Iterative Deepening Dynamic Time Warping for Time Series

***Selina Chu*[†]*, Eamonn Keogh*[‡]*, David Hart*[†]*, and Michael Pazzani*[†]

1 Introduction

Time series are a ubiquitous form of data occurring in virtually every scientific discipline and business application. There has been much recent work on adapting data mining algorithms to time series databases. For example, Das et al. attempt to show how association rules can be learned from time series [7]. Debregeas and Hebrail [8] demonstrate a technique for scaling up time series clustering algorithms to massive datasets. Keogh and Pazzani introduced a new, scalable time series classification algorithm [16]. Almost all algorithms that operate on time series data need to compute the similarity between them. Euclidean distance, or some extension or modification thereof, is typically used. However as we will demonstrate in Section 2.1, Euclidean distance can be an extremely brittle distance measure.

The reason why Euclidean distance may fail to produce an intuitively correct measure of similarity between two sequences is that it is very sensitive to small distortions in the time axis. Consider Figure 1.A, the two sequences have approximately the same overall shape, but the shapes are not aligned in the time axis. The nonlinear alignment shown in Fig 1.B would allow a more intuitive distance measure to be calculated.

A method that allows this elastic shifting of the X-axis is desired in order to detect similar shapes with different phases. Such a technique has long been known in the speech processing community [29, 26]. The technique, Dynamic Time Warping (DTW), was

[†] Department of Information and Computer Science, University of California, Irvine, California 92697, {selina, dhart, pazzani}@ics.uci.edu.
[‡] Department of Computer Science and Engineering, University of California, Riverside, California 92521, eamonn@cs.ucr.edu.

introduced to the data mining community by Berndt and Clifford [4]. Although they demonstrate the utility of the approach, they acknowledge that the algorithm's time complexity is a problem and that "*…performance on very large databases may be a limitation*". Despite this shortcoming of DTW, it is still widely used in various fields. In bioinformatics, Aach and Church successfully applied DTW to RNA expression data [1]. In chemical engineering, it has been used for the synchronization and monitoring of batch processes in polymerization [14]. DTW has been successfully used to align biometric data, such as gait, signatures and even fingerprints [11, 14, 22, 19]. Several researchers including Vullings et al. [30] and Caiani et al. [5] have demonstrate the use of DTW for ECG pattern matching. Finally in robotics, Oates et al. demonstrated that DTW may be used for clustering an agent's sensory outputs [23].

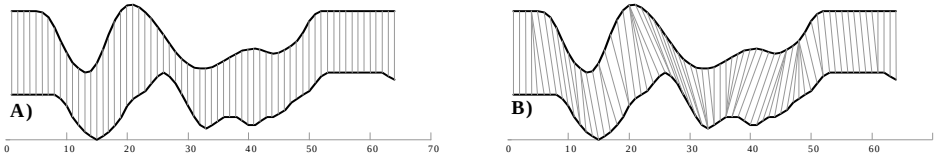

Figure 1: Note that while the sequences have an overall similar shape, they are not aligned in the time axis. Euclidean distance, which assumes the i^{th} point on one sequence is aligned with the i^{th} point on the other (A), will produce a pessimistic dissimilarity measure. A nonlinear alignment (B) allows a more intuitive distance measure to be calculated.

Because of the algorithm's relative lethargy, various modifications to the algorithm have been suggested. Yi et al. proposed a technique that maps the data into Euclidean space such that their DTW distance is approximately preserved [33]. Unfortunately the method allows false dismissals, and more importantly, does not allow user to have direct control over the quality of the results returned. Rafiei and Mendelzon proposed an extension to their work on similarity searching to allow shifting and scaling of the time axis; however their method only handles global scaling of the time axis, and not the more general case of (local) time warping [27].

In previous work we introduced piecewise dynamic time warping (PDTW), a modification of DTW that performs warping on a reduced dimensionality representation of the data [17]. (Chan and Fu [6] later introduced a similar technique that operates on a different, but logically equivalent representation [15].) We demonstrated that this approach allows dramatic speedup with low probability of false dismissal. One important limitation of this work, however, is that the user must carefully specify the compression ratio used. If the user chooses too fine of an approximation, the gains in speed are negligible. In contrast, if the user chooses too coarse of an approximation, the number of false dismissals becomes unacceptable.

It is this limitation that motivates our current work. The intuition behind IDDTW is that for any given level of approximation, we can create a model describing the distribution of the distance approximation errors. Then, for any two time series, we can calculate the approximations of DTW at increasingly finer levels of representation and use the stored distribution models to filter out poor matches with some user-specified tolerance. All that the user needs to specify to the algorithm is simply the query and their tolerance for false dismissals. In the worst case, if the user has zero tolerance for false dismissals, our algorithm degrades to the classic DTW algorithm. However, as the users' tolerance for false dismissals increases, the search technique becomes much faster. The surprising result is that, even a very small tolerance for false dismissals results in extremely large gains in speed.

The rest of this paper is organized as follows. Section 2 provides more detailed motivation for IDDTW and a review of the classic DTW algorithm. Section 3 introduces the IDDTW algorithm. In section 4 we experimentally compare IDDTW to DTW and Euclidean distance on real and synthetic datasets. Section 5 contains a discussion of related work. Finally section 6 contains our conclusions and directions for future work.

2 Background

In this section, we motivate the utility of DTW with three original data mining experiments. For completeness, we will then review the classic dynamic time warping algorithm.

2.1 Dynamic Time Warping vs. Euclidean Distance

Although the utility of dynamic time warping has been extensively demonstrated in many domains [1, 5, 11, 14, 22, 23, 29, 30], for completeness we will provide brief motivating examples here. In particular, we consider the three most common data mining tasks, classification, clustering and the discovery of association rules. In each example we compare DTW to Euclidean distance, the most commonly used distance measure for time series [2, 7, 8, 10, 15].

2.1.1 Classification

There has been much work on classification of time series. The most commonly studied benchmark dataset is Cylinder-Bell-Funnel, a synthetic dataset introduced in [28] and used by [21, 13, 9] and others. The dataset consists of a 3-class problem, with the classes generated by the following equations:

$$c(t) = (6+\eta) \bullet X_{[a,b]}(t) + \varepsilon(t)$$
$$b(t) = (6+\eta) \bullet X_{[a,b]}(t) \bullet (t-a)/(b-a) + \varepsilon(t)$$
$$f(t) = (6+\eta) \bullet X_{[a,b]}(t) \bullet (b-a)/(b-t) + \varepsilon(t)$$
$$X_{[a,b]} = \{\ 1, \text{ if } a \le t \le b, \text{ else } 0\ \}$$

Where η and $\varepsilon(t)$ are drawn from a standard normal distribution $N(0,1)$, a is an integer drawn uniformly from the range [16, 32] and $(b-a)$ is an integer drawn uniformly from the range [32, 96]. Figure 2 shows an example from each class.

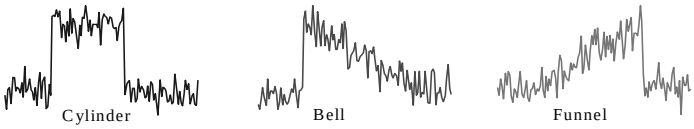

Figure 2: Examples of the Cylinder-Bell-Funnel dataset.

We performed a simple classification experiment on this dataset, using the one nearest neighbor algorithm. The dataset consists of ten instances of each class, and the classifier was evaluated using the "leave one out" strategy. Since we had the luxury of unlimited data, we averaged the results over 100 runs. Although several researchers have suggested methods to tune DTW to particular problems, we simply used the classic "off-the-shelf" algorithm as described in Section 2.2. Table 1 contains the results.

Two observations are immediately apparent. Using DTW can reduce the error rate on this problem by a factor of 10. However DTW is also many orders of magnitude slower than Euclidean distance for this task.

Technique	Mean Error Rate	Mean Time (seconds)
Euclidean Distance	0.2610	1
DTW Distance	0.0287	4320

Table 1: A comparison of Euclidean distance and DTW distance on a classification task.

2.1.2 Clustering

For this experiment we attempted to cluster four power-demand time series. Each sequence corresponds to a week's demand for power in a Dutch research facility in 1997 [31]. Typically the weekly patterns vary little, with peaks beginning at 9:00am and lasting until 5:00pm each weekday, and relatively little activity on the weekends (sequences 1 and 2). However, in this experiment we included two anomalous sequences, one in which Wednesday was a national holiday (sequence 3), and one in which Monday was a national holiday (sequence 4). The results of clustering are shown in Figure 3.

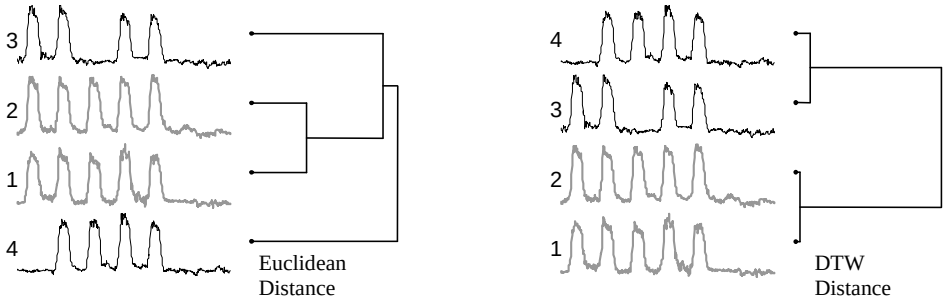

Figure 3: A comparison of Euclidean distance and DTW distance on a clustering task.

The Euclidean distance correctly identifies the two normal weeks as a cluster, but fails to group the two vacation weeks together because the vacation days are different. In contrast, DTW is able to discover the similarity between the two vacation weeks by warping Monday in sequence 3 to Tuesday in sequence 4 etc. Note that the grouping between sequences 3 and 4 is not as tight as the grouping between sequences 1 and 2, because there is a penalty paid for the warping effort. This maps nicely on to our intuitions of similarity.

2.1.3 Mining Association Rules

Although the discovery of association rules in discrete sequences is an area that has attracted extensive research [3], less work has been done on the discovery of association rules in time series. The most frequently referenced work in this area is by Das et al. [7], so we have implemented this method for our experiment. The method works by clustering the subsequences of a time series using K-means (or a greedy approximation thereof), then assigning a label to the centroid of each cluster. Once the data has been discretized in this fashion, classic association rule mining algorithms are used to find rules with a minimum support and confidence. The rules are then ranked by their J-measure, a measure of their interestingness.

We experimented on an ECG dataset because there are simple and obvious rules in such data. For example, if we see the first half of a single heartbeat (the RST wave), we should expect to see the second half of a heartbeat (the PQR wave) with very high confidence and support. Figure 4 shows the results of a single experiment (only the top three rules are shown).

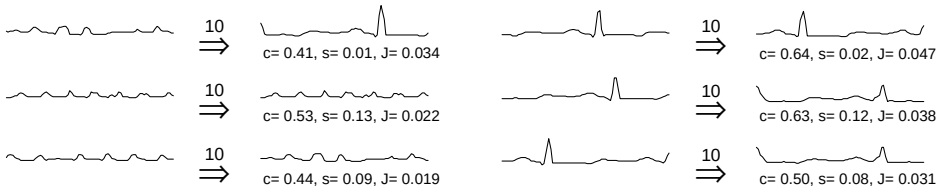

Figure 4: A comparison of Euclidean distance and DTW distance on an association rule task, where c, s, and J are confidence, support and "interestingness" respectively. (Parameter settings were w (length of prototype shapes) = 80, k (size of alphabet) = 10, T (maximum distance between antecedent and consequent) =10).

The most obvious result is that, with the Euclidean version, the prototypical shapes do not resemble the familiar electrocardiogram (with the exception of the highest ranked consequent). This is because the Euclidean centroid of the clusters do not produce shapes that are representative of the cluster. In contrast, the DTW centroid of the clusters produces a prototype that closely resembles all the members of the cluster. This allows DTW to find higher quality, more intuitive rules. To be fair, by carefully adjusting the three input parameters, we can make Euclidean distance produce more intuitive results but clearly DTW is more robust at finding meaningful rules.

2.2 The Dynamic Time Warping Algorithm

For completeness, we now review the classic DTW algorithm. The reader may skip this section without loss of continuity.

Suppose we have two time series Q and C, of length n and m respectively, where:

$$Q = q_1,q_2,\ldots,q_i,\ldots,q_n \quad (1)$$

$$C = c_1,c_2,\ldots,c_j,\ldots,c_m \quad (2)$$

To align these two sequences using DTW, we construct an n-by-m matrix where the (i^{th}, j^{th}) element of the matrix contains the distance $d(q_i,c_j)$ between the two points q_i and c_j (Typically the Euclidean distance is used, so $d(q_i,c_j) = (q_i - c_j)^2$). Each matrix element (i,j) corresponds to the alignment between the points q_i and c_j. This is illustrated in Figure 5. A warping path, W, is a contiguous (in the sense stated below) set of matrix elements that defines a mapping between Q and C. The k^{th} element of W is defined as $w_k = (i,j)_k$, so we have:

$$W = w_1, w_2, \ldots,w_k,\ldots,w_K \qquad \max(m,n) \le K < m+n-1 \quad (3)$$

The warping path is typically subjected to several constraints.

- **Boundary conditions:** $w_1 = (1,1)$ and $w_K = (m,n)$. Simply stated, this requires the warping path to start and finish in diagonally opposite corner cells of the matrix.
- **Continuity:** Given $w_k = (a,b)$ then $w_{k-1} = (a',b')$, where $a–a' \le 1$ and $b-b' \le 1$. This restricts the allowable steps in the warping path to adjacent cells (including diagonally adjacent cells).
- **Monotonicity:** Given $w_k = (a,b)$ then $w_{k-1} = (a',b')$, where $a–a' \ge 0$ and $b-b' \ge 0$. This forces the points in W to be monotonically spaced in time.

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

$$DTW(Q,C) = \min\left\{\frac{1}{K}\sqrt{\sum_{k=1}^{K} w_k}\right\} \quad (4)$$

The *K* in the denominator is used to compensate for the fact that warping paths may have different lengths. This path can be found very efficiently using dynamic programming to evaluate the following recurrence which defines the cumulative distance $\gamma(i,j)$ as the distance $d(i,j)$ found in the current cell and the minimum of the cumulative distances of the adjacent elements:

$$\gamma(i,j) = d(q_i,c_j) + \min\{ \gamma(i-1,j-1) , \gamma(i-1,j) , \gamma(i,j-1) \} \quad (5)$$

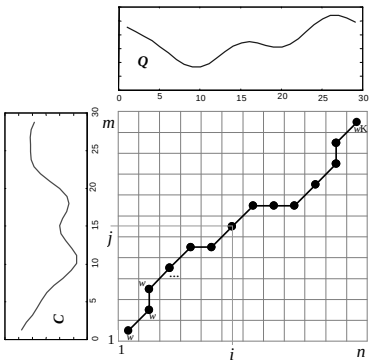

Figure 5: An example warping path.

The Euclidean distance between two sequences can be seen as a special case of DTW, where the k^{th} element of W is constrained such that $w_k = (i,j)_k$, i = j = k. Note that it is only defined in the special case where the two sequences have the same length. The time complexity of DTW is O(*nm*).

This review of DTW is necessarily brief; we refer the interested reader to [20] for a more detailed treatment.

2.3 Why Dynamic Time Warping is Resistant to Optimization

Because similarity measurement can be computationally expensive, we would like to optimize the calculations as much as possible. Euclidean distance (Equation 6), the most common distance measure, is amiable to several optimizations.

$$D(X,Y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \quad (6)$$

One obvious optimization is derived from the observation that the square root function is monotonic, thus we can remove it from the calculation and get identical rankings/classifications/clusterings. Once the square root step has been removed another optimization for similarly searching become apparent. The code to calculate the squared Euclidean distance can be modified to break out the loop if the partially accumulated distance exceeds the smallest distance encountered thus far.

```
accumulated_distance = 0;                                          (7)
for i in 1 .. length_of_time_series loop
   accumulated_distance = accumulated_distance + (qi – ci)2;
   if (accumulated_distance > best_so_far) then
      break;
   end if;
end for;
```

As soon as the accumulated distance exceeds the `best_so_far`, we can terminate and discard the candidate from further consideration. As the size of the database grows, this optimization reduces the effective complexity from linear to constant time. In contrast, DTW cannot be optimized using similar methods. The reason is that in order to establish the warping path, we must first calculate the distance between every point from the query sequence to every point in the candidate sequence. This alone guarantees that we are facing an $O(n^2)$ task. In addition, unlike the above method for Euclidean distance, DTW uses no accumulator. Instead it uses a stack as an implicit accumulator to store the entire set of distances before recursively retrieving them to determine the minimum warping path. It is not until $O(n^2)$ calls to the stack have been made that the first partial sum is available for consideration. This resistance of DTW to optimization suggests that we must look to approximation if we wish to speed up the distance calculations.

3 A New Approach: Iterative Deepening Dynamic Time Warping

In this section we introduce the IDDTW algorithm. Previous work allows us to approximate DTW at arbitrary levels of precision [17]. The idea behind the IDDTW algorithm is to obtain probabilistic models of the approximation errors for all levels of approximation prior to the querying process. The algorithm iteratively examines sequences at increasingly finer levels of approximation and compares the results to the probabilistic models and the user-specified tolerance for false dismissals. Using this information the algorithm iteratively considers whether the candidate sequence should either be dismissed as being unlikely to be a good match, or is worth considering at a yet finer approximation level. We begin by reviewing the technique that allows us to approximate DTW at arbitrary levels of precision.

3.1 Dimensionality Reduction

Keogh et al. [15] and Yi & Faloutsos [32] independently introduced a dimensionality reduction technique which approximates a time series by dividing it into equal-length segments and recording the mean value of the data points that fall within each segment. The authors use different names for this representation; for clarity we will refer to it as Piecewise Aggregate Approximation (PAA). Keogh & Pazzani [17] demonstrate a modification of DTW that operates on the PAA reduced dimensionality representation, which they called PDTW. They show that the resulting alignments are very similar to those produced by DTW. Figure 6 shows a typical example. PDTW has been shown to effectively generate a speedup of one to three orders of magnitude, compared to the classic DTW algorithm, with no significant loss of accuracy for classification and clustering tasks.

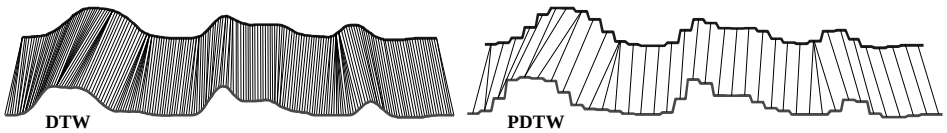

Figure 6: A) Two similar time series and the alignment between them, as discovered by DTW. B) The same time series in their PAA representation, and the alignment discovered by PDTW. This presents strong visual evidence that PDTW finds approximately the same warping as DTW.

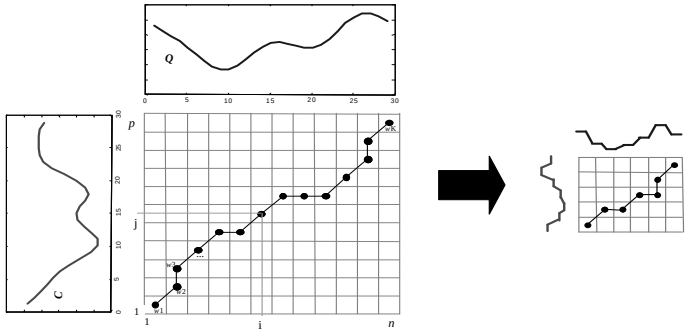

Figure 7: A) Example of warping path with DTW and B) PDTW.

We refer the reader to the original paper for full details of the PDTW algorithm. Here we just note that it can be thought of as the classic algorithm using the PAA coefficients as input. Figure 7 illustrates this idea.

A drawback of PDTW is that it requires the user to choose a compression rate for the dimensionality reduction, and the algorithm is very sensitive to the value chosen. A high compression rate means a coarser approximation, which leads to an increase in false dismissals. A lower compression rate signifies a finer approximation, but slower in computational time. The selection of the appropriate level of dimensionality reduction is dependent on the domain/dataset/query/task in question and requires careful fine-tuning to achieve the best result.

It is this drawback that motivates us to introduce IDDTW. With IDDTW, users only need to provide the system with a query and their tolerance for false dismissal. This option offers the user full control over the quality of their search result.

3.2 Iterative Deepening Algorithm Intuition

For notational simplicity we assume that the length n of the time series in question is a power of two. We can approximate a time series sequence using the PAA representation at any level of compression and use the PDTW algorithm to get an approximation to the true DTW distance. In general, the coarsest level approximation that is useful is to approximate the sequences with just 4 PAA coefficients (i.e. the original sequence is represented as four constant segments as in the top-right corner of Figure 8). We call this level of approximation d_1, (for "depth 1"). If we need to calculate a more accurate approximation we can double the resolution to 8 PAA coefficients, which we call d_2 etc. At $d_{\log2(n)-1}$ the "approximation" degenerates to the original data.

At any level of approximation we can use the PDTW algorithm to approximately calculate the true DTW distance *and* obtain an idea of the quality of the approximation. The idea is to obtain a distribution of approximation errors for the given level of approximation. This can be achieved by sampling the database, calculating the PDTW distance and the true DTW distance, and noting their difference.

During similarity search we can use the error distribution model to help prune off unpromising candidates without performing the full and expensive DTW calculation. The idea is that we perform the full DTW calculation on the first item in the database to

initialize a `best_so_far` variable. Thereafter, when presented with a candidate sequence to test, we first test it using PDTW at d_1, the coarsest resolution. The distance returned by PDTW is only an estimate but because we have an error distribution model for this estimate, we can calculate the probability that the new candidate sequence is actually a better match than the current `best_so_far`. We can compare this probability to the user-specified tolerance for false dismissal. There are only two possible outcomes to this test:

1) **The probability is out of tolerance**. In this case the sequence can be pruned, i.e., it is too unlikely to be a good match.
2) **The probability is within tolerance**. In this case we should compare the sequences at a higher resolution, so we compare them at d_2, d_3 , etc, each time testing to see if the sequence can be pruned, or if we must continue to the next level.

Eventually, as we compare the sequences at iteratively deeper levels of approximation, we will either prune away the candidate sequence, or be forced to compare them at depth $d_{\log2(n)-1}$. At that level there is no uncertainty about the calculated distance; if it is better than `best_so_far` we update it and continue to the next sequence.

The astute reader will note that, in the worst case, the iterative deepening could reach the maximum depth each time. Then it will be slower than the classic DTW algorithm. However as suggested below, the computation time for PDTW of the previous level is ¼ of that for the current one. Therefore, the amount of extra computation is at most 1/3 of the original cost for the original DTW.

$S = N + \frac{1}{4}N + \left(\frac{1}{4}\right)^2 N + \cdots + \left(\frac{1}{4}\right)^K N$ *S*=combined cost, *N*=cost for true DTW, *K*=max. depth

$S = N + \sum_{i=1}^{K} \left(\frac{1}{4}\right)^i N$

$S = \frac{4}{3}N$, as $K \to \infty$

This is a worst case bound. In practice, the algorithm rarely has to explore sequences all the way to depth $d_{\log2(n)-1}$ and dramatic speedup is observed.

3.3 Iterative Deepening Algorithm Details

We demonstrate our proposed method within the structure of a *K*-nearest neighbor algorithm. The algorithm begins by approximating the query *Q* and a candidate *C* using the PAA representation at the compression rate of W_{d1} to find the distance $D_{PDTW}(d_1)$. Using the error distribution, described in Table 2, for depth d_1 and the user confidence (or the acceptable tolerance for false dismissal), we determine whether *C* could be a potential object for expansion. If so, we compare the sequences at a higher resolution. We calculate *Q* and *C* with a more precise approximation using a finer compression rate of W_{d2} to determine D_{PDTW}. This process continues until *d* has reached the maximum depth. At that point, the "approximation" becomes the original data and we calculate the true warped distance D_{DTW} between *Q* and *C*.

The algorithm used to build the error distribution models of the approximated distance for all levels of approximation is described in Table 2. The inputs are the dataset **N** and the number of levels L. The output is a set of standard deviations **StdDev** from each of the error distribution models found.

```
Algorithm BuildErrorDistribution(N,L)
// N = dataset, L = number of depths
// d = {1,..,L}, C = {d range of compression rates}
for sample size i to build error distributions
  {Randomly pick 2 sequences of same lengths, N_a and N_b, where a != b}
  R_i := DTW(N_a, N_b);                    // True DTW of N_a and N_b
  for the range of d
    Approx_ N_a := PAA(N_a, C_d);          // Dimensionality reduced representation of N_a and N_b
    Approx_ N_b := PAA(N_b, C_d);
    RE_id := DTW(Approx_ N_a, Approx_ N_b);   // PDTW_d of N_a and N_b
    E_id :=RE_id – R_i;                       // Error between DTW and PDTW_d
    P_d :={P_d, E_id };                       // Accumulates each error at d in P_d
  end for;
  StdDev_d is found from P_d               // StdDev_d is the standard deviation for each P_d
end for;
```

Table 2: Algorithm to build the error distributions.

```
Algorithm IDDTW_K_NearestNeighbor(Q, N, UserConf, StdDev, K)
// N = dataset, Q = query sequence, UserConf = user confidence
// StdDev = standard deviations for each d, K = number of nearest neighbors
// d = {1,..,MAX_DEPTH}, C = {d range of compression rates}
R := {K*inf};
For i in size of N
C_better := true;
  While (c_better = true) and (d <= MAX_DEPTH)
    best_so_far  := argmax{R};                  // Use largest of R as reference for best_so_far
    if (best_so_far = inf) or (d = MAX_DEPTH)   // For first K R's, use full DTW
      D_DTW := DTW(Q, N_i);
      D := MAX_DEPTH;
    else
      Approx_ Q := PAA(Q, C_d);          // Dimension reduced representation of Q and N_i
      Approx_ N_i := PAA(N_i, C_d);
      D_DTW := DTW(Approx_ Q, Approx_ N_i);
      C_better := could_approx_be_better(D_DTW, best_so_far, UserConf, StdDev_d);
            // c_better := false, if estimated probability (using parameter StdDev_d) < UserConf
            // c_better := true, otherwise
    end if;
    d := d +1;
  end while;
  if c_better = true
    {updates R with D_DTW}      // replaces argmax{R} with current D_DTW, if D_DTW < argmax{R}
  end if;
end for;
```

Table 3: The IDDTW K-nearest neighbor algorithm

The IDDTW *K*-nearest neighbor algorithm used to query the dataset is described in Table 3. The inputs are the query *Q*, the dataset **N,** the user confidence *user_conf* (or tolerance for false dismissals), and the set of standard deviations **StdDev** obtained from `BuildErrorDistribution` in Table 2. The output is the *K* nearest neighbor matches. The algorithm for IDDTW begins by using the classic DTW on the first *K* candidates from the dataset. The results of the best matches to the query are contained in **R**, with |**R**|=*K*. The `best_so_far` is determined from `argmax`{**R**}. We want to use `argmax`{**R**} as an object for comparison so that we could always update the worst of the K matches each time we find a better result. ***W*** is the possible compression rates, from coarse to fine. Both *Q* and each subsequent candidate *C* after the first *K* candidates are approximated using PAA representations with W_d to determine the corresponding D_{PDTW}.

From this result, we can perform a test to determine whether *C* can be pruned off or to continue to the next level. If {result of the test is} found to have a probability that it could be a better match than the current `best_so_far`, a higher resolution of the approximation is required. The process of approximating *Q* and *C* to determine the D_{PDTW} and re-applying the test is repeated for all level of approximations until they fail the test or their true distance D_{DTW} is determined. Then we update **R** if D_{DTW} is found to be less than `best_so_far`. We repeat this process for each query-candidate pair until all the candidates in **N** are examined.

To determine whether a candidate is worth expanding, we perform a test, `could_approx_be_better`, on the approximated *Q* and *C* and comparing the result with the `best_so_far`. The `best_so_far` contains the true distance of the current "best match" so far. A worked-out example of this process is depicted in Figure 8. At depth 1 of the approximation, we find the estimated distance D_{PDTW} between query *Q* and candidate *C* to be 40. The `best_so_far` was found to be 30. The difference between the estimated distance and the `best_so_far` is 10. This is considered to be the estimated distance error. At that point we can view our error distribution as being centered around the approximated distance. By knowing that the `best_so_far` is 10 units away, we can determine the probability that the candidate could be better by examining the area beyond the location of the `best_so_far` (shown in solid black in Figure 8). We disqualify a candidate if the probability found is less than the user-specified acceptance for error. However, if determined to be a potential candidate, a finer approximation will be used and the test is re-applied to the next depth. This process continues until the full DTW is performed on the actual *C* and *Q*.

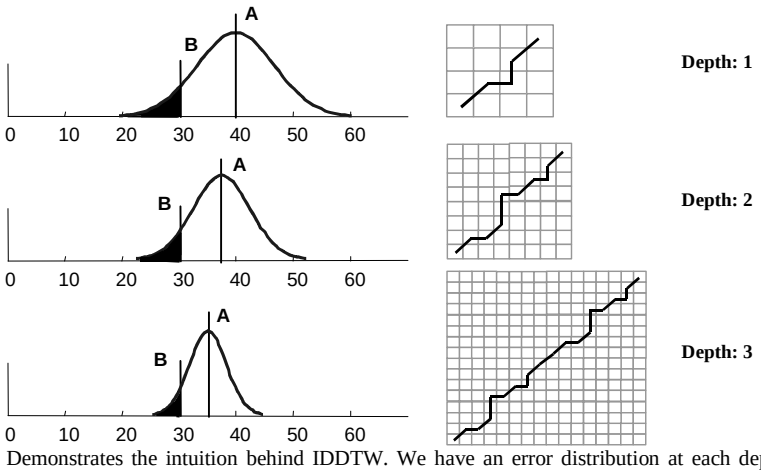

Figure 8: Demonstrates the intuition behind IDDTW. We have an error distribution at each depth. **A** is the approximated distance and **B** is the best_so_far. The mean of the distribution is at 0. If we center the distribution around **A**, then the distance between **A** and **B** is the approximated distance error. The shaded area can be seen as the probability that **A** could be better than the best_so_far. This also demonstrates the complexity of finding a warping path at each level of approximation.

A more detailed explanation might be in order. In addition to using PDTW values, (approximations of the true DTW distances) we compute a random sample of errors between PDTW and DTW values. We use the sample of errors to determine when we should keep a candidate even though we would discard the candidate if the PDTW computation were known to be exact. The possible error in PDTW value, which we estimate based on the random sample, causes us to keep a candidate.

If we were to take every pair (i,j) of time series in the database and compute both the DTW and PDTW for the pair (we will assume some fixed compression level for PDTW in this discussion), we would get a set of errors $e_{ij} = D_{DTW} - D_{PDTW}$ between the exact DTW and approximate PDTW values. Since we want to keep computation time small, we compute only a random sample of the e_{ij}, Specifically, we choose two time series uniformly at random from the database and we compute e_{ij}, repeating to obtain a set of samples. We do the sample computation (for each compression level) in just one initial step.

We apply the sample set as follows. We compute a PDTW distance $D_{PDTW}(C,Q)$ for a candidate relative to a query object, and we have some current threshold distance $D_{DTW}(C_{best},Q)$ {`best_so_far`} (typically the distance between a "good" candidate and the query item) that tells us how close the current candidate has to be for us to continue looking at it. Suppose $D_{PDTW}(C,Q) >$ `best_so_far`, but we also have an error e_C for the candidate, and it holds that e_C > $|D_{PDTW}(C,Q)$ - `best_so_far`|. In this case we do not want to discard the candidate, as it may be within our threshold distance to the query item. To make this work, we must have a way to estimate error e_C.

We set a decision error value to be $e_{decision} = |D_{PDTW}(C,Q)$ - `best_so_far`|. We then look at our precomputed sample set and count the number of error distances that satisfy $e_{ij} > e_{decision}$. The ratio r of this number to the total size of the sample set gives us a probability that we risk false dismissal.[1] We choose to discard a candidate only if the probability $p(e_C > e_{decision})$, which we estimate with MLE r, is less than our tolerance T for false dismissals.

4 Experimental Evaluation

In this section we conduct a number of experiments to compare IDDTW to classic DTW and Euclidean distance. We are interested in the speedup obtained over the classic DTW algorithm and the precision of the answer set. In general, techniques for similarity search tend to be either I/O bound or CPU bound, depending on the distance measure used. Since DTW is heavily CPU bound, we report only the CPU costs.

4.1 Results and Analysis

To demonstrate the effectiveness of IDDTW, we tested our algorithm on two datasets with varying properties. One is a homogeneous synthetic data set generated by the random walk expression, $x_t = x_{t-1} + z_t$, where z_t (t=1,2,..) are independent, identically distributed (uniformly) random variables. The other is a heterogeneous data set that is a concatenation of ten diverse datasets. The ten datasets were chosen to avoid any biased results and to represent the extremes along the following dimensions, stationary/non-stationary, noisy/smooth, cyclical/non-cyclical, symmetric/asymmetric, etc. In addition, the data sets represent the diverse areas in which data miners apply their algorithms including finance, medicine, manufacturing and science. Figure 9 illustrates the two

[1] The true probability θ that error e_C is greater than $e_{decision}$ is this same ratio computed for all pairs of items in the database. Since we have a random sample of the errors, our model is a Bernoulli trial for the event that e_{ij} is greater than $e_{decision}$. The ratio r is (as discussed in introductory statistics texts) a maximum likelihood estimate (MLE) of the true parameter θ. (We note that for a truly rigorous treatment we should look at confidence intervals for r.)

datasets used in the experiments. All the data, including the query sequences, are normalized to mean of zero and a standard deviation of one.

For each experiment, we created a database with 256 sequences. For the heterogeneous data, we randomly extracted sequences of length 512 and placed them in the database. For the random walk dataset, we randomly generated sequences, also of length 512. The query sequences are generated as follows: For the random walk data, we use the same generator and parameters to generate the queries. For the heterogeneous data, we randomly extracted sequences, rejecting sequences only if they already appeared in the database (this would make our results optimistic).

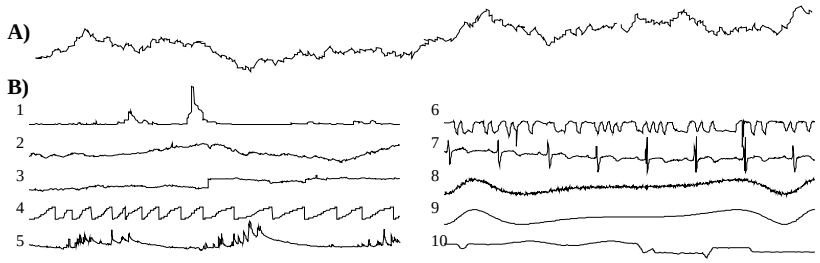

Figure 9: A) The random walk synthetic dataset used in the experiments. B) The heterogeneous datasets: 1. Radio Waves. 2. Exchange Rates. 3. Tickwise II. 4. Tickwise I. 5. Water Level. 6. Manufacturing. 7. ECG. 8. Noisy Sine Cubed. 9. Sine Cube. 10. Space Shuttle.

We compare three distance measures:

1) **DTW**: The classic dynamic time warping algorithm.
2) **IDDTW**: The iterative deepening dynamic time warping algorithm proposed in this work.
3) **Euclidean distance**: We used Euclidean distance as part of our experiment because it is the simplest distance measurement and a standard strawman commonly used in the literature.

The hardware used for all experiments was a Pentium III 500MHz with 256MB RAM. We used Matlab to implement all algorithms. Since Matlab is an interpreted language, the overall time is slow, but we are only interested in the relative performances.
For each experiment we ran eight different 10-Nearest Neighbor queries, testing the IDDTW algorithm at various tolerance levels for false dismissals. We define the accuracy of IDDTW, at various levels of tolerance for false dismissals, as the fraction of the number of objects retrieved by DTW alone which are also retrieved by IDDTW. An analogous definition is also made for the accuracy of retrievals using the Euclidean distance.

The results are summarized in Figures 10 and 11. Euclidean distance does a reasonable job considering that it has no flexibility to warp, however it is clearly not accurate enough a proxy for DTW to warrant serious consideration. In contrast IDDTW exhibits high accuracy for all tolerance levels. As we expect, the lower the tolerance for false dismissal specified by the user, the greater the accuracy. To understand the relationship between tolerance and speedup, we also compared the CPU time for using the classic DTW and IDDTW at each tolerance level. The results are summarized in Figure 11.

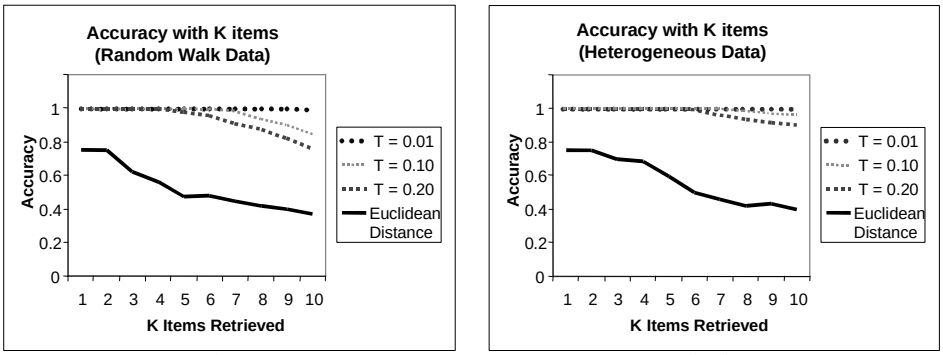

Figure 10: Comparison of Euclidean distance and IDDTW at various tolerances for false dismissals (T = 0.01, 0.10, 0.20) in terms of accuracy (to true DTW) for random walk data (left) and heterogeneous data (right).

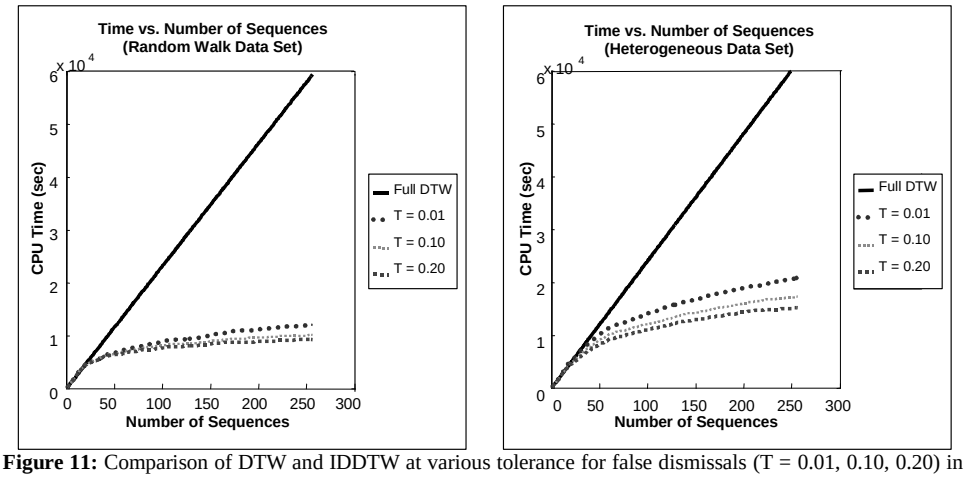

Figure 11: Comparison of DTW and IDDTW at various tolerance for false dismissals (T = 0.01, 0.10, 0.20) in terms of CPU cost (seconds) for random walk data set (left) and heterogeneous data set (right).

The time for DTW is linear in the number of objects in the database. In contrast, with IDDTW, the bound for filtering out poor matches becomes tighter as more sequences have been observed. Because of this property of IDDTW, the speedup over DTW is proportion to the size of the data. In other words, the larger the database of sequences, the greater the speedup obtained by IDDTW.

5 Related Work

Yi et al. introduced a method to speed up DTW by combining two techniques, which can be used in a pipelined manner [33]. The first technique is using FastMap to index and filter out non-qualifying sequences (FastMap is a dimensionally reduction technique that takes objects in a database and maps them onto a k-dimensional space, where k is the reduced dimension [10]). The objects are filtered out by comparing them in terms of their k-dimensional Euclidean distances. This technique results in possible false-dismissals and requires the parameter k to be pre-determined, either by being specified by the user or tuned for performance. The second technique is a lower bounding distance function, which underestimate the warping distance. Objects that pass both tests are then subjected to the normal DTW algorithm. While the work was pioneering, the approach was only tested on relatively short queries with the speedup reported being only a small constant.

In addition, there are several parameters to be set, and it is not clear how best to choose them.

Park et al. introduced a time warping measure that uses a symbolic representation that is itself obtained from a piecewise linear representation [24]. They demonstrate that this approach can be indexed in a suffix tree. However they only report the speedup for the indexing scheme (which is relatively small), and there is no comparison in terms of accuracy to the true DTW. Thus we can say nothing of the quality of the answer set. Their technique also allows false dismissals. They pointed out in their paper that "*…it is possible that a subsequence similar to a query in terms of the original time warping distance may not be included in the answer set in our approach*". In fact, this disclaimer greatly understates the problem with their approach, which rarely finds a true best match to a query.

Perng et al proposed the Landmark Model for pattern querying in time series databases [25]. This model does not rely on Euclidean distance. Rather it identifies features of landmarks that are invariant under the several transformations. Instead of working directly with raw data, they proposed a method for data representation using landmarks from the original data, such as local extrema, inflection points, etc. Using increasing types of landmarks will produce a more accurate representation. But fewer landmarks result in a smaller index tree. The most suitable selection of landmarks and the number of them used are domain-driven by the data. Since landmarks are sequential, it can reduce an indexing problem into a string-indexing problem. While the work does allow a flexible query language, the feature-extracting step requires the careful choice of several parameters, and it is not clear if the query language can emulate true DTW, which has well-documented success rate in real-world applications [1, 5, 11, 14, 22, 23, 29, 30].

6 Conclusions and Future Work

In this paper we introduced a modification of DTW that exploits the idea of iterative deepening along with a dimensionality reduction technique to produce a dramatic speedup which increases with database size. Our algorithm has the desirable properties of containing true DTW as a special case (when T = 0), and not requiring the careful adjustment of system parameters. In addition our approach gives the end-user complete and explicit control over the quality/time tradeoff through a single intuitive parameter, their tolerance for the probability of a false dismissal.

Future work includes a more detailed analysis of our approach and extensions to other similarity search problems that feature distance measures that are expensive, but can be approximated at different levels of precision.

References

[1] Aach, J. and Church, G. (2001). Aligning gene expression time series with time warping algorithms. *Bioinformatics*. Volume 17, pp 495-508.

[2] Agrawal, R., Lin, K. I., Sawhney, H. S., & Shim, K. (1995). Fast similarity search in the presence of noise, scaling, and translation in times-series databases. *Proc. of 21st International Conference on Very Large Data Bases.*

[3] Agrawal, R., Mannila, H., Srikant, R., Toivonen, H., & Verkamo, A. I. (1996). Fast discovery of association rules. *Advances in Knowledge Discovery and Data Mining*, Chapter 12, AAAI/MIT Press, pp 307-328.

[4] Berndt, D. & Clifford, J. (1994). Using dynamic time warping to find patterns in time series. *AAAI-94 Workshop on Knowledge Discovery in Databases*.

[5] Caiani, E.G., Porta, A., Baselli, G., Turiel, M., Muzzupappa, S., Pieruzzi, F., Crema, C., Malliani, A. & Cerutti, S. (1998). Warped-average template technique to track on a cycle-by-cycle basis the cardiac filling phases on left ventricular volume. *IEEE Computers in Cardiology*. Volume 25 Cat.

[6] Chan, K.P., Fu, A, Yu, C. (2001). Haar Wavelets for Efficient Similarity Search of Time-series: With and Without Time Warping. *IEEE Transactions on Knowledge and Data Engineering*, to appear.

[7] Das, G., Lin, K., Mannila, H., Renganathan, G. & Smyth, P. (1998). Rule discovery form time series. *Proc. of the 4th International Conference of Knowledge Discovery and Data Mining*. pp 16-22.

[8] Debregeas, A. & Hebrail, G. (1998). Interactive interpretation of Kohonen maps applied to curves. *Proc. of the 4th International Conference of Knowledge Discovery and Data Mining*. pp 179-183.

[9] Diez, J. J. R. & Gonzalez, C. A. (2000). Applying Boosting to Similarity Literals for Time Series Classification. *Multiple Classifier Systems, First International Workshop.* pp 210-219.

[10] Faloutsos, C., Ranganathan, M., & Manolopoulos, Y. (1994). Fast subsequence matching in time-series databases. *Proc. ACM SIGMOD International Conference on Management of Data*.

[11] Gavrila, D. M. & Davis,L. S.(1995). Towards 3-d model-based tracking and recognition of human movement: a multi-view approach. *International Workshop on Automatic Face- and Gesture-Recognition*. IEEE Computer Society.

[12] Gollmer, K., & Posten, C. (1995). Detection of distorted pattern using dynamic time warping algorithm and application for supervision of bioprocesses. *IFAC Workshop on On-Line Fault Detection and Supervision in Chemical Process Industries.*

[13] Kadous, M. W. (1999). Learning comprehensible descriptions of multivariate time series. *Proc. of the 16th International Machine Learning Conference*. Morgan Kaufmann. pp 454-463.

[14] Kassidas, A., MacGregor, J. F., & Taylor, P. A. (1998). Synchronization of Batch Trajectories Using Dynamic Time Warping. *American Institute of Chemical Engineers*. Volume 44, pp 864-874.

[15] Keogh, E., Chakrabarti, K., Pazzani, M., & Mehrotra, S. (2001). Dimensionality Reduction for Fast Similarity Search in Large Time Series Databases. *Knowledge and Information Systems*, Volume 3, Issue 3, pp 263-286.

[16] Keogh, E., & Pazzani, M. (1998). An enhanced representation of time series which allows fast and accurate classification, clustering and relevance feedback. *Proc. of the 4th International Conference of Knowledge Discovery and Data Mining*. pp 239-241, AAAI Press.

[17] Keogh, E. & Pazzani, M. (2000). Scaling up Dynamic Time Warping for Datamining Applications. *Proc. of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.*

[18] Keogh, E., Smyth, P. (1997). A probabilistic approach to fast pattern matching in time series databases. *Proc. of the 3rd International Conference of Knowledge Discovery and Data Mining*. pp 24-20, AAAI Press.

[19] Kovacs-Vajna, Z.M. (2000). A Fingerprint Verification System Based on Triangular Matching and Dynamic Time Warping. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Volume 22, Number 11, November 2000, pp 1266-1276.

[20] Kruskall, J. B. & Liberman, M. (1983). The symmetric time warping algorithm: From continuous to discrete. *Time Warps, String Edits and Macromolecules.* Addison-Wesley.

[21] Manganaris, S. (1997). Supervised classification with temporal data. PhD thesis, Computer Science Department, School of Engineering, Vanderbilt University.

[22] Munich, M. E. & Perona, P. (1999). Continuous Dynamic Time Warping for translation-invariant curve alignment with applications to signature verification. *Proc. of the 8th IEEE International Conference on Computer Vision.*

[23] Oates, T., Schmill, M. D., and Cohen, P. R. (2000). A Method for Clustering the Experiences of a Mobile Robot that Accords with Human Judgments. *Proc. of the 17th National Conference on Artificial Intelligence*, pp 846-851.

[24] Park, S., Chu, W. W., Yoon, J., & Hsu, C. (2000). Efficient Searches for Similar Subsequences of Different Lengths in Sequence Databases. *Proc. of the IEEE 16th International Conference on Data Engineering*.

[25] Perng, C., Wang, H., Zhang, S., & Parker, S. (2000). Landmarks: a new model for similarity-based pattern querying in time series databases. *Proc. 16th International Conference on Data Engineering*.

[26] Rabiner, L. & Juang, B. (1993). Fundamentals of speech recognition. Englewood Cliffs, N.J, Prentice Hall.

[27] Rafiei, D. and Mendelzon, A.O. (1997). Similarity-Based Queries for Time Series Data. *Proc. of the ACM SIGMOD international conference on Management of data*.

[28] Saito, N. (1994). Local feature extraction and its application using a library of bases. PhD thesis, Yale University.

[29] Sakoe, H. & Chiba, S. (1978) Dynamic programming algorithm optimization for spoken word recognition. *IEEE Trans. Acoustics, Speech, and Signal Processing*., Volume 26, pp 143-165.

[30] Vullings, H.J.L.M., Verhaegen , M.H.G., & Verbruggen, H.B. (1997). ECG Segmentation Using Time-Warping. A*dvances in Intelligent Data Analysis*. pp 275-285.

[31] Wijk, J.J. van & E. van Selow (1999). Cluster and Calendar-based Visualization of Time Series Data. In: G. Wills, D. Keim (eds.), *Proc. IEEE Symposium on Information Visualization (InfoVis'99), IEEE Computer Society*, pp 4-9.

[32] Yi, B, K., & Faloutsos, C.(2000). Fast time sequence indexing for arbitrary Lp norms. *Proc. of the 26th International Conference on Very Large Databases.*

[33] Yi, B. K., Jagadish, H. V., Faloutsos, C. (1998). Efficient retrieval of similar time sequences under time warping. *Proc. of the 14th International Conference on Data Engineering*. pp. 201-208.