Approximating Dynamic Time Warping and Edit Distance for a Pair of Point Sequences *

Pankaj K. Agarwal, Kyle Fox, Jiangwei Pan, and Rex Ying

Department of Computer Science, Duke University
[pankaj,kylefox,jwpan]@cs.duke.edu, zhitao.ying@duke.edu

Abstract

We give the first subquadratic-time approximation schemes for dynamic time warping (DTW) and edit distance (ED) of several natural families of point sequences in $\mathbb{R}^d$, for any fixed $d \geq 1$. In particular, our algorithms compute $(1 + \varepsilon)$-approximations of DTW and ED in time near-linear for point sequences drawn from k-packed or k-bounded curves, and subquadratic for backbone sequences. Roughly speaking, a curve is κ-packed if the length of its intersection with any ball of radius r is at most $\kappa \cdot r$, and a curve is κ-bounded if the sub-curve between two curve points does not go too far from the two points compared to the distance between the two points. In backbone sequences, consecutive points are spaced at approximately equal distances apart, and no two points lie very close together. Recent results suggest that a subquadratic algorithm for DTW or ED is unlikely for an arbitrary pair of point sequences even for $d = 1$.

Our algorithms work by constructing a small set of rectangular regions that cover the entries of the dynamic programming table commonly used for these distance measures. The weights of entries inside each rectangle are roughly the same, so we are able to use efficient procedures to approximately compute the cheapest paths through these rectangles.

1998 ACM Subject Classification F.2.2 [Nonnumerical Algorithms and Problems] Geometrical problems and computations

Keywords and phrases Dynamic time warping, Edit distance, Near-linear-time algorithm, Dynamic programming, Well-separated pair decomposition

Digital Object Identifier 10.4230/LIPIcs.xxx.yyy.p

1 Introduction

Motivation. Trajectories are functions from a time interval to $\mathbb{R}^d$, for $d \geq 1$, and they describe how physical systems change over time. Trajectories are being recorded and inferred from numerous sources and are often represented as ordered sequences of points. These sources include GPS sensors in smart phones and vehicles, surveillance videos, shape-based touch screen authentication patterns, hurricane patterns, and time series data. A fundamental task for analyzing trajectory data is to measure the similarity between trajectories. For example, computing trajectory similarity is an important step in object segmentation from video trajectories [14], smart phone authentication using touch screen trajectories [17], and stock price prediction using historical patterns [3]. In many applications, it is not enough to merely *quantify* how similar pairs of trajectories are; we need to compute correspondences between their sample points as well. These correspondences represent shared structures between trajectories, which can be present not only in trajectories with physical constraint such as vehicle trajectories, but also in trajectories representing the movement of pedestrians [23] or hurricanes [30].

Conference title on which this volume is based on.
Editors: Billy Editor and Bill Editors; pp. 1–16

Having an effective way to identify similar portions between a pair of trajectories can greatly aid in identifying and understanding these shared structures.

Problem statement. Let $P = \langle p_1, \ldots, p_m \rangle$ and $Q = \langle q_1, \ldots, q_n \rangle$ be two sequences of points in $\mathbb{R}^d$ for some fixed $d \geq 1$. We define a *correspondence* as a pair (p_i, q_j). A set C of correspondences is *monotone* if for any pair of correspondences $(p_i, q_j), (p_{i'}, q_{j'})$ with $i' \geq i$ we have $j' \geq j$. We define the cost of C to be $\sum_{(p,q) \in C} ||pq||$, where $|| \cdot ||$ is the Euclidean norm. The similar portions of P and Q are represented by a set C of monotone correspondences, with the cost of C quantifying the quality of similarity. The goal is to compute a monotone set of correspondences with certain properties. While numerous criteria for computing correspondences have been proposed, we focus on two, which are widely used: dynamic time warping (DTW) and edit distance (ED). They are used for matching various types of sequences such as speech signals, DNA and protein sequences, protein backbones, time-series data, GPS/video trajectories, touch screen authentication trajectories, etc [15, 17, 19, 20, 22, 26, 27, 29, 31–33].

DTW computes a monotone set of correspondences in which every point in P and Q appears at least once, and minimizes the sum of distances of corresponding pairs of points. Formally, the cost of DTW, denoted by $\mathsf{dtw}(P, Q)$, is $\mathsf{dtw}(P, Q) = \min_C \sum_{(p,q) \in C} ||pq||$, where the minimum is taken over all sets C of monotone correspondences that cover all points of P and Q. DTW allows a point to appear in multiple correspondences, so it matches two sequences effectively even if the sampling rates are different.

Edit distance (also called Levenshtein distance) seeks a monotone *matching* on the points in P and Q of minimum cost; each point in P corresponds to at most one point in Q and vice versa. It also adds an additional constant *gap* penalty g for each point in $P \cup Q$ that does not appear in any correspondence. Formally, the cost of ED, denoted by $\mathsf{ed}(P, Q)$, is $\mathsf{ed}(P, Q) = \min_C \sum_{(p,q) \in C} ||pq|| + g(m + n - 2|C|)$, where the minimum is taken over all sets C of monotone matchings in the complete bipartite graph $P \times Q$. More sophisticated gap penalty functions have been proposed, but we focus on the simple linear gap penalty function. By tuning g correctly, meaningful correspondences can be computed even when faced with outlier points that arise from measurement errors or short deviations in otherwise similar trajectories.

Given a parameter $\varepsilon \in (0, 1)$, we wish to develop efficient $(1 + \varepsilon)$-approximation algorithms for computing $\mathsf{dtw}(P, Q)$ and $\mathsf{ed}(P, Q)$, i.e., they return a value Δ such that $\mathsf{dtw}(P, Q) \leq \Delta \leq (1 + \varepsilon)\mathsf{dtw}(P, Q)$ or $\mathsf{ed}(P, Q) \leq \Delta \leq (1 + \varepsilon)\mathsf{ed}(P, Q)$. We are also interested in computing correspondences that realize these distances.

Prior results. It is well-known that both $\mathsf{dtw}(P, Q)$ and $\mathsf{ed}(P, Q)$, as well as the relevant correspondences, can be computed in $O(mn)$ time using dynamic programming [28]. A series of recent papers show that there exists no algorithm for computing $\mathsf{dtw}(P, Q)$ or $\mathsf{ed}(P, Q)$ in time $O(n^{2-\delta})$ for any $\delta > 0$ unless the Strong Exponential Time Hypothesis (SETH) of Impagliazzo and Paturi [25] is false. In particular, Backurs and Indyk [9] showed a lower bound for edit distance, and Abboud *et al.* [4] and Bringmann and Künnemann [12] independently showed lower bounds for DTW. While most of these lower bounds were presented for the string versions of their respective problems, the DTW lower bound of Bringmann and Künnemann uses sequences of points in $\mathbb{R}$. Unless SETH is false, there exists no strictly subquadratic time algorithm for DTW, even in our setting of point sequences in $\mathbb{R}^d$. Similar conditional lower bounds have been shown for other distance and similarity problems [4, 5, 9, 10, 12]. Some of these results suggest that even strongly subquadratic approximation schemes seem unlikely [4, 10].

In view of the recent lower bounds, a natural question to ask is whether near-linear, or even subquadratic, algorithms exist for certain natural families of sequences. Aronov *et al.* [8] gave subquadratic-time approximation schemes for the discrete Fréchet distance of κ-*bounded* and *backbone* point sequences. Discrete Fréchet distance is similar to DTW except that one uses max instead

of sum in the definition. Restricting themselves to these families of sequences allowed them to subvert the hardness result of Bringmann [10] mentioned above. Driemel *et al.* [18] extended the approximation results to the continuous Fréchet distance and the even more natural family of κ-*packed* curves (see below for definitions of these families of curves); see also Bringmann and Künnemann [11] and Bringmann and Mulzer [13]. We note that while (discrete) Fréchet distance is a reasonable measure to compute the similarity between two sequences, it is not effective in identifying similar portions of the sequences. Roughly speaking, these algorithms guess the value of (discrete) Fréchet distance, say, Δ, simplify the two sequences within $\frac{\varepsilon}{2}\Delta$ error, and show that near-linear number of entries in the dynamic-programming table need to be computed by observing that each point p in one sequence can be matched with a point in the other (simplified) sequence that lies within distance Δ from p.

Currently, no subquadratic-time approximation results are known for DTW, although there are a number of heuristics designed to speed up its exact computation in practice; see Wang *et al.* [34]. Subquadratic-time approximation algorithms are known for variants of edit distance, but all of these algorithm have at least a polylogarithmic approximation ratio [7]. It is not clear how to extend the above algorithms for (discrete) Fréchet distance to DTW or ED because a point p in P may be matched to a point in Q that is far away from p and thus their idea of computing a small number of entries of the table does not work.

Our results. We present algorithms for computing $\mathsf{dtw}(P, Q)$ and $\mathsf{ed}(P, Q)$ approximately which have subquadratic running time for several "well-behaved" families of input sequences. The correspondences realizing these distances can also be recovered. The two algorithms are almost identical except a few implementation details. In the worst case, their running time is quadratic for arbitrary point sequences, but it is near-linear if P and Q are κ-packed or κ-bounded sequences and subquadratic when P and Q satisfy the conditions for a backbone sequence. These are the first approximation algorithms that compute $\mathsf{dtw}(P, Q)$ and $\mathsf{ed}(P, Q)$ for such point sequences in subquadratic time.

For $x \in \mathbb{R}^d$ and $r \in \mathbb{R}^+$, let $\mathsf{B}(x, r)$ denote the ball of radius r centered at x. Given $\kappa \in \mathbb{R}^+$, a curve γ in $\mathbb{R}^d$ is κ-*packed* if the length of γ inside any ball of radius r is bounded by κr [18], and γ is κ-*bounded* if for any $0 \le t < t' \le 1$, $\gamma[t : t'] \subseteq \mathsf{B}\left(\gamma(t), \frac{\kappa}{2}||\gamma(t)\gamma(t')||\right) \cup \mathsf{B}\left(\gamma(t'), \frac{\kappa}{2}||\gamma(t)\gamma(t')||\right)$, where $\gamma : [0, 1] \to \mathbb{R}^d$ and $\gamma[t : t']$ is the portion of γ between $\gamma(t)$ and $\gamma(t')$ [6]. We say a point sequence P is κ-packed (resp. κ-bounded) if the polygonal curve that connects points of P is κ-packed (resp. κ-bounded). A point sequence $P = \langle p_1, \ldots, p_m \rangle$ is said to be a backbone sequence if it satisfies the following two conditions: (i) for any pair of non-consecutive integers $i, j \in [1, m]$, $||p_i p_j|| > 1$; (ii) for any integer i in $(1, m]$, $c_1 \le ||p_{i-1} p_i|| \le c_2$, where c_1, c_2 are positive constants [8]. These sequences are commonly used to model protein backbones where each vertex represents a C_α atom, connected to its neighbors via covalent bonds. See Figure 1(a), (b) for examples of κ-packed and κ-packed curves and (c) for an example of backbone sequence. We use γ_P to denote the polygonal curve connecting the points of sequence P. Our results are summarized in the following theorems.

▶ **Theorem 1.** *Let P and Q be two point sequences of length at most n in $\mathbb{R}^d$, and let $\varepsilon \in (0, 1)$ be a parameter. An $(1 + \varepsilon)$-approximate value of $\mathsf{dtw}(P, Q)$ can be computed in $O(\frac{\kappa}{\varepsilon} n \log n)$, $O(\frac{\kappa^d}{\varepsilon^d} n \log n)$, and $O(\frac{1}{\varepsilon} n^{2-1/d} \log n)$ time if P, Q are κ-packed, κ-bounded, and backbone sequences, respectively.*

▶ **Theorem 2.** *Let P and Q be two point sequences of length at most n in $\mathbb{R}^d$, and let $\varepsilon \in (0, 1)$ be a parameter. An $(1 + \varepsilon)$-approximate value of $\mathsf{ed}(P, Q)$ can be computed in $O(\frac{\kappa}{\varepsilon} n \log n)$, $O(\frac{\kappa^d}{\varepsilon^d} n \log n)$, and $O(\frac{1}{\varepsilon} n^{2-1/(d+1)} \log n)$ time if P, Q are κ-packed, κ-bounded, and backbone sequences, respectively.*

Recall that the standard dynamic programming algorithm for computing $\mathsf{dtw}(P, Q)$ or $\mathsf{ed}(P, Q)$ constructs a weighted grid $\mathsf{V} = \{(i, j) \mid 1 \le i \le m, 1 \le j \le n\}$ and formulates the problem as
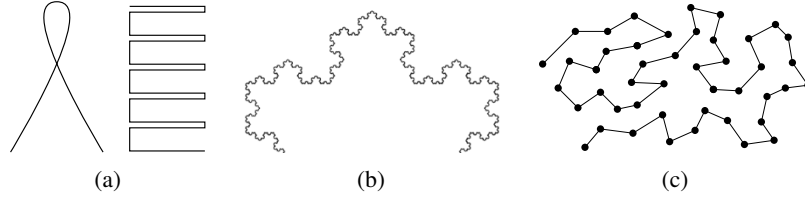
■ **Figure 1.** (a) κ-packed curves that are not κ-bounded. (b) The top half of the Koch snowflake is a κ-bounded curve that is not κ-packed. (c) A backbone sequence that is neither κ-bounded nor κ-packed.

computing a minimum-weight path from $(1, 1)$ to (m, n). Based on the observation that nearby grid points may have similar weights when P, Q are "well-behaved", our main idea is to construct a small number of potentially overlapping rectangular regions of V, whose union contains the minimum-weight path in V, such that all grid points within each rectangle have similar weights. We show how to construct the rectangles so that the number of "boundary points" of the rectangles is near linear when P, Q are κ-packed or κ-bounded and subquadratic when P, Q are backbone sequences. We then describe an efficient procedure to compute approximate minimum-weight paths from $(1, 1)$ to all boundary points.

The algorithm framework is quite general and can work for a variety of similar distance measures based on monotone correspondences. For example, our results immediately generalize to variants of dynamic time warping and edit distance that use the k-th powers of distance between points instead of their Euclidean distance for any constant $k > 0$. Moreover, the framework may prove useful in designing subquadratic-time algorithms for other problems that can be solved with standard dynamic programming.

2   Algorithm for DTW

Let $P = \langle p_1, \ldots, p_m \rangle$ and $Q = \langle q_1, \ldots, q_n \rangle$ be two point sequences in $\mathbb{R}^d$, and let $\varepsilon \in (0, 1)$ be a parameter. We present a $(1 + \varepsilon)$-approximation algorithm for computing $\mathrm{dtw}(P, Q)$, which takes subquadratic time if P, Q are "well-behaved". Without loss of generality, assume that $m \le n$ and $\varepsilon \ge 1/n$. If $\varepsilon < 1/n$, we can simply compute $\mathrm{dtw}(P, Q)$ in $O(mn) = O(n/\varepsilon)$ time via dynamic programming.

Given positive integers $i < i'$, let $[i : i'] := \{i, i + 1, \ldots, i'\}$, and let $[i] := [1 : i]$. Let $\mathsf{V} = [m] \times [n]$ denote a set of grid points[1] in $\mathbb{R}^2$, and define a weight function $\omega : \mathsf{V} \to \mathbb{R}_{\ge 0}$ where $\omega(i, j)$ is the Euclidean distance between p_i and q_j. Two different grid points in V are said to be *neighboring* if they differ by at most 1 in each coordinate. We say (i, j) dominates (i', j') if $i \ge i'$ and $j \ge j'$. A path π is a sequence of neighboring grid points; π is *admissible* if it is non-decreasing in both coordinates. Define the weight of the path π, $\omega(\pi)$, as the sum of the weights of the grid points along π. Define $\mu(i, j)$ as the minimum weight of an admissible path from $(1, 1)$ to (i, j). So $\mathrm{dtw}(P, Q) = \mu(m, n)$ and $\mu(m, n)$ can be computed in $O(n^2)$ time via dynamic programming.

For $1 \le i_1 \le i_2 \le m$ and for $1 \le j_1 \le j_2 \le n$, the set of grid points $[i_1 : i_2] \times [j_1 : j_2]$ is called a *rectangle*. A point $(i, j) \in \mathsf{V}$ is a *boundary point* of this rectangle if $i \in \{i_1, i_2\}$ or $j \in \{j_1, j_2\}$. We first outline the algorithm for computing an $(1 + \varepsilon)$-approximate value of $\mu(m, n)$, and then describe it in detail in Sections 2.1-2.3. Section 2.4 analyzes its running time for well-behaved point sequences.

(i) Compute an estimate $\underline{\mathsf{d}}$ of $\mathrm{dtw}(P, Q)$ such that $\underline{\mathsf{d}} \le \mathrm{dtw}(P, Q) \le 4n\underline{\mathsf{d}}$. Let $\bar{\mathsf{d}} = 4n\underline{\mathsf{d}}$.

(ii) Compute a set $\mathcal{R}$ of (possibly overlapping) rectangles and a weight ω_{R} for each rectangle $\mathsf{R} \in \mathcal{R}$, such that:

---

[1] Note that in this paper, a point can refer to either a grid point in V or a sequence point from $P \cup Q$.
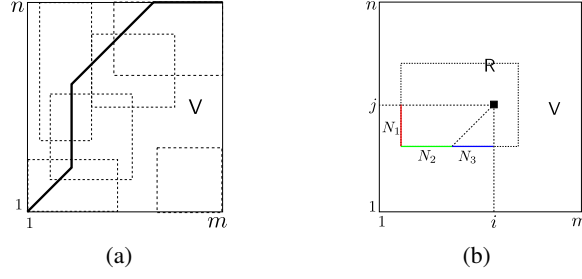
Figure 2. (a) Illustration of our algorithm: compute a small set of rectangles that covers the optimal admissible path from $(1,1)$ to (m,n) (drawn in bold); (b) partitioning the boundary points of rectangle R dominated by (i,j).

(a) for all R $\in \mathcal{R}$ and $(i,j) \in$ R, $|\omega(i,j) - \omega_\mathsf{R}| \leq \frac{\varepsilon}{2} \max\{\omega_\mathsf{R}, \underline{\mathsf{d}}/2n\}$,

(b) if $(i,j) \in$ V and $\omega(i,j) \leq \overline{\mathsf{d}}$, then there exists rectangle R $\in \mathcal{R}$ such that $(i,j) \in$ R.

Conditions (a) and (b) above ensure that the weights of grid points in each rectangle are roughly the same, and the minimum-weight admissible path between $(1,1)$ and (m,n) is contained in the union of the rectangles. See Figure 2(a).

Let $\mathcal{B} = \bigcup_{\mathsf{R} \in \mathcal{R}} \partial \mathsf{R}$ be the set of boundary points of the rectangles in $\mathcal{R}$. The sizes of $\mathcal{R}$ and $\mathcal{B}$ depend on the input sequences P, Q. In the worst case $|\mathcal{R}|, |\mathcal{B}| = \Theta(mn)$, but they are subquadratic if P, Q are well-behaved.

(iii) For every $(i,j) \in \mathcal{B}$, compute a $(1+\varepsilon)$-approximate value $\tilde{\mu}(i,j)$ of $\mu(i,j)$, i.e., $\mu(i,j) \leq \tilde{\mu}(i,j) \leq (1+\varepsilon)\mu(i,j)$.

(iv) Return $\tilde{\mu}(m,n)$.

2.1 An $O(n)$ approximation

Let $\mathsf{dfr}(P,Q)$ denote the discrete Fréchet distance between P and Q, i.e., replace sum with max in the definition of $\mathsf{dtw}(P,Q)$.

▶ **Lemma 3.** $\mathsf{dfr}(P,Q) \leq \mathsf{dtw}(P,Q) \leq 2n \cdot \mathsf{dfr}(P,Q)$

Proof: Let π^* be the minimum-weight admissible path from $(1,1)$ to (m,n) corresponding to $\mathsf{dtw}(P,Q)$. Then

$$\mathsf{dfr}(P,Q) \leq \max_{(i,j)\in\pi^*} ||p_i q_j|| \leq \sum_{(i,j)\in\pi^*} ||p_i q_j|| = \mathsf{dtw}(P,Q).$$

Similarly, let $\overline{\pi}$ be the admissible path corresponding to $\mathsf{dfr}(P,Q)$. Then

$$\mathsf{dtw}(P,Q) \leq \sum_{(i,j)\in\overline{\pi}} ||p_i q_j|| \leq 2n \max_{(i,j)\in\overline{\pi}} ||p_i q_j|| = 2n\mathsf{dfr}(P,Q).$$

The second inequality follows because $|\overline{\pi} \cap \mathsf{V}| \leq m + n \leq 2n$.   □

Aronov *et al.* [8] gave a near-linear time algorithm for computing the approximate discrete Fréchet distance between κ-bounded point sequences. Their algorithm directly implies an $O(\kappa^d n \log n)$-time 2-approximation algorithm for computing $\mathsf{dfr}(P,Q)$ for κ-bounded sequences. They also prove that the same algorithm computes the 2-approximation of the discrete Fréchet distance between backbone sequences in $O(n^{2-2/d})$ time. With a slight modification of their analysis, it turns out that their algorithm also works for κ-packed sequences. The details can be found in the Appendix.

▶ **Lemma 4** ([8]). *A 2-approximation of $\mathsf{dfr}(P,Q)$ can be computed in time $O(\kappa n \log n)$, $O(\kappa^d n \log n)$, and $O(n^{2-2/d})$ if P, Q are κ-packed, κ-bounded, and backbone sequences, respectively.*

Let $\overline{\mathsf{dfr}}(P, Q)$ be the 2-approximate discrete Fréchet distance computed using Lemma 4, i.e., $\mathsf{dfr}(P, Q) \leq \overline{\mathsf{dfr}}(P, Q) \leq 2 \cdot \mathsf{dfr}(P, Q)$. Set $\underline{\mathsf{d}} = \overline{\mathsf{dfr}}(P, Q)/2$. By Lemma 3, $\underline{\mathsf{d}} \leq \mathsf{dtw}(P, Q) \leq 4n\underline{\mathsf{d}}$.

2.2 Computing rectangles $\mathcal{R}$

Let H be an axis-aligned hypercube in $\mathbb{R}^d$ that contains $P \cup Q$. Let $\mathcal{T}$ be a *quadtree*, a 2^d-way tree, on $P \cup Q$. Each node v of $\mathcal{T}$ is associated with an axis-aligned box $\square_v$. The root of $\mathcal{T}$ is associated with H. A node v is a leaf if $|\square_v \cap (P \cup Q)| \leq 1$. The boxes associated with the children of a node v are obtained by partitioning $\square_v$ into 2^d congruent hypercubes — the side length of each resulting box is half that of $\square_v$. For a node $v \in \mathcal{T}$, let $p(v)$ denote its parent, $\mathsf{ch}(v)$ the set of children of v, $\Delta(v)$ the side length of $\square_v$, $P_v = P \cap \square_v$, and $Q_v = Q \cap \square_v$. Let $m_v = |P_v|$ and $n_v = |Q_v|$. For two nodes $u, v \in \mathcal{T}$, let $\mathsf{d}_\square(u, v) = \min_{p,q \in \square_u \times \square_v} ||pq||$ denote the distance between $\square_u$ and $\square_v$. Two nodes u, v are said to be *neighboring* if u and v are in the same level of $\mathcal{T}$ and $\square_u$ and $\square_v$ share a facet. We do not construct the entire $\mathcal{T}$ but only a portion as described below.

Without loss of generality, suppose the side length of H is a power of 2. Let $\underline{r}$ and $\bar{r}$ be powers of 2 such that $\underline{r} \leq \varepsilon \underline{\mathsf{d}}/2n \leq 2\underline{r}$ and $\bar{r} \leq 4\overline{\mathsf{d}} \leq 2\bar{r}$. We call a node v of $\mathcal{T}$ *active* if $\Delta(v) \in [\underline{r}, \bar{r}]$ and $m_v + n_v > 0$. Let A denote the set of active nodes of $\mathcal{T}$. We construct the set A of active nodes of $\mathcal{T}$ and the sets P_v, Q_v for each active node $v \in A$. By definition, the active nodes lie in a portion of the quadtree $\mathcal{T}$ of height $O(\log(\bar{r}/\underline{r})) = O(\log n)$. Thus, $|A| = O(n)$ and $\sum_{v \in A}(m_v + n_v) = O(n \log n)$. Computing A and P_v, Q_v for all $v \in A$ takes $O(n \log n)$ time.

To compute the rectangles in $\mathcal{R}$, we first construct a family $\mathcal{F} = \{(u_1, v_1), \ldots, (u_s, v_s)\}$ of "well-separated" pairs of active nodes with the following properties:

(P1) For every $t \leq s$, $\max\{\Delta(u_t), \Delta(v_t)\} \leq \frac{\varepsilon}{16} \max\{\mathsf{d}_\square(u_t, v_t), \underline{\mathsf{d}}/2n\}$.

(P2) For all pairs $(i, j) \in \mathsf{V}$ with $||p_i q_j|| \leq \overline{\mathsf{d}}$, there exists a unique pair (u_t, v_t) such that $p_i \in P_{u_t}$ and $q_j \in Q_{v_t}$.

Intuitively, (u, v) is well-separated when for any $p \in u$ and $q \in v$, we have $||pq|| \approx \mathsf{d}_\square(u, v)$. Then, for each pair $(u_t, v_t) \in \mathcal{F}$, we construct a small number of rectangles.

Constructing $\mathcal{F}$. The properties (P1) and (P2) are similar to those for the so-called *well-separated pair decomposition* (WSPD) of a point set, introduced by Callahan and Kosaraju [16] (see also Har-Peled [24]). We therefore adapt their algorithm. We first describe a recursive procedure $\textsc{Pairing}(u, v)$, where u, v are two active nodes, which generates a family of pairs for P_u, Q_v.

$\textsc{Pairing}(u, v)$
if $\max\{\Delta(u), \Delta(v)\} \leq \frac{\varepsilon}{16} \max\{\mathsf{d}_\square(u, v), \underline{\mathsf{d}}/2n\}$
    add (u, v) to $\mathcal{F}$; return
if $\Delta(u) \geq \Delta(v)$, **then**
    $\forall w \in \mathsf{ch}(u)$ if $P_w \neq \varnothing$, do $\textsc{Pairing}(w, v)$
    else $\forall z \in \mathsf{ch}(v)$ if $Q_z \neq \varnothing$, do $\textsc{Pairing}(u, z)$


Let u_0 be a top-level active node with $\Delta(u_0) = \bar{r}$ and $P_{u_0} \neq \varnothing$. We call $\textsc{Pairing}(u_0, v_0)$ if $Q_{v_0} \neq \varnothing$ and either $v_0 = u_0$ or v_0 is a neighboring node of u_0.

(P1) is obviously true by the termination condition of the $\textsc{Pairing}$ procedure. (P2) is true because for each $(i, j) \in \mathsf{V}$ with $||p_i q_j|| \leq \overline{\mathsf{d}}$, it must be that p_i and q_j are contained in either the same active node or two neighboring active nodes of side length $\bar{r}$. The stopping criterion ensures that the $\textsc{Pairing}$ procedure never visits a node v with $\Delta(v) < \underline{r}$.

By adapting the analysis of the WSPD algorithm, the following lemma can be proven (see the appendix for the proof).

▶ **Lemma 5.** *If $(u, v) \in \mathcal{F}$, then (i) $\max\{\Delta(u), \Delta(v)\} \leq \min\{\Delta(p(u)), \Delta(p(v))\}$; (ii) $\Delta(u)/2 \leq \Delta(v) \leq 2\Delta(u)$; and (iii) there is a constant $c \geq 0$ such that $\mathsf{d}_\square(u, v) \leq \frac{c}{\varepsilon}\Delta(u)$.*
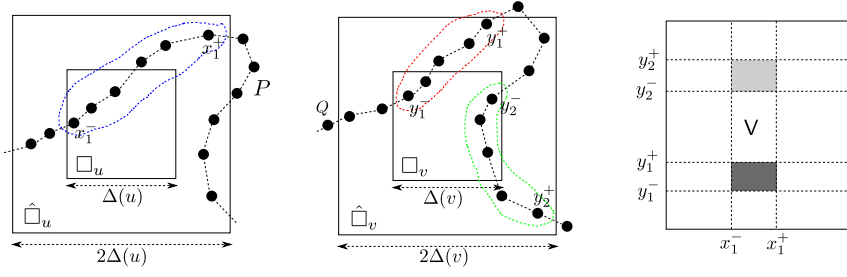
Figure 3. One MCS in $\hat{\Box}_u$ (left) and two MCSs in $\hat{\Box}_v$ (middle). Together, two rectangles R_{11}, R_{12} (right) are created (shaded areas).

Constructing $\mathcal{R}$. We describe how to construct rectangles from each well-separated pair $(u, v) \in \mathcal{F}$. Let $\hat{\Box}$ denote the concentric box to $\Box$ with twice the side length of $\Box$. The algorithm starts from the first unvisited point of P_u and walks along P until P exits $\hat{\Box}_u$; it then repeats this walk by jumping to the next point of P_u; this process stops when all points of P_u have been visited. Each walk corresponds to a maximal contiguous subsequence (MCS) of P in $\hat{\Box}_u$ with the first point inside $\Box_u$. Let $S_u(P) = \{[x_1^- : x_1^+], \ldots, [x_{\alpha_u}^- : x_{\alpha_u}^+]\}$ denote the MCSs as constructed above. Similarly, we compute $S_v(Q) = \{[y_1^- : y_1^+], \ldots, [y_{\beta_v}^- : y_{\beta_v}^+]\}$ denoting the MCSs of Q in $\hat{\Box}_v$. For every pair $a \in [\alpha_u], b \in [\beta_v]$, we define the rectangle $R_{ab} = [x_a^- : x_a^+] \times [y_b^- : y_b^+]$ and set its weight $\omega_{R_{ab}} = \mathsf{d}_\Box(u, v)$. Set $\mathcal{R}_{uv} = \{R_{ab} \mid a \in [\alpha_u], b \in [\beta_v]\}$ and $\mathcal{R} = \bigcup_{(u,v) \in \mathcal{F}} \mathcal{R}_{uv}$. See Figure 3 for an illustration of the MCSs of $S_u(P), S_v(Q)$, and the rectangles in $\mathcal{R}_{uv}$.

Remark. The rectangles in $\mathcal{R}_{uv}$ cover all the grid points corresponding to $P_u \times Q_v$, i.e., if $(p_i, q_j) \in P_u \times Q_v$ then $(i, j) \in \bigcup \mathcal{R}_{uv}$. Since $\bigcup \mathcal{R}_{uv}$ may also contain grid points that correspond to pairs in $(P \cap (\hat{\Box}_u \setminus \Box_u)) \times (Q \cap (\hat{\Box}_v \setminus \Box_v))$, a grid point may lie in multiple rectangles of $\mathcal{R}$, implying that the rectangles in $\mathcal{R}$ may overlap. Had we defined $S_u(P), S_v(Q)$ to be MCSs of P_u and Q_v respectively, the rectangles would have been disjoint, but we might have ended up creating $\Omega(n^2)$ rectangles in the worst case. As we will prove in Section 2.4, by allowing $\mathcal{R}_{uv}$ to cover extra points, we keep the size of $\mathcal{R}$ and $\mathcal{B}$ small.

We show that the set of rectangles $\mathcal{R}$ satisfies the conditions in step (ii) of the algorithm.

▶ **Lemma 6.** $\mathcal{R}$ *satisfies the following properties:*

(i) For all $R \in \mathcal{R}$ and for all $(i, j) \in R$, $|\omega(i, j) - \omega_R| \leq \frac{\varepsilon}{2} \max\{\omega_R, \underline{\mathsf{d}}/2n\}$.
(ii) If $(i, j) \in V$ and $\omega(i, j) \leq \overline{\mathsf{d}}$, then there exists a rectangle $R \in \mathcal{R}$ such that $(i, j) \in R$.

Proof: (i) Suppose R is constructed from some well-separated pair $(u_t, v_t) \in \mathcal{F}$. By construction, if $(i, j) \in R$, then $p_i \in \hat{\Box}_{u_t}$ and $q_j \in \hat{\Box}_{v_t}$. Therefore, $\omega(i, j) = ||p_i q_j|| \leq \mathsf{d}_\Box(u_t, v_t) + 2\sqrt{2}(\Delta(u_t) + \Delta(v_t))$. By property (P1) and $\omega_R = \mathsf{d}_\Box(u_t, v_t)$, we have $\omega(i, j) \leq \omega_R + 4\sqrt{2}\frac{\varepsilon}{16} \max\{\omega_R, \underline{\mathsf{d}}/2n\} \leq \omega_R + \frac{\varepsilon}{2} \max\{\omega_R, \underline{\mathsf{d}}/2n\}$. Similarly, we can prove $\omega(i, j) \geq \omega_R - \frac{\varepsilon}{2} \max\{\omega_R, \underline{\mathsf{d}}/2n\}$. (ii) By property (P2), there must exist a pair $(u, v) \in \mathcal{F}$ such that $p_i \in P_u$ and $q_j \in Q_v$. Since $\bigcup \mathcal{R}_{uv}$ "covers" $P_u \times Q_v$, there is a rectangle $R \in \mathcal{R}_{uv}$ that contains the grid point (i, j). ☐

The time taken to construct the set $\mathcal{R}$ is $O(|\mathcal{R}|)$ plus the time taken to generate $\mathcal{F}$. We bound the latter quantity in Section 2.4.

2.3 Computing admissible paths

We now describe an algorithm that for each $(i, j) \in \mathcal{B}$ computes $(1 + \varepsilon)$-approximate value $\tilde{\mu}(i, j)$ of $\mu(i, j)$ in amortized constant time.

We say a point $(i, j) \in V$ *hits* a rectangle $R = [i_1 : i_2] \times [j_1 : j_2]$ if $i_1 < i \leq i_2$ and $j_1 < j \leq j_2$, i.e., $(i, j) \in R$ but not on its left or bottom boundary. The algorithm sets $\tilde{\mu}(1, 1) = \omega(1, 1)$, and

processes the points of $\mathcal{B}$ from bottom to top and from left to right in a row. Suppose the current point is (i, j). There are two cases:

(i) If (i, j) does not hit any rectangle in $\mathcal{R}$, we set

$$\tilde{\mu}(i, j) = \min\{\tilde{\mu}(i - 1, j), \tilde{\mu}(i, j - 1), \tilde{\mu}(i - 1, j - 1)\} + \omega(i, j), \tag{1}$$

where $\tilde{\mu}(a, b) = \infty$ if $(a, b) \notin \mathcal{B}$.

(ii) Let $\mathsf{R} = [x^- : x^+] \times [y^- : y^+]$ be a rectangle hit by (i, j). We compute $\tilde{\mu}(i, j)$ using the computed $\tilde{\mu}$ values of the points on the left and bottom boundaries of R that are dominated by (i, j), and setting the weight of all points in R to be ω_{R}. So the weight of a path inside R is proportional to its length.

Without loss of generality, assume $i - x^- \geq j - y^-$. We divide the left and bottom boundary points that are dominated by (i, j) into three sets: $N_1 = \{(x^-, y) \mid y \in [y^- : j]\}$, $N_2 = \{(x, y^-) \mid x \in [x^- : i - (j - y^-) - 1]\}$, and $N_3 = \{(x, y^-) \mid x \in [i - (j - y^-) : i]\}$. See Figure 2(b).

The optimal admissible path from $(1, 1)$ to (i, j) must pass through a point in $N_1 \cup N_2 \cup N_3$. So we compute $\tilde{\mu}(i, j)$ as follows:

$$\tilde{\mu}(i, j) = \min \left\{ \begin{array}{ll} (i - x^-)\omega_{\mathsf{R}} & + \min_{(a,b) \in N_1} \tilde{\mu}(a, b), \\ i\omega_{\mathsf{R}} & + \min_{(a,b) \in N_2}(\tilde{\mu}(a, b) - a\omega_{\mathsf{R}}), \\ (j - y^-)\omega_{\mathsf{R}} & + \min_{(a,b) \in N_3} \tilde{\mu}(a, b) \end{array} \right\}. \tag{2}$$

The following lemma proves that our algorithm returns a $(1 + \varepsilon)$-approximation of $\mathrm{dtw}(P, Q)$.

▶ **Lemma 7.** *For each $(i, j) \in \mathcal{B}$, if $\mu(i, j) \leq \overline{\mathsf{d}}$, then*

$$|\tilde{\mu}(i, j) - \mu(i, j)| \leq \frac{\varepsilon}{2}(\mu(i, j) + (i + j)\underline{\mathsf{d}}/2n).$$

Proof: By induction on the order in which the $\tilde{\mu}$ values of points in $\mathcal{B}$ are computed, we prove $\tilde{\mu}(i, j) - \mu(i, j) \leq \frac{\varepsilon}{2}(\mu(i, j) + (i + j)\underline{\mathsf{d}}/2n)$. The lemma is obviously true for $(1, 1)$. Assume it is true for all points of $\mathcal{B}$ processed before (i, j). We prove that it is also true for (i, j).

If (i, j) does not hit any rectangle in $\mathcal{R}$, then $\tilde{\mu}(i, j)$ is computed using (1). Let $(a, b) \in \{(i - 1, j), (i, j - 1), (i - 1, j - 1)\}$ be the predecessor of (i, j) in the optimal admissible path from $(1, 1)$ to (i, j). Then $\mu(i, j) = \mu(a, b) + \omega(i, j)$. Since $\mu(a, b) \leq \mu(i, j) \leq \overline{\mathsf{d}}$, there is a rectangle R containing (a, b). Since (i, j) does not hit any rectangle, (a, b) must actually lie on the boundary of R, and thus in $\mathcal{B}$. So by induction hypothesis,

$$\tilde{\mu}(i, j) - \mu(i, j) = \tilde{\mu}(a, b) - \mu(a, b) \leq \frac{\varepsilon}{2}(\mu(a, b) + (a + b)\underline{\mathsf{d}}/2n) \leq \frac{\varepsilon}{2}(\mu(i, j) + (i + j)\underline{\mathsf{d}}/2n).$$

In the second case, let $\mathsf{R} \in \mathcal{R}$ be the rectangle hit by (i, j) and used to compute $\tilde{\mu}(i, j)$. Let (a, b) be the intersection of the optimal admissible path from $(1, 1)$ to (i, j) and the boundary of R. Then

$$\begin{aligned} \tilde{\mu}(i, j) &\leq \tilde{\mu}(a, b) + \max\{i - a, j - b\}\omega_{\mathsf{R}} \\ &\leq \mu(a, b) + \frac{\varepsilon}{2}(\mu(a, b) + (a + b)\underline{\mathsf{d}}/2n) + \max\{i - a, j - b\}\omega_{\mathsf{R}} \\ &\leq \mu(i, j) + \frac{\varepsilon}{2}(\mu(i, j) + (i + j)\underline{\mathsf{d}}/2n). \end{aligned}$$

The last inequality satisfies because $\omega_{\mathsf{R}} \leq \omega(h, k) + \frac{\varepsilon}{2}\max\{\omega(h, k), \underline{\mathsf{d}}/2n\}$ for any $(h, k) \in \mathsf{R}$ by Lemma 6. Similarly, we can prove that $\mu(i, j) - \tilde{\mu}(i, j) \leq \frac{\varepsilon}{2}(\mu(i, j) + (i + j)\underline{\mathsf{d}}/2n)$, and the lemma follows. □

▶ **Corollary 8.** $|\tilde{\mu}(m, n) - \mathrm{dtw}(P, Q)| \leq \varepsilon\mathrm{dtw}(P, Q)$.

We now describe how the algorithm for computing $\tilde{\mu}(i, j)$ is implemented efficiently.

Sorting points in $\mathcal{B}$.   Using radix sort, we sort the points of $\mathcal{B}$ in (y, x) lexicographical order, where x and y denote the first and second coordinates of points, so that they are sorted in the order in which they are processed. We also perform the same radix sort for (x, y) and $(y - x, x)$ lexicographical orderings. For each point in $\mathcal{B}$, we add a pointer to the previous point in each of the three sorted orders, namely, a pointer to the first point below, to the left of, and on the lower-left diagonal of the current point. These pointers are used to identify the $\tilde{\mu}$ values required in (1).

Finding a rectangle hit by a point.   The algorithm also needs to determine whether there exists a rectangle of $\mathcal{R}$ hit by (i, j). This can be achieved by maintaining the rectangle with the right-most right boundary when we traverse each row. More precisely, when processing the point $(i, j) \in \mathcal{B}$, we maintain a rectangle R_{curr} that is hit by (i, j) and whose right boundary spans the farthest; we denote the x-coordinate of the right boundary of R_{curr} by ξ_{curr}. If no rectangle hits (i, j), we set $\mathsf{R}_{curr} = \text{NULL}$. We update $\mathsf{R}_{curr}, \xi_{curr}$ while processing (i, j) as follows: If (i, j) is the left boundary point of a rectangle R with ξ being the x-coordinate of its right boundary and if $\xi > \xi_{curr}$, we set $\mathsf{R}_{curr} = \mathsf{R}$ and $\xi_{curr} = \xi$. Otherwise, if (i, j) is the right boundary point of R_{curr}, i.e., $i = \xi_{curr}$, we set $\mathsf{R}_{curr} = \xi_{curr} = \text{NULL}$. The total time spent at (i, j) is $O(1)$.

Range-min data structure.   If (i, j) hits the rectangle R_{curr}, we compute the minimum $\tilde{\mu}$ value in the interval $N_1 = [y^- : j]$ on the left boundary of R_{curr} and in the intervals $N_2 = [x^- : i - (j - y^-) - 1], N_3 = [i - (j - y^-) : i]$ on the bottom boundary of R_{curr}, by performing range-min queries. We use the data structure proposed by Fischer and Heun [21] (see also [1]), which answers a range-min query in $O(1)$ time after linear time preprocessing. Thus, a range-min query on N_2 or N_3 can be answered in $O(1)$ time by constructing a static range-min data structure on the points on the bottom boundary of R_{curr} (all $\tilde{\mu}$ values for these points have been computed before visiting any point that hit R_{curr}). On the other hand, to support range-min query on N_1, we need a range-min data structure on the left boundary points of R_{curr} that also supports inserting new points at the end when the $\tilde{\mu}$ values of the left boundary points are computed row by row.

We briefly describe their static data structure, and show how to extend it to support insertion in amortized $O(1)$ time. The input is an array of k real numbers. We say a data structure has time complexity $\langle p(k), q(k) \rangle$ if the preprocessing takes time $O(p(k))$ and each query takes time $O(q(k))$. The static data structure divides the array into blocks of size $b = \frac{1}{4} \log_2 k$. For each block, we construct a naive $\langle b^2, 1 \rangle$-time data structure. But Fischer and Heun show that many blocks can share the same data structure, so we create only much fewer than $k/b = O(k/\log k)$ copies of the data structure. Next the algorithm computes the minimum for each block and uses an $\langle k \log k, 1 \rangle$-time "exponential-range" data structure over the block minimums. We now describe each of the two structures in more detail.

A *Cartesian tree* of a length-b array stores the minimum element of the array at the root and the left (resp. right) child of the root is a Cartesian tree on the elements to the left (resp. right) of the minimum element. It can be built by a linear scan of the elements and pushing/popping elements into/from a stack at most $2b$ times; these push/pop operations serve as a fingerprint of the Cartesian tree. Thus, the number of different Cartisian trees for a length-b array is bounded by $2^{2b} = 4^b$. It turns out that all arrays of length b that have the same structured *Cartesian tree* [2] can share the same $\langle b^2, 1 \rangle$-time data structure. We thus build 4^b copies of the $\langle b^2, 1 \rangle$ data structure as follows: We go through each of the $O(k/\log k)$ blocks, and compute the fingerprint of the block in $O(b)$ time; if there is no data structure corresponding to the fingerprint, we build it in $O(b^2)$ time by computing the minimums for all possible $O(b^2)$ ranges. The exponential-range data structure maintains the minimums of $O(\log k)$ ranges starting at each index $i \in [k]$ of exponentially increasing sizes $1, 2, 2^2, \ldots, 2^{\log k}$. Then the minimum of a range $[i, j]$ can be obtained by taking the minimum of two ranges $[i, i + 2^\alpha - 1]$ and $[j - 2^\alpha + 1, j]$, where α is the largest integer such that $2^\alpha \leq j - i + 1$. The total preprocessing time is $O((k/b) \log(k/b) + 4^b b^2) = O(k)$.

To answer a range-min query, we compute the blocks containing the two end points of the query range; the minimum of the whole blocks in the range can be answered using the exponential-range data structure in $O(1)$ time; the minimums of the two partial blocks can also be answered in $O(1)$ time using the naive data structures associated with the two boundary blocks. So each query takes $O(1)$ time.

We now describe how to support inserting an element to the end of the array in amortized constant time. If the last block of the array contains less than b elements, the exponential-range data structure remains the same, and we just need to update the fingerprint of the last block. We can encode the fingerprint information (a sequence of pushes and pops) as a path from the root to an internal node in a full binary tree of depth $2b$, where a push corresponds to branching to the left child and a pop corresponds to branching to the right child. At each node of the binary tree, we store a pointer to the corresponding naive range-min data structure. Recall that the Cartesian tree is built by a linear scan of the elements; so inserting a new element just means going one more level down the binary tree, which takes constant time. On the other hand, when the last block is already full, the newly inserted element starts a new block. In this case, we also need to update the exponential-range data structure, which takes $O(\log(k/b))$ time; but since this only happens every $O(b) = O(\log k)$ elements, the amortized time per insersion is still constant. Therefore, we can insert an element to the data structure in amortized $O(1)$ time.

▶ **Lemma 9.** *For all $(i, j) \in \mathcal{B}$, $\tilde{\mu}(i, j)$ can be computed in a total time of $O(|\mathcal{B}|)$.*

2.4   Running time analysis

We now bound the size of $|\mathcal{B}|$, which by Lemma 9 will bound the running time of step (iii) of the algorithm. Similar argument will bound the time spent in generating the set $\mathcal{F}$, which will bound the running time of step (ii).

▶ **Lemma 10.** *The number of points in $\mathcal{B}$ is $O(\frac{\kappa}{\varepsilon} n \log n)$, $O(\frac{\kappa^d}{\varepsilon^d} n \log n)$, and $O(\frac{1}{\varepsilon} n^{2-1/d} \log n)$ for κ-packed, κ-bounded and backbone sequences, respectively.*

Proof: For any well-separated pair of quadtree nodes $(u, v) \in \mathcal{F}$, let $\hat{m}_u = |P \cap \hat{\square}_u|, \hat{n}_v = |Q \cap \hat{\square}_v|$. Recall that A denotes the set of active nodes of quadtree $\mathcal{T}$, and α_u (resp. β_v) is the number of maximal contiguous subsequences of P in $\hat{\square}_u$ (resp. Q in $\hat{\square}_v$) computed by our algorithm. Since $\hat{\square}_u$ is concentric with $\square_u$ with twice the side length of $\square_u$, $\sum_{u \in A} \hat{m}_u \leq 2^d \sum_{u \in A} m_u = O(m \log n)$. Let $N(u) = \{v \mid (u, v) \in \mathcal{F}\}$. The total number of rectangle boundary points is

$$|\mathcal{B}| \leq 2 \sum_{(u,v) \in \mathcal{F}} (\hat{m}_u \beta_v + \alpha_u \hat{n}_v) = 2 \sum_{u \in A} \hat{m}_u \sum_{v \in N(u)} \beta_v + 2 \sum_{v \in A} \hat{n}_v \sum_{u \in N(v)} \alpha_u. \tag{3}$$

We will show next that for any $u \in A$, $\sum_{v \in N(u)} \beta_v$ is bounded by $O(\kappa/\varepsilon)$ for κ-packed sequences, $O(\kappa^d/\varepsilon^d)$ for κ-bounded sequences, and $O(n^{1-1/d}/\varepsilon)$ for backbone sequences. The first part of (3) is then bounded by $O(\frac{\kappa}{\varepsilon} n \log n)$, $O(\frac{\kappa^d}{\varepsilon^d} n \log n)$, and $O(\frac{1}{\varepsilon} n^{2-1/d} \log n)$ for κ-packed, κ-bounded, and backbone sequences. Symmetrically, the second part of (3) has the same bound, and the lemma follows.

We now bound $\sum_{v \in N(u)} \beta_v$ for any $u \in A$. By Lemma 5, there exists a constant c such that for any $v \in N(u)$, $\hat{\square}_v$ is contained in a ball B concentric with $\square_u$ of radius $\frac{c}{\varepsilon} \Delta(u)$.

There are two types of maximal contiguous subsequence $[y_b^- : y_b^+]$ of $Q \cap \hat{\square}_v$ computed by our algorithm: (i) $q_{y_b^+} = q_n$ is the last point of Q, and (ii) $q_{y_b^+ + 1}$, the point of Q after the last point in the MCS lies outside of $\hat{\square}_v$. The first type of MCS is bounded by the number of v's such that $\hat{\square}_v$ contains the last point of Q, q_n. Suppose node u is at level t of the quadtree. By Lemma 5(ii), such v's can be from levels $t - 1, t, t + 1$. Moreover, at each level, q_n can be in the $\hat{\square}_v$ of at most 2^d v's.

Thus, the number of maximal contiguous subsequences of the first type is at most $2^d \times 3 = O(1)$. In the following, we bound the second type separately for each family of input sequence.

κ-**packed sequences**. Since the MCS starts inside $\square_v$ and leaves $\hat{\square}_v$ before the next MCS of $S_v(Q)$ starts, the length of γ_Q between $q_{y_b^-}$ and $q_{y_b^+}$ is at least $\Delta(v)/2$ (see Figure 3). Let $\hat{L}_v$ be the length of $\gamma_Q \cap \hat{\square}_v$. Then

$$\sum_{v \in N(u)} \beta_v \leq O(1) + \sum_{v \in N(u)} \frac{\hat{L}_v}{\Delta(v)/2} \leq O(1) + \frac{4}{\Delta(u)} \sum_{v \in N(u)} \hat{L}_v.$$

The last inequality follows from Lemma 5(ii). Because of the following four conditions: the side length of $\hat{\square}_u$ is twice that of $\square_u$, the nodes in $N(u)$ belong to three levels of $\mathcal{T}$ (Lemma 5(ii)), the cells of the nodes at the same level of $\mathcal{T}$ are disjoint, and $\hat{\square}_v \subseteq \mathsf{B}$ for all $v \in N(u)$, we can conclude that

$$\sum_{v \in N(u)} \hat{L}_v \leq 3 \cdot 2^d |\gamma_Q \cap \mathsf{B}| \leq \frac{3 \cdot 2^d c \kappa}{\varepsilon}.$$

The last inequality follows because P is κ-packed sequence. Hence $\sum_{v \in N(u)} \beta_v = O(\kappa/\varepsilon)$, as claimed.

κ-**bounded sequences**. We first show that for any two MCSs $[y_1^- : y_1^+]$ and $[y_2^- : y_2^+]$, $||q_{y_1^-} q_{y_2^-}|| \geq \Delta(v)/(\kappa + 2)$. This is because between points $q_{y_1^-}$ and $q_{y_2^-}$, the curve γ_Q goes from inside $\square_v$ to outside $\hat{\square}_v$ which spans distance at least $\Delta(v)/2$. Let q be the intersection of this portion of γ_Q with the boundary of $\hat{\square}_v$. By κ-boundedness, $\gamma_Q(y_1^- : y_2^-) \subseteq \mathsf{B}(q_{y_1^-}, \frac{\kappa}{2}||q_{y_1^-} q_{y_2^-}||) \cup \mathsf{B}(q_{y_2^-}, \frac{\kappa}{2}||q_{y_1^-} q_{y_2^-}||)$. Therefore, $(1 + \kappa/2)||q_{y_1^-} q_{y_2^-}|| \geq ||q_{y_1^-} q|| \geq \Delta(v)/2$, and the claim follows. By a packing argument, the number of MCSs in $\hat{\square}_v$ is bounded by $O(\kappa^d)$. Finally, $|N(u)| = O(1/\varepsilon^d)$ by another packing argument in the ball B. So the number of second-type MCSs in all $\hat{\square}_v$'s for $v \in N(u)$ is $O(\kappa^d/\varepsilon^d)$.

Backbone sequences. By the property that two consecutive points on a backbone sequence have distance between c_1 and c_2, there must exist one point on any MCS in the shell along the boundary of $\square_v$ with thickness c_2. The volumn of the shell is $O(\Delta(v)^d - (\Delta(v) - c_2)^d) = O(\Delta(v)^{d-1})$. Furthermore, any two points on Q are at least distance 1 apart. So the number of MCSs is bounded by $O(\Delta(v)^{d-1})$. Since $|N(u)| = O(1/\varepsilon^d)$, the number of MCSs in all $\hat{\square}_v$'s for $v \in N(u)$ is $O(\frac{\Delta(v)^{d-1}}{\varepsilon^d})$. On the other hand, each second-type MCS consumes a portion of γ_Q of length at least $\Delta(v)/2$; this means that the subsequence contains $\Omega(\Delta(v)/c_2) = \Omega(\Delta(v))$ points of Q. Since there are a total of n points in Q, the total number of MCSs in all $\hat{\square}_v$'s with $(u, v) \in \mathcal{F}$ is also bounded by $O(\frac{n}{\Delta(v)})$. The worst case happens when $\frac{\Delta(v)^{d-1}}{\varepsilon^d} = (\frac{n}{\Delta(v)})$, which leads to $\Delta(v) = \varepsilon n^{1/d}$. So the total number of second-type MCSs is $O(\frac{1}{\varepsilon} n^{1-1/d})$. $\square$

To bound the running time for constructing the family $\mathcal{F}$ of well-separated node pairs, we bound, for each active node $u \in A$, the number of times PAIRING(u, v) is called for some $v \in A$. This can be bounded using similar arguments as the proof of Lemma 10, and we can show that the time for constructing $\mathcal{F}$ is $O(\frac{\kappa}{\varepsilon} n \log n)$, $O(\frac{\kappa^d}{\varepsilon^d} n \log n)$, and $O(\frac{1}{\varepsilon} n^{2-1/d})$ for κ-packed, κ-bounded, and backbone sequences, respectively. Combining this bound with Lemmas 4, 9, and 10, we obtain Theorem 1.

3 Edit Distance

We now show how our DTW algorithm can be extended to compute a $(1 + \varepsilon)$-approximate value of ed(P, Q). Define $\mathsf{V} = [m + 1] \times [n + 1]$. For $i < m + 1$ and $j < n + 1$, we have $\omega(i, j) = ||p_i q_j||$. Otherwise $\omega(i, j)$ is undefined (e.g., $\omega(m + 1, \cdot)$ and $\omega(\cdot, n + 1)$ undefined). We add an edge between

every pair of neighboring points in V. The weight of a horizontal or vertical edge is set to g and the weight of a diagonal edge $\langle (i,j),(i+1,j+1) \rangle$ is set to $\omega(i,j)$. The weight of an admissible path π is defined as the sum of weights of the edges along π. As earlier, we define $\mu(i,j)$ to be the minimum weight of an admissible path from $(1,1)$ to (i,j). Then $\mathsf{ed}(P,Q) = \mu(m+1, n+1)$.

We compute an approximate value of $\mathsf{ed}(P,Q)$ using the same 4-step algorithm as for $\mathsf{dtw}(P,Q)$, with a different implementation of each step. Step (ii) of the algorithm remains the same, except that we add all the points on the $(m+1)$-st column and $(n+1)$-st row of V to $\mathcal{B}$. In the following, we give a simple $O(n)$-approximation for step (i), and point out modifications needed in step (iii) to compute a value $\tilde{\mu}(i,j)$ for every $(i,j) \in \mathcal{B}$, such that $\mu(i,j) \le \tilde{\mu}(i,j) \le (1+\varepsilon)\tilde{\mu}(i,j)$.

$O(n)$**-approximation.** If $m = n$ and the weight of the monotone path $\pi = \langle (1,1),(2,2),\ldots,(n+1,n+1) \rangle$ is at most g, then it is returned as the optimal path. Otherwise, $\mathsf{ed}(P,Q) \ge g$, and we set $\underline{\mathsf{d}} = g$. Since $\mathsf{ed}(P,Q)$ is no larger than the weight of an all-gap admissible path from $(1,1)$ to $(m+1, n+1)$, we have $\mathsf{ed}(P,Q) \le 2(m+n)g \le \overline{\mathsf{d}} = 4n\underline{\mathsf{d}}$.

Computing admissible paths. We describe how to compute $\tilde{\mu}(i,j)$, for all $(i,j) \in \mathcal{B}$, in the same row by row order. The main difference from DTW is that, since ED allows gaps, it is possible for the optimal admissible path between $(1,1)$ and (i,j) to have rectilinear subpaths through grid points that are not covered by any rectangle. As in Section 2.3, we consider whether there there exists a rectangle hit by (i,j).

First, assume there exists a rectangle $\mathsf{R} = [x^- : x^+] \times [y^- : y^+] \in \mathcal{R}$ hit by (i,j). Similar to DTW, we divide the relevant points on the left and bottom boundaries of R into three disjoint subsets N_1, N_2 and N_3. If $2g \le \omega_\mathsf{R}$, it is always preferable to take a rectilinear path inside R. Thus we can assume the admissible path to (i,j) goes through either (x^-, j) or (i, y^-), and we set $\tilde{\mu}(i,j) = \min\{\tilde{\mu}(x^-, j) + (i - x^-), \tilde{\mu}(i, y^-) + (j - y^-)\}$. If $2g > \omega_\mathsf{R}$, the minimum-weight path inside R should take as many diagonal steps as possible. So we set

$$\tilde{\mu}(i,j) = \min \left\{ \begin{array}{l} j\omega_\mathsf{R} + (i-j)g + \min_{(a,b) \in N_1}(\tilde{\mu}(a,b) + (g - \omega_\mathsf{R})b), \\ j\omega_\mathsf{R} + (i-j)g + \min_{(a,b) \in N_2}(\tilde{\mu}(a,b) + (g - \omega_\mathsf{R})b - ag), \\ i\omega_\mathsf{R} + (j-i)g + \min_{(a,b) \in N_3}(\tilde{\mu}(a,b) + (g - \omega_\mathsf{R})a) \end{array} \right\}.$$

We use the same range-min data structure of Fischer and Heun to compute each $\tilde{\mu}(i,j)$ in amortized $O(1)$ time. The key used for the data structure is $\tilde{\mu}(a,b) + (g - \omega_\mathsf{R})b$, $\tilde{\mu}(a,b) + (g - \omega_\mathsf{R})b - ag$, and $\tilde{\mu}(a,b) + (g - \omega_\mathsf{R})a$ for N_1, N_2, and N_3, respectively.

Next, assume (i,j) does not hit any rectangle in $\mathcal{R}$. If $\{(i-1,j),(i,j-1),(i-1,j-1)\} \subset \mathcal{B}$, and thus their $\tilde{\mu}$ values have been computed, it is trivial to compute $\tilde{\mu}(i,j)$ in $O(1)$ time. We now focus on the case where one of the predecessors of (i,j) is not in $\mathcal{B}$. Let $U = \bigcup_{\mathsf{R} \in \mathcal{R}} \mathsf{R}$ denote the union of all rectangles in $\mathcal{R}$. A point $(h,k) \in U$ is on the boundary of U, denoted by ∂U, if (h,k) does not lie in the interior of any rectangle of $\mathcal{R}$; so at least one point of $\{(h-1,k),(h+1,k),(h,k-1),(h,k+1)\}$ is not in U. Consider any admissible path π from $(1,1)$ to (i,j) whose total weight is at most $\overline{\mathsf{d}}$. Let (a,b) denote the last point of $\mathcal{B}$ on π before reaching (i,j). The subpath of π between (a,b) and (i,j) must be outside U, and it can only contain gaps since the weight of any point outside U is greater than $\overline{\mathsf{d}}$, except the first step out of (a,b), which costs $\omega(a,b)$ if it is diagonal. Let (i_0, j) (resp. (i, j_0)) be the first point of $\mathcal{B}$ to the left of (resp. below) (i,j) on row j (resp. column i). Let $\partial U_{ij} = [i-1] \times [j-1] \cap \partial U$ denote the points on ∂U that are lower-left of (i,j). We set

$$\tilde{\mu}(i,j) = \min \left\{ \begin{array}{l} \tilde{\mu}(i_0, j) + (i - i_0)g, \\ \tilde{\mu}(i, j_0) + (j - j_0)g, \\ \min_{(a,b) \in \partial U_{ij}}(\tilde{\mu}(a,b) + (i + j - a - b)g + \min(0, \omega(a,b) - 2g)) \end{array} \right\}. \quad (4)$$

To compute $\tilde{\mu}(i,j)$, we use a different and simpler range-min data structure for ∂U with key $\tilde{\mu}(a,b) - (a+b)g + \min(0, \omega(a,b) - 2g)$, that supports the decrease-key operation and query in $O(\log n)$ time.

More specifically, we maintain a minimum over points of ∂U in each column as we traverse $\mathcal{B}$ row by row. Then we maintain the column minimums in a complete binary tree where each leaf corresponds to a column and an internal node stores the minimum over the leaves of the subtree rooted at this node. Note that the column minimums are always non-increasing while we perform the row by row computation. When the minimum of a column corresponding to some leaf v gets decreased, we update the minimum information stored at each node along the path from v to the root of the binary tree. This takes $O(\log n)$ time. The last term in (4) can be computed by querying the the complete binary tree with range $[i - 1]$ in $O(\log n)$ time.

▶ **Lemma 11.** *The number of points on ∂U is $O(\frac{\kappa}{\varepsilon} n)$, $O(\frac{\kappa^d}{\varepsilon^d} n)$, and $O(\frac{1}{\varepsilon^{1-1/(d+1)}} n^{2-1/(d+1)})$ for κ-packed, κ-bounded, and backbone sequences, respectively.*

Proof: We analyze the number of times we enter of U as we (conceptually) traverse each row j of V from column 1 to column m. Let $\mathsf{B} = \mathsf{B}(q_j, 4\sqrt{d}\,\overline{\mathsf{d}})$ be a ball centered at q_j with radius $4\sqrt{d}\,\overline{\mathsf{d}}$. Let (i_1, j) be an entering point of U, i.e., $(i_1, j) \in U$ and $(i_1 - 1, j) \notin U$, and let (i_2, j) be the first point to the right of (i_1, j) that is not in U. We claim that the polygonal curve of the subsequence $P(i_1 : i_2)$ has length $\Omega(\varepsilon\overline{\mathsf{d}})$ inside B. Let R denote a rectangle of which (i_1, j) is a (left) boundary point, and let (u, v) denote the quadtree node pair from which R_1 is constructed, where $p_{i_1} \in \hat{\square}_u$ and $q_j \in \hat{\square}_v$. If $\Delta(u) \leq \frac{\varepsilon}{2c}\overline{\mathsf{d}}$, where c is the constant in Lemma 5(iii), then by Lemma 5(iii), $\mathsf{d}_\square(u, v) \leq \frac{c}{\varepsilon}\Delta(u) \leq \overline{\mathsf{d}}/2$. Note that $\Delta(v) \leq 2\Delta(u) = O(\varepsilon\overline{\mathsf{d}})$. So $\|p_{i_1} q_j\| \leq \mathsf{d}_\square(u, v) + O(\varepsilon\overline{\mathsf{d}}) \leq \frac{3}{4}\overline{\mathsf{d}}$. Moreover, since $(i_2, j) \notin U$, it must be that $\|p_{i_2} q_j\| > \overline{\mathsf{d}}$. Therefore, the polygonal curve of $P(i_1 : i_2)$ contains $\Omega(\overline{\mathsf{d}})$ length inside ball B. On the other hand, suppose $\Delta(u) > \frac{\varepsilon}{2c}\overline{\mathsf{d}}$. Then $\hat{\square}_u$ must be contained in ball B because the family of well-separated pairs are constructed by invoking the PAIRING procedure on neighboring nodes of size $\overline{\mathsf{d}}$. So the polygonal curve of $P(i_1 : i_2)$ has $\Omega(\varepsilon\overline{\mathsf{d}})$ length inside B because $P(i_1 : i_2)$ goes from inside $\square_u$ to outside of $\hat{\square}_u$. Thus, the claim is true.

Using the same argument as in Lemma 10, we can show that the number of such entering points of U is $O(\frac{\kappa}{\varepsilon})$ for κ-packed sequences and $O(\frac{\kappa^d}{\varepsilon^d})$ for κ-bounded sequences. For backbone sequences, the bound is slightly weaker as we can not guarantee that there is a point in $P(i_1 : i_2)$ within distance c_2 of the boundary of B. Instead, the number of points of P inside B is bounded by $O(\overline{\mathsf{d}}^d)$ since any two non-consecutive points of P have distance at least 1 (by definition of backbone sequences); this value is also an upper bound on the number of entering points of U on row j. A second upper bound of $O(m/(\varepsilon\overline{\mathsf{d}}))$ can be obtained similarly as in Lemma 10. The worst case happens when $\overline{\mathsf{d}} = (\frac{m}{\varepsilon})^{1/(d+1)}$, which gives a bound of $(\frac{m}{\varepsilon})^{1-\frac{1}{d+1}}$. Summing over n rows, the bounds follows. □

Note that both the number of updates and the number of queries in the binary tree are bounded by $|\partial U|$, and each update or query takes $O(\log n)$ time. Moreover, the case when there exists a rectangle hit by the current point takes the same time as for DTW. Therefore Theorem 2 follows.

4 Conclusion

In this paper, we presented $(1 + \varepsilon)$-approximate algorithms for computing the dynamic time warping (DTW) and edit distance (ED) between a pair of point sequences. The running time of our algorithms is near-linear when the input sequences are κ-packed or κ-bounded, and subquadratic when the input sequences are protein backbone sequences. Our algorithms are the first near-linear or subquadratic-time algorithms known for computing DTW and ED for "well-behaved" sequences. One interesting open question is whether there exists a near-linear algorithm for computing DTW and ED for backbone sequences in $\mathbb{R}^2$. Another interesting open problem is to identify other dynamic-programming based geometric optimization problems that can be solved using our approach, i.e., by visiting a small number of entries of the dynamic programming table using geometric properties of the input.

References

1 http://web.stanford.edu/class/archive/cs/cs166/cs166.1146/lectures/01/Slides01.pdf.

2 https://en.wikipedia.org/wiki/Cartesian_tree.

3 Time series matching with dynamic time warping. https://systematicinvestor.wordpress.com/2012/01/20/time-series-matching-with-dynamic-time-warping/.

4 A. Abboud, A. Backurs, and V. V. Williams. Quadratic-time hardness of LCS and other sequence similarity measures. In *IEEE Symposium on Foundations of Computer Science*, 2015.

5 A. Abboud, V. V. Williams, and O. Weimann. Consequences of faster alignment of sequences. In *International Colloquium on Automata, Languages, and Programming*, 2014.

6 H. Alt, C. Knauer, and C. Wenk. Comparison of distance measures for planar curves. *Algorithmica*, 38(1):45–58, 2004.

7 A. Andoni, R. Krauthgamer, and K. Onak. Polylogarithmic approximation for edit distance and the asymmetric query complexity. In *IEEE Symposium on Foundations of Computer Science*, 2010.

8 B. Aronov, S. Har-Peled, C. Knauer, Y. Wang, and C. Wenk. Fréchet distance for curves, revisited. In *European Symposium on Algorithms*, 2006.

9 A. Backurs and P. Indyk. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). In *Symposium on Theory of Computing Conference*, 2015.

10 K. Bringmann. Why walking the dog takes time: Frechet distance has no strongly subquadratic algorithms unless SETH fails. In *IEEE Symposium on Foundations of Computer Science*, 2014.

11 K. Bringmann and M. Künnemann. Improved approximation for fréchet distance on c-packed curves matching conditional lower bounds. *CoRR*, abs/1408.1340, 2014.

12 K. Bringmann and M. Künnemann. Quadratic conditional lower bounds for string problems and dynamic time warping. In *IEEE Symposium on Foundations of Computer Science*, 2015.

13 K. Bringmann and W. Mulzer. Approximability of the Discrete Fréchet Distance. In *Symposium on Computational Geometry*, 2015.

14 T. Brox and J. Malik. Object segmentation by long term analysis of point trajectories. In *Proc. ECCV*. Springer, 2010.

15 E. 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 *IEEE Computers in Cardiology*, 1998.

16 P. B. Callahan and S. R. Kosaraju. A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields. *J. ACM*, 42(1):67–90, 1995.

17 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 *SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2012.

18 A. Driemel, S. Har-Peled, and C. Wenk. Approximating the Fréchet distance for realistic curves in near linear time. *Discrete & Computational Geometry*, 48(1):94–127, 2012.

19 R. Durbin, S. R. Eddy, A. Krogh, and G. Mitchison. *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Camridge University Press, New York, 1998.

20 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.

21 J. Fischer and V. Heun. Theoretical and practical improvements on the rmq-problem, with applications to lca and lce. In *Combinatorial Pattern Matching*. Springer, 2006.

22 T. Gasser and K. Wang. Alignment of curves by dynamic time warping. *Annals of Statistics*, 25(3):1251–1276, 1997.

23 M. C. Gonzalez, C. A. Hidalgo, and A.-L. Barabasi. Understanding individual human mobility patterns. *Nature*, 453(7196):779–782, 2008.

24 S. Har-Peled. *Geometric approximation algorithms*. American Mathematical Society Providence, 2011.

25 R. Impagliazzo and R. Paturi. On the complexity of k-SAT. *Journal of Computer and System Sciences*, 62(2):367–375, 201.

26 E. J. Keogh and M. J. Pazzani. Scaling up dynamic time warping for datamining applications. In *Proc. KDD*, 2000.

27 E. J. Keogh and C. A. Ratanamahatana. Exact indexing of dynamic time warping. *Knowl. Inf. Syst.*, 7(3):358–386, 2005.

28 J. Kleinberg and E. Tardos. *Algorithm Design*. Addison-Wesley Longman Publishing Co., Inc., 2005.

29 R. Kolodny, P. Koehl, and M. Levitt. Comprehensive evaluation of protein structure alignment methods: Scoring by geometric measures. *Journal of Molecular Biology*, 346(4):1173 – 1188, 2005.

30 J.-G. Lee, J. Han, and K.-Y. Whang. Trajectory clustering: a partition-and-group framework. In *Proc. SIGMOD*, 2007.

31 M. Müller. *Information Retrieval for Music and Motion*. Springer-Verlag Berlin Heidelberg, 2007.

32 M. E. Munich and P. Perona. Continuous dynamic time warping for translation-invariant curve alignment with applications to signature verification. In *Proc. ICCV*, 1999.

33 L. Rabiner and B. Juang. *Fundamentals of Speech Recognition*. PTR Prentice Hall, 1993.

34 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.

A   2-approximation of $\mathsf{dfr}(P, Q)$

Given a sequence $Q = \langle q_1, \ldots, q_n \rangle$ in $\mathbb{R}^d$, a subsequence $\tilde{Q} = \langle Q_{i_1}, \ldots, Q_{i_k} \rangle$ is a *left μ-simplification* of Q if

(i) $1 \le i_1 < i_2 < \cdots < i_k \le n$
(ii) for any integer $j \in (i_{t-1}, i_t)$, $d(q_j, q_{i_{t-1}}) \le \mu$.
(iii) for any $t \in [2 : k]$, we have $d(q_{i_{t-1}}, q_{i_t}) > \mu$.

We solve the decision version of the problem, i.e., deciding whether $\mathsf{dfr}(P, Q) < \delta$. we perform a left μ-simplification on both sequences P and Q to obtain $\tilde{P}$ and $\tilde{Q}$ with $\mu = \varepsilon\delta/2$. According to [8], it suffice to solve the decision problem on $\tilde{P}$ and $\tilde{Q}$ to obtain a $(1 + \varepsilon)$-approximation. The time taken to solve the decision problem is proportional to the number of entries in the dynamic programming table that have value at most δ; we call them reachable entries. This is equal to the sum of the number of points of $\tilde{P}$ in the ball $\mathsf{B}(q, \delta)$ for all $q \in \tilde{Q}$.

Let γ denote the κ-packed polygonal curve that connects the points of $\tilde{P}$. Consider another ball B' with the same center as B, but with a radius $\delta + \mu$. By property of μ-simplification, the portion of γ between consecutive points in the sequence is of length at least μ. Thus every point of the simplified sequence inside B is the left end-point of a unique portion of γ of length at least μ inside ball B'. By κ-packedness, the total length of γ inside B' is $O(\kappa(\delta + \mu))$. With $\mu = O(\varepsilon\delta)$, the total number of points in B is bounded by $O(\kappa(\delta + \mu)/\mu) = O(\kappa(1 + \varepsilon)/\varepsilon) = O(\kappa/\varepsilon)$, for $\epsilon \le 1$. Using the approximation algorithm for spherical range query introduced in the same paper, these reachable entries can be computed in $O(\kappa/\varepsilon)$ time. Since there are at most n points in $\tilde{P}$, the decision problem takes $O(n\kappa/\varepsilon)$ time.

Using the technique of well-separated pair decomposition, we can obtain a set of $O(n)$ distances such that the distance between any pair of points from P and Q is $(1 + 1/5)$-approximated by one of the distances in the set. Therefore, by performing $O(\log n/\varepsilon) = O(\log n)$ binary searches, and using the above decision problem, we can obtain a $(1 + \varepsilon)$-approximation of $\mathsf{dfr}(P, Q)$ in $O(\frac{\kappa}{\varepsilon} n \log n)$ time. Setting ε to a small constant, we obtain a 2-approximation.

B   Proof of Lemma 5

First, we prove (i). Suppose it is not true, i.e., $\max\{\Delta(u), \Delta(v)\} > \min\{\Delta(p(u)), \Delta(p(v))\}$. Without loss of generality, assume $\Delta(u) \ge \Delta(v)$; then $\Delta(u) > \Delta(p(v))$. This means that the algorithm should have called the PAIRING procedure between the children of u and $p(v)$, which is a contradiction to the fact that PAIRING(u, v) is called.

Next, we prove (ii). Suppose $\Delta(v) < \Delta(u)/2$. So $\Delta(p(v)) = 2\Delta(v) < \Delta(u)$. This is again a contradiction because the algorithm should have called the PAIRING procedure between $p(v)$ and the children of u. Similarly $\Delta(v) > 2\Delta(u)$ also leads to a contradiction.

Finally, we prove (iii). Without loss of generality, suppose PAIRING(u, v) is recursively called by PAIRING$(p(u), v)$. Since the algorithm does not terminate at pair $(p(u), v)$, it must be that $\max\{\Delta(p(u)), \Delta(v)\} > \varepsilon\mathsf{d}_{\square}(p(u), v)$. So

$$\mathsf{d}_{\square}(u, v) \le \mathsf{d}_{\square}(p(u), v) + O(\Delta(u)) \le \frac{1}{\varepsilon} \max\{\Delta(p(u)), \Delta(v)\} + O(\Delta(u)),$$

which is $O(\Delta(u)/\varepsilon)$ by (ii).