LB_Keogh Supports Exact Indexing of Shapes under Rotation Invariance with Arbitrary Representations and Distance Measures

Eamonn Keogh Li Wei Xiaopeng Xi Sang-Hee Lee[1] Michail Vlachos
Department of Computer Science & Engineering
[1]Department of Anthropology
University of California - Riverside
Riverside, CA 92521
{*eamonn*, *wli*, *xxi*}*@cs.ucr.edu*, *shlee@ucr.edu* , *vlachos@us.ibm.com*

Dear Reader, this is an expanded version of the VLDB 2006 paper

ABSTRACT

The matching of two-dimensional shapes is an important problem with applications in domains as diverse as biometrics, industry, medicine and anthropology. The distance measure used must be invariant to many distortions, including scale, offset, noise, partial occlusion, etc. Most of these distortions are relatively easy to handle, either in the representation of the data or in the similarity measure used. However rotation invariance seems to be uniquely difficult. Current approaches typically try to achieve rotation invariance in the representation of the data, at the expense of discrimination ability, or in the distance measure, at the expense of efficiency. In this work we show that we can take the slow but accurate approaches and dramatically speed them up. On real world problems our technique can take current approaches and make them four orders of magnitude faster, without false dismissals. Moreover, our technique can be used with any of the dozens of existing shape representations and with all the most popular distance measures including Euclidean distance, Dynamic Time Warping and Longest Common Subsequence.

Keywords

Shape, Indexing, Rotation Invariance, Dynamic Time Warping

1. INTRODUCTION

The matching of two-dimensional shapes is an important problem with applications in domains as diverse as biometrics, industry, medicine and anthropology. The distance measure used must be invariant to many distortions, including scale, offset, noise, partial occlusion, etc. Most of these distortions are relatively easy to handle, particularly if we use the well-known technique of converting the shapes into time series as in Figure 1. However, no matter what representation is used, rotation invariance seems to be uniquely difficult to handle. For example [14] notes "*rotation is always something hard to handle compared with translation and scaling*", and the literature abounds with similar statements. Many current approaches try to achieve rotation invariance in the representation of the data, at the expense of discrimination ability [19], or in the distance measure, at the expense of efficiency [1][2][3][7].

As an example of the former, the very efficient rotation invariant technique of [19] cannot differentiate between the shapes of the lowercase letters "**d**" and "**b**". As an example of the latter, the work of Adamek and Connor [1], which is state of the art in terms of accuracy or precision/recall takes an untenable O(n^3) for each shape comparison.

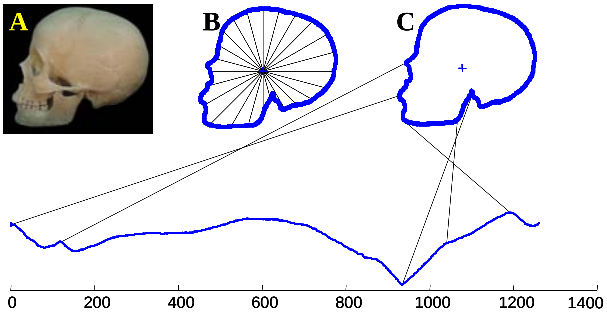


Figure 1: Shapes can be converted to time series. **A**) A bitmap of a human skull. **B**) The distance from every point on the profile to the center is measured and treated as the Y-axis of a time series of length *n* (**C**)

In this work we show that we can take the slow but accurate approaches and dramatically speed them up. For example we can take the O(n^3) approach of [1] and on real world problems bring the average complexity down to O($n^{1.06}$). This dramatic improvement in efficiency does not come at the expense of accuracy; we can prove that we will always return the same answer set as the slower methods.

We achieve speedup over the existing methods in two ways, dramatically decreasing the CPU requirements, and allowing indexing. Our technique works by grouping together similar rotations, and defining an admissible lower bound to that group. Given such a lower bound, we can utilize the many search and indexing techniques known in the database community.

Our technique has the following advantages:

- There are dozens of techniques in the literature for converting shapes to time series [1][3][6][24][25][28], including some that are domain specific [4][21]. Our approach works for *any* of these representations.
- While there are many distance measures for shapes in the literature, Euclidean distance, Dynamic Time Warping [2][4] [20][21] and Longest Common Subsequence [23] accounts for the majority of the literature. Our approach works for *any* of these distance measures.
- Our approach uses the idea of LB_Keogh lower bounding as its cornerstone. Since the introduction of this idea a few years ago [11], dozens of researchers world wide have adopted and extended this framework for applications as diverse as motion capture indexing [13], P2P searching [9], handwriting retrieval [21], dance indexing, and query by

humming and monitoring streams [26]. This widespread adoption of LB_Keogh lower bounding has insured that it has become a mature and widely supported technology, and suggests that any contributions made here can be rapidly adopted and expanded.

- In some domains it may be useful to express *rotation-limited* queries. For example, in order to robustly retrieve examples of the number "8", without retrieving infinity symbols "∞", we can issue a query such as: "Find the best match to this shape allowing a maximum rotation of ± 15 degrees". Our framework supports such rotation-limited queries.

The rest of this paper is organized as follows. In Section 2 we discuss background material and related work. In Section 3 we formally introduce the problem and in Section 4 we offer our solution. Section 5 offers a comprehensive empirical evaluation of our technique. Finally Section 6 offers some conclusions and directions for future work.

2. BACKGROUND AND RELATED WORK

The literature on shape matching is vast; we refer the interested reader to [6][22] and [28] for excellent surveys. While not all work on shape matching uses a 1D representation of the 2D shapes, an increasingly large majority of the literature does. We therefore only consider such approaches here. Note that we lose little by this omission. The two most popular measures that operate directly in the image space, the Chamfer [5] and Hausdorff [18] distance measures, require O(n^2log$n$) time[1] and recent experiments (including some in this work) suggest that 1D representations can achieve comparable or superior accuracy.

In essence there are three major techniques for dealing with rotation invariance, landmarking, rotation invariant features and brute force rotation alignment. We consider each below.

2.1 Landmarking

The idea of "landmarking" is to find the one "true" rotation and only use that particular alignment as the input to the distance measure. The idea comes in two flavors, domain dependent and domain independent.

In domain dependent landmarking, we attempt to find a single (or very few) fixed feature to use as a starting point for conversion of the shape to a time series. For example, in face profile recognition the most commonly used landmarks (fiducial points) are the chin or nose [4]. In limited domains this may be useful, but it requires building special purpose feature extractors. For example, even in a domain as intuitively well understood as human profiles, accurately locating the nose is a non-trivial problem, even if we discount the possibility of mustaches and glasses. Probably the only reason any progress has been made in this area is that most work reasonably assumes that faces presented in an image are likely to be upright. For shape matching in skulls, the canonical landmark is called the Frankfurt Horizontal [27], which is defined by the right and left porion (the highest point on the margin of the external auditory meatus) and the left orbitale (the lowest point on the orbital margin). However, a skull can be missing the relevant bones to determine this orientation and still have enough global information to match its shape to similar examples. Indeed the famous Skhul V skull shown in Figure 12 is such an example.

In domain independent landmarking, we align all the shapes to some cardinal orientation, typically the major axis. This approach may be useful for the limited domains in which there is a well-defined major axis, perhaps the indexing of hand tools. However there is increasing recognition that the "*…major axis is sensitive to noise and unreliable*" [28]. For example a recent paper shows that under some circumstances, a single extra pixel can change the rotation by ± 90 degrees [29].

To show how brittle landmarking can be we performed a simple clustering experiment where we clustered three primate skulls using Euclidean distance with both the major axis technique, and the minimum distance of all possible rotations (as found by brute force). Figure 2 shows the result.

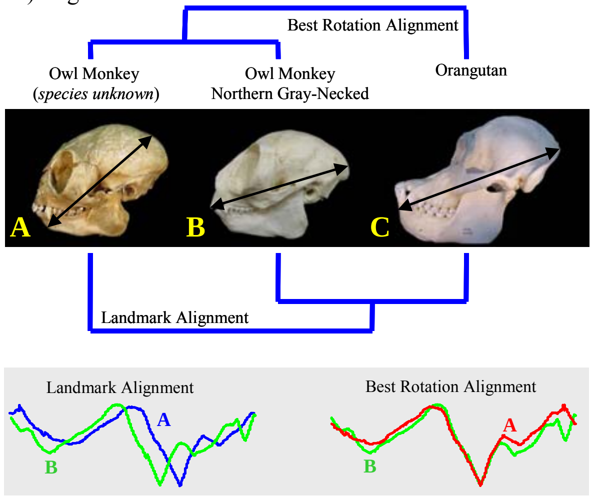


Figure 2: *Top*) Three primate skulls, two of them from the same genus, are clustered using both the landmark rotation beginning at the major axis, and the best rotation. *Bottom*) The landmark-based alignment of **A** and **B** explains why the landmark-based clustering is incorrect: a small amount of rotation error results in a large difference in the distance measure

The most important lesson we learned from this experiment (and dozens of other similar experiments on diverse domains [10]) is that rotation (mis)alignment is the most important invariance for shape matching, unless we have the best rotation then nothing else matters.

2.2 Rotation Invariant Features

A large number of papers achieve fast rotation invariant matching by extracting only rotation invariant features and indexing them with a feature vector [6]. This feature vector is often called the shapes "signature". There are literally dozens of rotation invariant features including ratio of perimeter to area, fractal measures, elongatedness, circularity, min/max/mean curvature, entropy, perimeter of convex hull etc. In addition many researchers have attempted to frame the shape-matching problem as a more familiar histogram-matching problem. For example in [19] the authors build a histogram containing the distances between two randomly chosen points on the perimeter of the shapes in question. The approach seems to be attractive, for example it can trivially also handle 3D shapes, however it suffers from extremely poor precision. For example, it cannot differentiate between the shapes of the lowercase letters "**d**" and "**b**", or "**p**" and "**q**", since these pairs of shapes have identical histograms. In general, all

[1] More precisely the time complexity is O(Rplog$p$), where p is the number of pixels in the perimeter and R is the number of rotations that need to be executed. Here $p = n$, and while R is a user defined parameter, it should be approximately equal n to guarantee all rotations (up to the limit of rasterization) are considered.

these methods suffer from very poor discrimination ability [6]. In retrospect this is hardly surprising. In order to achieve rotation invariance, all information that contains rotation information must be discarded; inevitably, some useful information will also be discarded in this process. Our experience with these methods suggests that they can be useful for making quick coarse discriminations, for example differentiating between skulls and vertebrae. However we could not get these methods to distinguish between the skulls of humans and orangutan, a trivial problem for human or the brute force algorithm discussed in the next section.

2.3 Brute Force Rotation Alignment

There are a handful of papers that recognize that the above attempts at approximating rotation invariance are unsatisfactory for most domains, and they achieve true rotation invariance by exhaustive brute force search over all possible rotations, but only at the expense of computational efficiency and indexability [1][2][3][7][25]. For example, paper [1] uses DTW to handle nonrigid shapes in the time series domain, while they note that most invariances are trivial to handle in this representation, they state "*rotation invariance can (only) be obtained by checking all possible circular shifts for the optimal diagonal path*." This step makes the comparison of two shapes O(n^3) and forces them to abandon hope of indexing. Similarly paper [25] notes "*In order to find the best matching result, we have to shift one curve* n *times, where* n *is the number of possible start points*." All the techniques introduced thus far to mitigate this untenable computational complexity do so at the expense of introducing false dismissals. Typically they offer some implicit or explicit trick to find a one (or a small number of) of starting point(s) [2][3][7]. For example paper [2] suggests "*In order to avoid evaluation of the dissimilarity measure for every possible pair of starting contour points ...we propose to extract a small set of the most likely starting points for each shape.*" Furthermore, both the heuristic used and the number of starting points must "*be adjusted to a given application*", and it is not obvious how to best achieve this.

In forceful experiments on publicly available datasets it has been demonstrated that brute force rotation alignment produces the best precision/recall and accuracy in diverse domains [1][2]. In retrospect this is not too surprising. The rival techniques with rotation invariant features are all using some lossy transformation of the data. In contrast the brute force rotation alignment techniques are using a (potentially) lossless transformation of the data. With more high quality information to use, any distance measures will have an easer time reflecting the true similarity of the original images.

The contribution of this work is to speed up these accurate but slow methods by many orders of magnitude while producing identical results.

3. ROTATION INVARIANT MATCHING

We begin by formally defining the rotation invariant matching problem. We begin by assuming the Euclidean distance, and generalize to other distance measures later. For clarity of presentation we will generally refer to "time series", which the reader will note can be mapped back to the original shapes.

Suppose we have two time series, Q and C of length n, which were extracted from shapes by an arbitrary method.

$$Q = q_1, q_2, \ldots, q_i, \ldots, q_n$$
$$C = c_1, c_2, \ldots, c_j, \ldots, c_n$$

As we are interested in large data collections we denote a database of m such time series as $\overline{Q}$.

$$\overline{Q} = \{Q_1, Q_2, \ldots Q_m\}$$

If we wish to compare two time series, and therefore shapes, we can use the ubiquitous Euclidean distance:

$$ED(Q,C) \equiv \sqrt{\sum_{i=1}^{n}(q_i - c_i)^2}$$

When using Euclidean distance as a subroutine in a classification or indexing algorithm, we may be interested in knowing the exact distance only when it is eventually going to be less than some threshold r. For example, this threshold can be the "range" in range search or the "best-so-far" in nearest neighbor search. If this is the case, we can potentially speed up the calculation by doing early abandoning [12].

Definition 1. *Early Abandon*: During the computation of the Euclidean distance, if we note that the current sum of the squared differences between each pair of corresponding data points exceeds r^2, then we can stop the calculation, secure in the knowledge that the exact Euclidean distance had we calculated it, would exceed r.

While the idea of early abandoning is fairly obvious and intuitive, it is so important to our work we illustrate it in Figure 3 and provide pseudocode in Table 1.

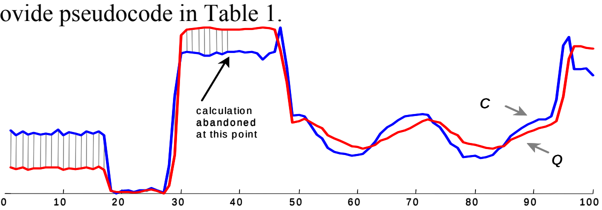


Figure 3: A visual intuition of early abandoning. Once the squared sum of the accumulated gray hatch lines exceeds r^2, we can be sure the full Euclidean distance exceeds r

Note that the "num_steps" value returned by the optimized Euclidean distance in Table 1 is used only to tell us how useful the optimization was. If its value is significantly less than n this suggests dramatic speedup.

Table 1: Euclidean distance optimized with early abandonment

```
algorithm [dist, num_steps] = EA_Euclidean_Dist($Q$, $C$, $r$)
accumulator = 0
for $i$ = 1 to length($Q$)                 // Loop over time series
  accumulator += $(q_i - c_i)^2$             // Accumulate error contribution
  If accumulator > $r^2$                     // Can we abandon?
     disp('doing an early abandon')
     num_steps = $i$
     return [ infinity, num_steps ]        // Terminate and return an
  end                                      // infinite error to signal the
end                                        // early abandonment.
return [ sqrt(accumulator), length($Q$) ]  // Terminate with true dist
```

While the Euclidean distance is a simple distance measure it produces surprisingly good results for clustering, classification and query by content of shapes, *if* the time series in question happen to be rotation aligned. For example, in an experiment in [20] we manually performed rotation alignment of the time series extracted from face profiles by explicitly showing the algorithm the beginning and endpoint of a face (the nape and Adams apple respectively).

However if the shapes are not rotation aligned, this method can produce extremely poor results. To overcome this problem we need to hold one shape fixed, rotate the other, and record the minimum distance of all possible rotations.

For reasons that will become apparent later, we achieve this by expanding one time series into a matrix $\boldsymbol{C}$ of size n by n.

$$\boldsymbol{C} = \begin{Bmatrix} c_1, c_2, \ldots, c_{n-1}, c_n \\ c_2, \ldots, c_{n-1}, c_n, c_1 \\ \vdots \\ c_n, c_1, c_2, \ldots, c_{n-1} \end{Bmatrix}$$

Note that each row of the matrix is simply a time series, shifted (rotated) by one from its neighbors. It will be useful below to address the time series in each row individually, so we will denote the i^{th} row as C_i, which allows us to denote the matrix above in the more compact form of $\boldsymbol{C} = \{C_1, C_2, \ldots, C_n\}$.

We can now define the Rotation invariant Euclidean Distance (RED) as:

$$RED(\mathrm{Q}, \boldsymbol{C}) = \min_{1 \le j \le n} \left\{ ED\left(Q, C_j\right) \equiv \sqrt{\sum_{i=1}^{n} \left(q_i - c_i\right)^2} \right.$$

Table 2 shows the pseudocode to calculate this.

Table 2: An algorithm to find the rotated match between two time series

```
algorithm: [bestSoFar] = Test_All_Rotations(Q,$\boldsymbol{C}$,$r$)
bestSoFar = $r$
for $j$ = 1 to $n$
   distance = EA_Euclidean_Dist(Q, $C_j$, bestSoFar)     // As in Table 1
      if distance <  bestSoFar
         bestSoFar = distance;
      end;
end;
return[bestSoFar]
```

Note that the algorithm tries to take advantage of early abandoning by passing EA_Euclidean_Dist the value of r, the best rotation alignment discovered thus far.

If we are simply measuring the distance between two time series then the algorithm is invoked with r set to infinity, however, as we shall see below, if the algorithm is being used as a subroutine in a linear scan of a large dataset $\overline{Q}$, the calling routine can set the value of r to achieve speedup. In particular the calling function sets r to the value of the best match (under any rotation) discovered thus far. Table 3 shows the pseudocode. Note that the time complexity for this algorithm is $O(mn^2)$. This is simply untenable for large datasets.

Table 3: An algorithm to find the best rotated match to query from a database of possible matches

```
algorithm: [best_match_loc, bestSoFar]= Search_Database_for_Rotated_Match($\boldsymbol{C}$, $\overline{Q}$)
  best_match_loc = null
  bestSoFar        = inf
  for $i$ = 1 to number_of_time_series_in_database($\overline{Q}$)
    distance  = Test_All_Rotations($\overline{Q}_i$,$\boldsymbol{C}$, bestSoFar);       // As in Table 2
      if distance <  bestSoFar
         best_match_loc = $i$
         bestSoFar = distance
      end;
  end;
return[best_match_loc, bestSoFar]
```

Before continuing we will review the notation introduced thus far in Table 4.

Table 4: Notation Table

C	A time series $c_1, c_2, \ldots, c_i, \ldots, c_n$
$\boldsymbol{C}$	A n by n matrix containing every rotation of C
C_i	The i^{th} row of the above
Q	Another time series $q_1, q_2, \ldots, q_i, \ldots, q_n$
$\overline{Q}$	A database containing many time series $\overline{Q} = \{Q_1, .., Q_m\}$

Note that our notation seems somewhat space inefficient in that it expands time series C, of length n, to a matrix of size n by n. However the rest of the database uses the original (arbitrary rotation) time series, and since the size of the database is assumed to be large, this overhead is asymptotically irrelevant.

There are two simple and useful generalizations of definitions thus far.

Mirror Image Invariance: Depending on the application we may wish to retrieve shapes that are enantiomorphic (mirror images) to the query. For example, in matching skulls, the best match may simply be facing the opposite direction. In contrast when matching letters we *don't* want to match a "d" to a "b". If enantiomorphic invariance is required we can trivially achieve this by augmenting matrix $\boldsymbol{C}$ to contain C_i and reverse(C_i) for $1 \le i \le n$.

Rotation-Limited Invariance: In some domains it may be useful to express *rotation-limited* queries. For example, in order to robustly retrieve examples of the number "**6**", without retrieving examples of the number "**9**", we can issue a query such as: "Find the best match to this shape allowing a maximum rotation of ± 15 degrees". Our framework trivially supports such rotation-limited queries, by removing from the matrix $\boldsymbol{C}$ all time series that correspond to the unwanted rotations.

Thus far we have shown a brute force search algorithm that can support rotation invariance, rotation-limited invariance and/or mirror image invariance. We simply put the appropriate time series into matrix $\boldsymbol{C}$ and invoke the algorithm in Table 3. This algorithm, even though speeded up by the early abandoning optimization, is too slow for large datasets. In the next section we introduce our novel search mechanism.

4. WEDGE BASED ROTATION MATCHING

We will begin by showing how we can efficiently search for the best match in main memory. Since large datasets may not fit on disk we will further show how we can index the data.

4.1 Fast and Exact Main Memory Search

We begin by defining time series *wedges*. Imagine that we take several time series, $C_1, .., C_k$, from our matrix $\boldsymbol{C}$. We can use these sequences to form two new sequences U and L:

$$U_i = \max(C_{1i}, .., C_{ki})$$
$$L_i = \min(C_{1i}, .., C_{ki})$$

U and L stand for Upper and Lower respectively. We can see why in Figure 4. They form the smallest possible bounding envelope that encloses all members of the set $C_1, .., C_k$ from above and below. More formally:

$$\forall_i \quad U_i \ge C_{1i}, .., C_{ki} \ge L_i$$

For notational convenience, we will call the combination of U and L a *wedge*, and denote a wedge as W:

$$W = \{U, L\}$$

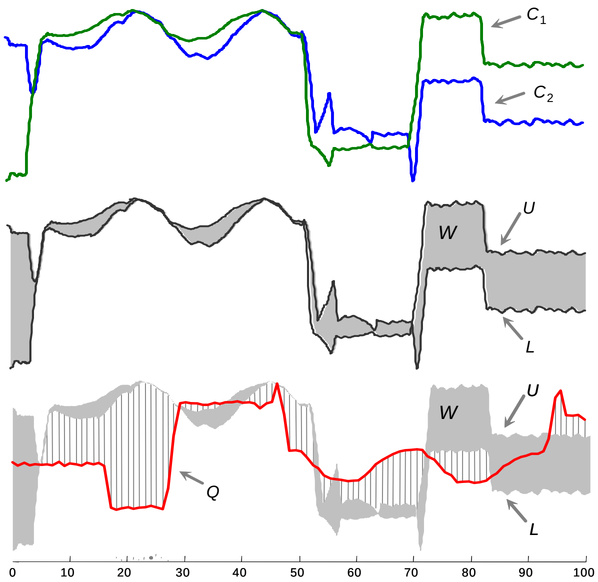


Figure 4: *Top*) Two time series C_1 and C_2. *Middle*) A time series wedge *W*, created from C_1 and C_2. *Bottom*) An illustration of LB_Keogh

We can now define a lower bounding measure between an arbitrary time series Q and the entire set of candidate sequences contained in a wedge *W*:

$$LB\_Keogh(Q,W) = \sqrt{\sum_{i=1}^{n} \begin{cases} (q_i - U_i)^2 & if\ q_i > U_i \\ (q_i - L_i)^2 & if\ q_i < L_i \\ 0 & otherwise \end{cases}}$$

For brevity we do not show a proof of this lower bounding property. A proof appears in [10] and also in [15], where the authors use this representation for different problem.

Note that the *LB_Keogh* function has been used before to support DTW [11][20][21][23], uniform scaling [13], and query filtering [26]. For these tasks the lower bounding distance function is the same, but the definition of *U* and *L* are different.

There are two important observations about LB_Keogh. First, in the special case where *W* is created from a single candidate sequence, it degenerates to the Euclidean distance. Second, not only does LB_ Keogh lower bound all the candidate sequences C_1,..,C_k, but we can also do *early abandon* with LB_Keogh. While the latter fact might be obvious, for clarity we make it explicit in Table 5.

Table 5: LB_Keogh optimized with early abandonment

```
algorithm [dist, num_steps] = EA_LB_Keogh(Q, W, r)
accumulator = 0
for i = 1 to length(Q)                        // Loop over time series
  if $q_i$ > W.$U_i$                          // Accumulate error contribution
        accumulator += ($c_i$ - W.$U_i$)$^2$
  elseif $q_i$ < W.$L_i$
        accumulator += ($c_i$ - W.$L_i$)$^2$
  end
  if accumulator > $r^2$                      // Can we abandon?
        return [ infinity, i]                 // Terminate and return an infinite error
  end                                         // to signal the early abandonment.
end
return [ sqrt(accumulator), length(Q) ]  // Terminate with true dist
```

Note once again that the value returned in "num_steps" is merely a bookkeeping device to allow a post mortem evaluation of efficiency.

Suppose we have just two time series C_1 and C_2 of length *n*, and we know that in future we will be given a time series query Q and asked if one (or both) of C_1 and C_2 are within *r* of the query. We naturally wish to minimize the number of steps we must perform ("steps" are measured by "num_steps"). We are now in a position to outline two possible approaches to this problem.

- We can simply compare the two sequences, C_1 and C_2 (in either order) to the query using the early abandon algorithm introduce in Table 1. We will call this algorithm, *classic*.
- We can combine the two candidate sequences into a wedge, and compare Q to the wedge using LB_Keogh. If the LB_Keogh function early abandons, we are done. We can say with absolute certainty that neither of the two candidate sequences is within *r* of the query. If we cannot early abandon on the wedge, we need to individually compare the two candidate sequences, C_1 and C_2 (in either order) to the query. We will call this algorithm, *Merge*.

Let us consider the best and worst cases for each approach. For *classic* the worst case is if both candidate sequences are within *r* of the query, which will require 2*n* steps. In the best case, the first point in the query may be radically different to the first point in either of the candidates, allowing immediate early abandonment and giving a total cost of 2 steps.

For *Merge*, the worst case is also if both candidate sequences are within *r* of the query, because we will waste *n* steps in the lower bounding test between the query and the wedge, and then *n* steps for each individual candidate, for a total of 3*n*. However the best case, also if the first point in the query is radically different, would allow us to abandon with a total cost of 1 step.

Which of the two approaches is better depends on:

- The shapes of C_1 and C_2. If they are similar, this greatly favors *Merge*.
- The shape of Q. If Q is truly similar to one (or both) of the candidate sequences, this would greatly favor *classic*.
- The matching distance *r*. Here the effect is non monotonic and dependent on the two factors above.

We can generalize the notion of wedges by hierarchically nesting them. Let us begin by augmenting the notation of a wedge to include information about the sequences used to form it. For example, if a wedge is built from C_1 and C_2, we will denote it as $W_{(1,2)}$. Note that a single sequence is a special case of a wedge, for example the sequence C_1 can also be denoted as W_1. We can combine $W_{(1,2)}$ and W_3 into a single wedge by finding maximum and minimum values for each i^{th} location, from *either* wedge. More concretely:

$$U_{\mathrm{i}} = \max(W_{(1,2)\mathrm{i}}, W_{3\mathrm{i}})$$
$$L_{\mathrm{i}} = \min(W_{(1,2)\mathrm{i}}, W_{3\mathrm{i}})$$
$$W_{((1,2),3)} = \{U, L\}$$

In Figure 5 we illustrate this notation. We call $W_{(1,2)}$ and W_3 *children* of wedge $W_{((1,2),3)}$. Since individual sequences are special cases of wedges, we can also call C_1 and C_2 children of $W_{(1,2)}$.

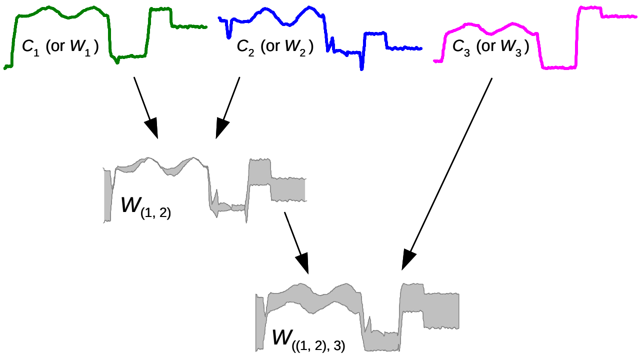


Figure 5: An illustration of hierarchically nested wedges

Given the generalization to hierarchal wedges, we can now also generalize the *Merge* approach. Suppose we have a time series Q and a wedge $W_{((1,2),3)}$. We can compare the query to the wedge using LB_Keogh. If the LB_Keogh function early abandons, we are done. We know with certainty that none of the three candidate sequences is within r of Q. If we cannot early abandon on the wedge, we need to compare the two child wedges, $W_{(1,2)}$ and W_3 to the query. Again, if we cannot early abandon on the wedge $W_{(1,2)}$, we need to individually compare the two candidate sequences, C_1 and C_2 (in either order) to the query. We call this algorithm *H-Merge* (Hierarchal Merge).

The utility of a wedge is strongly correlated to its area. We can get some intuition as to why by visually comparing LB_Keogh(Q, $W_{(1,2)}$) with LB_Keogh(Q, $W_{((1,2),3)}$) as shown in Figure 6. Note that the area of $W_{((1,2),3)}$ is much greater than that of $W_{(1,2)}$, and that this reduces the value returned by the lower bound function and thus the possibility to early abandon.

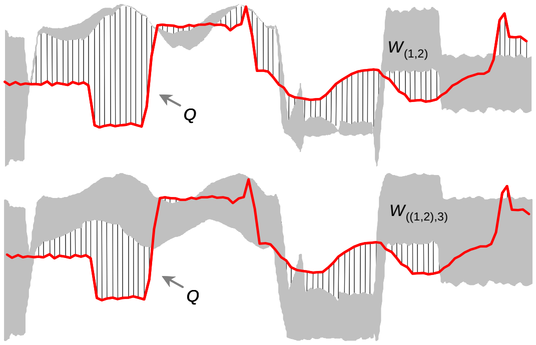


Figure 6: *Top*) An illustration of LB_Keogh(Q, $W_{(1,2)}$). *Bottom*) An illustration of LB_Keogh(Q, $W_{((1,2),3)}$). Note that the tightness of the lower bound is proportion to the number and (squared) length of vertical lines

For some problems, the *H-Merge* algorithm can give exceptionally poor performance. If the wedge $W_{(1,2)}$, created from C_1 and C_2 has an exceptional large area (i.e. C_1 and C_2 are very dissimilar), it is very unlikely to be able to prune off any steps.

At this point we can see that the efficiency of *H-Merge* is dependent on the candidate sequences and Q itself. In general, merging similar sequences into a hierarchal wedge is a good idea, but merging dissimilar sequences is a bad idea.

The observations above motivate a final generalization of *H-Merge*. Recall that to achieve rotation invariance we expanded our time series C into a matrix with n time series. Given these n sequences, we can merge them into K hierarchal wedges, where $1 \le K \le n$. This merging forms a partitioning of the data, with each sequence belonging to exactly one wedge. We will use **W** to denote a set of hierarchal wedges:

$$\mathbf{W} = \{W_{\text{set}(1)}, W_{\text{set}(2)}, .., W_{\text{set}(K)}\}, \qquad 1 \le K \le n$$

where $W_{\text{set}(i)}$ is a (hierarchically nested) subset of the n candidate sequences. Note that we have

$$W_{\text{set}(i)} \cap W_{\text{set}(j)} = \varnothing \;\text{ if } i \neq j, \text{ and}$$

$$| W_{\text{set}(1)} \cup W_{\text{set}(2)} \cup .. \cup W_{\text{set}(K)} | = n$$

We will attempt to merge together only similar sequences. We can then compare this set of wedges against our query. Table 6 formalizes the algorithm.

Table 6: Algorithm *H-Merge*

```
algorithm   [dist] = H-Merge($Q$, W,$K$, $r$)
$S$ = {$empty$}                                  // Initialize a stack.
for $i$ = 1 to K                                 // Place all the wedges into the stack.
    enqueue($W_{set(i)}$ ,$S$ )
end
while not empty($S$ )
    $T$ = dequeue($S$ )
    dist =  EA_LB_Keogh($Q$,$T$,$r$)             // Note that is early abandon version.
    if isfinite(dist)                            //  We did not early abandon.
        if cardinality($T$ ) = 1                 // T  was an individual sequence.
            disp('The sequence ', $T$, 'is ', dist, ' units from the query')
            return[dist]
        else                                     // T was a wedge, find its children
            enqueue(children($T$ ) ,$S$ )        // and push them onto the stack.
        end
    end
end
```

Note that this algorithm is designed to replace the **Test_All_Rotations** algorithm that is invoked as a subroutine in the **Search_Database_for_Rotated_Match** algorithm shown in Table 3.

As we shall see in our empirical evaluations, *H-Merge* can produce very impressive speedup if we make judicious choices in the set of hierarchal wedges that make up W. However, the number of possible ways to arrange the hierarchal wedges is greater than K^K, and the vast majority of these arrangements will be very poor, so specifying a good arrangement of W is critical.

A simple observation alleviates the need to invent a new algorithm to find a good arrangement of W. Note that hierarchal clustering algorithms have very similar goals to an ideal wedge-producing algorithm. In particular, hierarchal clustering algorithms can be seen as attempting to minimize the distances between objects in each subtree. A wedge-producing algorithm should attempt to minimize the area of each wedge. However the area of a wedge is simply the maximum Euclidean distance between any sequences contained therein (i.e Newton-Cotes rule from elementary calculus). This motivates us to derive wedge sets based on the result of a hierarchal clustering algorithm. Figure 8 shows wedge sets W, of every size from 1 to 5, derived from the dendrogram shown in Figure 7.

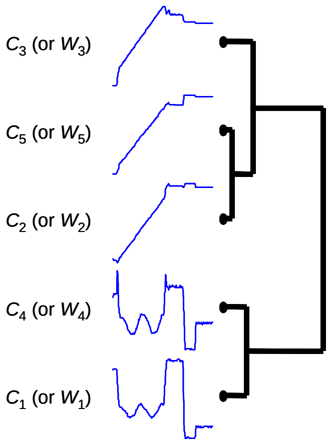


Figure 7: A dendrogram of five sequences C_1, C_2,..., C_5, clustered using group average linkage

Given that the clustering algorithm produces the tentative wedge sets, all we need to do is to choose the best one. We could attempt to do this by eye, for example in Figure 8 it is clear that any sequence that early abandons on W_3, will almost certainly also early abandon on both W_2 and W_5; similar remarks apply to W_1 and W_4. At the other extreme, the wedge at $K = 1$ is so "fat" that it is likely have poor pruning power. The set W = $\{W_{((2,5),3)}, W_{(1,4)}\}$ is probably the best compromise. However because the set of time series might be very large, such visual inspection is not scalable.

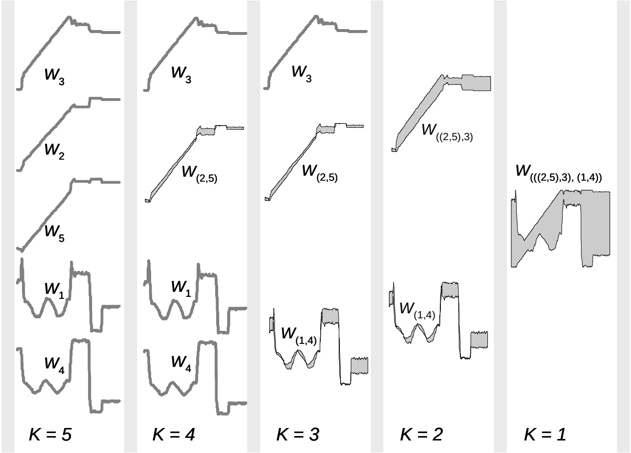


Figure 8: Wedge sets **W**, of size 1 to 5, derived from the dendrogram shown in Figure 7

The problem is actually even more complex, in that the best value for K also depends on the current value of *r* (Recall *r* is the "best-so-far" in nearest neighbor search.). If *r* is large then very little early abandoning is possible and this favors a large value for K. In contrast, if *r* is small we can do a lot of early abandoning, and we are better off having many sequences in a single wedge so we can early abandon all of them with a single calculation. Note however that for nearest neighbor search the value of *r* will get smaller as we search through the database.

With this in mind, we dynamically choose the wedge set based on a fast empirical test. We start with the wedge set where K = 2. Each time the bestSoFar value changes, we test a subset of the possible values of K and choose the most efficient one (as measured by num_steps) as the next K to use. Which subset to test is decided on-the-fly based on the current K value. They are the values which evenly divide the ranges [1, current_K] and [current_K, max_K] into 5 intervals. Note that on average the bestSoFar value only changes log(m) during a linear search, so this slight overhead in adjusting the parameter is not too burdensome, however we do include this cost in all experiments in Section 5.

4.2 Lower Bounding in Index Space

True rotation invariance has traditionally been so demanding in terms of CPU time that little or no effort was made to index it (or it was indexed with the possibility of false dismissals). As we shall see in the experiments in Section 5.2, the ideas presented in the last section produce such dramatic reductions in CPU time that it is worth considering indexing the data.

There are several possible techniques we could consider for indexing. Recent years have seen dozens of papers on indexing time series envelopes that we could attempt to leverage off [11][15][20][21][23]. The only non-trivial adaptation to be made is that instead of the query being a single envelope, it would be necessary to search for the best match to K envelopes in the wedge set **W**.

Note however that we do not necessarily have to use the enveloping idea in the indexing phase. So long as we can lower bound in the index space we can use an arbitrary technique to get (hopefully a small subset of) the data from disk to main memory, where our *H-Merge* can very efficiently find the distance to the best rotation. One possible method to achieve this indexable lower bound is to use Fourier methods. Many authors have independently noted that transforming the signal to the Fourier space and calculating the Euclidean distance between the *magnitude* of the coefficients produces a lower bounds to any rotation [24]. We can leverage of this lower bound to use a VP-tree to index our time series as shown in Table 7.

Table 7: A Vantage Point Tree for Indexing Shapes

```
Algorithm [BSF] = NNSearch(C)
    BSF.ID = null;                        // BSF is the Best-So-Far variable
    BSF.distance = infinity;
    W = convert_time_series_to_wedge_set(C);
     Search($Q_{root}$,W, BSF);     // Invoke subroutine on the root of index tree
Subroutine Search(NODE, W, BSF)
if NODE.isLeaf                                    // we are at a leaf node.
    for each compressed time-series cT in node
       LB = computeLowerBound(cT, W);
       queue.push(cT,LB);                         // sorted by lower bound.
    end
    while (not (queue.empty()) and (queue.top().LB < BSF.distance))
      if (BSF.distance > queue.top().LB)
        retrieve full time series Q of queue.top() from disk;
        distance = H-Merge(Q, W, BSF.distance )  // calculate full distance.
            if distance < BSF.distance             // update the best-so-far
                BSF.distance = distance;           // distance and location.
                BSF.ID = Q;
            end
       end
     end
else                                               // we are at a vantage point.
    LB = computeLowerBound(VP, W);
    queue.push(VP,LB);
        if LB < (node.median + BSF.distance)
               search(NODE.left, W, BSF);     // recursive search left.
       else
               search(NODE.right, W, BSF);    // recursive search right.
       end
end
```

This technique is adapted from [24], and we refer the reader to this work for a more complete treatment.

4.3 Generalizing to other Distance Measures

As we shall see in Section 5, the Euclidean distance is typically very effective and intuitive as a distance measure for shapes. However in some domains it may not produce the best possible precision/recall or classification accuracy [2][20]. The problem is that even after best rotation alignment, subjectively similar shapes may produce time series that are globally similar but contain local "distortions". These distortions may correspond to local features in that are present in both shapes but in different proportions. For example in Figure 9 we can see that the larger brain case of the Lowland Gorilla changes the locations in which the brow ridge and jaw map to in a time series relative to the Mountain Gorilla.

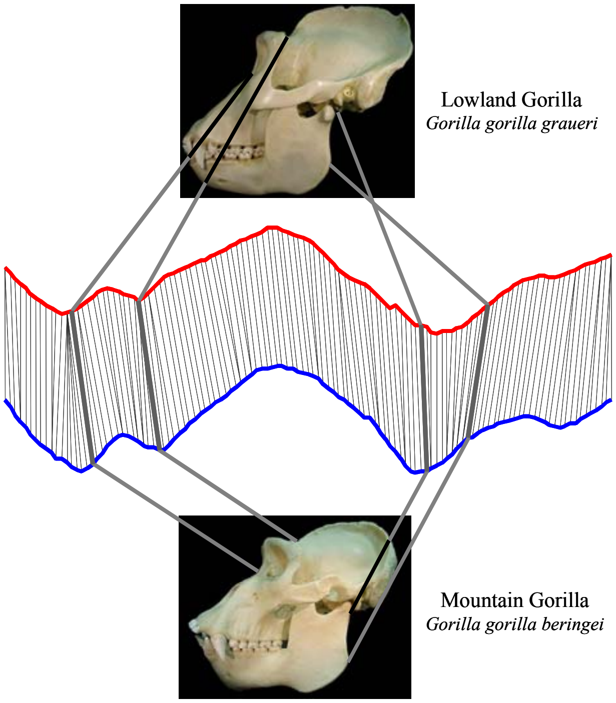


Figure 9: The Lowland Gorilla and Mountain Gorilla are morphologically similar, but have slightly different proportions. Dynamic Time Warping can be used to align homologous features in the time series representation space

Even if we assume that the database contains the actual object used as a query, it is possible that the two time series are distorted versions of each. Here the distortions may be caused by camera perspective effect, differences in lighting causing shadows which appear to be features, parallax etc.

Fortunately there is a well-known technique for compensating such local misalignments, Dynamic Time Warping (DTW) [11][20]. While DTW was invented in the context of 1D speech signals others have noted its utility for matching shapes, including face profiles [4], leafs [20], handwriting [21] and general shape matching [1].

To align two sequences using DTW, an n-by-n matrix is constructed, where the (i^{th}, j^{th}) element of the matrix is the distance $d(q_i, c_j)$ between the two points q_i and c_j (i.e. $d(q_i, c_j) = (q_i - c_j)^2$). Each matrix element (i, j) corresponds to the alignment between the points q_i and c_j, as illustrated in Figure 10.

A warping path P is a contiguous set of matrix elements that defines a mapping between Q and C. The t^{th} element of P is defined as $p_t = (i, j)_t$ so we have:

$$P = p_1, p_2, \dots, p_t, \dots, p_T \quad n \le T < 2n\text{-}1$$

The warping path that defines the alignment between the two time series is subject to several constraints. For example, the warping path must start and finish in diagonally opposite corner cells of the matrix; the steps in the warping path are restricted to adjacent cells (including diagonally adjacent cells); the points in the warping path must be monotonically spaced in time. In addition to these constraints, virtually all practitioners using DTW also constrain the warping path in a global sense by limiting how far it may stray from the diagonal [11][20][21]. A typical constraint is the Sakoe-Chiba Band which states that the warping path cannot deviate more than R cells from diagonal.

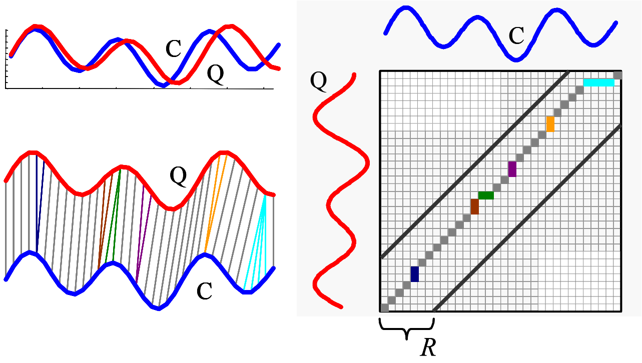


Figure 10: *Left)* Two time series sequences which are similar but out of phase. *Right)* To align the sequences we construct a warping matrix, and search for the optimal warping path, shown with solid squares. Note that Sakoe-Chiba Band with width R is used to constrain the warping path

The optimal warping path can be found in O(nR) time by dynamic programming [11]. As we shall show experimentally in the Section 5, DTW can significantly outperform Euclidean distance on real datasets.

Based on an arbitrary wedge W and the allowed warping range R, we define two new sequences, DTW_U and DTW_L:

$$DTW\_U_i = \max(U_{i-R} : U_{i+R})$$

$$DTW\_L_i = \min(L_{i-R} : L_{i+R})$$

They form an additional envelope above and below the wedge, as illustrated in Figure 11.

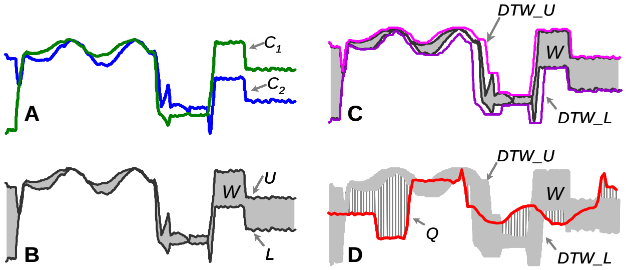


Figure 11: The idea of bounding envelopes introduced in Figure 4 is generalized to allow DTW. **A**) Two time series C_1 and C_2. **B**) A time series *wedge W,* created from C_1 and C_2. **C**) In order to allow lower bounding of DTW, an additional envelope is created above and below the wedge. **D**) An illustration of LB_Keogh_{DTW}

We can now define a lower bounding measure for DTW distance between an arbitrary query Q and the entire set of candidate sequences contained in a wedge W:

$$LB\_Keogh_{DTW}(Q,W)=\sqrt{\sum_{i=1}^{n}\begin{cases}(q_i-DTW\_U_i)^2 & \text{if } q_i>DTW\_U_i\\ (q_i-DTW\_L_i)^2 & \text{if } q_i<DTW\_L_i\\ 0 & \text{otherwise}\end{cases}}$$

We make the following claim:

Proposition 1: For any sequence Q of length n and a wedge W containing a set of time series $C_1, \ldots, C_k$ of the same length n, for any global constraint on the warping path of the form $j-R \le i \le j+R$, the following inequality holds:

$LB\_Keogh_{DTW}(Q, W) \le \min(DTW(Q, C_s))$, where $s = 1, 2, \ldots, k$.

Because of space limitations we refer the interested reader to [10] for the proof. In addition, space limitations also prohibit a discussion of the minor modifications required to index $LB\_Keogh_{DTW}(Q,W)$, however [23] contains the necessary modifications for both DTW and for LCSS which is discussed below.

To facilitate later efficiency comparisons to Euclidean distance and other methods, it will be useful to define the time complexity of DTW in terms of "num_steps" as returned by Table 1 and Table 5. The variable "num_steps" is the number of real-value subtractions that must be performed, and completely dominates the CPU time, since the square root function is only performed once (and can be removed, see [12]). If we construct a full n by n warping matrix, then DTW clearly requires at least n^2 steps. However as we noted above and illustrated in Figure 10, we can truncate the corners of the matrix to reduce this number to approximately nR, where R is the width of the Sakoe-Chiba Band. While nR is the number of steps for a single DTW, we expect the average number of steps to be less, because some full DTW calculations will not be needed if the lower bound test fails. Since the lower bound test requires n steps, the average number of steps when doing m comparisons should be:

$$\frac{m*a(nR)+m(n)}{m}$$

Where a is the fraction of the database that requires the full DTW calculated. Note that even this is pessimistic, since both DTW[2] and LB_Keogh_{DTW} are implemented as early abandoning (recall Table 5). We therefore simply count the "num_steps" required by each approach and divide it by m to get the average number of steps required for one comparison.

In addition to DTW, several researchers have suggested using Longest Common SubSequence (LCSS) as a distance measure for shapes. The LCSS is very similar to DTW except that while DTW insists that every point in C maps onto one (or more) point(s) in Q, LCSS allows some points to go unmatched. The intuition behind this idea in a time series domain is that subsequences may contain additions or deletions, for example an extra (or forgotten) dance move in a motion capture performance, or a missed beat in ECG data. Rather than forcing DTW to produce an unnatural alignment between two such sequences, we can use LCSS, which simply ignores parts of the time series that are too difficult to match. In the image space the missing section of the time series may correspond to a partial occlusion of an object, or to a physically missing part of the object, as shown in Figure 12.

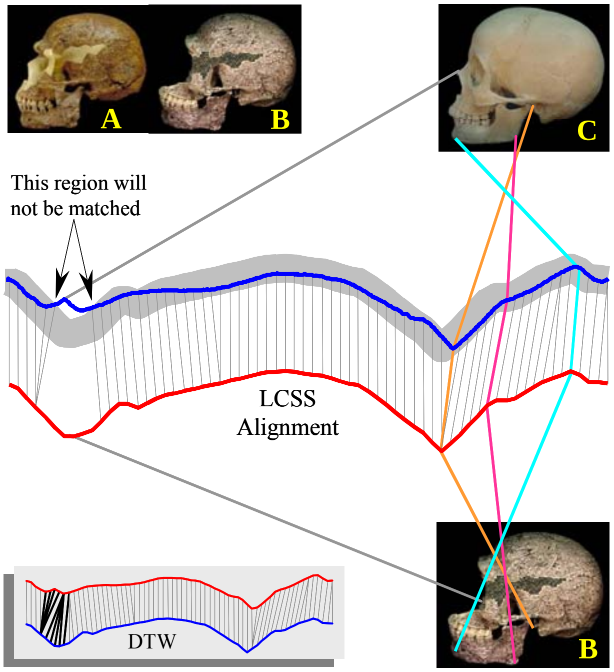


Figure 12: **A**) The famous Skhul V is generally reproduced with the missing bones extrapolated in epoxy, however the original Skhul V (**B**) is missing the nose region, which means it will match to a modern human (**C**) poorly, even after DTW alignment (inset). In contrast, LCSS alignment will not attempt to match features that are outside a "matching envelope" (heavy gray line) created from the other sequence.

While we considered LCSS for generality, we will not further explain how to incorporate it into our framework. It has been shown in [23] that it is trivial to lower bound LCSS using the envelope-based techniques described above. The minor changes include reversing some inequality signs since LCSS is a similarity measure, not a distance measure. Our omission here of a detailed discussion is due to space limitations and to a slight bias against the method. Unlike Euclidean distance which has no parameters, or DTW, which has one intuitive and easy to set parameter, LCSS requires 2 parameters, and tuning them is nontrivial. In experiments we found that we could sometimes tune LCSS to *slightly* beat DTW on *some* problems, however we did not have large enough datasets to allow training/test splits that guarded against overfitting to a statistically significant standard.

5. EXPERIMENTAL RESULTS

In this section we empirically evaluate our approach. We begin by stating our experimental philosophy. In a recent paper Veltkamp and Latecki attempted to reproduce the accuracy claims of several shape matching papers but discovered to their dismay that they could not match the claimed accuracy for any approach [22]. One suggested reason is the observation that many approaches have highly tuned parameters, a fact which we believe makes Euclidean distance (zero parameters) and DTW (one parameter) particularly attractive. Veltkamp and Latecki conclude "*It would be good for the scientific community if the reported test results are made reproducible and verifiable by publishing data sets and software along with the articles*". We completely concur and have placed *all* datasets at the following URL [10].

[2] Note that a *recursive* implementation of DTW would always require nR steps, however *iterative* implementation (as used here) can potentially early abandon with as few as R steps.

5.1 Effectiveness of Shape Matching

In general this paper is not making any claims about the *effectiveness* of shape matching. Because we are simply speeding up arbitrary distance calculations on arbitrary 1-dimensional representations of shapes, we automatically inherit the well-documented effectiveness of other researchers published work [1][2][3][7][8][20][24].

Nevertheless, for completeness and in order to justify the extra computational expense of DTW, we will show the effectiveness of shape matching on several publicly available datasets.

Table 8 shows the error rate of one-nearest neighbor classification as measured using leaving-one-out evaluation. Recall that Euclidean distance has no parameters, DTW has a single parameter (the warping window width *R*) which was learned by looking only at the training data. For the Face and Leaf datasets the (approximate) correct rotation was known [20]. We removed this information by randomly rotating the images.

Table 8: The Error of Euclidean distance and DTW on several publicly available datasets

Name	Number of Classes	Number of Instances	Euclidean Error (%)	DTW Error (%) {*R*}
Face	16	2240	3.839%	3.170% {3}
Swedish Leaves	15	1125	13.33%	10.84% {2}
Chicken	5	446	19.96%	19.96% {1}
MixedBag	9	160	4.375%	4.375% {1}
OSU Leaves	6	442	33.71%	15.61% {2}
Diatoms	37	781	27.53%	27.53% {1}

The MixedBag dataset is small enough to run the more computationally expensive Chamfer [5] and Hausdorff [18] distance measures. They achieved an error rate of 6.0% and 7.0% respectively [24], slightly worse than Euclidean distance. Likewise the Chicken dataset allows us to compare directly to [17], which used identical experiments to test 6 different algorithms based on *discrete* sequences extracted from the shapes. The best of these algorithms had an error rate of 20.5% and took over a minute for each distance calculation, whereas our approach takes an average time of 0.0039 seconds for each distance calculation[3]. For the Diatom dataset, the results are competitive with human experts, whose error rates ranged from 57% to 13.5% [8], and only slightly worse than the Morphological Curvature Scale Spaces (MCSS) approach of [8], which got 26.0%. Note however that the Euclidean distance requires zero parameters once the time series have been extracted, whereas the MCSS has several parameters to set.

In general these experiments show two things (which had been noted before), the extra effort of DTW is useful in some domains, and very simple time series representations of shapes are completive to other more complex representations.

We also performed extensive "sanity check" experiments using a large database of primate skulls. For all species where we have at least two examples we perform a hierarchal clustering and check to see if both samples of the same species clustered together. Figure 13 shows a typical example.

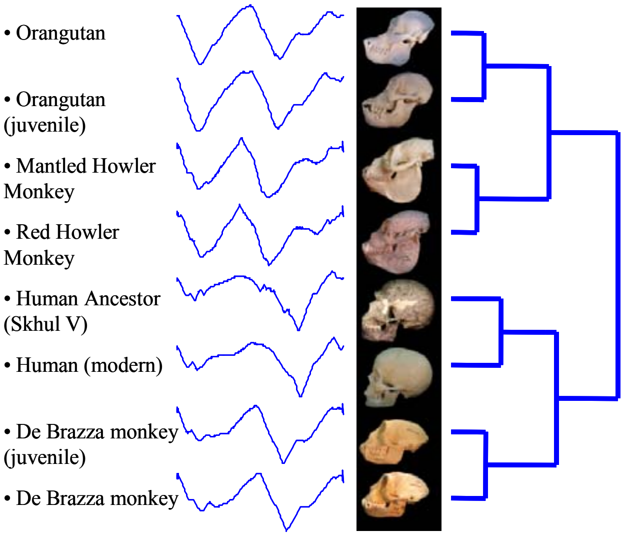


Figure 13: A group average hierarchal clustering of eight primate skulls based on the lateral view, using Euclidean distance

It is important to recall that Figure 13 shows a phenogram, *not* a phylogenetic tree. However on larger scale experiments in this domain (shown in [10]) we found that large subtrees of the dendrograms did conform to the current consensus on primate evolution.

5.2 Main Memory Experiments

There is increasing awareness that comparing two competing approaches using only CPU time opens the possibility of implementation bias [12]. As a simple example, while the Haar wavelet transform is O(n) and DFT is O($n\log n$), the DFT is *much* faster in the popular language Matlab, simply because it is a highly optimized subroutine. For this reason many recent papers compare approaches with some implementation-free metric [11][20][23][24]. As we noted earlier, the variable "num_steps" returned by Table 1 and Table 5 allows an implementation free measure to compare performance.

For Euclidean distance queries we compare to *brute force* and *Fourier* (FFT) methods, which are the only competitors to also guarantee no false dismissals. The cost model for the FFT lower bound is $n\log n$ steps. If the FFT lower bound fails we allow the approach to avail of our early abandoning techniques discussed in Section 3.

We tested on two datasets, a homogeneous database of 16,000 projectile point images, all of length 251 and a heterogeneous dataset consisting of all the data used in the classification experiments, plus 1,000 projectile points. In total the heterogeneous dataset contains 5,844 objects of length 1,024. To measure the performance we averaged over 50 runs, with the query object randomly chosen and removed from the dataset.

We measure the average number of steps required by each approach for a single comparison of two shapes, divided by the number of steps require by brute force. For our method, we include a startup cost of O(n^2), which is the time require to build the wedges. Because the utility of early abandoning depends on the value of the best-so-far, we expect our method to do better as we see larger and larger datasets.

Figure 14 shows the results on the projectile points dataset using Euclidean distance.

[3] We are aware that one should normally not compare CPU times from different computers, however here the 4 orders of magnitude offers a comfortable margin that dwarfs implementation details.

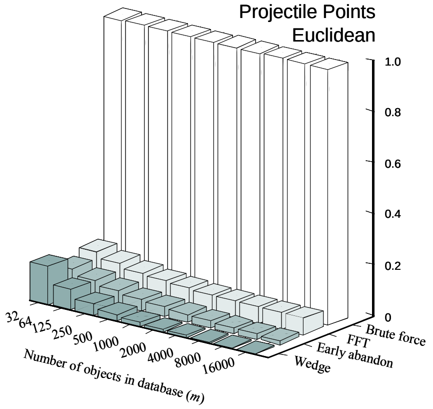


Figure 14: The relative performance of four algorithms on the Projectile Points dataset using the Euclidean distance measure

We can see that for small datasets our approach is slightly worse than *FFT* and simple *Early abandon* because we had to spend some time building the wedges. However, by the time we have seen 64 objects we have already broken even, and thereafter rapidly race towards beating *FFT* and *Early abandon* by one order of magnitude and *Brute force* by two orders of magnitude.

The results on the projectile points dataset using DTW are shown in Figure 15, and are even more dramatic.

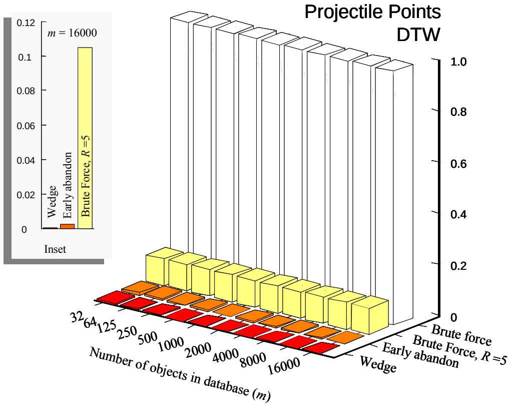


Figure 15: The relative performance of four algorithms on the Projectile Points dataset using the DTW distance measure. The inset shows a zoom-in of the 3 best algorithms when m = 16,000

Here the cost of building the wedges is dwarfed by a single brute force DTW-rotation-invariant comparison, so our approach is faster even for a database of size 3. By the time we have examined the entire database, our approach is more than 5,000 times faster than the brute force approach. It is interesting to note that the early abandoning strategy is by itself quite competitive, yet to our knowledge no one uses it. We suspect this is because most people are more familiar with the elegant and terse recursive version of DTW, which does not allow early abandoning, than the iterative implementation, which does. Note however that even though our highly optimized early abandoning strategy is competitive, our wedge approach is still an order of magnitude faster once the dataset is larger than 500 objects.

Sometimes indexing methods that work well for highly homogeneous datasets do not work well for heterogenous datasets, and vice versa. We consider this possibility by testing on the heterogenous dataset in Figure 16.

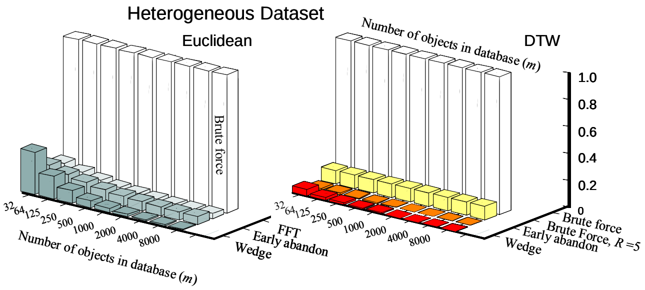


Figure 16: The relative performance of four algorithms on the Heterogeneous dataset using Euclidean distance (*left*) and DTW (*right*)

In this dataset it takes our wedge approach slightly longer to beat *Early abandon* (and *FFT* for Euclidean search), however by the time we have seen 8,000 objects our approach is two orders of magnitude faster than its Euclidean competitors, and for DTW it is an order of magnitude faster than *Early abandon* and 3,976 times faster than brute force.

Recall that our algorithm requires the setting of a single parameter, the number of intervals to search for a new value for K every time the bestSoFar variable is updated. In all the experiments above this value was set to 5. We found that we can change this value to any number in the range 3 to 20 without affecting the performance of our algorithm by more than 4%, we therefore omit further discussion of this parameter setting.

As a final sanity check we also measured the wall clock time of our best implementation of all method. The results are essentially identical to those shown above.

5.3 Disk Access Experiments

The results in the previous section show that we can do true rotation invariant matching so fast that CPU time is no longer the bottleneck, and we should therefore also attempt to minimize disk accesses. We will compare to Linear Scan, which is the only other competitor that we are aware of that allows exact rotation invariant indexing under Euclidean distance and DTW with a guarantee of no false dismissals. Recall that the lower bound used by the VP-tree requires transforming the signal to the Fourier space and calculating the Euclidean distance between the coefficient magnitudes [24]. It is well understood that most of the energy of the signal will be concentrated in a relatively small number of these coefficients [23] and that using just a few large valued coefficients is better than using all of them. We therefore will perform experiments keeping just the first D coefficients, were D = {4, 8, 16, 32}.

We count the fraction of items that must be retrieved from disk. Figure 17 illustrates the results for the full projectile points and heterogeneous datasets over a range of dimensionalities.

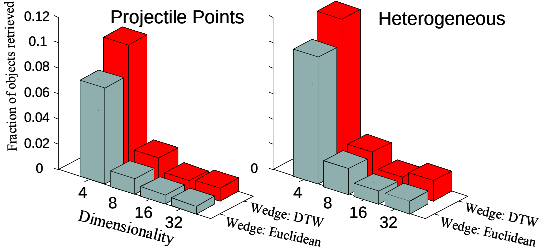


Figure 17: The fraction of items retrieved from disk to answer a 1-nearest neighbor query, using dimensionalities D = {4, 8, 16, 32}.

6. CONCLUSIONS AND FUTURE WORK

We have introduced a method to support fast rotation-invariant search of large shape datasets with arbitrary representations and distance functions. Our method supports rotation limited queries and mirror image invariance *if desired*.

Future work includes both extensions and applications of the current work. We will attempt to extend this approach to the indexing of 3D shapes, and we have begun to use our algorithm as a subroutine in several data mining algorithms which attempt to cluster, classify and discover motifs in a variety of anthropological datasets, including petroglyph and projectile point databases.

Reproducible Research Statement: All datasets and images used in this work are freely available at this URL [10].

Acknowledgements and Dedication: We would like to acknowledge Chotirat Ann Ratanamahatana and Longin Jan Latecki for useful suggestions and Jason Dorff for help with skull images. In addition we thank the many donors of datasets.

This paper, together with [11] and [13] is the final part of the LB_Keogh/VLDB trilogy. I would like to thank the VLDB reviewers and chairs that made this possible. I composed most of this paper in my head while on a flight to Dublin to see my mother for the last time (in order to distract myself). She was proud to see her last name in print. This paper is dedicated to Emily (Peggy) Keogh, 1927 to 2005.

8. REFERENCES

[1] Adamek, T. and O'Connor, N.E. A multiscale representation method for nonrigid shapes with a single closed contour. *IEEE Circuits and Systems for Video Technology*, 14(5): 742-753, 2004.

[2] Adamek, T. and O'Connor, N.E. Efficient contour-based shape representation and matching. *Multimedia Information Retrieval 2003*: 138-143.

[3] Attalla, E. and Siy, P. Robust shape similarity retrieval based on contour segmentation polygonal multiresolution and elastic matching. *Pattern Recognition*, 38(12): 2229-2241, 2005.

[4] Bhanu, B. and Zhou, X. Face recognition from face profile using dynamic time warping. In *Proceedings of International Conference on Pattern Recognition (ICPN'04)*, pp. 499-502, 2004.

[5] Borgefors, G. Hierarchical Chamfer Matching: A Parametric Edge Matching Algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 10(6): 849-865, November 1988.

[6] Cardone, A., Gupta, S.K., and Karnik, M. A survey of shape similarity assessment algorithms for product design and manufacturing applications. *ASME Journal of Computing and Information Science in Engineering*, 3(2): 109-118, 2003.

[7] Gdalyahu, Y. and Weinshall, D. Flexible syntactic matching of curves and its application to automatic hierarchical classification of silhouettes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(12): 1312-1328, Dec. 1999.

[8] Jalba, A.C., Wilkinson, M.H.F., Roerdink, J.B.T.M., Bayer, M.M., and Juggins, S. Automatic Diatom Identification using Contour Analysis by Morphological Curvature Scale Spaces. *Machine Vision and Applications*, 16(4): 217-228, 2005.

[9] Karydis, Y., Nanopoulos, A., Papadopoulos, A.N., and Manolopoulos, Y. Evaluation of Similarity Searching Methods for Music Data in Peer-to-Peer Networks. *International Journal of Business Intelligence and Data Mining*, 1(2): 210-228, 2005.

[10] Keogh, E. www.cs.ucr.edu/~eamonn/shape/shape.htm, 2006.

[11] Keogh, E. Exact indexing of dynamic time warping. In *Proceedings of the 28th International Conference on Very Large Data Bases*, Hong Kong. pp 406-417, 2002.

[12] Keogh, E. and Kasetty, S. On the Need for Time Series Data Mining Benchmarks: A Survey and Empirical Demonstration. In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Edmonton, Canada. pp 102-111, 2002.

[13] Keogh, E., Palpanas, T., Zordan, V., Gunopulos, D., and Cardle, M. Indexing Large Human-Motion Databases. In *Proceedings of the 30th International Conference on Very Large Data Bases*, Toronto, Canada, pp 780-791, 2004.

[14] Li, D. and Simske, S. *Shape Retrieval Based on Distance Ratio Distribution*. HP Tech Report. HPL-2002-251, 2002.

[15] Li, Q., Lopez, I., and Moon, B. Skyline Index for Time Series Data. *IEEE Transactions on Knowledge and Data Engineering*. 16(6): pp 669-684, 2004.

[16] Ling, H. and Jacobs, D.W. Using the Inner-Distance for Classification of Articulated Shapes. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Vol. II, pp 719-726, 2005.

[17] Mollineda, R. A., Vidal, E., and Casacuberta, F. Cyclic Sequence Alignments: Approximate Versus Optimal Techniques. International Journal of Pattern Recognition and Artificial Intelligence (IJPRAI), 16(3): 291-299, 2002.

[18] Olson, C. F. and Huttenlocher, D. P. Automatic Target Recognition by Matching Oriented Edge Pixels. *IEEE Transactions on Image Processing*, 6(1): 103-113, January 1997.

[19] Osada, R., Funkhouser, T., Chazelle, B., and Dobkin, D. Shape Distributions. *ACM Transactions on Graphics*, 21(4): 807-832, October, 2002.

[20] Ratanamahatana, C. A. and Keogh, E. Three Myths about Dynamic Time Warping. In *Proceedings of SIAM International Conference on Data Mining (SDM '05)*, Newport Beach, CA, April 21-23, pp 506-510, 2005.

[21] Rath, T. and Manmatha, R. *Lower-Bounding of Dynamic Time Warping Distances for Multivariate Time Series.* Tech Report MM-40, University of Massachusetts Amherst, 2002.

[22] Veltkamp, R. C. and Latecki, L. J. Properties and Performance of Shape Similarity Measures. In *Proceedings of IFCS 2006 Conference: Data Science and Classification*. July, 2006.

[23] Vlachos, M., Hadjieleftheriou, M., Gunopulos, D. and Keogh. E. Indexing Multi-Dimensional Time-Series with Support for Multiple Distance Measures. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp 216-225, August 24-27, 2003, Washington, DC, USA.

[24] Vlachos, M., Vagena, Z., Yu, P. S., and Athitsos, V. Rotation invariant indexing of shapes and line drawings. In *Proceedings of ACM Conference on Information and Knowledge Management (CIKM)*, pp 131-138, 2005.

[25] Wang, Z., Chi, Z., Feng, D., and Wang, Q. Leaf Image Retrieval with Shape Features. In *Proceedings of the 4th International Conference on Advances in Visual Information Systems*, pp 477- 487, 2000.

[26] Wei, L., Keogh, E., Van Herle, H., and Mafra-Neto, A. Atomic Wedgie: Efficient Query Filtering for Streaming Time Series. In *Proceedings of the 5th IEEE International Conference on Data Mining (ICDM 2005)*, pp 490-497, 2005.

[27] White, T. D. *Human Osteology*. 2nd edition. San Diego: Academic Press, 2000.

[28] Zhang, D. and Lu, G. Review of shape representation and description techniques. *Pattern Recognition*, 37(1): 1-19, 2004.

[29] Zunic, J., Rosin, p., and Kopanja, L. Shape Orientability. ACCV (2) 2006: pp 11-20.

Appendix

Below we present some additional materials that we could not fit into the conference version of this paper. To enhance readability we have repeated some text from above.

EUCLIDEAN DISTANCE LOWER BOUND

Definition 1. *Euclidean Distance:* given two time series (or time series subsequences) both of length *n*, the Euclidean Distance between them is the square root of the sum of the squared differences between each pair of corresponding data points:

$$ED(Q,C) \equiv \sqrt{\sum_{i=1}^{n}(q_i - c_i)^2}$$

Figure 18 gives a visual intuition behind the Euclidean distance.

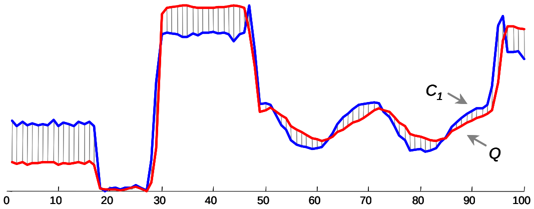


Figure 18: The visual intuition behind the Euclidean distance. The Euclidean distance is the square root of the sum of the square lengths of the gray hatch lines

Given a set of time series $C_1,..,C_k$, we can form two new sequences *U* and *L*:

$$U_i = \max(C_{1i},..,C_{ki})$$
$$L_i = \min(C_{1i},..,C_{ki})$$

U and *L* stand for Upper and Lower respectively, as shown in Figure 4. They form the smallest possible bounding envelope that encloses all members of the set $C_1,..,C_k$ from above and below. More formally:

$$\forall_i \quad U_i \geq C_{1i},..,C_{ki} \geq L_i$$

For notational convenience, we will call the combination of *U* and *L* a *wedge*, and denote a wedge as *W*:

$$W = \{U, L\}$$

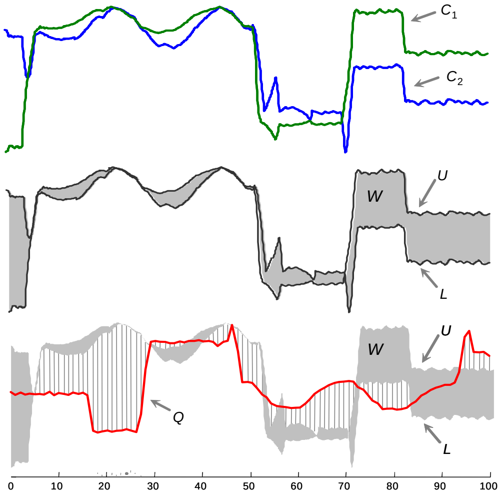


Figure 19: *Top*) Two time series C_1 and C_2. *Middle*) A time series wedge *W*, created from C_1 and C_2. *Bottom*) An illustration of LB_Keogh

Definition 2. *LB_Keogh:* we can now define a lower bounding measure between an arbitrary time series *Q* and the entire set of candidate sequences contained in a wedge *W*:

$$LB\_Keogh(Q,W) = \sqrt{\sum_{i=1}^{n}\begin{cases}(q_i - U_i)^2 & \text{if } q_i > U_i \\ (q_i - L_i)^2 & \text{if } q_i < L_i \\ 0 & \text{otherwise}\end{cases}}$$

We will now prove the claim of the lower bounding.

Proposition 1: For any sequence *Q* of length *n* and a wedge *W* containing a set of time series $C_1,..,C_k$ of the same length *n*, the following inequality holds:

$$LB\_Keogh(Q,W) \leq ED(Q,C_s)\text{, where } s = 1, 2, ..., k.$$

Proof:

Suppose we know that among the *k* time series $C_1,..,C_k$, C_s has the minimal Euclidean distance to query *Q*. And we wish to prove

$$\sqrt{\sum_{i=1}^{n}\begin{cases}(q_i - U_i)^2 & \text{if } q_i > U_i \\ (q_i - L_i)^2 & \text{if } q_i < L_i \\ 0 & \text{otherwise}\end{cases}} \leq \sqrt{\sum_{i=1}^{n}(q_i - C_{si})^2}$$

Since the terms under radicals are positive, we can square both sides:

$$\sum_{i=1}^{n}\begin{cases}(q_i - U_i)^2 & \text{if } q_i > U_i \\ (q_i - L_i)^2 & \text{if } q_i < L_i \\ 0 & \text{otherwise}\end{cases} \leq \sum_{i=1}^{n}(q_i - C_{si})^2$$

Below we will show that every term in the left summation can be matched with some greater or equal term in the right summation.

There are three cases to consider, for the moment we will just consider the case when $q_i > U_i$. We want to show:

$$(q_i - U_i)^2 \leq (q_i - C_{si})^2$$

$(q_i - U_i) \leq (q_i - C_{si})$ — Since $q_i > U_i$, we can take square roots on both sides

$-U_i \leq -C_{si}$ — Subtract q_i from both sides

$C_{si} \leq U_i$ — Add $U_i + C_{si}$ to both sides

$C_{si} \leq \max(C_{1i},...,C_{ki})$ — By definition $U_i = \max(C_{1i},..,C_{ki})$

This is obviously true.

The case $q_i < L_i$ yields to a similar argument. This final case is simple to show, since clearly $0 \leq (q_i - C_{si})^2$ because $(q_i - C_{si})^2$ must be nonnegative.

Thus we have shown that each term on the left side is matched with an equal or larger term on the right side. Our inequality holds. ■

DTW DISTANCE LOWER BOUND

LB_Keogh can be generalized to Dynamic Time Warping distance (DTW). Below we will first give some definitions and then provide the proof.

Suppose we have two time series *Q* and *C*, both of length *n*, where:

$$Q = q_1, q_2, \ldots, q_i, \ldots, q_n$$
$$C = c_1, c_2, \ldots, c_j, \ldots, c_n$$

To align these two sequences using DTW, an n-by-n matrix is constructed, where the (i^{th}, j^{th}) element of the matrix is the distance $d(q_i, c_j)$ between the two points q_i and c_j (i.e. $d(q_i, c_j) = (q_i - c_j)^2$). Each matrix element (i, j) corresponds to the alignment between the points q_i and c_j, as illustrated in Figure 10.

Definition 3. *Warping path:* a warping path P is a contiguous set of matrix elements that defines a mapping between Q and C. The t^{th} element of P is defined as $p_t = (i, j)_t$ so we have:

$$P = p_1, p_2, \ldots, p_t, \ldots, p_T \qquad n \leq T < 2n\text{-}1$$

The warping path that defines the alignment between the two time series is subject to several constraints. For example, the warping path must start and finish in diagonally opposite corner cells of the matrix; the steps in the warping path are restricted to adjacent cells (including diagonally adjacent cells); the points in the warping path must be monotonically spaced in time. In addition to these constraints, virtually all practitioners using DTW also constrain the warping path in a global sense by limiting how far it may stray from the diagonal [11][20][21].

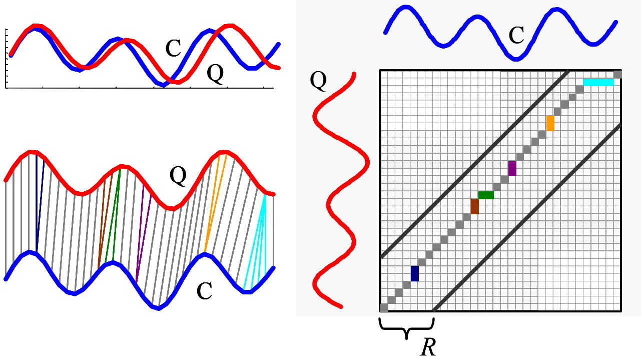


Figure 20: *Left)* Two time series sequences which are similar but out of phase. *Right)* To align the sequences we construct a warping matrix, and search for the optimal warping path, shown with solid squares. Note that Sakoe-Chiba Band with width R is used to constrain the warping path

Definition 4. *Warping Window*: the subset of matrix that the warping path is allowed to visit is called the warping window.

The warping window constrains the indices of the warping path $p_t = (i, j)_t$ such that $j - R \leq i \leq j + R$, where R is a term defining the *reach*, or allowed range of warping, for a given point in a sequence.

There are exponentially many warping paths that satisfy the above conditions, however we are only interested in the path that minimizes the warping cost.

Definition 5. *DTW Distance:* given two time series (or time series subsequences), the *DTW Distance* between them is the minimal cost of all warping paths:

$$DTW(Q,C) = \min\left\{ \sqrt{\sum_{t=1}^{T} p_t} \right\}$$

In Section 0, we have shown that given a set of time series $C_1,..,C_k$, we can form two new sequences U and L:

$$U_i = \max(C_{1i},..,C_{ki})$$
$$L_i = \min(C_{1i},..,C_{ki})$$

Based on the wedge W and the allowed warping range R, we can now define two new sequences, DTW_U and DTW_L:

$$DTW\_U_i = \max(U_{i-R} : U_{i+R})$$
$$DTW\_L_i = \min(L_{i-R} : L_{i+R})$$

They form an additional envelope above and below the wedge, as illustrated in Figure 11.

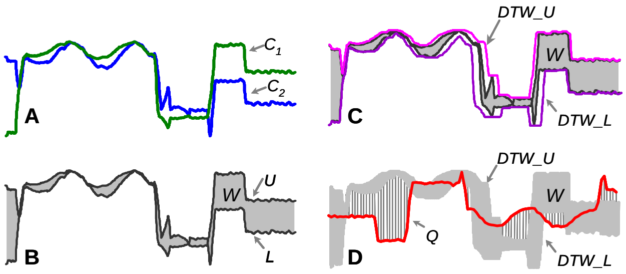


Figure 21: The idea of bounding envelopes introduced in Figure 4 is generalized to allow DTW. **A**) Two time series C_1 and C_2. **B**) A time series *wedge W,* created from C_1 and C_2. **C**) In order to allow lower bounding of DTW, an additional envelope is created above and below the wedge. **D**) An illustration of LB_Keogh_{DTW}

We can now define a lower bounding measure for DTW distance between an arbitrary query Q and the entire set of candidate sequences contained in a wedge W:

$$LB\_Keogh_{DTW}(Q,W) = \sqrt{\sum_{i=1}^{n} \begin{cases} (q_i - DTW\_U_i)^2 & \text{if } q_i > DTW\_U_i \\ (q_i - DTW\_L_i)^2 & \text{if } q_i < DTW\_L_i \\ 0 & \text{otherwise} \end{cases}}$$

We will now prove the claim of the lower bounding.

Proposition 2: For any sequence Q of length n and a wedge W containing a set of time series $C_1, \ldots, C_k$ of the same length n, for any global constraint on the warping path of the form $j - R \leq i \leq j + R$, the following inequality holds:

$LB\_Keogh_{DTW}(Q,W) \leq DTW(Q,C_s)$, where $s = 1, 2, ..., k$.

Proof:

Suppose we know that among the k time series $C_1, \ldots, C_k$, C_s has the minimal DTW distance to query Q. And we wish to prove

$$\sqrt{\sum_{i=1}^{n} \begin{cases} (q_i - DTW\_U_i)^2 & \text{if } q_i > DTW\_U_i \\ (q_i - DTW\_L_i)^2 & \text{if } q_i < DTW\_L_i \\ 0 & \text{otherwise} \end{cases}} \leq \sqrt{\sum_{t=1}^{T} p_{st}}$$

Since the terms under radicals are positive, we can square both sides:

$$\sum_{i=1}^{n} \begin{cases} (q_i - DTW\_U_i)^2 & \text{if } q_i > DTW\_U_i \\ (q_i - DTW\_L_i)^2 & \text{if } q_i < DTW\_L_i \\ 0 & \text{otherwise} \end{cases} \leq \sum_{t=1}^{T} p_{st}$$

From Definition 3 we know that $n \leq T$, so our strategy will be to show that every term in the left summation can be matched with some greater or equal term in the right summation.

There are three cases to consider, for the moment we will just consider the case when $q_i > DTW\_U_i$. We want to show:

$$(q_i - DTW\_U_i)^2 \leq p_{st}$$

$$(q_i - DTW\_U_i)^2 \leq (q_i - C_{sj})^2 \qquad \text{By Definition 3}$$

$$(q_i - DTW\_U_i) \leq (q_i - C_{sj})$$

Since q_i > DTW_U_i , we can take square roots on both sides

$$-DTW\_U_i \leq -C_{sj}$$

Subtract q_i from both sides

$$C_{sj} \leq DTW\_U_i$$

Add $DTW\_U_i + C_{sj}$ to both sides

By definition DTW_U_i = $\max(U_{i-R} : U_{i+R})$

$$C_{sj} \le \max(U_{i-R} : U_{i+R})$$

Since the query sequence Q and all the candidate sequences C_1, …, C_k are of the same length and $j\text{-}R \le i \le j\text{+}R$, we know $i\text{-}R \le j \le i\text{+}R$. So we can rewrite the right side and the inequality becomes

$$C_{sj} \le \max(U_{i-R}, U_{(i+1)-R}, \ldots, U_j, \ldots, U_{i+R})$$

If we remove all terms except U_j from the RHS we are left with $C_{sj} \le \max(U_j)$ which is obviously true since $U_j = \max(C_{1j},..,C_{kj})$.

The case $q_i < DTW\_U_i$ yields to a similar argument. The final case is simple to show, since clearly $0 \le (q_i - C_{sj})^2$ because $(q_i - C_{sj})^2$ must be nonnegative.

Thus we have shown that each term on the left side is matched with an equal or larger term on the right side. Our inequality holds. ■

ADDITIONAL SHAPE MATCHING EXAMPLES

Figure 22 shows an alternative figure to explain and motivate DTW for shapes.

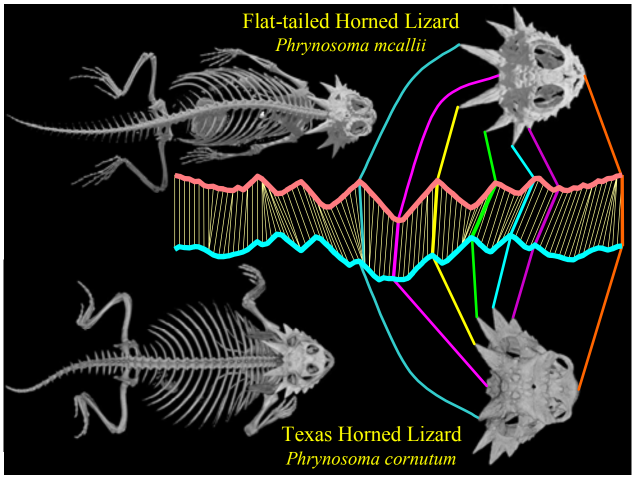


Figure 22: This figure was an alternative to Figure 9 above, however we decided to stick with the primate motif for this paper

With Dr. Wendy Hodges, a noted herpetologist at the University of Texas, we have begun to study reptilian morphology using our shape matching tools. Figure 23 shows a simple sanity check clustering of some reptile skulls.

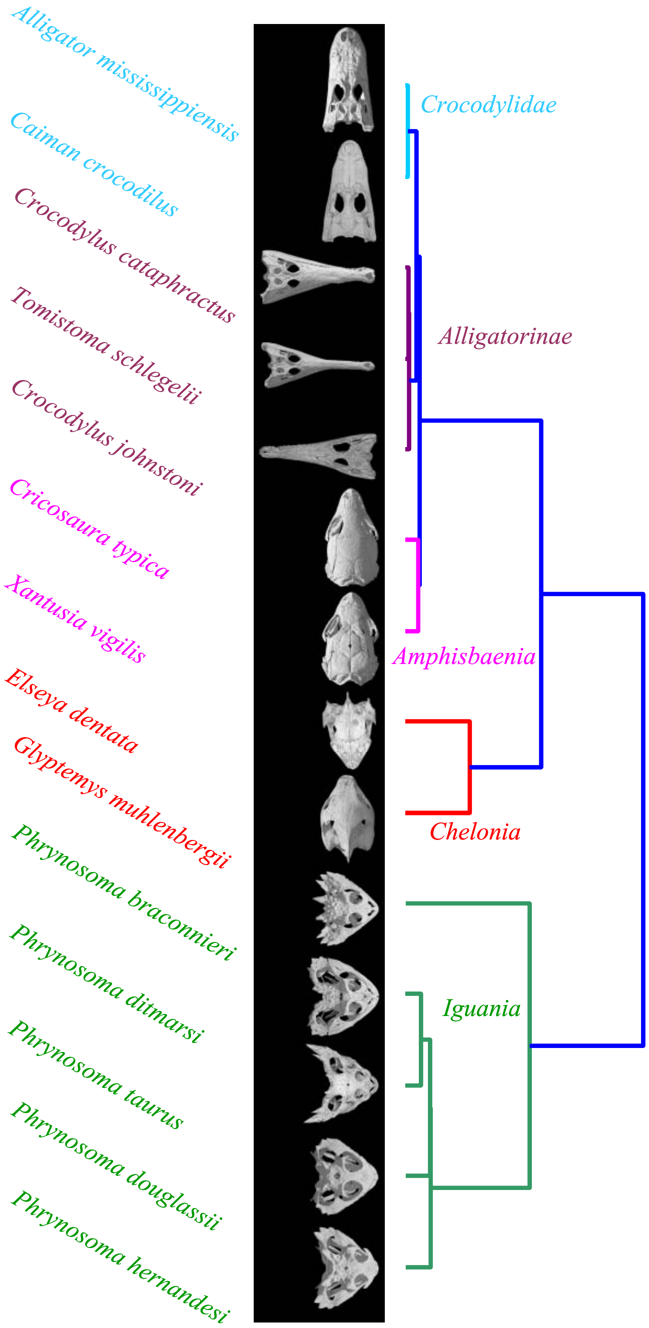
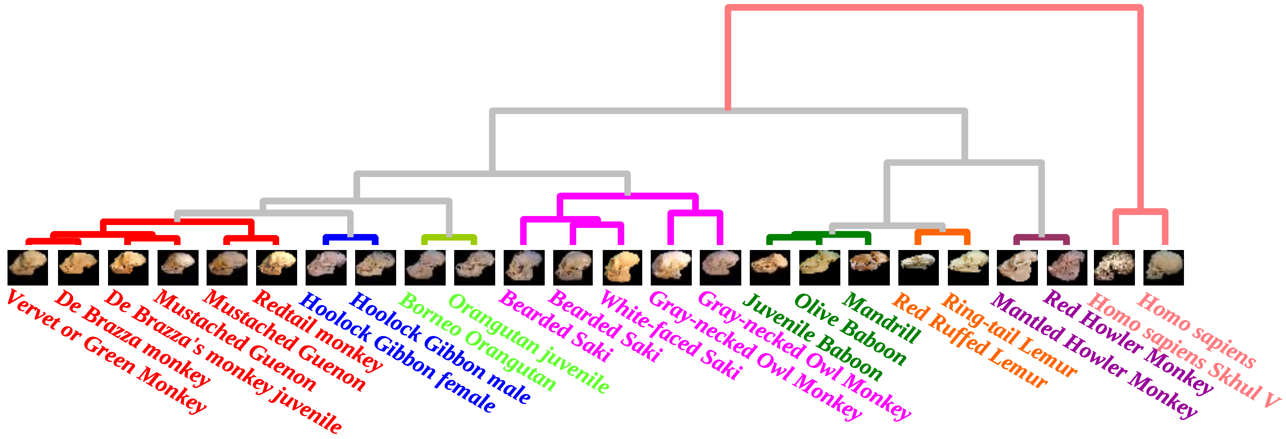


Figure 23: A group average hierarchal clustering of fourteen reptile skulls using rotation invariant Euclidean distance

Finally, in Figure 24 we show a clustering of twenty-four primate skulls, including all the skulls previously featured in the paper.



All these are in the genus Cercopithecus, except for the skull identified as being either a Vervet or Green monkey, both of which belong in the Genus of Chlorocebus which is in the same Tribe (***Cercopithecini***) as *Cercopithecus*.

Tribe *Cercopithecini*

Cercopithecus

De Brazza's Monkey, *Cercopithecus neglectus*
Mustached Guenon, *Cercopithecus cephus*
Red-tailed Monkey, *Cercopithecus ascanius*

Chlorocebus

Green Monkey, *Chlorocebus sabaceus*
Vervet Monkey, *Chlorocebus pygerythrus*

These are the same species ***Bunopithecus hooloc*** (Hoolock Gibbon)

These are in the Genus ***Pongo***

All these are in the family ***Cebidae***

Family *Cebidae (New World monkeys)*

Subfamily Aotinae
Aotus trivirgatus
Subfamily Pitheciinae sakis
Black Bearded Saki, Chiropotes satanas
White-nosed Saki, Chiropotes albinasus

All these are in the tribe ***Papionini***

Tribe Papionini
Genus Papio – baboons
Genus Mandrillus- Mandrill

These are in the family ***Lemuridae***

These are in the genus ***Alouatta***

These are in the same species ***Homo sapiens*** (Humans)

Important Note: This key requires color viewing of this graphic

Figure 24: A group average hierarchal clustering of twenty-four reptile skulls using rotation invariant Euclidean distance. Note that this is a phenogram, not a phylogenetic tree. For the current best phylogenetic tree, see the Tree of Life project, starting at their primate page.