Intelligent Data Analysis 11 (2007) 561–580
IOS Press

Toward accurate dynamic time warping in linear time and space

Stan Salvador[a,*] and Philip Chan[b]
[a]*General Dynamics, Armament and Technical Products, 4205 Westinghouse Commons Dr., Charlotte, NC 28269, USA*
[b]*Department of Computer Sciences, Florida Institute of Technology, 150 W University Blvd., Melbourne, FL 32901, USA*
E-mail: pkc@cs.fit.edu

Received 22 October 2006
Revised 10 March 2007
Accepted 2 June 2007

Abstract. Dynamic Time Warping (DTW) has a quadratic time and space complexity that limits its use to small time series. In this paper we introduce FastDTW, an approximation of DTW that has a linear time and space complexity. FastDTW uses a multilevel approach that recursively projects a solution from a coarser resolution and refines the projected solution. We prove the linear time and space complexity of FastDTW both theoretically and empirically. We also analyze the accuracy of FastDTW by comparing it to two other types of existing approximate DTW algorithms: constraints (such as Sakoe-Chiba Bands) and abstraction. Our results show a large improvement in accuracy over existing methods.

Keywords: Dynamic time warping, time series, time series alignment, time series similarity

1. Introduction

Dynamic time warping (DTW) is a technique that finds an optimal alignment between two time series in which one time series may be “warped” non-linearly by stretching or shrinking its time axis. This alignment can be used to find corresponding regions or to determine the similarity between the two time series. DTW is frequently used in speech recognition to determine if two waveforms represent the same spoken phrase. In a speech waveform, the duration of each spoken sound and the interval between sounds are permitted to vary, but the overall speech waveforms must be similar. DTW is also used in many other disciplines [11], including data mining, gesture recognition, robotics, manufacturing, and medicine. An example of one time series “warped” to another is shown in Fig. 1.

In Fig. 1, each vertical line connects a point in one time series to its correspondingly similar point in the other time series. The lines have similar y-axis values, but have been separated so the vertical lines between them can be viewed more easily. If the two time series in Fig. 1 were identical, all of the lines would be straight vertical lines because no warping would be necessary to ‘line up’ the two time series.

[*]Corresponding author. 3639 Robin Ln., Charlotte, NC 28269, USA. Tel.: +1 704 599 2993; Fax: +1 980 235 2396; E-mail: ssalvador@gdatp.com.

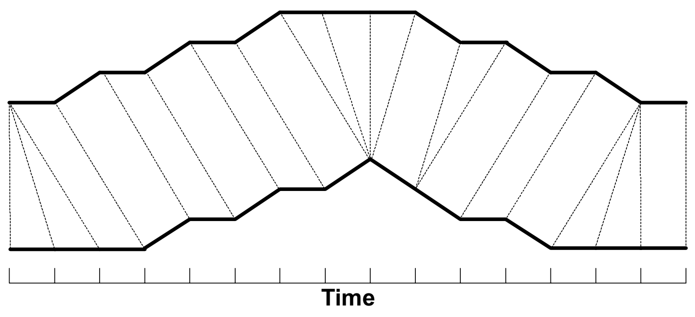

Fig. 1. A warping between two time series.

The warp path distance is the sum of the distances between the corresponding points. Thus, two time series that are identical except for localized stretching of the time axis (as in Fig. 1) will have warp path distances of zero. The DTW algorithm finds the optimal alignment between two time series, but has an O(N^2) time and space complexity that limits its usefulness to small time series.

We aim to develop a dynamic time warping algorithm that has a linear time and space complexity, and can find a warp path between two time series that is nearly optimal. In this paper we introduce FastDTW, which is able to efficiently find an accurate warp path between two time series that is close to an optimal solution. The FastDTW algorithm avoids the brute-force dynamic programming approach of DTW by using a multilevel approach. In our approach, the time series are initially sampled down to a very low resolution. A warp path is found for this lowest resolution and *projected* onto an incrementally higher resolution. The projected warp path is *refined* and projected again to a higher resolution. The process of projecting and refining is repeated until a warp path is found for the full resolution time series.

Our main contribution is the introduction of the FastDTW algorithm, an accurate approximation of DTW that runs in linear time and space. We prove the O(N) time and space complexity both theoretically and empirically. We also demonstrate that FastDTW produces an accurate warp path between two time series that is nearly optimal (classic DTW is optimal, but has a quadratic time and space complexity). We also evaluate other existing approximate DTW algorithms on a large and diverse set of time series data.

The next section describes the classic dynamic time warping algorithm and existing approaches to speed it up. Section 3 provides a detailed explanation of our FastDTW algorithm. Section 4 discusses experimental evaluations of FastDTW based on accuracy, and time/space complexity, and Section 5 summarizes our study.

2. Related work

2.1. *Dynamic Time Warping* (*DTW*)

A distance measurement is required to determine the similarity between time series and for time series classification. Euclidean distance is an efficient distance measurement that can be used. The Euclidian distance between two equal length time series is simply the sum of the distances from each nth point in one time series to the nth point in the other. Euclidean distance is appropriate for some domains [11], but its results are often unintuitive [12]. If two time series are identical, but one is shifted slightly along the time axis, then Euclidean distance will consider them to be very different. Dynamic time warping (DTW) was introduced [15] to overcome this limitation and give intuitive distance measurements between time series by ignoring both global and local shifting of the time dimension.

The time warping problem is stated as follows: Given two time series X and Y, of lengths $|X|$ and $|Y|$,

$$X = x_1, x_2, \dots, x_i, \dots, x_{|X|}$$
$$Y = y_1, y_2, \dots, y_j, \dots, y_{|Y|}$$

construct a warp path W,

$$W = w_1, w_2, \dots, w_K \quad \max(|X|, |Y|) \leqslant K < |X| + |Y|$$

where K is the length of the warp path, and the k^{th} element of the warp path is

$$w_k = (i, j)$$

where i is an index of time series X, and j is an index of time series Y. The warp path starts at the beginning of each time series at w_1 =(1, 1) and finishes at the end of both time series at w_K = ($|X|$, $|Y|$). There is also a constraint on the warp path that forces i and j to be monotonically increasing in the warp path, which is why the lines representing the warp path in Fig. 1 do not overlap. Every index of both time series must be used. Stated more formally:

$$w_k = (i, j),\ w_{k+1} = (i', j') \quad i \leqslant i' \leqslant i + 1,\ j \leqslant j' \leqslant j + 1$$

An optimal warp path is a minimum distance warp path, where the distance (or cost) of a warp path W is

$$\textit{Dist}(W) = \sum_{k=1}^{k=K} \textit{Dist}(w_{ki}, w_{kj})$$

Dist(w_{ki}, w_{kj}) is the distance between the two data point indexes (one from X and one from Y) in the k^{th} element of the warp path.

Dynamic programming is used to find this minimum-distance warp path between two time series. Rather than solving the entire problem all at once, solutions to sub-problems (portions of the time series) are found, and used to build solutions to larger sub-problems until a solution is found for the entire time series. A two-dimensional $|X|$ by $|Y|$ cost matrix D, is created where the value at $D(i,\ j)$ is the minimum distance of a warp path for the two time series X' = $x_1, \dots, x_i$ and Y' = $y_1, \dots, y_j$. $D(|X|$, $|Y|)$ contains the minimum distance of a warp path between time series X and Y. Both axes of D represent time. The x-axis is the time of series X, and the y-axis is the time of series Y. Figure 2 shows an example of a cost matrix and a minimum distance warp path traced through it from D(1, 1) to $D(|X|$, $|Y|)$.

The time series and warp path in Fig. 2 are the same as in Fig. 1. The warp path is W = {(1, 1), (2, 1), …, (15, 15), (16, 16)}. If the warp path passes through cell $D(i,\ j)$ of the cost matrix, then the i^{th} point in time series X is warped to the j^{th} point in time series Y. If X and Y were identical, the warp path would be a straight diagonal line (a linear warp path). Single points in one time series can map to several points in the other (vertical and horizontal sections of the warp path). Since a single point may map to multiple points in the other time series, the time series do not need to be of equal length.

To find a minimum distance warp path, every cell in the cost matrix must be filled. The rationale behind using dynamic programming is that if the solutions are already known for all slightly smaller

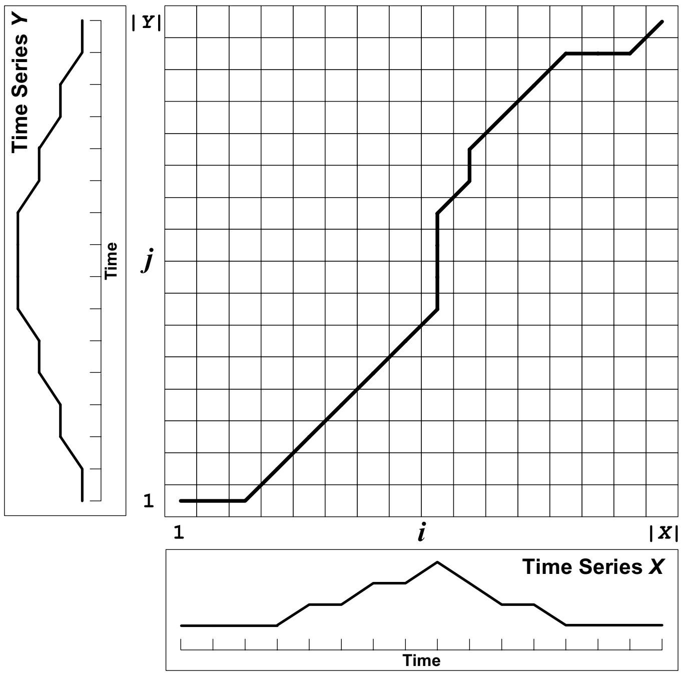

Fig. 2. A cost matrix with a warp path traced through it.

portions of that time series that are a single data point away from lengths i and j, then the value at $D(i, j)$ is the minimum distance for all these smaller time series, plus the distance between the points i_i and j_j. Since the warp path must either increase by one or stay the same along the i and j axes, the distances of the optimal warp paths one data point smaller than lengths i and j are contained in the matrix at $D(i-1, j)$, $D(i, j-1)$, and $D(i-1, j-1)$. So the value of a cell in the cost matrix is

$$D(i, j) = \textit{Dist}(i, j) + \min[D(i-1, j), D(i, j-1), D(i-1, j-1)]$$

The warp path to $D(i, j)$ must pass through one of those three cells, and since the minimum warp path distance is already known for them, all that is needed is to add the distance between the current pair of points, *Dist*(i, j), to the smallest value in those three cells. The cost matrix is filled one column at a time from the bottom up, from left to right. After the entire matrix is filled, a warp path must be found from D(1, 1) to $D(|X|, |Y|)$. The warp path is calculated backwards, starting at $D(|X|, |Y|)$. A greedy search evaluates three nearby cells: to the left, below, and diagonally to the bottom-left. Whichever of these three cells has the smallest value is then added to the beginning of the warp path, and the search continues from that cell until D(1, 1) is reached.

The time and Space complexity of the dynamic time warping algorithm is straightforward to determine. Each cell in the $|X|$ by $|Y|$ cost matrix is filled exactly once, and each cell is filled in constant time. This yields both a time and space complexity of $|X|$*$|Y|$, which is O(N^2) if $N = |X| = |Y|$. The quadratic space complexity is particularly prohibitive because memory requirements are in the *terabyte* range for time series containing only 177,000 measurements. A linear space-complexity implementation of the classic DTW algorithm is possible by only keeping the current and previous columns in memory as the cost matrix is filled from left to right. However, this implementation can only calculate the warp distance, and cannot reconstruct the warp path because the information required to do so is thrown away with the

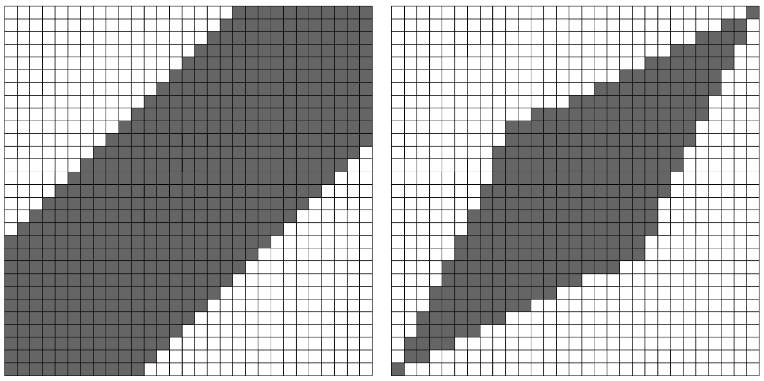
Fig. 3. Two constraints: Sakoe-Chiba Band (left) and Itakura Parallelogram (right), both have a width of 11.

discarded columns. This is not a problem if only the distance between two time series is required, but applications that find corresponding regions between time series [21] or merge time series together [1] require the warp path to be found.

2.2. *Speeding up dynamic time warping*

The quadratic time and space complexity of DTW creates the need for more efficient alternatives. The approaches used make DTW faster fall into three categories:

1) *Constraints* – Limit the number of cells that are evaluated in the cost matrix.
2) *Abstraction* – Perform DTW on a reduced representation of the data.
3) *Indexing* – Use lower bounding functions to reduce the number of times DTW must be run during time series classification or clustering.

Constraints are widely used to speed up DTW. Two of the most commonly used constraints are the Sakoe-Chiba Band [18] and the Itakura Parallelogram [7], which are shown in Fig. 3.

The shaded areas in Fig. 3 are the cells of the cost matrix that are filled in by the DTW algorithm for each constraint. The width of each shaded area, or window, is specified by a parameter. When constraints are used, DTW finds an optimal warp path through the constraint window. However, the globally optimal warp path will not be found if it is not entirely inside the window. Using constraints speeds up DTW by a constant factor, but DTW is still O(N^2) if the size of the window is a function of time series length.

Constraints work well in domains where time series have only a small variance in their temporal alignment, and an optimal warp path is expected to be close to a linear warp and passes through the cost matrix diagonally in a relatively straight line. In fact, for many applications, using constraints leads to better classification accuracy than unconstrained DTW [17,18]. However, constraints work poorly if time series are of events that start and stop at radically different times. In this scenario, the warp path can stray very far from a linear warp and nearly the entire cost matrix must be evaluated to find an optimal warp path. Figure 4 depicts a simplified example of a situation where constraining DTW does not work well, and the entire cost matrix must be evaluated for good results. This can occur in applications where the time series are of monitored devices that are issued commands (such as on/off) in a predictable order, but with an unspecified amount of time (steady-state conditions) between commands. Examples of such data include space shuttle valve signatures [4]. Our previous work [21] uses DTW

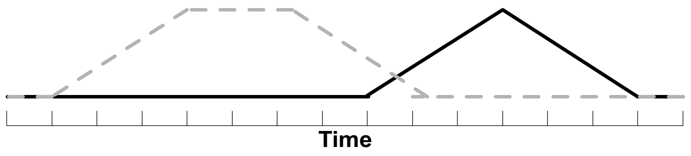

Fig. 4. Two time series significantly out of phase.

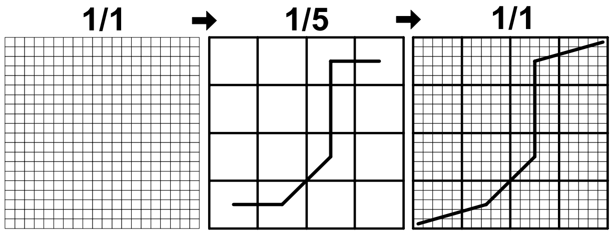

Fig. 5. Speeding up DTW by abstraction.

to locate corresponding states between these time series and to merge the time series into an "average" time series. The valve time series contain as many as 20,000 data points, which makes the calculation of the warp path impossible with the standard DTW algorithm. FastDTW was developed to warp these long time series.

Abstraction speeds up the DTW algorithm by operating on a reduced representation of the data. These algorithms include IDDTW [3], PDTW [13], and COW [2]. The left side of Fig. 5 shows a full-resolution cost matrix for which a minimum-distance warp path must be found. Rather than running DTW on the full resolution (1/1) cost matrix, the time series are reduced in size to make the number of cells in the cost matrix more manageable. A warp path is found for the lower-resolution time series and is mapped back to full resolution.

The resulting speedup depends on how much abstraction is used. Obviously, the calculated warp path becomes increasingly inaccurate as the level of abstraction increases. Projecting the low resolution warp path to the full resolution usually creates a warp path that is far from optimal. This is because even *IF* an optimal warp path passes through the low-resolution cell, projecting the warp path to the higher resolution ignores local variations in the warp path that can be very significant.

Indexing [9,14] uses lower-bounding functions to prune the number of times DTW is run for similarity search [17]. Indexing speeds up applications in which DTW is used, but it does not make up the actual DTW calculation any more efficient.

Our FastDTW algorithm uses ideas from both the constraints and abstraction approaches. Using a combination of both overcomes many limitations of using either method individually, and yields an accurate algorithm that is O(N) in both time and space complexity. Our multi-level approach is superficially similar to IDDTW [3] because they both evaluate several different resolutions. However, IDDTW simply executes PDTW [13] at increasingly higher resolutions until a desired "accuracy" is achieved. *IDDTW does not project low resolution solutions to higher resolutions*. In Section 4, we will demonstrate that these methods are more inaccurate than our method given the same amount of execution time. Projecting warp paths to higher resolutions is also done in the construction of "Match Webs" [15]. However, their approach is still O(N^2) due to the simultaneous search for many warp paths (they call them "chains"). A multi-resolution approach in their application also could not continue down to the low resolutions without severely reducing the number of "chains" that could be found.

Some recent research [18] asserts that there is no need to speed up the original DTW algorithm. However, this is only true under the following (common) conditions:

1) *Tight Constraints* – A relatively strict near-linear warp path is allowable.
2) *Short Time Series* – All time series are short enough for the DTW algorithm to execute quickly. ($\sim$3,000 points if a warp path is needed, or $\sim$100,000 if no warp path is needed and the user has a *lot* of patience).

Under these conditions, current research has contributed DTW methods that have an amortized linear time complexity for time series similarity search. This is possible only when performing similarity searches on many time series when in which DTW evaluations can be pruned from the search. However, all runs of DTW cannot be pruned, and running DTW between even a single pair of time series becomes prohibitive if both of the following conditions are true:

1) *No/Loose Constraints* – The warp path through the DTW cost matrix cannot be tightly constrained.
2) *Long Time Series* – DTW is being used with long time series (more than a few thousand points).

The NASA valve data described earlier [4] can contain 20,000 data points for 2 seconds of monitoring. This data set is an example of a data set that meets both of these conditions and requires a more efficient algorithm to perform dynamic time warping. Other researchers [2,3,13] have also tried to speed up DTW in previous research.

3. Approach

The multilevel approach that FastDTW uses is inspired by a multilevel graph bisection algorithm [8]. Graph bisection is the task of splitting a graph into roughly equal portions, such that the sum of the edges that would be broken is as small as possible. Efficient and accurate algorithms exist for small graphs, but for large graphs, the solutions found are typically far from optimal. A multilevel approach can be used to find an optimal solution for a small graph, and then repeatedly expand the graph and “fix” the pre-existing solution for the slightly larger problem. A multilevel approach works well if a large problem is difficult to solve all at once, but partial solutions can effectively be refined at different levels of resolution. The dynamic time warping problem can also be solved with a multilevel approach. Our FastDTW algorithm uses the multilevel approach and is able to find an accurate warp path in linear time and space.

3.1. FastDTW algorithm

FastDTW is a linear approximate DTW algorithm that uses a multilevel approach with three key operations:

1) *Coarsening* – Shrink a time series into a smaller time series that represents the same curve as accurately as possible with fewer data points.
2) *Projection* – Find a minimum-distance warp path at a lower resolution, and use it as an initial guess for a higher resolution’s minimum-distance warp path.
3) *Refinement* – Refine the warp path projected from a lower resolution by locally adjusting the warp path.

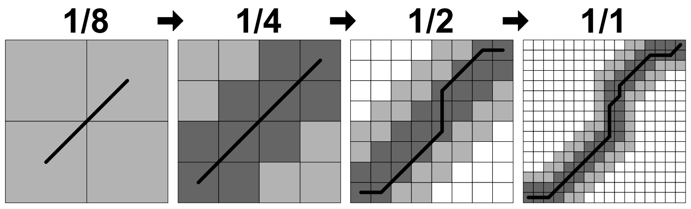

Fig. 6. The four different resolutions evaluated during a complete run of the FastDTW algorithm.

Coarsening reduces the length (or resolution) of a time series by averaging adjacent pairs of points. The resulting time series is a factor of two smaller than the original time series. Coarsening is run many times to produce different resolutions of the time series. *Projection* takes a warp path calculated at a lower resolution and determines what cells it passes through in a higher resolution. Since the resolution is increasing by a factor of two, a single point in the low-resolution warp path will map to at least four points at the higher resolution (possibly > 4 if $|X| \neq |Y|$). This projected path is then used as a heuristic during refinement to find a warp path at the higher resolution. *Refinement* finds an optimal warp path *in the neighborhood* of the projected path, where the size of the neighborhood is controlled by the *radius* parameter.

DTW is O(N^2) because every cell in the cost matrix must be filled to ensure an optimal answer is found, and the size of the matrix grows quadratically with the size of the time series. In our multilevel approach, the cost matrix is only filled in the neighborhood of the path projected from the previous resolution. Since the length of the warp path grows *linearly* with the length of the time series, our multilevel approach is an O(N) algorithm.

The FastDTW algorithm first uses coarsening to create all of the resolutions that will be evaluated. Figure 6 shows four resolutions that are created when running the FastDTW algorithm on the time series that were previously used in Figs 1 and 2. DTW is run to find an optimal warp path for the lowest resolution time series. This lowest resolution warp path is shown in the left of Fig. 6. After the warp path is found for the lowest resolution, it is projected to the next higher resolution. In Fig. 6, the projection of the warp path from a resolution of 1/8 is shown as the heavily shaded cells at 1/4 resolution.

To refine the projected path, a constrained DTW algorithm is run with the very specific constraint that only cells in the projected warp path are evaluated. This will find an optimal warp path *through the area of the warp path that was projected from the lower resolution*. However, the globally optimal warp path may not be entirely contained within the projected path. To increase the likelihood of finding a globally optimal solution, there is a *radius* parameter that controls the *additional* number of cells on each side of the projected path that will also be evaluated when refining the warp path. In Fig. 6, the *radius* parameter is set to 1. The cells included during warp path refinement due to the *radius* are lightly shaded. Once the warp path is refined at the 1/4 resolution, that warp path is projected to the 1/2 resolution, expanded by a *radius* of 1, and refined again. Finally, the warp path is projected to the full resolution (1/1) matrix in Fig. 6. The projection is expanded by the *radius* and refined one last time. This refined warp path is the output of the algorithm.

The warp path found by FastDTW in Fig. 6 is the same optimal warp path that was found by DTW in Fig. 2. FastDTW evaluated $4 + 16 + 44 + 100 = 164$ cells at all resolutions, while DTW evaluated 256 (16^2) cells. This increase in efficiency is not very significant for this small problem, especially considering the overhead of creating all four resolutions. However, the number of cells that FastDTW evaluates scales linearly with the length of the time series, while the classic dynamic time warping algorithm always evaluates N^2 cells (if both time series are of length N).

```
                    Function FastDTW
Input:   X   a TimeSeries of length |X|
         Y   a TimeSeries of length |Y|
         radius   distance to search outside of the projected
                  warp path from the previous resolution
                  when refining the warp path
Output:  1) A min. distance warp path between X and Y
         2) The warped path distance between X and Y

 1| // The min size of the coarsest resolution.
 2| Integer minTSsize = radius+2
 3|
 4| IF (|X|<=minTSsize OR |Y|<=minTSsize)
 4| {
 5|   // Base Case: for a very small time series run
 6|   //   the full DTW algorithm.
 7|   RETURN DTW(X, Y)
 8| }
 8| ELSE
 9| {
10|   // Recursive Case: Project the warp path from
11|   //   a coarser resolution onto the current
12|   //   current resolution.  Run DTW only along
13|   //   the projected path (and also  radius   cells
14|   //   from the  projected pat).
15|   TimeSeries shrunkX = reduceByHalf(X)
16|   TimeSeries shrunkY = reduceByHalf(Y)
17|
18|   SearchWindow window =
19|       ExpandedResWindow(
20|         FastDTW(shrunkX, shrunkY, radius),
21|         X, Y, radius)
22|
23|   RETURN DTW(X, Y, window)
24| }
```

Fig. 7. The FastDTW algorithm.

The example in Fig. 6 finds the optimal warp path. Although FastDTW is not guaranteed to always find a warp path that is optimal, the path found is usually very close to optimal. The larger the value of the *radius* parameter, the more accurate the warp path will be. If the *radius* parameter is set to be as large as one of the input time series, then FastDTW generalizes to DTW (optimal but O(N^2)). The accuracy of FastDTW using different settings for the *radius* parameter is demonstrated in Section 4.

The pseudocode for FastDTW is shown Fig. 7. The input is two time series and a *radius*. The output is a warp path and the distance between the two time series along that warp path. Line 2 determines the length of a time series at the lowest resolution, where decreasing the resolution further would be pointless because full DTW would need to be run at multiple resolutions.

FastDTW has a straightforward recursive implementation. In the base case, when one of the input time series has a length less than or equal to *minTSsize*, the algorithm simply returns the result of the standard DTW algorithm. The recursive case has three steps. First, half-resolution time series are created from the input time series. This is performed by lines 15–16 in Fig. 7. Next, the search window is determined by recursively calculating the warp path for the lower-resolution time series that were just created (line 20), expanding it by *radius* cells (line 20), and projecting that path to the current resolution (line 19).

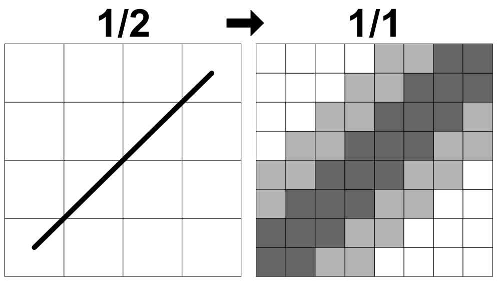

Fig. 8. Maximum (worst-case) number of cells evaluated for a *radius* (r) of 1.

Finally, the search window is used by a constrained version of DTW that only evaluates the cells in the search window. The result of this constrained DTW algorithm is returned at line 23.

Time Complexity of FastDTW. To simplify the calculations, we will assume that the two time series X and Y are both of length N. All analysis will be performed on worst-case behavior. The number of cells in the cost matrix that are filled by FastDTW in a single resolution is equal the number of cells in the projected warp path and *radius* (denoted as r in the rest of this analysis) cells away from the projected path. The worst case, a straight diagonal projected warp path is depicted in Fig. 8.

The lightly shaded cells in Fig. 8 are the 2*Nr* cells on each side of the projected path (heavily shaded cells), which itself has $3N$ cells. The projected path therefore has the following maximum number of cells at a resolution where both series contain N points:

$$3N + 2(2Nr) = N(4r + 3) \tag{1}$$

The length of the time series at each resolution (*res*) follows the sequence (N points are contained in the original time series):

$$\left\{\frac{N}{2^{res}}\right\}_{res=0}^{res=\infty} = N, \frac{N}{2}, \frac{N}{2^2}, \frac{N}{2^3}, \frac{N}{2^4}, \ldots \tag{2}$$

Therefore, the number of cells evaluated at all resolutions is (combine Eqs (1) and (2))

$$\sum_{res=0}^{\infty} \frac{N}{2^{res}}(4r + 3) = N(4r + 3) + \frac{N}{2}(4r + 3) + \frac{N}{2^2}(4r + 3) + \ldots \tag{3}$$

The series in Eq. (3) is very similar to the series

$$\sum_{x=0}^{\infty} \frac{1}{2^x} = 1 + \frac{1}{2} + \frac{1}{2^2} + \frac{1}{2^3} + \frac{1}{2^4} + \ldots = 2 \tag{4}$$

Multiplying Eq. (4) by Eq. (1) yields

$$N(4r + 3) + \frac{N}{2}(4r + 3) + \frac{N}{2^2}(4r + 3) + \ldots = 2N(4r + 3) \tag{5}$$

Since the sequence in Eq. (5) is identical to the sequence in Eq. (3), the number of cells evaluated at all resolutions is

$$\textit{Total number of cells filled} = 2N(4r + 3) \tag{6}$$

In addition to the number of cells calculated there is also time complexity for creating the coarser resolutions and determining the warp path by tracing through the matrix. The time complexity of creating the resolutions is proportional to the number of points in all of the resolutions, which is the series in Eq. (2). The solution of Eq. (2) is obtained by multiplying Eq. (4) by N, which yields $2N$. Multiple resolutions of both time series are created, so $2N$ is doubled to obtain the final time complexity.

$$\textit{Time to create all resolutions} = 4N \tag{7}$$

The time complexity needed to trace the warp path back through a matrix is measured by the length of the warp path. A resolution containing N points has a length of $2N$ in the worst case. Multiplying Eq. (4) by $2N$ gives the worst-case length of all warp paths added together from every resolution:

$$\textit{Time to trace warp paths} = 4N \tag{8}$$

Adding Eqs (6), (7), and (8) gives the total worst-cast time complexity of FastDTW

$$\textit{FastDTW time complexity} = N(8r + 14) \tag{9}$$

which is O(N) if r (*radius*) is a small constant value.

Space Complexity of FastDTW. The space complexity of FastDTW consists of the space required to store the resolutions (other than the full resolution input time series), the maximum amount of cells that are used at any one time in a cost matrix, and the size of the warp path stored in memory. The space complexity of storing all extra resolutions other than the full resolution for one input time series is Eq. (2) without the first term, which is $2N - N = N$. For both input time series the space complexity is

$$\textit{Space of resolutions} = 2N \tag{10}$$

The space complexity of the cost matrix is the maximum size cost matrix that is created for the full resolution matrix. The number of cells in the matrix is Eq. (1).

$$\textit{Space of cost matrix} = N(4r + 3) \tag{11}$$

The space complexity of storing the warp path is equal to the longest possible warp path at full resolution:

$$\textit{Space complexity of storing the warp path} = 2N \tag{12}$$

and adding Eqs (10), (11), and (12) gives the total worst-cast space complexity of

$$\textit{FastDTW space complexity} = N(4r + 7) \tag{13}$$

which is also O(N) if r (*radius*) is a small constant value.

4. Empirical evaluation

The goal of this evaluation is to demonstrate the efficiency and accuracy of FastDTW on a wide range of time series. To ensure reproducibility, all datasets and programs used in this evaluation can be found online at “http://cs.fit.edu/˜pkc/FastDTW/”.

4.1. Accuracy of FastDTW

4.1.1. Procedures and criteria

The accuracy of an approximate DTW algorithm can be measured by determining how much the approximate warp path distance differs from the optimal warp path distance:

$$\mathit{Error\ of\ a\ warp\ path} = \frac{\mathit{approxDist} - \mathit{optimalDis}\,t}{\mathit{optimalDis}\,t} \times 100 \tag{14}$$

If the approximate algorithm finds a warp path with a distance equal to the optimal warp path distance, then there is zero error. The optimal warp path distance can be found by running DTW. The error of a warp path will always be $\geqslant$0%, and will exceed 100% if the distance of the approximate warp path is more than twice the optimal distance. Our accuracy evaluation measures the deviation from the optimal warp-path found by DTW. It does not measure classification or clustering accuracy because these measures can be ambiguous and are open to interpretation. Our algorithm is intended to be a fast, accurate replacement for DTW in applications where DTW runs too slow. Improving clustering/classification accuracy is beyond the scope of this paper.

FastDTW is evaluated against Sakoe-Chiba Bands and Abstraction. Sakoe-Chiba Bands (see left side of Fig. 3) constrain DTW to only evaluate a specified *radius* away from a linear warp within the cost matrix. Itakura Parallelograms (see right side of Fig. 3) are not evaluated because a Band will always find a warp path equal to or better than that of a parallelogram for a given *radius*. The Abstraction algorithm used in this evaluation first samples the data, and then runs DTW to find a warp path on the sampled data similar to the Piecewise Dynamic Time Warping algorithm (PDTW) [13]. This warp path is then projected to the full resolution, as shown in Fig. 5.

The *radius* parameter performs a similar function for all three algorithms. It expands the evaluated region of the cost matrix from an initial "guess". For a Band, the initial guess is a linear warp. For Abstraction, it is the projected warp path from the sampled data, and for FastDTW it is the projected warp path from a lower resolution. Each algorithm is run with multiple *radius* parameters on a wide range of data sets. All three algorithms evaluate roughly the same number of cells in the cost matrix for a particular *radius*, and all three algorithms are linear with respect to the length of the input time series. The number of cells evaluated for a given *radius* does not differ by more than a factor of two between any pair of algorithms. The Abstraction algorithm is made O(N) by sampling the data down to $\sqrt{N}$ points before performing quadratic DTW (O($\sqrt{N}^2$) $=$ O(N)).

The data sets used to evaluate the accuracy of the FastDTW algorithm include very similar time series that are from the same domain, as well as dissimilar time series that are from different domains. Both types of data are used to show that FastDTW works well on a wide range of data, regardless of the similarity or characteristics of the time series. The accuracy of each algorithm is measured on three groups of data:

1) *Random* – 45 dissimilar time series from different domains (eeg, random walk, earthquake, speech, tide, etc.). The average length is 1128 points. All data sets in this group are very dissimilar to each other, which makes the error of the optimal warp paths calculated between these time series very large. Optimal warp paths between these time series are also most likely very far from a linear warp path which can be challenging for approximate DTW algorithms. Details and references to the papers in which each data set first appeared can be found at the UCR Time Series Data Mining Archive [10].

Table 1
Average error of the three algorithms at selected radius values (errors of the 3 groups of data are averaged)

	Radius				
	0	1	10	20	30
FastDTW	19.2%	8.6%	1.5%	0.8%	0.6%
Abstraction	983.3%	547.9%	6.5%	2.8%	1.8%
Band	2749.2%	2385.7%	794.1%	136.8%	9.3%

2) *Trace* – 200 similar time series data sets. The Trace domain contains real instrumentation data from a nuclear power plant with several types of simulated transient events inserted that simulate instrumentation failure. All time series have a length of 275 points. Time series in this group have a similar overall shapes to the time series with short-term transient spikes in randomly inserted into the data. This group of data is included to test the ability of FastDTW and existing approximate DTW algorithms from handling short-term discontinuities in the data that can make the construction of the entire minimum-cost warp path difficult. Additional details on this data can be found in [19].
3) *Gun* – 200 very similar time series data sets. All time series have a length of 150 points. The Gun domain contains 2 classes, a gun being drawn from a holster and a gun being pointed. The movements are represented by a time series of the hand's x-axis position over time. Additional details of this data can be found in [17].

All data sets used in this evaluation are publicly available [10]. Each algorithm and group of data is evaluated with various settings for the *radius* parameter. For a given algorithm, group of data, and *radius*, the average error of all possible warp paths between pairs of time series in the group of data is recorded.

4.1.2. Results and analysis

FastDTW is very accurate for all three groups of data, with an average error ranging from 0.0% to 19.2%, depending on the value of the *radius* parameter. For all algorithms, the error decreases as the *radius* parameter increases. However, FastDTW converges to 0% error much faster than the other two algorithms. A summary of the results for several *radius* settings is contained in Table 1.

Table 1 shows the average error for all three algorithms when run with a *radius* of 0, 1, 10, 20, and 30. FastDTW has a small error for all *radius* settings, and begins to approach 0% error when *radius* is set at or above 10. Abstraction is inaccurate for small *radius* values (983%), but is reasonably accurate when run with larger *radius* settings. The Band algorithm is very inaccurate for all *radius* settings except for 30. For a small *radius* of 0 or 1, the error for the Band algorithm is greater than two thousand percent.

The accuracy of each algorithm on the three different groups of data is displayed in Fig. 9, in which the x-axis is the *radius* parameter, and the y-axis is the error of the algorithm. Each group of data is shown graph in Fig. 9, and each of the three algorithms can be identified by the style of the line. Dashed lines are used for the Band algorithm, dotted lines are used for Abstraction, and solid lines are used for FastDTW.

The solid lines in Fig. 9 show the error of FastDTW on all three groups of data. The errors Fig. 9 are the mean error of all warp paths for every pair of time series in a group of data. The mean error of FastDTW for all *radius* parameters is smaller than the mean error of the Abstraction and Band algorithms over all three groups of data. The largest average improvement in error with a *radius* of 1 was on the Trace data set where FastDTW was 135 times more accurate than Abstraction, and 616 times more accurate than Bands. The smallest average improvement in error with a *radius* of 1 was on the "Random" data set

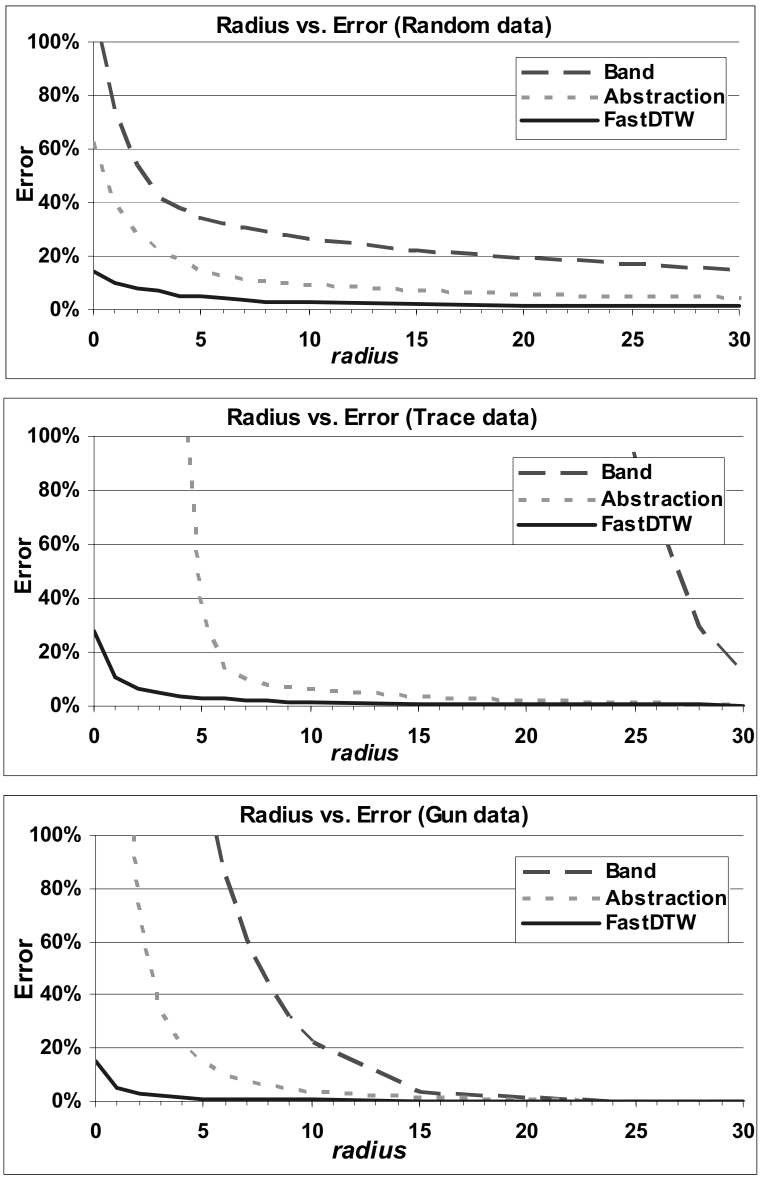

Fig. 9. Accuracy of FastDTW compared to Bands and Abstraction on all three groups of data.

where FastDTW was 4.4 times more accurate than Abstraction, and 7.4 times more accurate than Bands. The difference between the average errors in Fig. 9 is very significant due to the averages being calculated on large samples containing 990 (Random data set) to 19900 (Trace and Gun data sets) warp paths for each algorithm. As an example, the 95% confidence intervals for FastDTW are 0.3% to 1.5% with a *radius* of 1, while the error of FastDTW is 30% to 6508% less than the other two algorithms at the same *radius*. All 95% confidence intervals for the three algorithms are well separated except for large *radius* values on the Trace and Gun data sets where the error of each algorithm converges to zero (due to nearly all cells in the cost matrix being evaluated for the relatively short time series). The error of FastDTW also varies much less between different time series than the other two algorithms. Over all of the warp paths generated for the three data sets shown in Fig. 9 (40,790 total warp paths for each algorithm), the average standard deviation for a *radius* parameter of 1 was 8373% for the Band algorithm, 2749% for the Abstraction algorithm, and only 26% for the FastDTW algorithm This means that the accuracy of FastDTW is affected by the characteristics or similarity/dissimilarity of the input time series less than the Abstraction and Band algorithms. FastDTW is much more accurate than the other two methods when the *radius* parameter is set to small values. When the *radius* parameter is large, the Abstraction method begins to approach the accuracy of FastDTW. However, FastDTW was more accurate than Abstraction for all *radius* values, with the exception of *radius* values that were large enough for both the Abstraction and FastDTW algorithms to converge to 0.0% error).

Abstraction (dotted lines in Fig. 9) has large errors for small *radius* values, but converges to less than 5% error on all data sets as the *radius* is increased to 30. Small *radius* values have large errors because Abstraction blindly projects the warp path from a sampled time series onto a full resolution cost matrix. This projection may be "in the neighborhood" of a near-optimal warp path, but it fails to take into consideration any local variation in the warp path that is obscured by sampling. Local variations in the warp path can have a huge impact on the accuracy of a warp path. Increasing the *radius* setting can greatly increase the accuracy because this begins to adjust the warp path to cover local variations. However, the accuracy is still worse than FastDTW for a given *radius* because FastDTW projects the "neighborhood" of the near-optimal warp path from the previous resolution in several small steps rather than a single large step. Abstraction does perform reasonably well in our evaluation if the *radius* is increased to at least 10. The ability of Abstraction to locally refine its projected path within the neighborhood of *radius* cells is not a part of the original algorithm, and is introduced in this paper. The original Abstraction algorithm runs identically to our improved implementation when using a *radius* of 0, which has a very large average error of 983.3% over the three groups of data.

Sakoe-Chiba Bands can only find a near-optimal warp path if one exists that is almost entirely contained within *radius* cells from a linear warp. When the Band algorithm is used with a *radius* of 0, and the two time series are of equal length, it generalizes to Euclidean distance. A slight misalignment between the two time series can cause a very large amount of error in the warp path. Bands (dashed lines in Fig. 9) have errors greater than 100% (as high as 7225%) for small *radius* values, and converge very slowly to 0% error as the *radius* increases. The Band method performs best on random data because if two time series have almost nothing in common, an arbitrary warp path probably has a warp path distance that is relatively close to both the minimum-distance and maximum-distance warp paths. The other two groups of data contain time series that are similar to the rest of the time series in the group, which means that the optimal warp distance can be very small. Due to the way that error is calculated in Eq. (14), if the optimal warp distance is very small, then the potential error can be very large. The Band approach on the Trace data group has extremely poor accuracy because the time series contain events that are shifted in time, and Bands only work well if a near-optimal warp path exists that is close to a linear warp. The Gun data group also does not work very well with the Band algorithm, which is unexpected since the time series seem to be reasonably in phase with each other.

The *radius* parameter determines how many cells in the cost matrix are evaluated. Any approach will find an accurate warp path if a very large portion of the cost matrix is evaluated (large *radius* values). The algorithm that needs to evaluate the fewest cells in the cost matrix to find an accurate warp path is the most efficient algorithm. Each graph in Fig. 10 plots the percentage of the cost matrix evaluated against the error of the warp path that is found. Figure 10 is similar to Fig. 9, except that in this case the x-axis is the percentage of the cost matrix that is evaluated rather than the *radius* parameter. The "percentage of the cost matrix evaluated" is the number of cells evaluated (at all resolutions) divided by

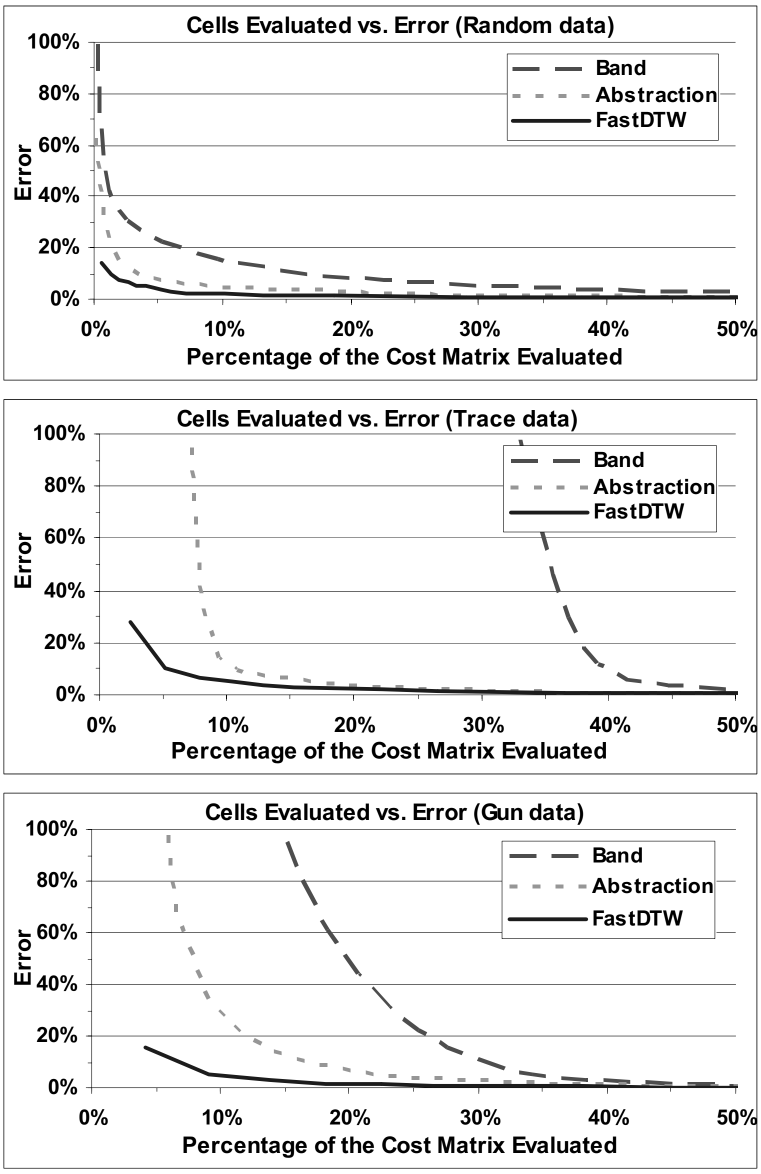

Fig. 10. Error vs. the % of the cost matrix evaluated.

the number of cells in the full resolution cost matrix. Comparing the algorithms in this way it is possible to compare the accuracy of all three algorithms when each uses the same amount of CPU resources. The number of cells evaluated (which is representative of the execution time) by FastDTW is approximately 50% greater than Bands and Abstraction for the same *radius*.

The Band algorithm (dashed line in Fig. 10) has a large amount of error if less than 40% of the cost matrix is evaluated. This percentage varies significantly between the groups of data. Time series that are in phase with each other (Gun data) have less error than time series that are out of phase (Trace data). Abstraction (dotted lines in Fig. 10) has a large error until about 10–30% of the cells are evaluated, at which point the error decreases sharply.

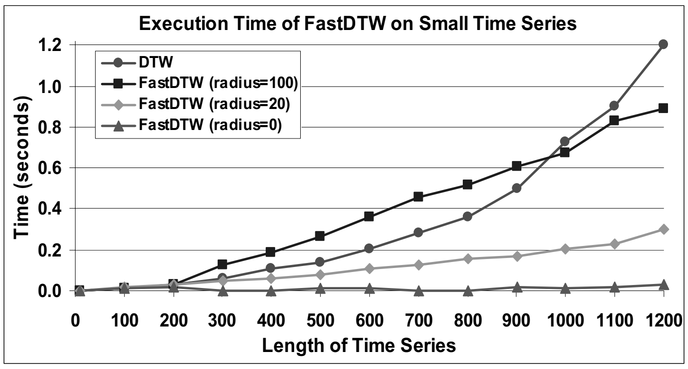

Fig. 11. Execution time on small time series.

The FastDTW algorithm has a small amount of error, even when the smallest possible fraction of cells is evaluated. This allows FastDTW to use a fixed *radius* parameter to achieve good results even as the length of the input time series are increasing; and since the *radius* is fixed, the percentage of the cost matrix that is evaluated decreases as the lengths of the time series increase.

4.2. *Efficiency*

4.2.1. *Procedures and criteria*

The efficiency of FastDTW will be measured in seconds, and compared to DTW. FastDTW will be run with the *radius* parameter set to 0, 20, and 100 over a range of varying-length time series. The data sets used are synthetic sine waves with Gaussian noise inserted. Only the lengths of the time series are significant because the shape of the time series has little influence on the run-time of either algorithm. The lengths of the time series evaluated vary from 10 to 180,000.

In this evaluation, DTW is the linear-space implementation that only retains the last two columns of the cost matrix. When the standard implementation is used, the test machine runs out of memory when the length of the time series exceeds ∼3,000. FastDTW is implemented as described in this paper except that the cost matrix is filled using secondary storage if the time series are so large that the number of cells in the search window is larger than can fit into main memory. Both algorithms are implemented in Java, and the runtime is measured using the system clock on a machine with minimal background processes running.

4.2.2. *Results and analysis*

The FastDTW algorithm was significantly faster than the classic DTW algorithm for all but the smallest time series. FastDTW is 50 to 150 times faster than classic DTW (using *radius* values of 100 and 0 respectively) for a time series with 150,000 points. The exact time series length at which FastDTW runs quicker than DTW depends on the *radius* parameter. Figure 11 shows the critical region where one algorithm is faster than the other depending on the *radius* parameter.

FastDTW, with a *radius* of 100, is slower than DTW until the size of the time series exceeds approximately 950 points (see Fig. 11). However, with a *radius* of 0 or 20, DTW is never faster than FastDTW. For small time series it makes more sense to use DTW rather than FastDTW. FastDTW is not significantly faster than DTW for small time series, and DTW is guaranteed to always find an optimal warp path. However, for large time series, FastDTW is much more efficient.

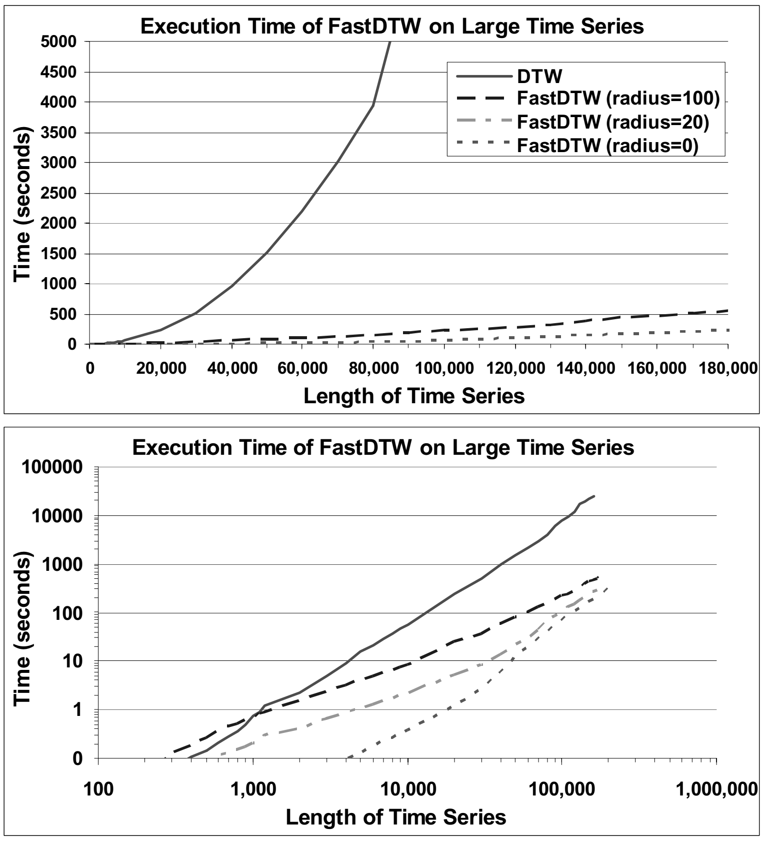

Fig. 12. The efficiency of FastDTW on long time series.

Figure 12 shows the execution times of FastDTW, with *radius* values of 0, 20, and 100 on time series ranging in length from 10 to 180,000. In Fig. 12, the y-axis is the execution time in seconds, and the x-axis is the length of the time series. The top graph is scaled normally, and the bottom graph is log-scaled. At the top of Fig. 12, DTW has a quadratic curve, while all three FastDTW curves are approximately straight lines. The *radius* parameter increases the execution of FastDTW by a constant factor, which is why the three FastDTW lines in the log-scaled graph seem to be converging as length of the time series increase.

In Section 3.1, we proved theoretically that the FastDTW algorithm was O(N). Using the empirical data in Fig. 12, the equation of the FastDTW curve with a *radius* of 100 is

$$y = 0.00000001x^2 + 0.001x - 0.7337$$

The coefficient of the squared term is very small, and the linear term appears to be the most significant term in the equation (which would empirically prove that the FastDTW algorithm is O(N)). However, values for x can be very large, and the squared term dominates the equation when $x >$100,000. The reason for this slight sub-linearity is that for large time series, the number of cells in the search window does not fit into main memory, and must be saved to the disk. Reading non-sequential cells from the disk cannot be performed in linear time. Larger time series create larger swap files, which require the disk head to move further to perform each random-access read operation. The number of cells in the cost matrix that must be filled/read is linear with respect to the length of the time series. So the *algorithm* is O(N), but the *implementation* is not quite O(N) for large time series when the entire search window will not fit into main memory.

5. Concluding remarks

In this paper we introduced FastDTW, a linear and accurate approximation of dynamic time warping (DTW). FastDTW uses a multilevel approach that recursively projects a warp path to a higher resolution and refines it. While the quadratic time and space complexity of DTW limits its use to only the smallest time series data sets, FastDTW can be run on much larger data sets. DTW is an order of magnitude faster than DTW, and it also compliments existing indexing methods that speed up time series similarity search and classification.

Our theoretical and empirical analysis show that FastDTW has a linear time and space complexity. Expirical results also show that FastDTW is accurate when warping both similar and dissimilar time series. With a *radius* of only 1, FastDTW had an average error of 8.6%, and increasing the *radius* to 20 lowered the error to under 1%. FastDTW's accuracy was compared to two existing methods, Abstraction and Sakoe-Chiba Bands. FastDTW had smaller (or equal) error than both existing approaches for all *radius* values, and was many times more accurate than either approach when using small *radius* values.

The main limitation of FastDTW is that it is an approximate algorithm and is not guaranteed to find an optimal solution (although it very often does). Future work will look into increasing the accuracy of FastDTW. Possibilities to increase the accuracy of FastDTW include changing the step size between resolutions between resolutions and changing the size of the initial resolution. Initial tests have indicated that starting at a (coarsest) resolution of $\sqrt{N}$ improves the accuracy of the FastDTW, while retaining its linear time complexity. Using piecewise linear approximations to more accurately represent time series structure at lower resolutions is another possibility for increasing the accuracy of FastDTW. In some applications, an optimal warp path may not always provide the best results due to discontinuities in the warp path where the warp path is only incrementing one of the time series for a relative long sequence. Modifications that balance the tradeoff between a minimum-error warp path and a smooth warp path [6] are another important area for future research.

References

[1] W. Abdulla, D. Chow and G. Sin, *Cross-Words Reference Template for DTW-Based Speech Recognition Systems*, in Proc. IEEE TENCON, Bangalore, India, 2003.

[2] D. Bylund, R. Danielsson, G. Malmquist and K.E. Markides, Chromatographic Alignment by Warping and Dynamic Programming as a Pre-Processing Tool for PARAFAC Modeling of Liquid Chromatography-Mass Spectrometry Data, *In Journal of Chromatography A* (2002), 237–244.

[3] S. Chu, E. Keogh, D. Hart and M. Pazzani, *Iterative Deepening Dynamic Time Warping for Time Series*, In Proc. of the Second SIAM Intl. Conf. on Data Mining. Arlington, Virginia, 2002.

[4] B. Ferrell and S. Santuro, NASA Shuttle Valve Data [http://www.cs.fit.edu/˜pkc/nasa/data/], 2005.

[5] L. Gupta, D. Molfese, R. Tammana and P. Simos, Nonlinear Alignment and Averaging for Estimating the Evoked Potential, *In IEEE Transactions on Biomedical Engineering* **43**(4) (1996), 346–356.

[6] A. Helair, N. Courty, S. Gibet and F. Multan, Temporal alignment of communicative gesture sequences, *In Computer Animation and Virtual Worlds* **17**(3–4) 347–357.

[7] F. Itakura, Minimum Prediction Residual Principle Applied to Speech Recognition, *In IEEE Trans Acoustics, Speech, and Signal Proc* **ASSP-23** (1975), 52–72.

[8] G. Karypis, R. Aggarwal, V. Kumar and S. Shekhar, *Multilevel Hypergraph Partitioning: Application in VLSI Domain*, In Design Automation Conl, 526–530, Anaheim, California, 1997.

[9] E. Keogh, *Exact Indexing of Dynamic Time Warping*, In VLDB, 406–417, Hong Kong, China, 2002.

[10] E. Keogh and T. Folias, The UCR Time Series Data Mining Archive [http://www.cs.ucr.edu/˜eamonn/TSDMA/ index.html], Riverside, CA, University of California – Computer Science and Engineering Department, 2003.

[11] E. Keogh and S. Kasetty, *On the Need for Time Series Data Mining Benchmarks: A Survey and Empirical Demonstration*, In Proc. of the 8th ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining, 102–111, Edmonton, Canada, 2002.

[12] E. Keogh and M. Pazzani, *Relevance Feedback Retrieval of Time Series*, In 22nd ACM-SIGIR Conf. on Research Development in Information Retrieval, 183–190, Berkeley, California, 1999.

[13] E. Keogh and M. Pazzani, *Scaling up Dynamic Time Warping for Datamining Applications*, In Proc. of the Sixth ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining, 285–289, Boston, Massachuseetts, 2000.

[14] S. Kim, S. Park and W. Chu, *An Index-based Approach for Similarity Search Supporting Time Warping in Large Sequence Databases*, In Proc. 17th Intl. Conf. on Data Engineering, 607–614, Heidelberg, Germany, 2001.

[15] L. Kovar and M. Gleicher, *Automated Extraction and Parameterization in Large Data Sets*, In Proc SIGGRAPH, 559–568, Vol. 23, issue 3, 2004.

[16] J. Kruskall and M. Liberman, *The Symmetric Time Warping Problem: From Continuous to Discrete*, In Time Warps, String Edits and Macromolecules: The Theory and Practice of Sequence Comparison, pp. 125–161, Addison-Wesley Publishing Co., Reading, Massachusetts, 1983.

[17] C. Ratanamahatana and E. Keogh, *Making Time-series Classification More Accurate Using Learned Constraints*, In Proc. of SIAM Intl. Conf. on Data Mining, pp. 11–22, Lake Buena Vista, Florida, 2004.

[18] C. Ratanamahatana and E. Keogh, *Three Myths about Dynamic Time Warping*, In Proc of SIAM Intl. Conf. on Data Mining, Newport Beach, California, 2005.

[19] D. Roverso, *Multivariate Temporal Classification by Windowed Wavelet Decomposition and Recurrent Neural Networks*, In Proc. Of the 3rd ANS Intl. Topical Meeting on Nuclear Plant Instrumentation, Control and Human-Machine Interface Technologies, Washington, D.C., 2000.

[20] H. Sakoe and S. Chiba, *Dynamic Programming Algorithm Optimization for Spoken Word Recognition*, In IEEE Trans. Acoustics, Speech, and Signal Processing, Vol. ASSP-26, 1978.

[21] S. Salvador, Learning States for Detecting Anomalies in Time Series, Master's Thesis, CS-2004-05, Dept. of Computer Sciences, Florida Institute of Technology, 2004.