A Simple Efficient Approximation Algorithm for Dynamic Time Warping

Rex Ying
Stanford University
rexying@stanford.edu

Jiangwei Pan
Duke University
jwpan@cs.duke.edu

Kyle Fox
Duke University
kylefox@cs.duke.edu

Pankaj K. Agarwal
Duke University
pankaj@cs.duke.edu

ABSTRACT

Dynamic time warping (DTW) is a widely used curve similarity measure. We present a simple and efficient $(1+\epsilon)$-approximation algorithm for DTW between a pair of point sequences, say, P and Q, each of which is sampled from a curve. We prove that the running time of the algorithm is $O(\frac{\kappa^2}{\epsilon} n \log \sigma)$ for a pair of κ-packed curves with a total of n points, assuming that the spreads of P and Q are bounded by σ. The spread of a point set is the ratio of the maximum to the minimum pairwise distance, and a curve is called κ-packed if the length of its intersection with any disk of radius r is at most κr. Although an algorithm with similar asymptotic time complexity was presented in [1], our algorithm is considerably simpler and more efficient in practice.

We have implemented our algorithm. Our experiments on both synthetic and real-world data sets show that it is an order of magnitude faster than the standard exact DP algorithm on point sequences of length $5,000$ or more while keeping the approximation error within 5–10%. We demonstrate the efficacy of our algorithm by using it in two applications — computing the k most similar trajectories to a query trajectory, and running the iterative closest point method for a pair of trajectories. We show that we can achieve 8-12 times speedup using our algorithm as a subroutine in these applications, without compromising much in accuracy.

CCS Concepts

•Theory of computation → Approximation algorithms analysis; Computational geometry; •Information systems → *Spatial-temporal systems; Information retrieval query processing;*

Keywords

Curve matching, dynamic time warping; approximation algorithm; trajectory analysis

SIGSPATIAL'16, October 31-November 03, 2016, Burlingame, CA, USA

DOI: http://dx.doi.org/10.1145/2996913.2996954

1. INTRODUCTION

Curves have become ubiquitous representations of complex shapes. They are simpler structures than the shapes they represent, and can often be used directly in computations even when processing their original shapes is infeasible. Matching similar curves is particularly useful. The notion of curve matching allows one to compare, cluster, and summarize curves, and it is used in a variety of applications, including learning and recognizing signals from speech, handwriting, fingerprints, and image features.

One application of curve matching of particular interest is in the area of trajectory analysis. Trajectories are functions from a time interval to $\mathbb{R}^d$, for $d \geq 1$, and they describe how physical systems change over time. They are sensed or inferred, often as ordered sequences of points, from a variety of sources including GPS sensors in smart phones and vehicles, surveillance videos, and the observed movement of hurricanes. Fundamental tasks for analyzing trajectory data include measuring the similarity between a pair of trajectories and computing similar portions between them. These steps are important for many applications such as subtrajectory clustering of GPS trajectories, detecting anomalous trajectories, similarity queries on trajectories, object segmentation from video trajectories [4], and smart phone authentication using touch screen trajectories [6]. This paper presents a simple and efficient algorithm for computing similarity between two curves using dynamic time warping that is easy to implement and works well in practice.

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 (p_i, q_j), $(p_{i'}, q_{j'}) \in C$ with $i' > i$, we have $j' \geq j$. In other words, the pairs of correspondences do not cross; see Figure 1. 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 computing the dynamic time warping (DTW) between P and Q, a widely used criterion [6, 9, 18, 21]. DTW computes a monotone set C of correspondences for which every point of P and Q appear at least once, and it minimizes the cost of C subject to this constraint, denoted by $\mathsf{dtw}(P, Q)$.

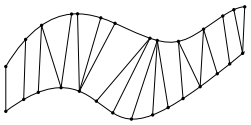


Figure 1. Example of DTW correspondences between two sequences of points sampled from two underlying curves.

It is well-known that $\mathsf{dtw}(P, Q)$ can be computed in $O(mn)$ time by a simple dynamic programming (DP) algorithm [15]. The goal of this paper is to present a fast approximation algorithm for computing $\mathsf{dtw}(P, Q)$ as well as the corresponding set of correspondences. Given $\epsilon > 0$, we call a monotone sequence of correspondences $\tilde{C}$ an *ϵ-approximate DTW*, or *ϵ-DTW* for brevity, if $\tilde{C}$ covers all points of $P \cup Q$ and its cost is at most $(1 + \epsilon)\mathsf{dtw}(P, Q)$.

Motivation. Often, the input point sequences are not very long, say, less than 100000 points, and the naive quadratic-time algorithm is computationally feasible. In many applications, however, the DTW computation is used as a routine that is invoked many times. We give two examples for which a slow DTW procedure is undesirable due to repeated DTW computation.

In the curve nearest neighbor problem [7], we have a large database of point sequences representing different types of signatures, trajectories, protein structures, etc. Given an input sequence, we wish to compute the k closest curves in our database. One method is to find the k point sequences that give the k smallest DTW values when compared against the input sequence. The database may contain thousands of sequences, and thus it is inefficient to use the naive DTW algorithm to compare the input sequence with all sequences in the database.

In many applications of DTW (e.g. analyzing video trajectories), each curve is represented in its own coordinate system, and certain transforms (e.g. translation, rotation, scaling) need to be applied on one of the input sequences before computing DTW. More specifically, let M be the family of transforms that can be applied to Q. We wish to compute $\min_{\mu \in M} \mathsf{dtw}(P, \mu(Q))$. A commonly used method to solve this problem is the iterative closest point (ICP) algorithm, which as the name suggests is an iterative method. In each iteration, ICP finds correspondences between points of the curves using similarity measures such as DTW. It then finds the transformation that minimizes the total squared distance of the correspondences between the two curves. In this task, the number of times that DTW is computed between the two curves after different transformations is equal to the number of iterations it take to converge. A fast DTW algorithm can significantly speed up the task.

Related work. Several methods have been proposed for measuring the similarity between a pair of curves, such as Fréchet distance, kernel distance, edit distance, dynamic time warping (DTW), etc. As mentioned above, DTW is used in a wide range of applications. An advantage of DTW over some other methods is that it not only computes a distance between two curves but also identifies the portion of them that are similar by returning a set of correspondences.

The standard DP algorithm for computing $\mathsf{dtw}(P, Q)$ can be viewed as constructing a weighted grid $\mathsf{V} = \{(i, j) | 1 \leq i \leq m, 1 \leq j \leq n\}$, where the weight $w(i, j)$ of (i, j) is $||p_i q_j||$. Each grid point has outgoing edges to $(i, j+1)$, $(i+1, j)$, and $(i + 1, j + 1)$. The algorithm computes a minimum-weight path from $(1, 1)$ to (m, n) in this grid graph using dynamic programming (DP).

Because of the popularity of DTW, several heuristics have been proposed to expedite the DTW computation. For example, the commonly used library of Giorgino [12] uses constraints such as the Sakoe-Chiba Band [22] that only constructs a small "band" around the diagonal of the DP grid and restricts the search within this band. This approach is effective if the minimum-cost path stays near the diagonal; otherwise, it is as slow as the original DP algorithm or it does not compute an optimal or near optimal path from $(1, 1)$ to (m, n). Al-Nayamat [3] propose another heuristic that roughly speaking computes the entries of the DP grid in a lazy manner — initially only a few entries are computed for each point in one of the sequences and more entries are computed as needed. Although this heuristic speeds up the DTW computation by a factor of two on certain inputs, in general its running time is quadratic.

Recently, the authors [1] proposed an algorithm that computes an ϵ-DTW in near-linear time, assuming that the input sequences are "well-behaved", i.e., sampled from certain well-behaved curves such as κ-packed or κ-bounded curves (see below and [1] for the definitions of these curves). Not withstanding the $O(n \log n)$ running time in the worst case for κ-packed and κ-bounded curves with constant values of κ and ϵ, the algorithm is not efficient in practice compared with the DP algorithm until the input sequences are long enough, say $n \geq 10^6$. See Section 2 for a more detailed discussion on this algorithm.

Our results. Let P and Q be two point sequences in $\mathbb{R}^d$ as defined above, and let $\epsilon > 0$ be a parameter. Our main contribution is a simple, efficient algorithm for computing an ϵ-DTW of P and Q. It is effective in practice, even for moderately sized point sequences, and we can also mathematically bound its running time. Before stating the main result, we define two terms. For a point set X, its spread, denoted by $\mathsf{sp}(X)$, is the ratio of the maximum and the minimum pairwise distances of X. Given $\kappa > 0$, 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 [8]. We say a point sequence P is κ-packed if the polygonal curve that connects points of P is κ-packed. We now state our result:

THEOREM 1.1. *Let P and Q be two point sequences in $\mathbb{R}^d$ of length at most n each such that $\mathsf{sp}(P), \mathsf{sp}(Q) \leq \sigma$, and let $\epsilon > 0$ be a parameter. An ϵ-DTW of P and Q can be computed in $O(\frac{\kappa^2}{\epsilon} n \log \sigma)$ time if P and Q are κ-packed.*

In practice $\sigma = n^{O(1)}$, in which case the running time is $O(\frac{\kappa^2}{\epsilon} n \log n)$. Furthermore, the value of κ is experimentally observed to be a small constant for GPS trajectories (see Section 4).

Although the overall structure of the algorithm is similar to that in [1], a number of new ideas are needed to obtain a simpler and more efficient algorithm. As in [1], our technique is general enough to be used for other similarity measures such as edit distance.

We have implemented our algorithm. Our experimental results on both synthetic and real-world GPS trajectories show that our algorithm is roughly an order of magnitude faster than the DP algorithm even on point sequences of

length 5000, while returning an output whose approximation error is within 5-10%. The improvement is even more significant on larger sequences. We apply our algorithm to two different problems: computing the k most similar trajectories to a query trajectory, and running the iterative closest point method for a pair of trajectories. We show that we can achieve 8-12 times speedup using our algorithm as subroutine in these applications, without compromising much accuracy.

2. APPROX-DTW ALGORITHM

Let $P = \langle p_1, \ldots, p_m \rangle$ and $Q = \langle q_1, \ldots, q_n \rangle$ be two point sequences in $\mathbb{R}^d$. Let $\varepsilon \in (0,1)$ be a parameter. We present the ε-approximation algorithm of [1] for computing $\mathsf{dtw}(P,Q)$, which we refer to as Approx-DTW. Without loss of generality, assume that $m \leq n$ and $\varepsilon \geq 1/n$. If $\varepsilon < 1/n$, we can simply compute $\mathsf{dtw}(P,Q)$ 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 $\omega : \mathsf{V} \to \mathbb{R}_{\geq 0}$ 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 $i \geq i'$ and $j \geq j'$. 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 $\mathsf{dtw}(P,Q) = \mu(m,n)$.

For $1 \leq i_1 \leq i_2 \leq m$ and for $1 \leq j_1 \leq j_2 \leq n$, 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 $j \in \{j_1, j_2\}$. For a rectangle R, let $\partial\mathsf{R}$ denote the set of its boundary points.

We now outline the Approx-DTW algorithm.

Algorithm Approx-DTW(P,Q)

1. Compute an estimate $\underline{\mathsf{d}}$ of $\mathsf{dtw}(P,Q)$ such that $\underline{\mathsf{d}} \leq \mathsf{dtw}(P,Q) \leq 4n\underline{\mathsf{d}}$. Let $\overline{\mathsf{d}} = 4n\underline{\mathsf{d}}$.
2. Compute a set $\mathcal{R}$ of (possibly overlapping) rectangles and a weight ω_{R} for each rectangle $\mathsf{R} \in \mathcal{R}$, such that:
   (a) for all $\mathsf{R} \in \mathcal{R}$ and $(i,j) \in \mathsf{R}$, $|\omega(i,j) - \omega_{\mathsf{R}}| \leq \frac{\varepsilon}{2} \max\{\omega_{\mathsf{R}}, \underline{\mathsf{d}}/2n\}$,
   (b) if $(i,j) \in \mathsf{V}$ and $\omega(i,j) \leq \overline{\mathsf{d}}$, then there exists a rectangle $\mathsf{R} \in \mathcal{R}$ such that $(i,j) \in \mathsf{R}$.
3. Let $\mathcal{B} = \bigcup_{\mathsf{R} \in \mathcal{R}} \partial\mathsf{R}$ be the set of boundary points of the rectangles in $\mathcal{R}$.
   For every $(i,j) \in \mathcal{B}$, compute a value $\tilde{\mu}(i,j)$ such that $|\mu(i,j) - \tilde{\mu}(i,j)| \leq \epsilon\mu(i,j)$. Return $\tilde{\mu}(m,n)$.

Step 1 requires computing the approximate Fréchet distance as a subroutine. Although it takes $O(\kappa n \log n)$ time for κ-packed curves in theory, this step is expensive in practice.

[1]Note that in this paper, a point can refer to either a grid point in V or a sequence point from $P \cup Q$.

In Step 2, the algorithm stores points in P, Q in a quadtree and uses a procedure similar to the well-separated pair decomposition (WSPD) of [5] (see also [13]). This step uses quadtree boxes as the bounding volumes of subsequences of P and Q—often very loose bounding volumes. It is inefficient in practice and generates many more rectangles than necessary.

Each rectangle in $\mathcal{R}$ is specified by long contiguous subsequences of P and Q as directed by their intersections with the quadtree boxes. In this process, the rectangles generated may overlap with each other, because a contiguous subsequence for a quadtree box is allowed to travel out of the box, as long as it is within the box's neighborhood.

To compute the optimal path in the union of overlapping rectangles, the algorithm uses a dynamic data structure for answering range-min queries in $O(1)$ time which is too complicated to be used in practice.

Finally, although in [1] it is proved that the size of $\mathcal{B}$ constructed via the pairing procedure is bounded from above by $O(\frac{\kappa}{\epsilon} n \log n)$ for κ-packed curves, its hidden constant is observed to be large in practice. See Section 4.

3. THE SIMPLIFIED ALGORITHM

We now describe a simple ϵ-approximation algorithm for computing $\mathsf{dtw}(P,Q)$. We show that its running time is $O(\frac{\kappa^2}{\epsilon} n \log \sigma)$ if both P and Q are κ-packed and their spreads are bounded by σ. We first present an outline of the algorithm and then describe each of its steps in detail.

Algorithm FA-DTW(P,Q)

1. Compute a partition $\mathcal{R}$ of V into pairwise-disjoint rectangles and a weight ω_{R} for each rectangle $\mathsf{R} \in \mathcal{R}$ such that
$$|\omega(i,j) - \omega_{\mathsf{R}}| \leq \frac{\epsilon}{2}\omega_{\mathsf{R}} \quad \forall (i,j) \in \mathsf{R}, \forall \mathsf{R} \in \mathcal{R}. \qquad (1)$$
2. Let $\mathcal{B} = \bigcup_{\mathsf{R} \in \mathcal{R}} \partial\mathsf{R}$ be the set of boundary points of the rectangles in $\mathcal{R}$.
   For every $(i,j) \in \mathcal{B}$, compute a value $\tilde{\mu}(i,j)$ such that
$$|\mu(i,j) - \tilde{\mu}(i,j)| \leq \epsilon\mu(i,j). \qquad (2)$$
   Return $\tilde{\mu}(m,n)$.

Before describing the steps in detail, we note that the condition in Step 1 is simpler than the corresponding one stated in Section 2. A nice consequence of this simplification is that we do not have to estimate the value of $\mathsf{dtw}(P,Q)$ with $\underline{\mathsf{d}}$ and $\overline{\mathsf{d}}$ as in Section 2. Furthermore, it greatly simplifies the rectangle generation step. The cost we pay for this simplification is that the running time is $O(\frac{\kappa^2}{\epsilon} n \log \sigma)$ instead of $O(\frac{\kappa}{\epsilon} n \log n)$. In practice however, $\sigma = n^{O(1)}$, and therefore $\log \sigma = O(\log n)$. Furthermore, Step 1 generates pairwise-disjoint rectangles, which significantly simplifies Step 2. No complex data structures are needed to compute each $\tilde{\mu}(i,j)$.

3.1 Generating Rectangles

We now describe the algorithm for constructing the set $\mathcal{R}$ of rectangles. As mentioned earlier, this step is significantly simpler than that in [1] and leads to a more efficient

algorithm in practice. Roughly speaking, instead of working with a quadtree on $P \cup Q$, we build a bounding volume hierarchy independently on each of P and Q and use these hierarchies to construct the set $\mathcal{R}$. We now describe this step in detail.

We construct a tree $\mathcal{T}(P)$ on P in a top-down manner. Each node u of $\mathcal{T}(P)$ is associated with a contiguous subsequence P_u of P. If u is a leaf then $|P_u| = 1$, and if u is the root then $P_u = P$. If $P_u = \langle p_i, \ldots, p_j \rangle$ then we set $I_u = [i : j]$ and $\lambda_u = \sum_{k=i}^{j-1} ||p_k p_{k+1}||$. If $j > i$, i.e., u is an internal node, then let $r \in [i : j]$ be the largest index such that $\sum_{k=i}^{r-1} ||P_k P_{k+1}|| \leq \lambda_u/2$. We create two children v and z of u and set $P_v = \langle P_i, \ldots, P_r \rangle$ and $P_z = \langle P_{r+1}, \ldots, P_j \rangle$. The following lemma is straightforward:

LEMMA 3.1. *$\mathcal{T}(P)$ has $O(|P|)$ size and $O(\log \sigma)$ depth, and it can be constructed in $O(n)$ time.*

For each node $u \in \mathcal{T}(P)$, let B_u be a bounding volume of P_u, i.e., a simple-shaped region that contains P_u. We will consider B_u to be the minimum enclosing ellipsoid or the minimum enclosing box of P_u. The minimum enclosing ellipsoid can be constructed in linear time using a fixed-dimensional linear programming algorithm [17]. Although computing a minimum enclosing box is expensive, an ϵ-approximation can be computed very efficiently using a core-set [2]. We omit the details of computing B_u from here, and simply assume B_u can be computed in $O(|P_u|)$ time. Let Δ_u be the diameter of B_u. We can either compute B_u in advance for all $u \in \mathcal{T}(P)$ or compute in a lazy manner – computing it the first time it is needed. Similarly we compute an analogous tree $\mathcal{T}(Q)$ on Q. For a pair of nodes $u \in T(P)$, $v \in T(Q)$, we define $\mathsf{R}_{uv} = I_u \times I_v$.

After having constructed the trees $\mathcal{T}(P)$ and $\mathcal{T}(Q)$, we generate the rectangles of $\mathcal{R}$ using the recursive procedure GEN_RECTANGLES(u, v) described below, where $u \in \mathcal{T}(P)$ and $v \in \mathcal{T}(Q)$. Initially, we call GEN_RECTANGLES(root_P, root_Q) where root_P (resp. root_Q) is the root of $\mathcal{T}(P)$ (resp. $\mathcal{T}(Q)$) and set $\mathcal{R} = \varnothing$. GEN_RECTANGLES$(u, v)$ uses a procedure SEPARATED(u, v), which returns YES if the ratio of the maximum and the minimum distance between any pair of points in B_u and B_v is at most $(1+\varepsilon/2)$, and NO otherwise. The SEPARATED procedure takes $O(1)$ time if the bounding volumes are boxes or ellipsoids. In the code below, $\mathsf{ch}(u)$ refers to the children of tree node u.

```
GEN_RECTANGLES(u, v)
if SEPARATED(u, v)
    R = R ∪ {R_uv}; return
if Δ_u ≥ Δ_v
    ∀w ∈ ch(u)    GEN_RECTANGLES(w, v)
else ∀z ∈ ch(v)    GEN_RECTANGLES(u, z)
```

It is clear from the recursive procedure that it generates a partition of V into rectangles. For each rectangle $\mathsf{R} \in \mathcal{R}$, we set ω_{R} to be the value of an arbitrary node in R. Since every rectangle $\mathsf{R} \in \mathcal{R}$ is defined by two well-separated subsequences, it is easy to show that $\mathcal{R}$ satisfies condition (1).

3.2 Computing Paths to Boundary Points

We now describe the algorithm for computing $\tilde{\mu}(i, j)$ for all $(i, j) \in \mathcal{B}$. The algorithm consists of two steps. The first step sorts the rectangles of $\mathcal{R}$ in the order in which they will be processed. The second step computes the values $\tilde{\mu}(i, j)$.

Ordering Rectangles. Let $\mathcal{R} = \{\mathsf{R}_1, \ldots, \mathsf{R}_s\}$. We construct a directed graph $H = (X, \Gamma)$ where $X = \{1, \ldots, s\}$ and (u, v), for $1 \leq u \neq v \leq s$, is an edge in Γ if there are nodes $(i, j) \in \partial\mathsf{R}_u$ and $(i', j') \in \partial\mathsf{R}_v$ such that $((i, j), (i', j'))$ is an edge in the original grid graph G on V.

LEMMA 3.2. *(i) H is a planar directed acyclic graph. (ii) Let $\mathsf{R}_u, \mathsf{R}_v \in \mathcal{R}$ be two rectangles such that a path in G from $(1, 1)$ to a node of R_v passes through a node of R_u, then there is a path from u to v in H.*

Lemma 3.2 implies that a topological sorting of vertices of H gives a desired ordering of rectangles in $\mathcal{R}$, namely, if a path in G from s to a point of R_v passes through R_u, then R_u appears before R_v in the ordering.

The coordinates of the grid points within $\mathcal{B}$ all lie in $[m] \times [n]$, so the points of $\mathcal{B}$ can be sorted in $O(|\mathcal{B}|)$ time. Afterward, we can compute the edges of G whose both endpoints are boundary points. Next, we scan these edges to construct the edges of H using the criterion specified in the definition of H. We then topologically sort the vertices of H to compute a linear ordering on the rectangles $\mathcal{R}$.

Let $\mathcal{R} = \langle \mathsf{R}_1, \ldots, \mathsf{R}_s \rangle$ be the resulting sequence of rectangles. We process the rectangles of $\mathcal{R}$ in this order. Suppose we have processed $\mathsf{R}_1, \ldots, \mathsf{R}_{u-1}$, i.e. we have computed $\tilde{\mu}(i, j)$ for all $(i, j) \in \mathcal{B}_{u-1} = \bigcup_{i=1}^{u-1} \partial\mathsf{R}_i$. Let $\mathsf{R}_u = [x^- : x^+] \times [y^- : y^+]$ be the next rectangle in $\mathcal{R}$. Without loss of generality, assume that $x^+ - x^- \geq y^+ - y^-$. We first compute $\tilde{\mu}(\cdot)$ for points on the left and bottom boundaries of R_u and then on the right and top boundaries.

Left and bottom boundaries. All three incoming neighbors of (x^-, y^-) belong to $\mathcal{B}_{u-1}$, so $\tilde{\mu}(x^-, y^-)$ can be computed in $O(1)$ time. For a point (x^-, j) with $j \in (y^- : y^+]$, we have $(x^- - 1, j - 1), (x^- - 1, j) \in \mathcal{B}_{u-1}$. So each $\tilde{\mu}(x^-, j)$, for $j \in (y^- : y^+]$, can be computed in $O(1)$ time by iteratively increasing the value of j. Similarly, $\tilde{\mu}(i, y^-)$, for $i \in (x^- : x^+]$, can be computed in $O(1)$ time each. Hence, left and right boundaries of R_u can be processed in a total of $O(|\partial\mathsf{R}_u|)$ time.

Right boundary. We fix a point (x^+, j), for $j \in [y^-, y^+]$, on the right boundary of R_u. Let $N(j) \subset \partial\mathsf{R}_u$ be the set of points dominated by (x^+, j). For $(a, b) \in N(j)$, let $\lambda(a, b)$ be the length of the shortest path (i.e., with the minimum number of grid points) from (a, b) to (x^+, j) in R_u. Then we set

$$\tilde{\mu}(x^+, j) = \min_{(a,b) \in N(j)} \tilde{\mu}(a, b) + \lambda(a, b)\omega_{\mathsf{R}}. \tag{3}$$

The computation of $\tilde{\mu}(x^+, j)$ is greatly simplified by partitioning $N(j)$ into three subsets N_1, N_2 and N_3 as defined below; see Figure 2 (a).

(i) $N_1 = \big\{(x^-, l) \mid l \in [y^- : j]\big\}$. For all points $(a, b) \in N_1$, we have $\lambda(a, b) = x^+ - x^-$. For $j \in [y^- : y^+]$, let $\phi_j = \min_{y^- \leq l \leq j} \tilde{\mu}(x^-, l)$. We have

$$\min_{(a,b) \in N_1} \tilde{\mu}(a, b) + \lambda(a, b)\omega_{\mathsf{R}} = \phi_j + (x^+ - x^-)\omega_{\mathsf{R}}.$$

We note that $\phi_j = \min\{\phi_{j-1}, \tilde{\mu}(x^-, j)\}$. Hence, ϕ_j can be computed from ϕ_{j-1} in $O(1)$ time.

(ii) $N_2 = \big\{(i, y^-) \mid i \in [x^- : x^+ - (j - y^- - 1)]\big\}$. There is a shortest path from (a, b) to (x^+, j) that passes through the point $(x^+ - (j - y^-) - 1, y^-)$. Hence

$$\min_{(a,b) \in N_2} \tilde{\mu}(a, b) + \lambda(a, b) = \tilde{\mu}(x^+ - (j - y^-) - 1, y^-) + (j - y^- + 1)\omega_{\mathsf{R}}.$$

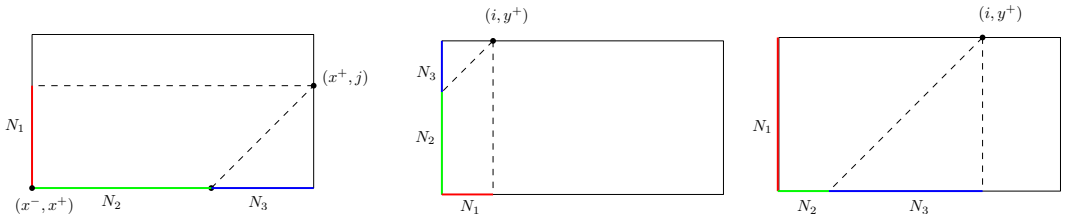


Figure 2. Partition of boundary points dominated by point on: (a) right edge, (b) top edge with diagonal intersecting the left edge, (c) top edge with diagonal intersecting the bottom edge.

(iii) $N_3 = \{(i, y^-) \mid i \in [x^+ - (j + y^-) : y^+]\}$. There is a shortest path from $(a, b) \in N_3$ to (x^+, j) that passes through the point $(x^+, j-1)$. Hence

$$\min_{(a,b)\in N_3} \tilde{\mu}(a, b) + \lambda(a, b) = \tilde{\mu}(x^+, j-1) + \omega_{\mathsf{R}}.$$

Putting everything together,

$$\begin{aligned}\tilde{\mu}(i, j) = \min\{&\phi_j + (x^+ - x^-)\omega_{\mathsf{R}},\\ &\tilde{\mu}(x^+ - (j - y^-) - 1, y^-) + (j - y^- + 1)\omega_{\mathsf{R}},\\ &\tilde{\mu}(x^+, j-1) + \omega_{\mathsf{R}}\}. \end{aligned} \quad (4)$$

Overall, the algorithm spends $O(1)$ time for each value of $j \in [y^-, y^+]$.

Top boundary. There are two cases depending on whether the ray from (i, j) in the southwest direction intersects the left or the bottom boundary of R_u, i.e. whether $i \le x^- + (y^+ - y^-)$.

Case (i): $i \le x^- + (y^+ - y^-)$. See Figure 2 (b). This case is symmetric to the right boundary case. We can therefore use the same procedure as in the case of the right boundary (with the first and the second coordinates switched).

Case (ii): $i > x^- + (y^+ - y^-)$. Again, we partition $N(j)$ into three subsets

$$\begin{aligned} N_1 &= \{(x^-, j) \mid j \in [y^- : y^+]\},\\ N_2 &= \{(l, y^-) \mid l \in (x^- : i - (y^+ - y^-))\},\\ N_3 &= \{(l, y^-) \mid l \in (i - (y^+ - y^-) : i]\} \end{aligned}$$

See Figure 2 (c).

For all $(a, b) \in N_1 \cup N_2$, there is a shortest path from (a, b) to (i, y^+) that passes through $(i-1, y^+)$. Hence the best path from a point in $N_1 \cup N_2$ to (i, j) in $N_1 \cup N_2$ has value $\tilde{\mu}(i-1, j^+) + \omega_{\mathsf{R}}$.

For all $(a, b) \in N_3$, we compute $\min_{(a,b)\in N_3} \tilde{\mu}(a, b) + (y^+ - y^-)\omega_{\mathsf{R}}$, as follows: The set N_3 shifts to the right as (i, j) moves to the right along the top edge. Computing the minimum in N_3 requires maintaining the minimum of the set of items in a "window" of length $y^+ - y^-$ as we slide the window. This task can be accomplished as follows. We maintain the subsequence $Q = \langle (i_1, y^-), (i_2, y^-), \ldots, (i_r, y^-)\rangle$ of N_3 such that (i) $i_1 > i - (y^+ - y^-)$, (ii) $\tilde{\mu}(i_l, y^-) < \tilde{\mu}(i_{l+1}, y^-)$ for $1 \le l < r$, and (iii) for all $i_{l-1} < b \le i_l - 1$ (we assume $i_0 = i - (y^+ - y^-)$), we have $\tilde{\mu}(b, y^-) \ge \tilde{\mu}(i_l, y^-)$. Then,

$$\min_{(a,b)\in N_3} \tilde{\mu}(a, b) = \tilde{\mu}(i_1, y^-). \quad (5)$$

When we process $(i+1, y^-)$, we remove (i_1, y^-) from Q if $i_l = i + 1 - y^+ + y^-$. Next, we scan Q backwards and remove the current item (i_l, y^-) being scanned if $\tilde{\mu}(i_l, y^-) \ge \tilde{\mu}(i+1, y^-)$. Finally, we insert $(i+1, y^-)$ into Q as its last item. The time spent in updating Q while processing p_i is $O(1 + k_i)$ where k_i is the number of items deleted from Q while processing p_i. Since each item is deleted at most once, the total running time is $O(x^+ - x^-)$.

Hence, $\tilde{\mu}(i, j)$, for all $(i, j) \in \partial\mathsf{R}_u$, can be computed in $O(|\partial\mathsf{R}_u|)$ time. Summing over all rectangles of $\mathcal{R}$, the total time spent is $O(|\mathcal{B}|)$. We reiterate that the procedure is quite simple and does not rely on any complicated data structures. After having computed $\tilde{\mu}(m, n)$, $\tilde{\Pi}$ can be computed in $O(m + n)$ time, using a straightforward backtracking procedure.

3.3 Running Time Analysis

We bound the number of boundary points of $\mathcal{R}$ for well behaved curves. The same argument will also bound the time spent in generating $\mathcal{R}$. For simplicity, we assume that each node of $\mathcal{T}(P)$ or $\mathcal{T}(Q)$ maintains the minimum enclosing box of points associated with that node. For each node $u \in \mathcal{T}(P)$, let $m_u = |P_u|$, let λ_u be the arc length of the sequence P_u, let B_u be the minimum enclosing box of P_u, and let $\Delta(u)$ be the diameter of B_u. Similarly we define $n_v = |Q_v|$ for a node v of $\mathcal{T}(Q)$; λ_v, B_v, and $\Delta(v)$ are defined analogously. Since P and Q are κ-packed, for any node $v \in \mathcal{T}(P) \cup \mathcal{T}(Q)$,

$$\Delta_u \le \lambda_u \le \kappa\Delta_u. \quad (6)$$

For a pair of nodes $u \in \mathcal{T}(P)$ and $v \in \mathcal{T}(Q)$, let $d(u, v) = \min_{a\in B_u, b\in B_v} ||ab||$. Let $p(u)$ be the parent of tree node u. The following lemma can be proven using similar arguments as those given for WSPDs [5, 13].

LEMMA 3.3. *Let R_{uv} be a rectangle in $\mathcal{R}$ for some $u \in \mathcal{T}(P)$ and $v \in \mathcal{T}(Q)$. Then*
(i) $\max\{\Delta(u), \Delta(v)\} \le \min\{\Delta(p(u)), \Delta(p(v))\}$.
(ii) There is a constant $c \ge 0$ such that $d(u, v) \le \frac{c\kappa}{\epsilon}\Delta(u)$.

LEMMA 3.4. $|\mathcal{B}| = O(\frac{\kappa^2}{\epsilon} n \log\sigma)$.

PROOF. For a node $u \in \mathcal{T}(P)$, let ν_u be the number of rectangles R_{uv} in $\mathcal{R}$ for some $v \in \mathcal{T}(Q)$. Similarly we define ν_v for a node $v \in \mathcal{T}(Q)$. Then

$$|\mathcal{B}| \le \sum_{\mathsf{R}_{uv}\in\mathcal{R}} 2(m_u + n_v) = \sum_{u\in\mathcal{T}(P)} 2m_u\nu_u + \sum_{v\in\mathcal{T}(Q)} 2n_v\nu_v. \quad (7)$$

We prove below in Lemma 3.5 that $\nu_u, \nu_v = O(\frac{\kappa^2}{\epsilon})$ for any $u \in \mathcal{T}(P)$ and $v \in \mathcal{T}(Q)$. Since $\sum_{u\in\mathcal{T}(P)} m_u = O(m \log \sigma)$ and $\sum_{v\in\mathcal{T}(Q)} n_v = O(n \log \sigma)$, we obtain

$$|\mathcal{B}| = O((m+n)\tfrac{\kappa^2}{\epsilon} \log \sigma) = O(\tfrac{\kappa^2}{\epsilon} n \log \sigma).$$

□

LEMMA 3.5. *For any node $u \in \mathcal{T}(P) \cup \mathcal{T}(Q)$, $\nu_u = O(\frac{\kappa^2}{\epsilon})$.*

PROOF. We prove the bound for a node $u \in \mathcal{T}(P)$. Let $N(u) = \{\mathsf{R}_{uv} \in \mathcal{R} \mid v \in \mathcal{T}(Q)\}$; $\nu_u = |N(u)|$. Let O be the center of the box B_u, and let B be the ball of radius $c' \frac{\kappa}{\epsilon} \Delta(u)$ for some sufficiently large constant c'.

By Lemma 3.3, if $\mathsf{R}_{uv} \in N(u)$ then $d(u, v) \leq \frac{c\kappa}{\epsilon}\Delta(u)$, and

$$\Delta(v) \leq \Delta(p(u)) \leq \lambda_{p(u)} \leq 2\lambda_u \leq 2\kappa\Delta(u).$$

Therefore the box $B_v \subseteq \mathsf{B}$.

We note that the boxes B_v for $B_v \in N(u)$ are pairwise disjoint. Since Q is κ-packed,

$$\sum_{v\in N(u)} \lambda_v \leq \kappa c' \frac{\kappa}{\epsilon}\Delta(u) = c' \frac{\kappa^2}{\epsilon}\Delta(u).$$

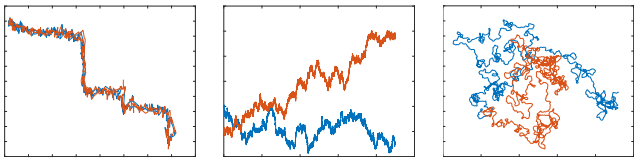
On the other hand, by (6) and Lemma 3.3, $\lambda_v \geq \frac{\lambda_{p(v)}}{2} \geq \frac{\lambda_u}{2} \geq \frac{\Delta_u}{2}$.

Therefore, $\nu_u = |N_u| \leq \frac{c'\kappa^2}{\epsilon}\Delta_u / \frac{\Delta_u}{2} = O(\frac{\kappa^2}{\epsilon})$. □

The overall running time is thus $O(\frac{\kappa^2}{\epsilon} n \log \sigma)$, thereby proving Theorem 1.1.

The analysis works even if we use another bounding volume such as ball or ellipsoid instead of bounding box. However a tighter bounding volume will give a better performance in practice, which is why we chose bounding boxes.

4. EXPERIMENTAL RESULTS



(a) (b) (c)

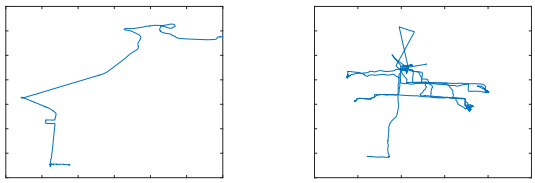
Figure 3. Synthetic trajectories: (a) similar underying curves; (b) Martingale sequences; (c) random walks.

We present our experimental results that evaluate the efficacy and efficiency of our algorithm on both synthetic and real-world data sets. We first visually compare the quality of the output of our algorithm with the optimal solution as well as with a few other algorithms (Section 4.1), and then quantitatively compare the performance of our algorithm with others (Section 4.2). We also study how the performance (both accuracy and efficiency) of our algorithm changes as we vary the parameter ϵ (Section 4.3). Finally, we demonstrate the advantage of our algorithm over the standard DP algorithm by applying it two problems in which DTW is computed repeatedly (Section 4.4).

Datasets. The datasets we used consist of synthetic curves of length 500 to 20000 and real-world trajectories in the GeoLife GPS trajectories dataset [24].

The synthetic curve dataset is organized into the following three categories.

1. *Similar curves:* pairs of curves sampled from the same underlying curve, with different sampling rates and additive Gaussian noise. The standard deviation of the noise is equal to 0.3% of the length of the underlying curve. The underlying curves chosen include parametric curves as well as the spline-interpolated curves in the GeoLife dataset (Figure 3 (a)). The estimated κ value is 4.8.
2. *Martingale sequence:* x-monotone curves with the y values determined by Martingale sequences. The x-values are uniformly sampled. Each y-value is sampled from a normal distribution whose mean is equal to its previous y-value (Figure 3 (b)). The estimated κ value is 3.8.
3. *Random walk:* at each step, the curves turn a normally distributed random angle and move for a normally distributed random distance. These curves simulate Brownian motion in 2D. The mean and standard deviation for the angle and distance vary for different input sequences (Figure 3 (c)). The estimated κ value is 13.7.



(a) (b)

Figure 4. Examples of GeoLife trajectories with (a) low and (b) high κ parameter.

In the GeoLife GPS trajectories dataset, trajectories are represented by a sequence of time-stamped points in 3D, representing latitude, longitude, and altitude information. The number of points of trajectories in the dataset range from 100 to 30000. For our experiments, we only keep the latitude and longitude information of each sample, and represent them as points in $\mathbb{R}^2$.

We estimated the κ parameter of 500 real-world trajectories from the GeoLife dataset, and the percentiles of κ values are shown in Table 1. We note that most of the trajectories, such as the one shown in Figure 4 (a), have low κ parameters. In contrast, the κ parameter for a small portion of trajectories such as in Figure 4 (b) is larger than 100. However, in most trajectories with high κ parameters, only a very small portion of the curve is dense. Recall that κ is the *maximum* ratio between the length of a curve in a ball and the radius of the ball. If we instead compute the average ratio, $\tilde{\kappa}$, it is below 20 for all the curves. Our algorithm works well as long as $\tilde{\kappa}$ is small even if κ is large.

Base-line algorithms. We compare our algorithm FA-DTW with the following algorithms:

1. DP-DTW: the naive dynamic programming algorithm for DTW;

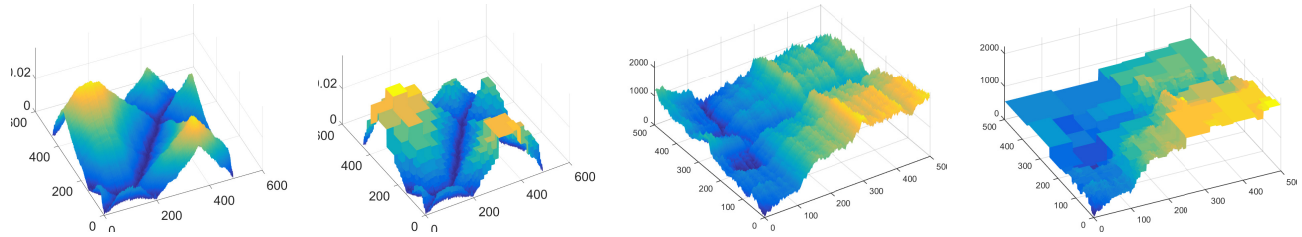


Figure 5. (a) PDS for a pair of similar curves (category 1); (b) corresponding APDS for $\epsilon = 1$; (c) PDS for a pair of random walk curves (category 3); (d) corresponding APDS for $\epsilon = 1$.

Percentiles	25%	50%	75%	90%	max
κ	6.1	9.3	17.9	25.4	114.9
$\tilde{\kappa}$	2.4	5.7	8.0	9.4	15.9

Table 1. Percentiles of the κ parameter for sampled input trajectories in GeoLife.

2. Band-DTW: the dynamic programming algorithm with Sakoe-Chiba band heuristic [22];
3. Approx-DTW: the algorithm introduced in [1];
4. Fréchet: the dynamic programming algorithm for computing Fréchet distance.

4.1 Output Visualization

The values in the dynamic programming table of the exact DTW computation form a surface with domain $[1, m] \times [1, n]$, which we call the *pairwise-distance surface* (PDS). The height of a point (i, j) on the surface equals the distance $||p_i - q_j||$. The DP-DTW algorithm thus computes an admissible path Π^* from $(1, 1)$ to (m, n) that minimizes the sum of the heights of all $(i, j) \in \Pi^*$. Our approximation algorithm finds a rectangular partition of the DP table, and assigns the same weight to the cells in each rectangle. This results in an approximate PDS (APDS) that consists of flat rectangular patches.

Figure 5 plots the PDS/APDS of two pairs of synthetic sequences (Figure 3(a, c)) as 3D surfaces. Note the deep trenches in Figure 5(a,b) that contain the (approximately) optimal admissible paths. Clearly, the APDS computed by FA-DTW approximates the PDS well. In regions where the distance variation is large, more rectangular patches are used; whereas in regions where distances are similar, only a few rectangular patches are sufficient to approximately describe the PDS.

We also visualize the correspondences computed by different algorithms in the 2D heatmap of the APDS (Figure 6). More specifically, the heatmap is visualized by coloring the cells in each rectangle computed by FA-DTW with the same color. A darker color at cell (i, j) corresponds to a smaller value of $||p_i q_j||$. The correspondences are plotted as admissible paths from $(1, 1)$ to (m, n) in the heatmap.

We observe that the paths computed by FA-DTW and DP-DTW mostly stay relatively close together. The Fréchet distance path, however, may be far from the DP-DTW path, and even contain long horizontal and vertical segments in areas where diagonal segments may result in fewer correspondences. Particularly, the FA-DTW path is closest to the DP-DTW path in darker regions where there are many small rectangles such as the upper-left region. In regions of large rectangles, the FA-DTW path can be less accurate. Fréchet does a particularly poor job approximating the path for DP-DTW when the input curves are not very similar (Figure 6(b)) as Fréchet is only concerned with the maximum distance between corresponding points and has no incentive to create a low weight path if even a single correspondence must have high weight anyway. Finally, we note that Band-DTW generates the same paths as DP-DTW in these example assuming the band width is sufficiently large.

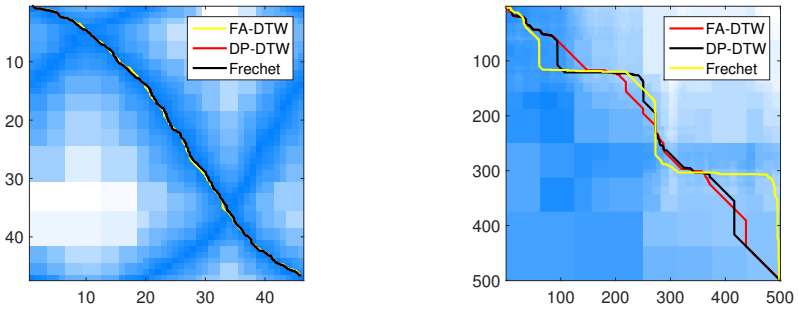
4.2 Quantitative Results

Comparison with DP-DTW. We use the running time of DP-DTW as a baseline, and compare it to the running time of FA-DTW for different input sequence categories in our synthetic dataset. In this experiment, we fix the value of ϵ to be 1. In Table 2, we note the average running time of DP-DTW across all inputs of each size; the running time of the algorithm is only sensitive to input size, not the actual data. The running time of FA-DTW, however, depends on the κ parameter of different input categories. Therefore, we note the running time of FA-DTW for increasing sizes of the input sequences of each category. In our experiments, we found that curves from the random walk category have the highest κ values in the synthetic dataset. This is confirmed by the runtime of FA-DTW on the random walk curves (category 3), which is higher than the other categories. The actual approximation error is much smaller than the ϵ parameter (see Section 4.3). In particular, the average approximation error is 9.5%; 90% of the approximate DTW values have error less than 17.4%, and the worst approximation error we observed is 28.3%.

It is clear that FA-DTW has a significant advantage for all synthetic input categories: its running time is an order of magnitude lower than that of DP-DTW for input sizes above 5000. It is also easy to verify that the running time of FA-DTW grows near-linear to the input size, consistent with our theoretical analysis.

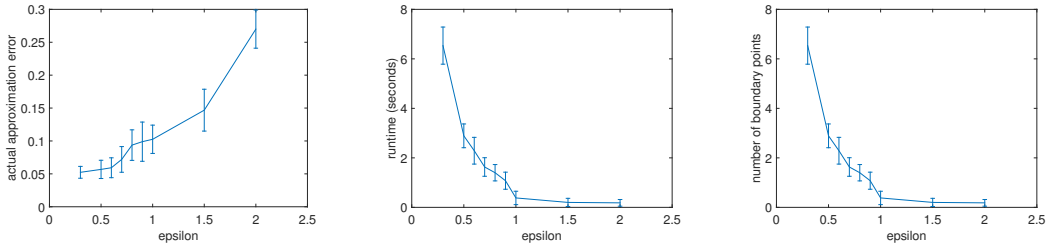
Size	FA-DTW for category			DP-DTW
	1	2	3	
1000	0·039	0·042	0·026	0.031
2000	0·093	0·090	0·084	0.096
5000	0·087	0·190	0·183	0.551
9000	0·429	0·412	0·535	1.811
15000	0·773	0·934	1·080	5.649
20000	1·248	1·236	1·391	11.713

Table 2. Comparison of runtimes (in seconds) for DP-DTW and FA-DTW for different input sequence categories.



(a) (b)

Figure 6. Rectangles and correspondences as a path on the APDS heatmap for: (a) two similar curves (category 1), and (b) two random walk sequences (category 3).



(a) (b) (c)

Figure 7. (a) Actual approximation error against ϵ parameter; (b) tradeoff between runtime and ϵ parameter for synthetic input datasets; (c) number of boundary points vs. ϵ parameter.

Comparison with Band-DTW. FA-DTW is also advantageous over the popular Band-DTW heuristic. The running time and error of Band-DTW depends on the band width parameter. For input size 5000, we observed that when the band is around 100-150 wide, the running time of Band-DTW is comparable to that of FA-DTW. However, we observed two synthetic input sequences of category 1 with wildly different sampling rates; the correspondence path was very far away from the diagonal in the DP table.

In fact, the DTW value computed using Band-DTW for these sequences was more than 3 times the exact DTW value, which is much worse than FA-DTW. Furthermore, the number of boundary points generated by FA-DTW was only 9.68×10^5, about 0.4% of the cells in the DP table. In contrast, to completely cover the optimal DTW correspondence path using a diagonal band in the DP table, we needed to compute the values of as many as 1.39×10^7 cells, which is more than half of the total number of cells.

In real trajectory datasets, it is common that the underlying curve is not uniformly sampled, due to varying travel speed of objects such as vehicles. In these cases, FA-DTW is more desirable than Band-DTW because it does not have parameters such as the band width, which requires prior knowledge of the input sequences, and it has guaranteed small approximation error.

Comparison with Approx-DTW. Part of our motivation for this work is that the new FA-DTW algorithm is much simpler than Approx-DTW and is expected to create far fewer boundary points in the DP table. This intuition is confirmed in Table 3. As the running time of both FA-DTW and Approx-DTW are proportional to the number of boundary points, the FA-DTW algorithm clearly is a much faster choice for computing the DTW. Unfortunately, we do not have exact timing results for Approx-DTW as the data structures it requires are too complicated to be implemented in a reasonable way.

Input size	2000	3000	5000	10000
FA-DTW(10^5)	0.9	1.5	3.1	11.5
Approx-DTW(10^5)	6.4	10.1	19.7	35.8

Table 3. Average number of boundary points generated by algorithms FA-DTW and Approx-DTW for 60 pairs of curves in the synthetic datasets when $\epsilon = 1$.

4.3 Parameter Sensitivity

When the parameter ϵ is set to 1, we guarantee our approximation algorithms to return solutions of value at most twice the optimal. In practice, however, for all categories of curves described above, the approximation error is usually less than 10%. In the worst case, the error is still within 15%. See Figure 7 (a) for the average approximation error for synthetic inputs against varying ϵ parameter.

The tradeoff between running time and ϵ is shown in Figure 7 (b). When ϵ increases from 0.3 to 1, the running time is significantly reduced, while the actual error only increases from 5% to 10%.

The above phenomena appear to derive directly from the effect different values of ϵ have on the number of rectangular boundary points our algorithm creates. We plot the number of rectangle boundary points created for varying values of ϵ in Figure 7 (c). We observe that the number of bound-

ary points decreases sharply as ϵ is increased to around 0.5. Afterwards, the decrease in boundary points levels off. The leveling off likely occurs because there are only so many rectangles needed to represent the varying distances between points on input curves when the threshold between relevant distances becomes large. When ϵ becomes sufficiently large, each additional rectangle contains a wide range of possible distances, and our algorithm does not need to search further for more rectangles.

4.4 Applications

We present experiments for two applications of our approximation algorithm: trajectory nearest neighbor queries and curve transformation.

Nearest-neighbor queries for trajectories. Suppose we keep a database of trajectories. Given an input trajectory, we want to know what are its k closest trajectories in our database. One method is to find the k trajectories that give the k smallest DTW values when compared against the input trajectory. This is a variant of the k-nearest neighbors problem applied to trajectories.

In this experiment, we include trajectories from the GeoLife dataset into a database, and compute the k closest trajectories to an input trajectory of size 5000. The size of the trajectories chosen from the GeoLife dataset ranges from 2000 to 20000. Even for a database of only 1000 trajectories, the task of comparing an input trajectory to all trajectories in the database using the exact DP algorithm takes 674 seconds.

In comparison, the FA-DTW algorithm with $\epsilon = 0.5$ takes only 38 seconds. The algorithm returns the closest trajectory correctly 98% of the time, and returns the 5 closest trajectories correctly 92% of the time. Most of the instances where FA-DTW returns a different set of nearest trajectories happen when the input trajectory is very different from all the trajectories in the database. Even in these cases, the difference between the nearest-neighbor distances using FA-DTW and DP-DTW is very small compared to the distance itself. Finally, when ϵ is decreased to 0.2, the closest 5 trajectories are always correctly returned, although the total running time increases to 110 seconds.

Iterative closest point. In many applications, we wish to minimize DTW between a pair of point sequences under translation and rotation of one of the point sequences. A widely used algorithm for curve similarity under translation/rotation is the *iterative closest point method* (ICP) [20]. In each iteration, ICP first finds the optimal DTW correspondences between the sequences and then computes the translation and rotation that minimizes the total squared distances between the correspondences. Such a translation and rotation can be computed by the Kabsch algorithm [14] using singular value decomposition. The algorithm stops when the DTW correspondences computed at the current and previous iterations are the same.

Consider the two curves of Figure 8(a) which are similar but embedded in different poses in $\mathbb{R}^2$. Since the distance between the two sequences is large, the majority of the distances between points in P and Q are within $1 + \epsilon$ of each other when $\epsilon = 0.5$. As shown in Figure 8(b), this small factor difference in distances leads our algorithm to finding only a few very large rectangles. Figure 8 (c) shows the correspondences after one iteration of ICP. When ICP converges, the positions of the two sequences are much closer to each other, and the correspondences result in a much smaller DTW distance; see Figure 8 (d-e).

The errors of FA-DTW are within 10% at each step of ICP. For sequences with small final DTW distance, the average number of steps for ICP to converge using FA-DTW is about 7 steps (the number can be large for sequences that are significantly different). On average, the ICP algorithm is 6 times faster using FA-DTW for sequences with small final DTW distance.

Pre-processing. We note that in both applications, we need to compute DTW repeatedly. In a k-nearest neighbors search, we compute DTW of an input sequence against all trajectories in a database; in ICP, we compute DTW between the same sequences after each step of transformation. These applications would benefit if we pre-compute the bounding volume hierarchies for all the sequences.

For the k-nearest neighbors search example, storing the bounding volume hierarchies takes only linear space, which is feasible even for large datasets. In the case of ICP for two input sequences, we can also pre-compute the bounding volume hierarchies. After each step, we transform each of the bounding volumes in the hierarchy according to the transformation found in the previous step.

After pre-processing, finding the children of a bounding volume now takes constant time, which reduces the running time of the GEN_RECTANGLES procedure. In our experiment, the FA-DTW algorithm with pre-processing is around 2 times faster compared to the same algorithm without pre-processing.

5. CONCLUSION

In this paper, we presented a simple ε-approximation algorithm for computing DTW between a pair of point sequences in $\mathbb{R}^d$, each having at most n points, and proved that its running time is $O(\frac{\kappa^2}{\varepsilon} n \log \sigma)$.

Experiments show our approximation algorithm gains a clear advantage over the naive dynamic programming algorithm when the input size is 2,000 or bigger. We observe a speedup ratio of 5-10 for most of the synthetic and real datasets. We present several applications as examples to prove the efficacy of the new algorithm. It is noteworthy that since most of the applications of DTW with large datasets have the common feature of computing DTW multiple times with at least one of the input sequences fixed, our algorithm which allows pre-processing and succinct representation of correspondences achieves even better speedup than that seen doing a single DTW computation.

Acknowledgements. Work on this paper is supported by NSF under grants CCF-11-61359, IIS-14-08846, CCF-15-13816, and ISS-14-47554 by an ARO grant W911NF-15-1-0408, and by Grant 2012/229 from the U.S.-Israel Binational Science Foundation.

6. REFERENCES

[1] P. K. Agarwal, K. Fox, J. Pan, and R. Ying. Approximating dynamic time warping and edit distance for a pair of point sequences. In *Proc. 32nd Int. Symp. Comp. Geom.*, pages 6:1–6:16, 2016.

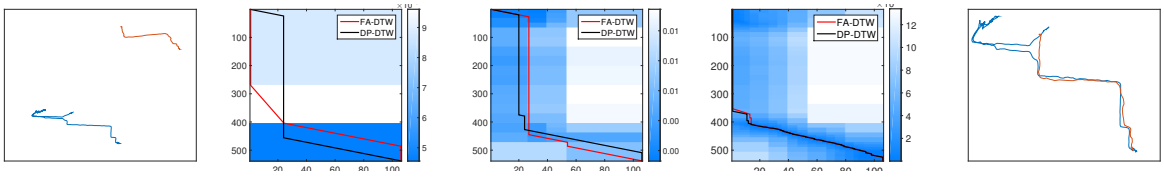


Figure 8. (a) Two trajectories in their initial poses; (b) initial rectangles and correspondences (approximate DTW = 12.69); (c) an intermediate step (approximate DTW = 4.33); (d) the final step (approximate DTW = 1.51). (e) The final pose of two trajectories after the red one undergoes a rigid transformation.

[2] P. K. Agarwal, S. Har-Peled, and K. R. Varadarajan. Geometric approximation via coresets. In *Combinatorial and Computational Geometry* (J. E. Goodman, J. Pach, and E. Welzl, eds.), Cambridge Univ. Press, New York, 1–30, 2005.

[3] G. Al-Naymat, S. Chawla, and J. Taheri. Sparsedtw: A novel approach to speed up dynamic time warping. In *Proc. 8th Aus. Data Mining Conf.*, pages 117–128, 2009.

[4] T. Brox and J. Malik. Object segmentation by long term analysis of point trajectories. In *Proc. 11th Euro. Conf. Comp. Vis.*, pages 282–295, 2010.

[5] 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.

[6] 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. *Proc. ACM Conf. Human Factors Comput.*, pages 987–996, 2012.

[7] H. Ding, G. Trajcevski, P. Scheuermann, X. Wang, and E. Keogh. Querying and mining of time series data: experimental comparison of representations and distance measures. *Proc. VLDB Endow.*, 1(2):1542–1552, 2008.

[8] A. Driemel, S. Har-Peled, and C. Wenk. Approximating the Fréchet distance for realistic curves in near linear time. *Discrete Comput. Geom.*, 48:94–127, 2012.

[9] R. Durbin, editor. *Biological Sequence Analysis: Probabalistic Models of Proteins and Nucleic Acids.* Cambridge Univ. Press, Cambridge, 1998.

[10] J. Fischer and V. Heun. Theoretical and practical improvements on the RMQ-problem, with applications to LCA and LCE. In *Proc. 17th Annu. Sympos. Combin. Pattern Match.*, pages 36–48, 2006.

[11] T. Gasser and K. Wang. Alignment of curves by dynamic time warping. *Annals Stat.*, 25:1251–1276, 1997.

[12] T. Giorgino. Computing and visualizing dynamic time warping alignments in R: The dtw Package. *J. Stat. Software*, 31(7), 2009.

[13] S. Har-Peled. *Geometric Approximation Algorithms.* American Mathematical Society, Providence, 2011.

[14] W. Kabsch. A solution for the best rotation to relate two sets of vectors. *Acta Crys. Sec. A*, 32(5):922–923, 1976.

[15] J. Kleinberg and E. Tardos. *Algorithm Design.* Pearson/Addison-Wesley, Boston, 2006.

[16] R. Kolodny, P. Koehl, and M. Levitt. Comprehensive evaluation of protein structure alignment methods: scoring by geometric measures. *J. Mol. Biol.*, 346(4):1173–1188, 2005.

[17] J. Matoušek, M. Sharir, and E. Welzl. A subexponential bound for linear programming. *Algorithmica*, 16(4-5):498–516, 1996.

[18] M. Munich and P. Perona. Continuous dynamic time warping for translation-invariant curve alignment with applications to signature verification. In *Proc. 7th IEEE Int. Conf. Comp. Vis.*, 1999.

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

[20] N. Padoy and G. D. Hager. Spatio-temporal registration of multiple trajectories. In *Proc. 14th Medical Image Comp. Comp.-Asst. Inter.*, pages 145–152. 2011.

[21] L. R. Rabiner and B. H. Juang. *Fundamentals of Speech Recognition.* Prentice Hall, Upper Saddle River, 1993.

[22] H. Sakoe and S. Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Trans. Acou., Speech, Signal Proc.*, 26:43–49, 1978.

[23] Y. Zheng, Q. Li, Y. Chen, X. Xie, and W.-Y. Ma. Understanding mobility based on GPS data. In *Proc. 10th ACM Conf. Ubi. Comp.*, pages 312–321, 2008.

[24] Y. Zheng, X. Xie, and W.-Y. Ma. Geolife: A collaborative social networking service among user, location and trajectory. *IEEE Data Eng. Bull.*, 33(2):32–39, 2010.

[25] Y. Zheng, L. Zhang, X. Xie, and W.-Y. Ma. Mining interesting locations and travel sequences from GPS trajectories. In *Proc. 18th Int. Conf. World Wide Web*, pages 791–800, 2009.

[26] F. Zhou and F. Torre. Canonical time warping for alignment of human behavior. In *Proc. 23rd Annu. Conf. Neural Info. Proc. Sys.*, pages 2286–2294, 2009.