *Algorithmica*, 1(2):133–162, 1986.

[61] B. Chazelle and L. J. Guibas. Fractional cascading: II. Applications. *Algorithmica*, 1(2):163–191, 1986.

[62] S. Chechik. New additive spanners. In *Proc. 24th Annu. ACM-SIAM Sympos. on Discrete Algorithms (SODA)*, pages 498–512, 2013.

[63] S. Chechik, D. Larkin, L. Roditty, G. Schoenebeck, R. E. Tarjan, and V. V. Williams. Better approximation algorithms for the graph diameter. In *Proc. 25th Annu. ACM-SIAM Sympos. on Discrete Algorithms (SODA)*, pages 1041–1052, 2014.

[64] K. Y. Chen, P. H. Hsu, and K. M. Chao. Approximate matching for run-length encoded strings is 3SUM-Hard. In *Proc. 20th Annu. Sympos. Combinatorial Pattern Matching (CPM)*, pages 168–179, 2009.

[65] K. Choudhary and O. Gold. Extremal distances in directed graphs: Tight spanners and near-optimal approximation algorithms. To appear in *Proc. of Annu. ACM-SIAM Sympos. on Discrete Algorithms (SODA)*, 2020. A preliminary version in arXiv:1812.01602.

[66] E. Cohen. Fast algorithms for constructing t-spanners and paths with stretch t. *SIAM Journal on Computing*, 28(1):210–236, 1998.

[67] E. Cohen. Polylog-time and near-linear work approximation scheme for undirected shortest paths. *J. ACM*, 47(1):132–166, 2000.

[68] D. Coleman, I. A. Şucan, M. Moll, K. Okada, and N. Correll. Experience-based planning with sparse roadmap spanners. In *Proc. IEEE International Conference on Robotics and Automation (ICRA)*, pages 900–905, 2015.

[69] L. J. Cowen. Compact routing with minimum stretch. *J. Algorithms*, 38(1):170–183, 2001.

[70] L. J. Cowen and C. G. Wagner. Compact roundtrip routing in directed networks. *J. Algorithms*, 50(1):7–95, 2004.

[71] A. De Luca, A. Hang, F. Brudy, C. Lindner, and H. Hussmann. Touch me once and I know it's you!: Implicit authentication based on touch screen patterns. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 987–996, 2012.

[72] E. D. Demaine, J. S. B. Mitchell, and J. O'Rourke. The open problems project. https://cs.smith.edu/~jorourke/TOPP/. Online; accessed 11-August-2019.

[73] A. Dobson and K. E. Bekris. Sparse roadmap spanners for asymptotically near-optimal motion planning. *The International Journal of Robotics Research*, 33(1):18–47, 2014.

[74] R. Duan and S. Pettie. Fast algorithms for (max, min)-matrix multiplication and bottleneck shortest paths. In *Proc. of the 20th Annu. ACM-SIAM Sympos. on Discrete Algorithms (SODA)*, pages 384–391, 2009.

[75] R. Durbin, S. R. Eddy, A. Krogh, and G. Mitchison. *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press, New York, 1998.

[76] H. Edelsbrunner, J. O'Rourke, and R. Seidel. Constructing arrangements of lines and hyperplanes with applications. *SIAM J. Comput.*, 15(2):341–363, 1986.

[77] A. Efrat, Q. Fan, and S. Venkatasubramanian. Curve matching, time warping, and light fields: New algorithms for computing similarity between curves. *Journal of Mathematical Imaging and Vision*, 27(3):203–216, 2007.

[78] T. Eiter and H. Mannila. Computing discrete Fréchet distance. Technical report, TU Vienna, Austria, 1994.

[79] M. Elkin. Computing almost shortest paths. In *Proc. 20th Annu. ACM Sympos. on Principles of Distributed Computing (PODC)*, pages 53–62, 2001.

[80] M. Elkin and D. Peleg. Approximating k-spanner problems for $k > 2$. In *Proc. 8th International Conference on Integer Programming and Combinatorial Optimization (IPCO)*, pages 90–104, 2001.

[81] M. Elkin and D. Peleg. The client-server 2-spanner problem with applications to network design. In *Proc. 8th International Colloquium on Structural Information and Communication Complexity (SIROCCO)*, pages 117–132, 2001.

[82] M. Elkin and D. Peleg. $(1+\epsilon, \beta)$-spanner constructions for general graphs. *SIAM J. Comput.*, 33(3):608–631, 2004.

[83] D. Eppstein, M. Overmars, G. Rote, and G. Woeginger. Finding minimum area k-gons. *Discrete Comput. Geom.*, 7(1):45–58, 1992.

[84] P. Erdős. Extremal problems in graph theory. In *Proc. Sympos. on Theory of Graphs and its Applications (Smolenice, Czechoslovakia)*, pages 29–36, 1963.

[85] J. Erickson. Lower bounds for linear satisfiability problems. *Chicago Journal of Theoretical Computer Science*, 1999(8), 1999.

[86] E. Ezra, S. Har-Peled, H. Kaplan, and M. Sharir. Decomposing arrangements of hyperplanes: VC-dimension, combinatorial dimension, and point location. *CoRR*, abs/1712.02913, 2017.

[87] E. Ezra and M. Sharir. A nearly quadratic bound for the decision tree complexity of k-SUM. In *Proc. 33rd International Symposium on Computational Geometry (SoCG)*, pages 41:1–41:15, 2017.

[88] E. Ezra and M. Sharir. A nearly quadratic bound for point-location in hyperplane arrangements, in the linear decision tree model. *Discrete Comput. Geom.*, 61(4):735–755, 2019.

[89] S. Fortune and J. Hopcroft. A note on Rabin's nearest-neighbor algorithm. *Inform. Process. Lett.*, 8(1):20–23, 1979.

[90] M. Fréchet. Sur quelques points du calcul fonctionnel. *Rendiconti del Circolo Mathematico di Palermo*, 22:1–74, 1906.

[91] G. N. Frederickson and D. B. Johnson. The complexity of selection and ranking in $X + Y$ and matrices with sorted columns. *J. Comput. Sys. Sci.*, 24(2):197–208, 1982.

[92] M. L. Fredman. How good is the information theory bound in sorting? *Theor. Comput. Sci*, 1(4):355–361, 1976.

[93] M. L. Fredman. New bounds on the complexity of the shortest path problem. *SIAM J. Comput.*, 5(1):83–89, 1976.

[94] A. Freund. Improved subquadratic 3SUM. *Algorithmica*, 77(2):440–458, 2017.

[95] A. Gajentaan and M. H. Overmars. On a class of $O(n^2)$ problems in computational geometry. *Comput. Geom.*, 5:165–185, 1995.

[96] J. Gao and D. Zhou. The emergence of sparse spanners and greedy well-separated pair decomposition. In *Proc. 12th Scandinavian Sympos. and Workshops on Algorithm Theory (SWAT)*, pages 50–61, 2010.

[97] C. Gavoille and C. Sommer. Sparse spanners vs. compact routing. In *Proc. 23rd Annu. ACM Sympos. on Parallelism in Algorithms and Architectures (SPAA)*, pages 225–234, 2011.

[98] E. N. Gilbert and E. F. Moore. Variable-length binary encodings. *Bell System Technical Journal*, 38(4):933–967, 1959.

[99] O. Gold and M. Sharir. On the complexity of the discrete Fréchet distance under L_1 and L_∞. *In Proc. 31st European Workshop on Computational Geometry (EuroCG)*, 2015.

[100] O. Gold and M. Sharir. Dominance product and high-dimensional closest pair under L_∞. In *Proc. of the 28th International Sympos. on Algorithms and Computation (ISAAC)*, pages 39:1–39:12, 2017.

[101] O. Gold and M. Sharir. Improved bounds for 3SUM, k-SUM, and linear degeneracy. In *Proc. 25th Annu. European Sympos. on Algorithms (ESA)*, pages 42:1–42:13, 2017. Also in arXiv:1512.05279, 2015.

[102] O. Gold and M. Sharir. Dynamic time warping and geometric edit distance: Breaking the quadratic barrier. *ACM Trans. Algorithms*, 14(4):50:1–50:17, 2018. Also in Proc. 44th International Colloquium on Automata, Languages, and Programming (ICALP), pages 25:1-25:14, 2017.

[103] S. Grabowski. New tabulation and sparse dynamic programming based techniques for sequence similarity problems. *Discrete Applied Mathematics*, 212:96–103, 2016.

[104] A. Grønlund and S. Pettie. Threesomes, degenerates, and love triangles. In *Proc. 55th Annu. Sympos. on Foundations of Computer Science (FOCS)*, pages 621–630, 2014.

[105] J. Hartmanis and R. E. Stearns. On the computational complexity of algorithms. *Trans. Amer. Math. Soc.*, 117:285–306, 1965.

[106] T. Haynes, S. Hedetniemi, and P. Slater. *Fundamentals of Domination in Graphs*. Chapman & Hall/CRC Pure and Applied Mathematics. Taylor & Francis, 1998.

[107] M. Henning and A. Yeo. *Total Domination in Graphs*. Springer Monographs in Mathematics. Springer New York, 2014.

[108] A. Hernández-Barrera. Finding an $o(n^2 \log n)$ algorithm is sometimes hard. In *Proc. 8th Canadian Conference on Computational Geometry*, pages 289–294, 1996.

[109] X. Huang and V. Y. Pan. Fast rectangular matrix multiplication and applications. *J. Complexity*, 14(2):257–299, 1998.

[110] R. Impagliazzo and R. Paturi. On the complexity of k-SAT. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001.

[111] R. Impagliazzo, R. Paturi, and F. Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001.

[112] P. Indyk, M. Lewenstein, O. Lipsky, and E. Porat. Closest pair problems in very high dimensions. In *Proc. 31st International Colloquium on Automata, Languages and Programming (ICALP)*, pages 782–792, 2004.

[113] Z. Jafargholi and E. Viola. 3SUM, 3XOR, triangles. *Algorithmica*, 74(1):326–343, 2016.

[114] D. M. Kane, S. Lovett, and S. Moran. Near-optimal linear decision trees for k-SUM and related problems. In *Proc. 50th Annu. ACM Sympos. on Theory of Computing (STOC)*, pages 554–563, 2018.

[115] H. Kaplan, L. Kozma, O. Zamir, and U. Zwick. Selection from heaps, row-sorted matrices, and $X+Y$ using soft heaps. In *2nd Symposium on Simplicity in Algorithms (SOSA@SODA)*, pages 5:1–5:21, 2019.

[116] H. Kaplan, S. Mozes, Y. Nussbaum, and M. Sharir. Submatrix maximum queries in Monge matrices and Monge partial matrices, and their applications. In *Proc. 23rd Annu. ACM-SIAM Sympos. on Discrete Algorithms (SODA)*, pages 338–355, 2012.

[117] M. J. Katz and M. Sharir. An expander-based approach to geometric optimization. *SIAM J. Comput.*, 26(5):1384–1408, 1997.

[118] E. Keogh and A. C. Ratanamahatana. Exact indexing of dynamic time warping. *Knowledge and Information Systems*, 7(3):358–386, 2005.

[119] E. J. Keogh and M. J. Pazzani. *Scaling up Dynamic Time Warping to Massive Datasets*, pages 1–11. Springer Berlin-Heidelberg, 1999.

[120] E. J. Keogh and M. J. Pazzani. Scaling up dynamic time warping for datamining applications. In *Proc. 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 285–289, 2000.

[121] D. E. Knuth. Optimum binary search trees. *Acta Informatica*, 1(3):270–270, 1972.

[122] T. Kopelowitz, S. Pettie, and E. Porat. Higher lower bounds from the 3SUM conjecture. In *Proc. 27th Annu. ACM-SIAM Sympos. on Discrete Algorithms (SODA)*, pages 1272–1287, 2016.

[123] K. Labib, P. Uznanski, and D. Wolleb-Graf. Hamming Distance Completeness. In *Proc. 30th Annu. Sympos. on Combinatorial Pattern Matching (CPM)*, pages 14:1–14:17, 2019.

[124] F. Le Gall. Faster algorithms for rectangular matrix multiplication. In *Proc. 53rd Annu. IEEE Sympos. on Foundations of Computer Science (FOCS)*, pages 514–523, 2012. Also in arXiv:1204.1111, 2012.

[125] F. Le Gall. Powers of tensors and fast matrix multiplication. In *Proc. 39th International Sympos. on Symbolic and Algebraic Computation (ISSAC)*, pages 296–303, 2014.

[126] F. Le Gall and F. Urrutia. Improved rectangular matrix multiplication using powers of the coppersmith-winograd tensor. In *Proc. 29th Annu. ACM-SIAM Sympos. on Discrete Algorithms (SODA)*, pages 1029–1046, 2018.

[127] A. Lincoln, V. V. Williams, J. R. Wang, and R. Williams. Deterministic time-space trade-offs for k-SUM. In *Proc. 43rd Int'l Colloq. on Automata, Languages, and Programming (ICALP)*, pages 58:1–58:14, 2016.

[128] W. J. Masek and M. S. Paterson. A faster algorithm computing string edit distances. *Journal of Computer and System Sciences*, 20(1):18–31, 1980.

[129] J. Matoušek. Computing dominances in E^n. *Inform. Process. Lett.*, 38(5):277–278, 1991.

[130] S. Meiser. Point location in arrangements of hyperplanes. *Information and Computation*, 106(2):286–303, 1993.

[131] A. Mirzaian and E. Arjomandi. Selection in $X + Y$ and matrices with sorted rows and columns. *Information processing letters*, 20(1):13–17, 1985.

[132] M. Müller. *Information Retrieval for Music and Motion*, pages 69–84. Springer Berlin-Heidelberg, 2007.

[133] D. Peleg and A. A. Schäffer. Graph spanners. *Journal of Graph Theory*, 13(1):99–116, 1989.

[134] D. Peleg and J. D. Ullman. An optimal synchronizer for the hypercube. *SIAM J. Comput.*, 18(4):740–747, 1989.

[135] D. Peleg and E. Upfal. A trade-off between space and efficiency for routing tables. *J. ACM*, 36(3):510–530, 1989.

[136] F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag New York, NY, 1985.

[137] M. Pătraşcu. Towards polynomial lower bounds for dynamic problems. In *Proc. 42nd ACM Sympos. on Theory of Computing (STOC)*, pages 603–610, 2010.

[138] M. Pătraşcu and R. Williams. On the possibility of faster SAT algorithms. In *Proc. 21st Annu. ACM-SIAM Sympos. on Discrete Algorithms (SODA)*, pages 1065–1075, 2010.

[139] M. Rabin. Probabilistic algorithms. In *Algorithms and Complexity, Recent Results and New Directions*, Academic Press, pages 21–39, 1976.

[140] C. A. Ratanamahatana and E. Keogh. Three myths about dynamic time warping data mining. In *Proc. 2005 SIAM International Conference on Data Mining*, pages 506–510, 2005.

[141] L. Roditty, M. Thorup, and U. Zwick. Roundtrip spanners and roundtrip routing in directed graphs. In *Proc. 13th Annu. ACM-SIAM Sympos. on Discrete Algorithms (SODA)*, pages 844–851, 2002.

[142] L. Roditty and V. V. Williams. Fast approximation algorithms for the diameter and radius of sparse graphs. In *Proc. 45th ACM Sympos. on Theory of Computing (STOC)*, pages 515–524, 2013.

[143] K. F. Roth. On certain sets of integers. *Journal of the London Mathematical Society*, 28(1):104–109, 1953.

[144] M. I. Shamos. Geometric complexity. In *Proc. of 7th Annu. ACM Sympos. on Theory of Computing (STOC)*, pages 224–233, 1975.

[145] M. A. Soss, J. Erickson, and M. H. Overmars. Preprocessing chains for fast dihedral rotations is hard or even impossible. *Comput. Geom.*, 26(3):235–246, 2003.

[146] T. Tao and V. Vu. *Additive Combinatorics*. Cambridge University Press, 2006.

[147] M. Thorup and U. Zwick. Compact routing schemes. In *Proc. 13th Annu. ACM Sympos. on Parallel Algorithms and Architectures (SPAA)*, pages 1–10, 2001.

[148] M. Thorup and U. Zwick. Approximate distance oracles. *J. ACM*, 52(1):1–24, 2005.

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

[150] T. K. Vintsyuk. Speech discrimination by dynamic programming. *Cybernetics*, 4(1):52–57, 1968.

[151] K. Wang and T. Gasser. Alignment of curves by dynamic time warping. *Annals of Statistics*, 25(3):1251–1276, 1997.

[152] X. Wang, A. Mueen, H. Ding, G. Trajcevski, P. Scheuermann, and E. Keogh. Experimental comparison of representation methods and distance measures for time series data. *Data Mining and Knowledge Discovery*, 26(2):275–309, 2013.

[153] O. Weimann, A. Abboud, and V. V. Williams. Consequences of faster sequence alignment. In *Proc. 41st Int'l Colloq. on Automata, Languages, and Programming (ICALP)*, pages 39–51, 2014.

[154] R. Wenger. Extremal graphs with no C4's, C6's, or C10's. *J. Combinatorial Theory, Series B*, 52(1):113–116, 1991.

[155] V. V. Williams. Multiplying matrices faster than Coppersmith-Winograd. In *Proc. 44th Sympos. on Theory of Computing (STOC)*, pages 887–898, 2012.

[156] V. V. Williams and R. Williams. Subcubic equivalences between path, matrix and triangle problems. In *Proc. 51st Annu. IEEE Sympos. on Foundations of Computer Science (FOCS)*, pages 645–654, 2010.

[157] V. V. Williams and R. Williams. Finding, minimizing, and counting weighted subgraphs. *SIAM J. Comput.*, 42(3):831–854, 2013.

[158] R. Yuster. Efficient algorithms on sets of permutations, dominance, and real-weighted APSP. In *Proc. 20th Annu. ACM-SIAM Sympos. on Discrete Algorithms (SODA)*, pages 950–957, 2009.

[159] U. Zwick. All pairs shortest paths using bridging sets and rectangular matrix multiplication. *J. ACM*, 49(3):289–317, 2002.

בפרק 6 אנו מראים את התוצאות הבאות.

1. לכל גרף מכוון לא ממושקל G על n קודקודים, אנו יכולים לחשב תת־גרף H שהוא $(3/2)$־פורש־קוטר של G, כך שמספר הקשתות ב H הוא $O(n^{3/2}\sqrt{\log n})$.

2. לכל גרף מכוון לא ממושקל G על n קודקודים עם קוטר D, אנו יכולים לחשב תת־גרף H שהוא $(5/3)$־פורש־קוטר של G, כך שמספר הקשתות של H הוא $O(D^{1/3}n^{4/3}\log^{2/3} n)$. חסם זה טוב יותר מהחסם בסעיף 1 עבור תת־גרף $(3/2)$־פורש־קוטר, כאשר $D = o(\sqrt{n/\log n})$.

3. בהינתן גרף מכוון לא ממושקל $G = (V, E)$ על n קודקודים, לכל $\delta, \epsilon \in [0, 1]$, אנו יכולים לחשב תת־גרף $H = (V, E_H \subseteq E)$ המקיים את אחד מהבאים.

   (א) H הוא $(1 + \delta)$־פורש־קוטר עם $O(n^{2-\epsilon}\log^{1-\epsilon} n)$ קשתות, או

   (ב) H הוא $(2 - \delta)$־פורש־קוטר עם $O(n^{1+\epsilon}\log^{\epsilon} n)$ קשתות.

לשם פשטות, התוצאות המבואות למעלה מוצגות כאן ללא זמני ריצה (אשר מובאים בפרק 6), ועבור גרפים מכוונים לא ממושקלים. עם זאת, התוצאות שלנו עובדות גם עבור גרפים עם משקולות חיוביים (לפחות 1) על הקשתות. אם G גרף מכוון עם משקולות מהתחום $[1, W]$ על הקשתות, אז המתיחה של H תגדל בנוסף לגורם הכפלי t בגורם חיבורי בגודל W.

ב [64] אנו גם מראים שהחסמים מסעיפים 1 ו 2 הם הדוקים, ואנו חוקרים סוגים נוספים של תתי־גרפים פורשים. אנו גם מראים כיצד ניתן לתחזק תתי־גרפים פורשי קוטר בסביבה דינמית (תחת הורדה/הוספה של קשתות בגרף), אך אנו לא כוללים תוצאות אלו בעבודה זו, ומפנים את הקורא ל [64] לפרטים נוספים.

לא זו בלבד שתתי־גרפים פורשי קוטר הם מעניינים מבחינה תיאורטית, הרבה מהשימושים של תתי־גרפים פורשים "רגילים" (שעובדים על גרפים לא מכוונים) מתאימים גם עבור תתי־גרפים פורשי קוטר. בפרט, תתי־גרפים פורשי קוטר עם קבוצת קשתות דלילה יכולים לשמש כמועמדים טובים לרשתות שדרה (backbone networks) [95].

4 תתי־גרפים פורשי קוטר

בפרק 6 אנו מציגים מחקר על תתי־גרפים "פורשי קוטר" (*diameter spanners*), אשר מיד נגדיר. פרק זה מבוסס על המאמר [64] של המחבר ו Keerti Choudhary.

תת־גרף פורש (נקרא גם *spanner*) של גרף לא מכוון $G = (V, E)$ הינו תת־גרף $H = (V, E_H \subseteq E)$ אשר משמר בקירוב את כל המרחקים (הקצרים ביותר) המקוריים בין כל זוגות הקודקודים בגרף G. באופן פורמלי, H הוא תת־גרף t־פורש (*t-spanner*) של G אם"ם לכל זוג קודקודים $u, v \in V$ מתקיים $d_H(u,v) \leq t \cdot d_G(u,v)$, כאשר $d_H(u,v)$ ו $d_G(u,v)$ הם המרחקים בין u ל v ב H ו G בהתאמה. הפרמטר t נקרא גם פרמטר ה"מתיחה" (*stretch*) של H. בהינתן גרף G ופרמטר מתיחה t, "תת־גרף פורש טוב" עבור G נחשב לאחד שיש לו כמות קשתות דלילה משמעותית (בגורם פולינומי) משל G.

תתי־גרפים פורשים הוצגו לראשונה בשנות ה 80 (ראו [21, 130, 131]). Althöfer et al. [14] הראו שלכל גרף לא מכוון (ייתכן עם קשתות ממושקלות) על n קודקודים יש תת־גרף $(2k-1)$־פורש עם $O(n^{1+1/k})$ קשתות, לכל מספר טבעי $k > 0$. אם מניחים את קיום השערת ה girth של Erdős [83], שקלול התמורות (ה trade-off) הזה בין מספר הקשתות בתת הגרף הפורש לפרמטר המתיחה (stretch) שלו הוא אופטימלי.

מלבד היותם מעניינים מבחינה תיאורטית, לתתי־גרפים פורשים יש הרבה מאד שימושים בתחומים שונים של מדעי המחשב, כגון מערכות מבוזרות, רשתות תקשורת וסכמות ניתוב [15, 68, 69, 95, 96, 132, 139, 145], תכנון תנועה [67, 72], קירוב מסלולים קצרים ביותר [65, 66, 78], ומבני נתונים העונים על שאילתות מרחק [29, 146].

עבור גרפים מכוונים, ניתן להראות שקיימים גרפים עם $\Omega(n^2)$ קשתות כך שאפילו הורדה של קשת אחת מהגרף תיתן לנו תת־גרף שפרמטר המתיחה שלו הוא בגודל של קוטר הגרף (המרחק הקצר ביותר המקסימלי בין זוג קודקודים בגרף). לכן, עבור גרפים מכוונים לא ניתן לקבל תתי־גרפים פורשים מעניינים (דלילים) עבור פרמטר מתיחה t כללי. מכאן מגיעה המוטיבציה לחקור תתי־גרפים t־פורשי־קוטר. בהינתן גרף מכוון $G = (V, E)$ עם קוטר D, תת־גרף $H = (V, E_H \subseteq E)$ הוא t־פורש־קוטר אם"ם הקוטר של H הוא לכל היותר $\lceil tD \rceil$.

עבור $t = 2$, קל לבנות H כזה עם $O(n)$ קשתות, כפי שנראה בפרק 6. זה מביא אותנו לשאלה המרכזית הבאה.

שאלה: בהינתן גרף מכוון $G = (V, E)$, ופרמטר מתיחה $t < 2$, האם יש ל G תת־גרף t־פורש־קוטר $H = (V, E_H \subseteq E)$? אם כן, כמה קטן הגודל של E_H יכול להיות? ומהו שקלול התמורות (ה trade-off) בין גודל זה ל t?

כולל גורם שתלוי בקוטר של קבוצת נקודות הקלט.

בפרק 5 אנו משפרים ומפשטים את התוצאה של Indyk *et al.* [111] עבור מציאת זוג נקודות קרובות ביותר במימד גבוה תחת מטריקת L_∞. אנו מראים שניתן לפתור בעיה זו על־ידי אלגוריתם דטרמיניסטי פולינומי־חזק שרץ בזמן $O(DP(n,d)\log n)$, או על־ידי אלגוריתם רנדומי שרץ בתוחלת זמן $O(DP(n,d))$, כאשר $DP(n,d)$ הוא חסם זמן הריצה לחישוב *dominance product* של n נקודות ב $\mathbb{R}^d$; זוהי מטריצה D כך שתא $[i,j]$ במטריצה מוגדר להיות $D[i,j]=|\{k \mid p_i[k] \le p_j[k]\}|$, כלומר $D[i,j]$ מכיל את מספר הקואורדינטות שבהן p_j שולט על p_i.

עבור גרסה של בעיית מציאת זוג נקודות ב $\mathbb{R}^d$ תחת מטריקת L_∞ בה כל הקואורדינטות של הנקודות הם מספרים שלמים מאיזשהו תחום $[-M,M]$, אנו מראים אלגוריתם שרץ בזמן $\widetilde{O}(\min\{Mn^{\omega(1,r,1)}, DP(n,d)\})$, כאשר $\omega(1,r,1)$ הוא המעריך בזמן הריצה של הכפלת מטריצה בגודל $n\times n^r$ במטריצה בגודל $n^r\times n$.[2]

בנוסף, אנו נותנים חסמים קצת יותר טובים עבור $DP(n,d)$, על־ידי ניתוח כללי יותר של האלגוריתם של Yuster [156], אשר משתמש בכפל מטריצות מלבניות. על־ידי הצבת חסמים עדכניים עבור הכפלת מטריצות מלבניות (ראו [122, 124]) בניתוח שאנו מביאים, ניתן לקבל חסמים משופרים עבור $DP(n,d)$. חישוב dominance product בעצמו הינו משימה חשובה, משום שמשתמשים בחישוב זה (כקופסא שחורה) בעוד אלגוריתמים לפתרון בעיות בסיסיות (בנוסף לאלגוריתם שלנו), כגון all-pairs-bottleneck-paths (APBP) [73], וגרסאות מסוימות של all-pairs-shortest-paths (APSP) [156].

לאחר המאמר שלנו [99], Graf, Labib, ו Uznański [121] הראו שב $\mathbb{R}^d$, חישוב dominance product שקול חישובית (עד כדי גורמים פולי־לוגריתמים) למציאת זוג קרובות ביותר תחת כל מטריקת L_p, כאשר $p\ge 3$ הינו קבוע אי־זוגי. (נעיר שעבור כל p קבוע וזוגי, חסם זמן הריצה עבור בעיה זו נמוך משמעותית מהחסם על $DP(n,d)$ [111].) בתוצאה שלנו, אנו למעשה מראים שחישוב dominance produt ב $\mathbb{R}^d$ הוא לפחות קשה כמו מציאת זוג נקודות קרובות ביותר תחת מטריקת L_∞ ב $\mathbb{R}^d$. התוצאות של Graf, Labib, ו Uznański יחד עם התוצאה שלנו מראות את הקשר בין חישוב dominance product לבעיות מציאת זוג נקודות קרובות ביותר תחת מטריקות שונות.

[2] הסימון $\widetilde{O}(\cdot)$ מסתיר גורמים פולי־לוגריתמים.

מעליה עובדים היא פוליהדרית[1] (למשל L_1, L_∞).

ייתכן מאד שלא ניתן לשפר בעיות אלו מעבר לחסם מסוג זה (או לכל היותר שיפור בגורם פולילוגריתמי), משום ש Bringmann ו Künnemann [39] הראו שאם מניחים את קיום SETH, אז אפילו את המקראה החד־מימדי של DTW לא ניתן לפתרון בזמן $O(n^{2-\Omega(1)})$. בהמשך, Abboud et al. [3], ו Abboud ו Bringmann [2] הראו שאפילו שיפור של גורם לוגריתמי בחזקה מספיק גבוהה עבור בעיות התאמה ריבועיות דומות, יכול להביא להשלכות משמעותיות, כגון אלגוריתמים מהירים יותר ל Formula-SAT וחסמים תחתונים חדשים בסיבוכיות מעגלים.

מדד ה־ Discrete Fréchet Distance מעל מטריקות פוליהדריות. מדד פופולרי נוסף בין סדרות של נקודות הוא Discrete Fréchet Distance. Bringmann ו Mulzer [40] הראו שאם מניחים את קיום SETH, לא ניתן לחשב מדד זה בזמן $O(n^{2-\Omega(1)})$, אפילו למקרה החד־מימדי בו נתונות שתי סדרות של n נקודות כל אחת על הישר $\mathbb{R}$ (עם המטריקה הסטנדרטית $d(x,y) = |x-y|$). בחלק 4.5 של העבודה אנו מראים שיש לבעיה זו עץ החלטה 2־לינארי פשוט עם עומק רק $O(n \log^2 n)$, ובאופן יותר כללי, עבור שתי סדרות של n נקודות ב $\mathbb{R}^d$, אנו מראים שקיים עץ החלטה $2d$־לינארי עם עומק $O(n \log^2 n)$, לכל קבוע d, כאשר המטריקה שעובדים מעליה היא פוליהדרית[1] (כגון L_1, L_∞).

3 זוג נקודות קרובות ביותר במימד גבוה תחת מטריקת L_∞

בפרק 5 אנו חוקרים את בעיית מציאת זוג נקודות קרובות ביותר במימד גבוה תחת מטריקת L_∞. פרק זה מבוסס על המאמר [99] של המחבר והמנחה שלו.

בהינתן n נקודות ב $\mathbb{R}^d$, בעיית "זוג נקודות קרובות ביותר" היא למצוא זוג נקודות שונות שהמרחק ביניהן הוא מינימלי. כאשר d הוא קבוע, ישנם אלגוריתמים יעילים שפותרים את הבעיה, כמו כן ישנם אלגוריתמים מהירים לחישוב פתרון מקורב עבור d כללי. אולם, מציאת פתרון מדויק עבור מימדים גבוהים (למשל $d = n$) נראה הרבה פחות מובן מבחינה אלגוריתמית. אנו עוסקים בבעיית זוג נקודות קרובות ביותר תחת מטריקת L_∞ ב $\mathbb{R}^d$, כאשר $d = n^r$, עבור איזשהו $r > 0$. קל לראות שהאלגוריתם הנאיבי לפתרון בעיה זו רץ בזמן $O(dn^2)$. עבור $d = n$, Indyk *et al.* [111] הראו את האלגוריתם הראשון הלא־טריוויאלי עבור מציאת זוג קרובות ביותר תחת מטריקת L_∞, אך התוצאה שלהם היא פולינומית־חלשה משום שזמן הריצה

[1] כלומר, שהמטריקה מושרית על־ידי נורמה שכדור היחידה שלה הוא פאון סימטרי קמור בעל מספר קבוע של פאות.

ל 3SUM שרץ בזמן $O(n^2(\log\log n)^{O(1)}/\log^2 n)$. להתפתחויות נוספות הקשורות בבעיות אלו ראו גם [25, 50, 120, 125].

2 אלגוריתמים להתאמה גיאומטרית

בפרק 4 אנו חוקרים בעיות בסיסיות להתאמה גיאומטרית בין שתי סדרות של נקודות. פרק זה מבוסס על המאמרים [98, 101] של המחבר והמנחה שלו.

חישוב התאמה גיאומטרית נותן מדד ל"דמיון" בין עקומים או סדרות של נקודות במרחב מטרי מסוים. מדדים פופולרים להתאמה גיאומטרית הם: Dynamic Time Warping, Geometric Edit Distance, ו Discrete Fréchet Distance. בפרק 4 אנו חוקרים את את סיבוכיות החישוב של מדדים אלו.

המדדים Dynamic Time Warping (DTW) ו Geometric Edit Distance (GED) הם מדדים בסיסיים לחישוב של "דימיון" בין עקומים או סדרות זמן המיוצגות על־ידי סדרות של נקודות במרחב מטרי מסוים. ספציפית, מדד DTW נמצא בשימוש רב בתחומים שונים של מדעי המחשב וביואינפורמטיקה, כגון זיהוי קול, התאמה בין צורות גיאומטריות, השוואה בין סדרות DNA וחלבונים, זיהוי מחוות על מסכי מגע, זיהוי ועיבוד מוסיקה, התאמת נתונים מבוססי סדרות בזמן, והרבה שימושים בתחום כרית נתונים. ראו דוגמאות לשימושים כאלה ב [47, 70, 76, 116־118, 129, 138, 149]. עד היום, חיפוש של הביטוי "dynamic time warping" (עם מרכאות) ב Google Scholar מניב כ 40,000 מאמרים, וחיפוש סטנדרטי ב Google מניב כ 270,000 תוצאות. זה מדגים את הפופולריות העצומה והחשיבות של מדד זה.

לכן, משימת החישוב של מדד ה DTW (או GED) היא בעיה מרכזית ב P. למרות מאמצים רבים למצוא אלגוריתמים יעילים יותר, האלגוריתם הכי טוב שהיה ידוע לחישוב DTW או GED בין שתי סדרות של n נקודות כל אחת ב $\mathbb{R}^d$ הוא אלגוריתם תכנון דינמי משנות ה 60 שרץ בזמן ריבועי (כלומר בזמן $\Theta(n^2)$), אפילו למקרה החד־מימדי שבו $d = 1$, שזה אחד המקרים הנפוצים בהרבה שימושים מעשיים.

האלגוריתמים הראשונים שרצים בזמן תת־ריבועי עבור חישוב DTW ו GED. בחלק 4.1 של עבודה זו אנו "שוברים" את החסם הריבועי בן הכ 50 שנים לחישוב DTW או GED בין שתי סדרות של n נקודות כל אחת ב $\mathbb{R}$, על־ידי פיתוח אלגוריתמים דטרמיניסטים שרצים בזמן $O(n^2/\log\log n)$. האלגוריתמים שלנו עובדים גם במימדים גבוהים יותר, כל עוד המטריקה

1 בעיות ה 3SUM, k-SUM, ובדיקת ניוון לינארית

בפרק 3 אנו חוקרים את הבעיות 3SUM, k-SUM, ובדיקת ניוון לינארית (נקראת גם Linear Degeneracy Testing). פרק זה מבוסס על המאמר [100] של המחבר והמנחה שלו.

בהינתן n מספרים ממשיים, הגרסה הכללית של בעיית ה 3SUM היא להכריע האם קיימים שלושה מספרים מתוכם שסכומם שווה לאפס. עד לפריצת דרך שקרתה ב 2014 על־ידי Grønlund ו Pettie [103], היתה השערה שאלגוריתם ידוע פשוט שרץ בזמן $O(n^2)$ הוא אופטימלי לבעיה זו. בחלוף השנים, חוקרים הראו שבעיות אלגוריתמיות רבות ניתנות לרדוקציה מבעיית ה 3SUM או גרסאות מוכללות שלה, כגון k-SUM ו k-linear degeneracy testing (k-LDT). בהינתן פונקציה לינארית $f(x_1,\ldots,x_k) = a_0 + \sum_{1\le i\le k} a_i x_i$ וקבוצה סופית $A \subset \mathbb{R}$, בעיית ה k-LDT היא להכריע האם $0 \in f(A^k)$. המקרה המיוחד בו $f(x_1,\ldots,x_k) = \sum_{i=1}^{k} x_i$ נקרא בעיית ה k-SUM. עבור k כללי, זו שאלה פתוחה האם בעיות אלו ניתנות לפתרון בזמן $o(n^{\lceil k/2\rceil})$ (כפי שהזכרנו מקודם, זה נכון ל $k = 3$). ההשערות על קושי החישוב של בעיות אלו הפכו להיות בסיס נפוץ מאד להוכחות של חסמים תחתונים תלויים עבור בעיות רבות ב P.

בפרק 3, אנו מראים שסיבוכיות עץ החלטה רנדומי 4־לינארי של 3SUM היא $O(n^{3/2})$, ושסיבוכיות עץ החלטה רנדומי $(2k-2)$־לינארי של k-SUM ושל k-LDT היא $O(n^{k/2})$, לכל $k \ge 3$ אי־זוגי. (ראו פרק 2.1 עבור הגדרה פורמלית של מודל עץ החלטה רנדומי r־לינארי.) חסמים אלו משפרים בהתאמה (אומנם במודל רנדומי) את החסם $O(n^{3/2}\sqrt{\log n})$ על סיבוכיות עץ החלטה 4־לינארי ואת החסם $O(n^{k/2}\sqrt{\log n})$ על סיבוכיות עץ החלטה $(2k-2)$־לינארי של Grønlund ו Pettie [103] (הם הראשונים שהראו שסיבוכיות עץ החלטה לינארי ל 3SUM היא תת־ריבועית). בנוסף, אנו מראים אלגוריתם דטרמיניסטי ל 3SUM שרץ בזמן $O(n^2 \log\log n/\log n)$, אשר משפר את החסם $O(n^2(\log\log n/\log n)^{2/3})$ של Grønlund ו Pettie [103]. חסם זה זהה לחסם שהתקבל באופן בלתי תלוי על־ידי Freund [93], אך האלגוריתם שלנו פשוט יותר, עקב ניצול עמוק יותר של מודל ה word-RAM.

לאחר המאמר שלנו [100], פורסמו התפתחויות רבות על בעיות אלו, שהמרכזית שבהן היא פריצת דרך של Kane, Lovett, ו Moran [113] שהראו שסיבוכיות עץ החלטה $2k$־לינארי של k-SUM היא רק $O(kn\log^2 n)$, והראו עצי החלטה כמעט אופטימליים גם עבור בעיות בסיסיות נוספות, כגון "למיין את $X+Y$" (למיין את הקבוצה $\{x+y \mid x\in X,\, y\in Y\}$, כאשר $X,\, Y$ הן קבוצות של n מספרים ממשיים כל אחת), ו־ APSP. התפתחות נוספת בוצעה על־ידי Chan [54] ששיפר גורם לוגריתמי נוסף בסיבוכיות הזמן של 3SUM, על־ידי פיתוח אלגוריתם דטרמיניסטי

בזמן $O(n^{2-\Omega(1)})$ את הבעיות שהזכרנו קודם, ביניהן Edit Distance, Longest Common Subsequence, Discrete Fréchet Distance, Dynamic Time Warping, ובעיות רבות נוספות. ראו [3-6, 16, 22, 26, 38, 62, 94, 112, 120, 134, 135, 140, 151, 154] עבור דוגמאות לחסמים תחתונים תלויים.

עבודות נוספות שנעשו לאחרונה על־ידי Abboud *et al.* [3], ו Abboud ו Bringmann [2] מראות שאפילו שיפור של $\text{polylog}(n)$ בזמן הריצה של אחת מהבעיות שהזכרנו לעיל, יוביל להשלכות משמעותיות, כגון אלגוריתמי Formula-SAT מהירים יותר, או חסמים תחתונים חדשים בסיבוכיות מעגלים (circuit complexity). עבודות אלו נותנות ראיות לקושי (וההשלכות) של שיפורים לאלגוריתמים הידועים כיום, לכך שייתכן שהם אופטימלים, ולכך ששיפורים של $\text{polylog}(n)$ בזמן הריצה הם אולי הדרך היחידה להגיע לפתרון האופטימלי.

הסקירה עד עכשיו מתייחסת למודל החישוב הסטנדרטי real-RAM (נקרא גם uniform model), אשר זמן החישוב שלו כולל את כל פעולות האריתמטיקה הסטנדרטיות והפעולות הבוליאניות אשר מתבצעות על־ידי האלגוריתם. מודל אחר, מנוון יותר, אך פופולרי מאד הוא מודל עץ החלטה (decision tree model), בו כל הריצות של האלגוריתם על כל קלט אפשרי מיוצגות על־ידי עץ, שבו כל הסתעפות מבוססת על בדיקת סימן של ביטוי אלגברי עם אילוצים מסוימים (ראו מטה). במודל זה, רק ההסתעפויות במסלול החישוב נספרות (מסלול חישוב מתחיל בשורש ומסתיים בעלה). הסיבוכיות של עץ ההחלטה היא העומק שלו (העומק נותן חסם עליון על מספר ההסתעפויות של כל מסלול חישוב). סוג מאד פופולרי של עץ החלטה, שעליו גם נתמקד בעבודה זו, הוא "עץ החלטה r־לינארי" או r-linear decision tree, במודל זה כל הביטויים האלגברים הם ביטויים לינארים עם לכל היותר r מחוברים. הגדרה פורמלית של מודל זה מובאת בחלק 2.1 של העבודה.

בעבודה זו אנו מראים עצי החלטה משופרים עבור k־SUM (ובפרט 3SUM), ועבור Discrete Fréchet Distance מעל מטריקות פוליהדריות, וכן אלגוריתמים משופרים לבעיות הפופולריות הבאות: 3SUM, Dynamic Time Warping, Geometric Edit Distance, Dominance Product, ו High Dimensional Closest Pair under L_∞.

כעת, ניתן סקירה קצרה על כל אחד מהפרקים בעבודה זו, על הבעיות הנלמדות בהם, ועל התוצאות שקיבלנו.

תקציר

פיתוח אלגוריתמים אופטימליים הינו נדבך מרכזי במדעי המחשב מאז התפתחותו כתחום מדעי. אולם עד היום, עבור מרבית הבעיות הנחקרות איננו יודעים אם האלגוריתמים הכי חדשים עבורן הם הטובים ביותר שניתן להשיג (אופטימליים). בין הבעיות הפופולריות ביותר במחלקה P הן אלו אשר ידועים עבורן אלגוריתמים סטנדרטים שרצים בזמן $O(n^c)$ עבור קלט בגודל n, כאשר $c = 2$ או $c = 3$. עבור $c = 3$, ניתן למצוא סוגים שונים של בעיות, כגון חישוב המסלול הקצר ביותר בין כל זוג קודקודים בגרף מכוון עם משקולות ממשיים (APSP), והכפלה קומבינטורית של מטריצות. עבור $c = 2$ (זמן ריבועי), ניתן למצוא בעיות בסיסיות רבות, כגון 3SUM, ובעיות התאמה בין מחרוזות, עקומים, וסדרות של נקודות, כגון Edit Distance, Longest Common Subsequence, Dynamic Time Warping, Geometric Edit Distance, ו Discrete Fréchet Distance. מכיוון שלבעיות אלו לא ידוע אלגוריתם שרץ בזמן $O(n^{2-\Omega(1)})$, בעיות אלו נקראות גם "בעיות ריבועיות".

מהמוטיבציה למצוא אלגוריתמים אופטימליים עבור בעיות בסיסיות אלו, חוקרים פיתחו אלגוריתמים משופרים עם זמן ריצה מהצורה $O(n^c/\text{polylog}(n))$, כאשר $\text{polylog}(n)$ מציין $\log^k n$ עבור איזשהו קבוע $k > 0$. כתוצאה מעבודות אלו, לסיבוכיות של הרבה בעיות ריבועיות קלאסיות יש כיום חסמים מהצורה $O(n^2/\text{polylog}(n))$. רק לאחרונה הראו Grønlund ו Pettie [103] שבעיית ה 3SUM המפורסמת (להכריע האם קיימים 3 מספרים בקבוצה של n מספרים ממשיים שסכומם הוא 0) ניתנת לפתרון ע"י אלגוריתם עם חסם תת־ריבועי מהצורה הזו. מכיוון שהסיבוכיות של בעיית ה 3SUM משמשת כחסם תחתון עבור הרבה בעיות אחרות [94, 107], הנקראות גם בעיות 3SUM-Hard (מראים זאת על־ידי רדוקציות מ 3SUM), כל התקדמות בהבנה שלנו על הסיבוכיות של 3SUM היא מאד רצויה.

עבודת מחקר משלימה לנסיון למצוא אלגוריתמים משופרים היא להבין טוב יותר עד כמה תיאורטית ניתן לשפר את האלגוריתמים הקיימים, על־ידי הוכחת חסמים תחתונים. אולם, נראה שהידע הקיים בהוכחת חסמים תחתונים "אמיתיים" (לא תלויים בהשערות לא מוכחות) הוא מוגבל מאד. עם זאת, בשנים האחרונות חלה התקדמות משמעותית בהבנה שלנו על סיבוכיות של בעיות בסיסיות ב P על ידי הוכחת "חסמים תחתונים תלויים", דרך רדוקציות מבעיות מרכזיות, כגון 3SUM, APSP, ו CNF-SAT. לדוגמא, בהינתן n נקודות במישור, להכריע האם קיימות שלוש נקודות קו־ליניאריות (שיש ישר העובר דרך שלושתן) זו בעיה שידועה להיות קשה לפחות כמו 3SUM (זו אחת הבעיות המפורסמות הידועות כ 3SUM-Hard). לאחרונה חוקרים הוכיחו כי בהנחה שלא ניתן לפתור את CNF-SAT בזמן $O(2^{(1-\Omega(1))n})$ (הנחה שנובעת ממה שנקרא Strong Exponentail Time Hypothesis או בקיצור SETH), נובע שלא ניתן לפתור

תמצית

פיתוח אלגוריתמים אופטימליים הינו נדבך מרכזי במדעי המחשב מאז התפתחותו כתחום מדעי. אולם עד היום, עבור מרבית הבעיות הנחקרות איננו יודעים אם האלגוריתמים הכי חדשים עבורן הם הטובים ביותר שניתן להשיג (אופטימליים). בין הבעיות הפופולריות ביותר במחלקה P הן אלו אשר ידועים עבורן אלגוריתמים סטנדרטים שרצים בזמן $O(n^c)$ עבור קלט בגודל n, כאשר $c = 2$ או $c = 3$. עבור $c = 3$, ניתן למצוא בעיות הכפלה קומבינטורית של מטריצות, ואת בעיית חישוב המסלול הקצר ביותר בין כל זוג קודקודים בגרף מכוון עם משקולות ממשיים (נקראת גם APSP). עבור $c = 2$ (זמן ריבועי), ניתן למצוא בעיות בסיסיות רבות, כגון 3SUM, ובעיות התאמה בין מחרוזות, עקומים, וסדרות של נקודות.

בעבודה זו, נציג עצי החלטה ואלגוריתמים משופרים עבור הבעיות הבסיסיות הבאות.

- עץ החלטה משופר עבור k-SUM ואלגוריתם עם זמן תת־ריבועי משופר עבור 3SUM.
- האלגוריתמים הראשונים שרצים בזמן תת־ריבועי ומחשבים את המדדים Dynamic Time Warping ו Geometric Edit Distance בין שתי סדרות של נקודות ב $\mathbb{R}$. תוצאה זו "שוברת" את החסם הריבועי הידוע כ־ 50 שנים עבור בעיות אלו. האלגוריתמים שלנו עובדים גם במימדים גבוהים יותר, כל עוד המטריקה שעובדים מעליה היא פוליהדרית (כגון L_1 ו־ L_∞).

  עבור בעיית חישוב מדד ה Discrete Fréchet Distance, אנו מראים עץ החלטה עם עומק כמעט לינארי, כאשר המימד קבוע והמטריקה שעובדים מעליה היא פוליהדרית.
- אלגוריתם משופר והראשון שהוא פולינומי־חזק שרץ בזמן $O(n^{3-\Omega(1)})$ עבור מציאת זוג נקודות קרובות ביותר (Closest Pair) מתוך n נקודות ב $\mathbb{R}^n$ תחת מטריקת L_∞.
- אנו מראים שלכל גרף מכוון לא ממושקל יש תתי־גרפים עם מספר קשתות תת־ריבועי אשר משמרים את הקוטר (המרחק הקצר ביותר המקסימלי בין זוג קודקודים בגרף) של הגרף המקורי עד כדי קירוב כפלי שקטן מ־2. אנו קוראים לתתי־גרפים אלו "פורשי קוטר" או *diameter spanners*, ומראים אלגוריתמים יעילים לחשב אותם.

אוניברסיטת תל אביב
הפקולטה למדעים מדויקים ע"ש ריימונד ובברלי סאקלר
ביה"ס למדעי המחשב ע"ש בלווטניק

אלגוריתמים חדשים למספר בעיות קלאסיות ב- P

חיבור לשם קבלת התואר
דוקטור לפילוסופיה
מאת
עומר גולד

מנחה: פרופ' מיכה שריר

הוגש לסנאט של אוניברסיטת תל אביב
אב תשע"ט