Dynamic Time Warping and Geometric Edit Distance: Breaking the Quadratic Barrier *

Omer Gold† Micha Sharir‡

November 5, 2016

Abstract

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 paper, 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 $O\left(n^2 \log\log\log n/\log\log n\right)$ time. 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_∞).

1 Introduction

Searching for optimal algorithms is a standard routine in the study of algorithm design. 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 algorithms, 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, Geometric Edit Distance (GED), Dynamic Time Warping (DTW), Discrete Fréchet Distance, and Longest Common Subsequence (LCS). These problems are usually referred to as "quadratic problems".

Motivated to find optimal algorithms for these basic problems, researchers have come up with time bounds of the form $O(n^c/\operatorname{polylog}(n))$, where $\operatorname{polylog}(n)$ stands for $\log^k n$, for some constant $k > 0$. By now, many classical quadratic problems have upper bounds of the form $O(n^2/\operatorname{polylog}(n))$, including all of the problems mentioned above, except for DTW and GED; see [3, 18, 19, 27] for such upper bounds. Among the very few archetypal quadratic problems for

*Work on this paper has been supported by Grant 892/13 from the Israel Science Foundation, by Grant 2012/229 from the U.S.-Israeli Binational Science Foundation, by the Israeli Centers of Research Excellence (I-CORE) program (Center No. 4/11), and by the Hermann Minkowski–MINERVA Center for Geometry at Tel Aviv University.

†Blavatnik School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel; omergold@post.tau.ac.il

‡Blavatnik School of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel; michas@post.tau.ac.il

which no $o(n^2)$-time algorithm is known, DTW and GED seem to be prominent examples, considering the decades of extensive efforts to break the quadratic barrier.

Motivation. Complementary to the standard theoretical interest in finding optimal algorithms for basic problems in P, a significant progress has been made in recent years towards a better understanding these problems, by proving conditional lower bounds via reductions from basic problems, such as 3SUM and CNF-SAT. Assuming that CNF-SAT takes $\Omega\left(2^{(1-o(1))n}\right)$ time (the so-called *Strong Exponential Time Hypothesis* (SETH) [20,21]), has led to recent lower bounds for a growing list of problems, including most the quadratic problems mentioned above. Specifically, assuming SETH, there is no $O\left(n^{2-\Omega(1)}\right)$-time algorithm for Discrete Fréchet Distance [8], Edit Distance [6], LCS [1,9], and DTW [1,9].

A recent seminal work by Abboud *et al.* [2] shows that even an improvement by a sufficiently large polylogarithmic factor for any of these basic problems would lead to major consequences, such as faster SAT algorithms, and new circuit lower bounds. For example, obtaining an $O\left(n^2/\log^k n\right)$-time algorithm for Edit Distance or LCS, for $k \approx 1000$, will immediately show that SAT on formulas of size $O(n^5)$ can be solved in $O(2^n/n^{15})$ time, which would imply that E^{NP} does not have such formulas.[1] Moreover, if Edit Distance, LCS on two binary sequences of length n can be solved in $O(n^2/\log^c n)$ time for *every* $c > 0$, then $\mathsf{NTIME}[2^{O(n)}]$ does not have non-uniform polynomial-size log-depth circuits. In fact, similar results are obtained for any problem that can implement *"alignment gadgets"* (see [2] for details), which includes DTW (see [9]), which we study in this paper. Hence, the work of Abboud *et al.* highly motivates and revives the study of polylogarithmic-factor improvements for these basic problems, as it can be seen as an effort to find new SAT algorithms, or alternatively, as the only way to push the efficiency of the solution "to the limit".

Problem Statement. 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

$$\mathsf{dtw}(A, B) = \min_{C:\, coupling} \sum_{(p_i, q_j) \in C} \mathsf{dist}(p_i,\ q_j). \tag{1}$$

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

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

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

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

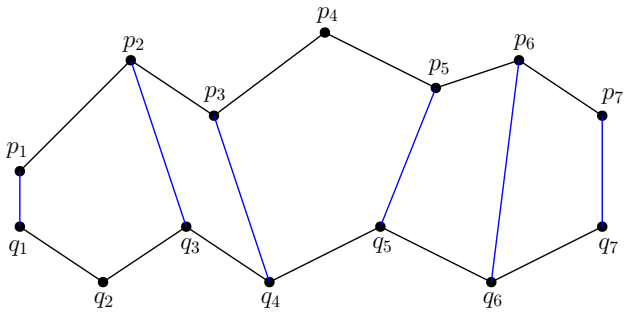

Figure 1.1: Example of a monotone matching (in blue) between two polygonal curves (represented by point-sequences) in the plane.

where the minimum is taken over all sets $\mathcal{M}$ of monotone matchings in the complete bipartite graph $A \times B$. The monotone matching $\mathcal{M}$ for which the above sum is minimized is called the *optimal matching*. The GED problem is to compute $\mathsf{ed}(A, B)$, and sometimes also the optimal matching. More sophisticated gap penalty functions have been proposed [14], 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 $A \cup B$. 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 [11, 13, 15, 23–26, 28, 30] for some examples.

The best-known worst-case running times for solving DTW or GED are given by long-standing dynamic programming algorithms that require $\Theta(nm)$ time. We review the standard quadratic-time DTW and GED algorithms in Sections 2.1 and 4, respectively.

DTW was perhaps first introduced as a speech discrimination method [31] 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 distance $\mathsf{dist}(x, y) = |x - y|$). Even for this special case, no subquadratic-time algorithms have been known. We mainly consider this case throughout the paper.

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.* [32] for a survey. Very recently, Agarwal *et al.* [4] gave a near-linear approximation scheme for computing DTW or GED for a restricted, although quite large, family of curves.

Recently, Bringmann and Künnemann [9] 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 [6]. This line of work was extended in a very recent work by Abboud *et al.* [2], mentioned above, where they show that even a sufficiently large polylog(n)-factor improvement over the quadratic time upper bound for Edit Distance or (the one-dimensional) DTW, will lead to major consequences.

Masek and Paterson [27] showed that Edit Distance between two strings of length at most n over $O(1)$-size alphabet can be solved in $O(n^2/\log n)$ time. More recent works attempt 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 (see [7] for example). 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 the distance between them.

Our Results and Related Work. 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.* [3] 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 recursive formula for the (discrete) Fréchet distance uses the max function, while the formula for DTW and GED involves the 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 [5], 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. We believe that our techniques open a door for improving other geometric problems that involve distances in $\mathbb{R}$ or polyhedral distance metrics in $\mathbb{R}^d$.

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 only when m is subpolynomial in n.

Theorem 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 *$\mathsf{ed}(A, B)$ (and optimal matching) can be computed by a deterministic algorithm in $O(n^2 \log\log\log n/\log\log n)$ time.*

Theorem 1.1 gives the very first subquadratic-time algorithm for solving DTW, breaking the nearly 50 years old $\Theta(n^2)$ time bound. In Section 3.1 we extend our algorithm to give a more general result, which supports high-dimensional polyhedral metric spaces, as stated in Theorem 1.2. In Section 4 we extend our algorithm for solving GED.

Theorem 1.2. *Let $A = (p_1, \ldots, p_n)$ and $B = (q_1, \ldots, q_n)$ be two sequences of n points in a polyhedral metric space*[2] *$(\mathbb{R}^d, \mathsf{dist})$, $\mathsf{dtw}(A, B)$ (and optimal coupling), or $\mathsf{ed}(A, B)$ (and optimal matching) can be computed by a deterministic algorithm in $O(n^2 \log\log\log n/\log\log n)$ time, for any constant d.*

2 Preliminaries

Throughout the paper, 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, $M[1, 1]$ the leftmost-bottom cell of a matrix M.

In Fredman's classic 1976 articles on the decision tree complexity of $(\min, +)$-matrix multiplication [17], and on sorting $X + Y$ [16], he often uses the simple observation that $a + b < a' + b'$ iff $a - a' < b' - b$. This observation is usually referred to as *Fredman's trick*. In our algorithm, we will

[2] 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_∞).

often use the following extension of Fredman's trick.

$$\begin{gathered} a_1 - b_1 + \cdots + a_r - b_r < a'_1 - b'_1 + \cdots + a'_t - b'_t \\ \textit{if and only if} \\ a_1 + \cdots + a_r - a'_1 - \cdots - a'_t < b_1 + \cdots + b_r - b'_1 - \cdots - b'_t. \end{gathered} \tag{3}$$

Our algorithm uses the following geometric domination technique, based on an algorithm by Chan [12]. Given a finite set Q of red points and blue points in $\mathbb{R}^d$, the *bichromatic dominating pairs reporting* problem is to report all the pairs $(p, q) \in Q^2$ such that p is red, q is blue, and p dominates q, i.e., p is greater than or equal to q at each of the d coordinates. A natural divide-and-conquer algorithm [29, p. 366] runs in $O(|Q| \log^d |Q| + K)$ time, where K is the output size. Chan [12] provided an improved *strongly* subquadratic time bound (excluding the cost of reporting the output) when $d = O(\log |Q|)$, with a sufficiently small constant of proportionality.

Lemma 2.1 (Chan [12])**.** *Given a finite set $Q \subset \mathbb{R}^d$ of red and blue points, one can compute all bichromatic dominating pairs $(p, q) \in Q^2$ in time $O(c_\varepsilon^d |Q|^{1+\varepsilon} + K)$, where K is the output size, $\varepsilon \in (0, 1)$ is an arbitrary prespecified parameter, and $c_\varepsilon = 2^\varepsilon/(2^\varepsilon - 1)$.*

Throughout the paper, we invoke Lemma 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; see below for details.

We denote by $[N] = \{1, \ldots, \lceil N \rceil\}$, the set of the first $\lceil N \rceil$ natural numbers, for any $N \in \mathbb{R}^+$.

Throughout the paper, we sometimes refer to a square matrix as a *box*.

Our model of computation is a simplified Real RAM model, in which "truly real" numbers are subject to only two unit-time operations: addition 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.

2.1 The Quadratic 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. This algorithm can be easily extended to return also the optimal coupling (see below). In Section 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 support 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.

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 support 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 each $i, j \in \{2, \ldots, s\}$, each consecutive pairs 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 *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 new leftmost column and new bottommost row. The common element $D_{1,1}[0,0]$ of these 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 $t = 0, \ldots, t^*-1$. The path starts at some point $P(0) = (\cdot, 1)$ or $(1, \cdot)$, which lies on either the left or the bottom boundary, and ends at some t^* (not necessarily the first such index) for which $P(t^*) = (\cdot, g)$ or $(g, \cdot)$; that is, P ends on either the right or the top boundary. Note that t^* can have any value in $[2g-2]$. The number of possible monotone staircase paths in a box $D_{i,j}$ is bounded by 3^{2g-1}, as is easily checked.[3] To simplify the notation we upper bound this quantity by 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 the boxes $D_{i,1}$ and $D_{1,j}$ for all $i, j \in [s]$.) In the algorithm that follows, we want to 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. We omit in this study perturbation techniques that can handle degenerate situations.

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

[3] Each staircase path can be encoded by its first position, followed by the sequence of its at most $2g-1$ moves, where each move is in one of the directions up/right/up-right.

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

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

Note that according the our general position assumption, the shortest path between v and w, withing 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 $P^*_{v,w}$ and $\mathrm{cost}_{i,j}(P^*_{v,w})$ 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 enumerate *every* possible staircase path in $S(v,w)$ as $P_{v,w} : [t^*] \to [g] \times [g]$, where we write $P_{v,w} = \left(P^{\mathrm{r}}_{v,w}, P^{\mathrm{c}}_{v,w}\right)$ as a pair of row and column functions $P^{\mathrm{r}}_{v,w}$, $P^{\mathrm{c}}_{v,w} : [t^*] \to [g]$, so that $P_{v,w}(k) = \left(P^{\mathrm{r}}_{v,w}(k), P^{\mathrm{c}}_{v,w}(k)\right)$, 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.) There are at most 3^{2g} possible staircase paths $P_{v,w}$ (for all admissible pairs $(v,w) \in L \times R$ combined), so in total, we enumerate at most 3^{2g} staircase paths. These enumerations can be done in a natural lexicographic order, so that they induce an order on the $< 4g^2$ admissible pairs of positions of $L \times R$, and for each such pair (v,w), an 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 in a box $D_{i,j}$, we want to use the extended Fredman's trick (as in (3)) to compare $\mathrm{cost}_{i,j}\left(P_{v,w}\right)$ with $\mathrm{cost}_{i,j}\left(P'_{v,w}\right)$, 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 starting positions $(\ell_0, m_0) = (\ell'_0, m'_0) = v$). We have

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

and

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

and we want to test whether, say, $\mathrm{cost}_{i,j}\left(P_{v,w}\right) < \mathrm{cost}_{i,j}\left(P'_{v,w}\right)$ (recall that we assume that equalities do not arise), that is, testing whether

$$\left|A_i(\ell_1) - B_j(m_1)\right| + \cdots + \left|A_i(\ell_r) - B_j(m_r)\right| < \left|A_i(\ell'_1) - B_j(m'_1)\right| + \cdots + \left|A_i(\ell'_t) - B_j(m'_t)\right|. \quad (4)$$

The last term in each side of (4) is actually unnecessary, since they are equal. In order to transform this inequality into a form suitable for applying the extended Fredman's trick (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 $A_i(\ell) - B_j(m)$ are all positive, so that (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 (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

$$A_i(P^{\mathrm{r}}_{v,w}(1)) + \cdots + A_i(P^{\mathrm{r}}_{v,w}(r)) - A_i(P'^{\mathrm{r}}_{v,w}(1)) - \cdots - A_i(P'^{\mathrm{r}}_{v,w}(t)) \tag{5}$$

$$< B_j(P^{\mathrm{c}}_{v,w}(1)) + \cdots + B_j(P^{\mathrm{c}}_{v,w}(r)) - B_j(P'^{\mathrm{c}}_{v,w}(1)) - \cdots - B_j(P'^{\mathrm{c}}_{v,w}(t)). \tag{6}$$

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 guess a staircase path $P_{v,w}$ as a candidate for being the shortest path from v to w. The overall number of such guesses is fewer than $(3^{2g})^{4g^2} = 3^{8g^3}$. For a fixed choice of paths, one for each admissible pair $(v,w) \in L \times R$, we want to test whether all the $< 4g^2$ guessed 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 guessed set of shortest paths). We will repeat the procedure for all $< 3^{8g^3}$ possible sets of guessed paths $P_{v,w}$.

Testing a fixed guess of shortest paths. For each group A_i, we create a (blue) point α_i, and for each group 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 guessed path. The value of α_i (resp., β_j) at that coordinate is the corresponding expression (5) (resp., (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 is 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. Clearly, $d_g < 3^{2g}$.

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 guessed (a path for every admissible pair $(v,w) \in L \times R$) are the shortest paths 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 the points, i.e., to compute all their coordinates, is $O(2s \cdot 3^{2g} \cdot g) = O(3^{2g} n)$.

By Lemma 2.1, we can report all pairs of points (α_i, β_j) such that α_i is dominated by β_j, in $O\left(c_\varepsilon^{3^{2g}} (n/g)^{1+\varepsilon} + K\right)$ time, where K is the number of boxes at which the test of our specific guesses comes out positive. As mentioned earlier, we use $\varepsilon = 1/2$, with $c_\varepsilon \approx 3.42$.

This runtime is for a specific guess 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 guesses is bounded by 3^{8g^3}, the overall runtime for all invocations of the bichromatic dominance reporting algorithm (including preparing the points) is

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

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's trick 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 guess 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

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

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, guessing sets of paths, but now we also guess the sign assignment 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 the guessed shortest path for some admissible pair of positions $(v, w) \in L \times R$, and let σ be the guessed sign assignment. Then, for every other path $P' \in S(v, w)$, we have the modified coordinates

$$\alpha_i = \big(\ldots, \sigma(P(1))A_i(P^{\mathrm{r}}(1)) + \cdots + \sigma(P(r))A_i(P^{\mathrm{r}}(r)) - \sigma(P'(1))A_i(P'^{\mathrm{r}}(1)) - \cdots - \sigma(P'(t))A_i(P'^{\mathrm{r}}(t)), \ldots\big),$$
$$\beta_j = \big(\ldots, \sigma(P(1))B_j(P^{\mathrm{c}}(1)) + \cdots + \sigma(P(r))B_j(P^{\mathrm{c}}(r)) - \sigma(P'(1))B_j(P'^{\mathrm{c}}(1)) - \cdots - \sigma(P'(t))B_j(P'^{\mathrm{c}}(t)), \ldots\big),$$

where we use the same notations as in (4), (5), and (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 coordinates

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

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

The runtime analysis is similar to the preceding one, but now we increase the number of guesses by a factor of 2^{g^2} for the 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 $\mathbb{R}^{d_g+g^2}$ ($d_g < 3^{2g}$ 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 paths to guess, and for each set, there are at most 2^{g^2} sign assignment guesses, 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

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

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 $< 4g^2$ guessed 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 $2g-1$ 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 an array in this order, and therefore, accessing a specific path (for some admissible pair) from the set takes $O(1)$ time.

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 guess a 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-value* 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-value* 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 $D_{i,j}$,

$$\mathrm{cost}_{i,j}(P) = V_i^{\mathrm{r}}(P, \sigma^*) - V_j^{\mathrm{c}}(P, \sigma^*). \tag{7}$$

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 guessed 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 $\kappa, \kappa' \in \left[\frac{n}{g-1}\right]$, we can retrieve, upon demand, the values $V_\kappa^{\mathrm{r}}(P, \sigma^*)$ and $V_{\kappa'}^{\mathrm{c}}(P, \sigma^*)$, and compute $\mathrm{cost}_{\kappa,\kappa'}(P)$ by using (7), in $O(1)$ time. In total, over all our guessed 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 $O(1)$ time, as well as compute $\mathrm{cost}_{i,j}(P^*_{v,w})$ 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 2.1) 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 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 positions of its left and bottom boundaries L (initialized to the same values as in the algorithm in Section 2.1), 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

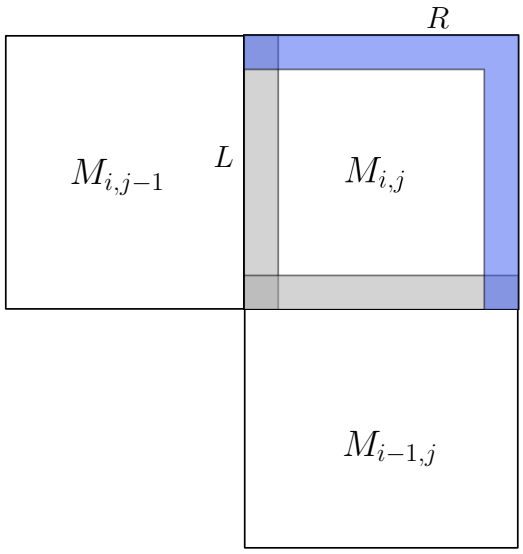

Figure 3.1: The L-boundary (shaded in gray) of box $M_{i,j}$ overlaps with the top boundary of $M_{i-1,j}$ and the right boundary of $M_{i,j-1}$. Once we have the values of M at the positions of the L-boundary of $M_{i,j}$, our algorithm computes the values of M at the positions of its R-boundary (shaded in blue).

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 $w \in R$, we have

$$M_{i,j}[w] = \min_{\substack{u\in L \\ (u,w) \text{ admissible}}} \Big\{ M_{i,j}[u] + \mathrm{cost}_{i,j}(P^*_{u,w}) \Big\}. \tag{8}$$

(Note that, by definition, the term $D_{i,j}[u]$ is included in $M_{i,j}[u]$ and not in $P^*_{u,w}$, 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 (8). 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 $M_{i,j}[v] + \mathrm{cost}_{i,j}(P^*_{v,w})$ as the *cumulative cost* of the pair (v,w), and denote it by c-cost(v,w).

We can rewrite (8), for each $w \in R$, as

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

where $M^B_{i,j}[w]$ is the minimum in (8) computed only over $u \in \{L(1),\ldots,L(g)\}$, which is the portion of L that overlaps the R-boundary of the bottom neighbor $M_{i-1,j}$ (when $i > 1$), and $M^L_{i,j}[w]$

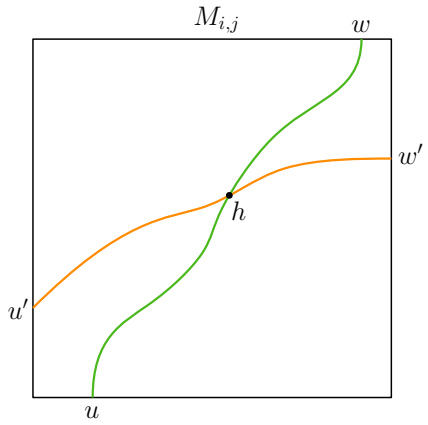

Figure 3.2: By Lemma 3.1, if (u, w) and (u', w') are minimal pairs in $M_{i,j}$, then the illustrated scenario is impossible, since the path $P^*_{u,w}$ (in green) is a portion of the shortest path from $M[0, 0]$ to $M_{i,j}[w]$, and the path $P^*_{u',w'}$ (in orange) is a portion of the shortest path 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, which contradicts the fact that both of these paths are shortest paths.

is computed over $u \in \{L(g), \ldots, L(2g-1)\}$, which overlaps the R-boundary of the left neighbor $M_{i,j-1}$ (when $j > 1$). See Figure 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 backward pointer tracing procedure, similar in principle to the one mentioned for the quadratic algorithm in Section 2.1.

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 $\text{c-cost}_{i,j}(v, w)$, 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.

Luckily, 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 3.1. *For a fixed box $D_{i,j}$, and for any two 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 3.1 asserts the so-called *Monge property* of shortest-path matrices (see, e.g., [10, 22]). See Figure 3.2 for an illustration and a sketch of a proof.

We can therefore use the following divide-and-conquer paradigm for computing the minimal pairs within a box $D_{i,j}$. We start by setting the median index $k = \lfloor |R|/2 \rfloor$ of $|R|$, and compute the minimal pair $(u, R(k))$ and c-cost$(u, R(k))$, naively, in $O(g)$ time, as explained above. The path $P^*_{u,R(k)}$ decomposes the box $D_{i,j}$ into two parts, so that one part, X, consists all the positions in $D_{i,j}$ that are (weakly) *above* $P^*_{u,R(k)}$, and the other part, Y, consists all the positions in $D_{i,j}$ 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 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 positions X and Y of L and R, respectively; there is no need to keep track of the corresponding portion of $D_{i,j}$ 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

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

It is an easy exercise to show 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 $O\left(n^2 \log \log \log n / \log \log n\right)$. This completes the proof of Theorem 1.1 for DTW. □

3.1 Extension to High-Dimensional Polyhedral Metric Spaces

The algorithm described above can be extended to work in higher dimensions $\mathbb{R}^d$, 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 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 guessing all the relevant signs, we get a sum of differences, and (the extended) Fredman's trick 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 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. Omitting further details, we get a subquadratic algorithm for DTW in such a higher-dimensional setup, with the same asymptotic time bound as that of the algorithm described above, but with the constant of proportionality depending 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 need to guess the facet of K 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 guess, $\mathsf{dist}(p_\ell, q_m)$ is a linear expression, and (the extended) Fredman's trick, with all the follow-up machinery, can then be applied, in a suitably modified but straightforward manner. 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 1.1 and as stated in Theorem 1.2, though the constant of

proportionality depends on the dimension d, and on the complexity of the unit ball K (i.e., its number of facets).

4 Geometric Edit Distance

We now show how our DTW algorithm can be extended to compute $\mathsf{ed}(A, B)$ (and optimal matching). Recall the definitions of monotone matching, $\mathsf{ed}(A, B)$, and optimal matching from Section 1. First, we overview the standard dynamic programming algorithm for computing GED between two sequences $A = (p_1, \ldots, p_n)$ and $B = (q_1, \ldots, q_n)$ of n points in $\mathbb{R}$.

The Quadratic 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. $M[\ell, 0] := \rho \cdot \ell$, $M[0, \ell] := \rho \cdot \ell$.

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", we obtain the optimal matching.

Recall the all-pairs-distances matrix D and its decomposition into boxes $D_{i,j}$, as defined in Section 3. 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 it equals to the cost of its corresponding monotone matching $\mathcal{M}(P)$ (see below how $\mathcal{M}(P)$ is defined).

We view D as a weighted directed grid graph G, whose vertices are the pairs $[n+1] \times [n+1]$, and its set of edges is

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

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 $|p_\ell - q_m|$. Each staircase path P in D is then a path in the graph G, whose corresponding monotone matching $\mathcal{M}(P)$ consists exactly all the pairs of points (p_ℓ, q_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 the cost of P in D to be the weight of its corresponding path in G, we obtain that the cost of P equals to the cost of $\mathcal{M}(P)$, and the corresponding dynamic programming matrix M is such 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). This implies that Lemma 3.1 holds in this setup too, and that once we have a corresponding data structure from the preprocessing procedure, we can apply the second stage of the DTW verbatim.

As for the preprocessing procedure, the cost of a staircase path is now a sum of distances $|p_\ell - q_m|$ plus a multiple of the parameter ρ. Since ρ is a fixed real number, we can guess all the relevant signs as before, get a linear expression in p_ℓ and q_m, and (the extended) Fredman's trick (3) can then be applied when comparing the costs of two staircase paths. (Our algorithm works also for more general gap penalty functions, as long as it is linear in the coordinates of the points $A \cup B$.) The rest of the preprocessing procedure and the extension to high-dimensional polyhedral metric spaces are similar to those for DTW.

References

[1] A. Abboud, A. Backurs, and V. V. Williams. Quadratic-time hardness of LCS and other sequence similarity measures. *CoRR*, abs/1501.07053, 2015.

[2] 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 *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing*, pages 375–388, 2016.

[3] 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.

[4] P. K. Agarwal, K. Fox, J. Pan, and R. Ying. Approximating dynamic time warping and edit distance for a pair of point sequences. In *Proceedings of the 32nd International Symposium on Computational Geometry, SoCG*, pages 6:1–6:16, 2016.

[5] 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.

[6] A. Backurs and P. Indyk. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). In *Proceedings of the 47th Annual ACM on Symposium on Theory of Computing, STOC*, pages 51–58, 2015.

[7] P. Bille and M. Farach-Colton. Fast and compact regular expression matching. *Theoretical Computer Science*, 409(3):486–496, 2008.

[8] K. Bringmann. Why walking the dog takes time: Fréchet distance has no strongly subquadratic algorithms unless SETH fails. In *Proceedings of the 55th IEEE Annual Symposium on Foundations of Computer Science, FOCS*, pages 661–670, 2014.

[9] K. Bringmann and M. Künnemann. Quadratic conditional lower bounds for string problems and dynamic time warping. In *Proceedings of the 56th Annual IEEE Symposium on Foundations of Computer Science, FOCS*, pages 79–97, 2015.

[10] R. E. Burkard, B. Klinz, and R. Rudolf. Perspectives of Monge properties in optimization. *Discrete Applied Mathematics*, 70(2):95–161, 1996.

[11] 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.

[12] T. M. Chan. All-pairs shortest paths with real weights in $O(n^3/\log n)$ time. *Algorithmica*, 50(2):236–243, 2008.

[13] 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.

[14] 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.

[15] 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.

[16] M. L. Fredman. How good is the information theory bound in sorting? *Theor. Comput. Sci*, 1(4):355–361, 1976.

[17] M. L. Fredman. New bounds on the complexity of the shortest path problem. *SIAM J. Comput.*, 5(1):83–89, 1976.

[18] O. Gold and M. Sharir. Improved bounds for 3SUM, k-SUM, and linear degeneracy. *CoRR*, abs/1512.05279, 2015.

[19] A. Grønlund and S. Pettie. Threesomes, degenerates, and love triangles. In *Proceedings 55th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 621–630, 2014.

[20] R. Impagliazzo and R. Paturi. On the complexity of k-SAT. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001.

[21] R. Impagliazzo, R. Paturi, and F. Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001.

[22] H. Kaplan, S. Mozes, Y. Nussbaum, and M. Sharir. Submatrix maximum queries in Monge matrices and Monge partial matrices, and their applications. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*, pages 338–355, 2012.

[23] E. Keogh and A. C. Ratanamahatana. Exact indexing of dynamic time warping. *Knowledge and Information Systems*, 7(3):358–386, 2005.

[24] E. J. Keogh and M. J. Pazzani. *Scaling up Dynamic Time Warping to Massive Datasets*, pages 1–11. Springer Berlin-Heidelberg, 1999.

[25] E. J. Keogh and M. J. Pazzani. Scaling up dynamic time warping for datamining applications. In *Proceedings of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 285–289, 2000.

[26] T. G. Kongming Wang. Alignment of curves by dynamic time warping. *Annals of Statistics*, 25(3):1251–1276, 1997.

[27] 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.

[28] M. Müller. *Information Retrieval for Music and Motion*, pages 69–84. Springer Berlin-Heidelberg, 2007.

[29] F. P. Preparata and M. I. Shamos. *Computational Geometry*. Springer, New York, NY, 1985.

[30] C. A. Ratanamahatana and E. Keogh. Three myths about dynamic time warping data mining. In *Proceedings of the 2005 SIAM International Conference on Data Mining*, pages 506–510, 2005.

[31] T. K. Vintsyuk. Speech discrimination by dynamic programming. *Cybernetics*, 4(1):52–57, 1968.

[32] 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.