FastDTW: Toward Accurate Dynamic Time Warping in Linear Time and Space

Stan Salvador and Philip Chan
Dept. of Computer Sciences
Florida Institute of Technology
Melbourne, FL 32901
{ssalvado, pkc}@cs.fit.edu

ABSTRACT

The dynamic time warping (DTW) algorithm is able to find the optimal alignment between two time series. It is often used to determine time series similarity, classification, and to find corresponding regions between two time series. DTW has a quadratic time and space complexity that limits its use to only small time series data sets. 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 coarse 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 compared to two other existing approximate DTW algorithms: Sakoe-Chuba Bands and Data Abstraction. Our results show a large improvement in accuracy over the existing methods.

Keywords

dynamic time warping, time series

1. INTRODUCTION

Motivation. Dynamic time warping (DTW) is a technique that finds the optimal alignment between two time series if one time series may be "warped" non-linearly by stretching or shrinking it along its time axis. This warping between two time series can then be used to find corresponding regions between the two time series or to determine the similarity between the two time series. Dynamic time warping is often 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. In addition to speech recognition, dynamic time warping has also been found useful in many other disciplines [8], including data mining, gesture recognition, robotics, manufacturing, and medicine. Dynamic time warping is commonly used in data mining as a distance measure between time series. An example of how one time series is "warped" to another is shown in Figure 1.

In Figure 1, each vertical line connects a point in one time series to its correspondingly similar point in the other time series. The lines actually have similar values on the *y*-axis but have been separated so the vertical lines between them can be viewed more easily. If both of the time series in Figure 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. The warp path distance is a measure of the difference between the two time series after they have been warped together, which is measured by the sum of the distances between each pair of points connected by the vertical lines in Figure 1. Thus, two time series that are identical except for localized stretching of the time axis will have DTW distances of zero.

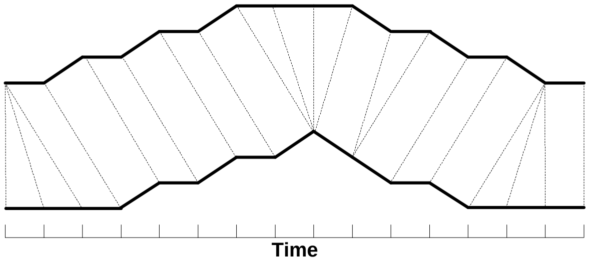

Figure 1. A warping between two time series.

Despite the effectiveness of the dynamic time warping algorithm, it has an O(N^2) time and space complexity that limits its usefulness to small time series containing no more than a few thousand data points. More details of the dynamic time warping algorithm are contained in Section 2.1.

Problem. We desire to develop a dynamic time warping algorithm that is linear in both time and space complexity and can find a warp path between two time series that is nearly optimal.

Approach. In this paper we introduce the FastDTW algorithm, which is able to find an accurate approximation of the optimal warp path between two time series. The FastDTW algorithm avoids the brute-force dynamic programming approach of the standard DTW algorithm by using a multilevel approach. The time series are initially sampled down to a very low resolution. A warp path is found for the lowest resolution and "projected" onto an incrementally higher resolution time series. The projected warp path is refined and projected again to yet a higher resolution. The process of refining and projecting is continued until a warp path is found for the full resolution time series.

Contributions. Our main contribution is the introduction of the FastDTW algorithm, which is 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 empirically demonstrate that FastDTW produces an accurate minimum-distance warp path between two time series than is nearly optimal (standard DTW is optimal, but has a quadratic time and space complexity). In addition to the FastDTW algorithm, we evaluate other existing approximate DTW algorithms, and compare their accuracy on a large and diverse group of time series data sets.

Organization. The next section describes the standard 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 the FastDTW algorithm 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 between time series is needed to determine 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 time series is simply the sum of the squared distances from each *n*th point in one time series to the *n*th point in the other. The main disadvantage of using Euclidean distance for time series data is that its results are very unintuitive. If two time series are identical, but one is shifted slightly along the time axis, then Euclidean distance may consider them to be very different from each other. Dynamic time warping (DTW) was introduced [11] to overcome this limitation and give intuitive distance measurements between time series by ignoring both global and local shifts in the time dimension.

Problem Formulation. The dynamic time warping problem is stated as follows: Given two time series *X*, and *Y*, of lengths |*X*| and |*Y*|,

$$X = x_1, x_2, \ldots, x_i, \ldots, x_{|X|}$$

$$Y = y_1, y_2, \ldots, y_j, \ldots, y_{|Y|}$$

construct a warp path *W*

$$W = w_1, w_2, \ldots, w_K \qquad \max(|X|, |Y|) \leq K < |X| + |Y|$$

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 from time series *X*, and *j* is an index from time series *Y*. The warp path must start at the beginning of each time series at $w_1 = (1, 1)$ and finish at the end of both time series at $w_K = (|X|, |Y|)$. This ensures that every index of both time series is used in the warp path. 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 Figure 1 do not overlap. Every index of each time series must be used. Stated more formally:

$$w_k = (i, j), w_{k+1} = (i', j') \qquad i \leq i' \leq i+1, \;\; j \leq j' \leq j+1$$

The optimal warp path is the warp path is the minimum-distance warp path, where the distance of a warp path *W* is

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

Dist(*W*) is the distance (typically Euclidean distance) of warp path *W*, and $Dist(w_{ki}, w_{kj})$ 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.

DTW Algorithm. A dynamic programming approach is used to find this minimum-distance warp path. Instead of attempting to solve the entire problem all at once, solutions to sub-problems (portions of the time series) are found, and used to repeatedly find solutions to a slightly larger problem until the solution is found for the entire time series. A two-dimensional |*X*| by |*Y*| cost matrix *D*, is constructed where the value at *D*(*i*, *j*) is the minimum-distance warp path that can be constructed from the two time series $X' = x_1, \ldots, x_i$ and $Y' = y_1, \ldots, y_j$. The value at *D*(|*X*|, |*Y*|) will contain the minimum-distance warp path between time series *X* and *Y*. Both axes of *D* represent time. The *x*-axis is the time of time series *X*, and the *y*-axis is the time of time series *Y*. Figure 2 *D* shows an example of a cost matrix and a minimum-distance warp path traced through it from *D*(1, 1) to *D*(|*X*|, |*Y*|).

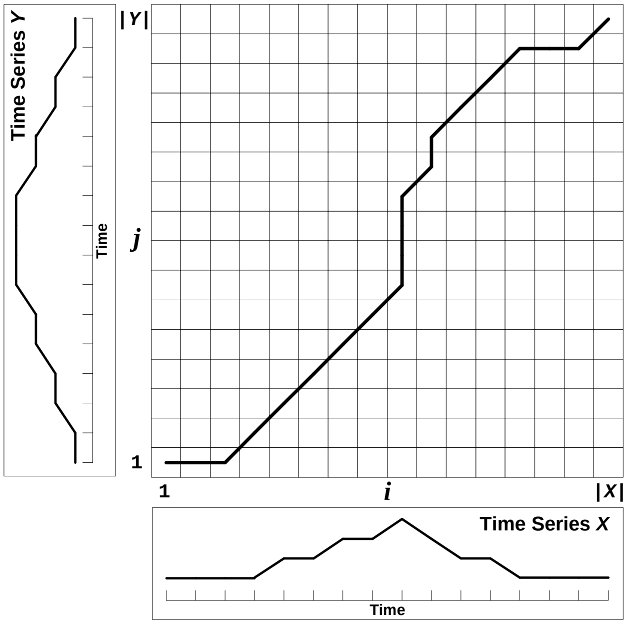

Figure 2. A cost matrix with the minimum-distance warp path traced through it.

The cost matrix and warp path in Figure 2 are for the same two time series shown in Figure 1. The warp path is *W* = {(1,1), (2,1), (3,1), (4,2), (5,3), (6,4), (7,5), (8,6), (9,7), (9,8), (9,9), (9,10), (10,11), (10,12), (11,13), (12,14), (13,15), (14,15), (15,15), (16,16)}. If the warp path passes through a cell *D*(*i*, *j*) in the cost matrix, it means that the i^{th} point in time series *X* is warped to the j^{th} point in time series *Y*. Notice that where there are vertical sections of the warp path, a single point in time series *X* is warped to multiple points in time series *Y*, and the opposite is also true where the warp path is a horizontal line. 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. If *X* and *Y* were identical time series, the warp path through the matrix would be a straight diagonal line.

To find the minimum-distance warp path, every cell of the cost matrix must be filled. The rationale behind using a dynamic programming approach to this problem is that since the value at *D*(*i*, *j*) is the minimum warp distance of two time series of lengths *i* and *j*, if the minimum warp distances are already known for all slightly smaller 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 of all possible warp paths for time series that are one data point smaller than *i* and *j*, plus the distance between the two points x_i and y_j. Since the warp past must either be incremented 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:

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

The warp path to *D*(*i*, *j*) must pass through one of those three grid cells, and since the minimum possible warp path distance is already known for them, all that is needed is to simply add the distance of the current two points to the smallest one. Since this equation determines the value of a cell in the cost matrix by using the values in other cells, the order that they are evaluated in is very important. The cost matrix is filled one column at a time from the bottom up, from left to right as depicted in Figure 3.

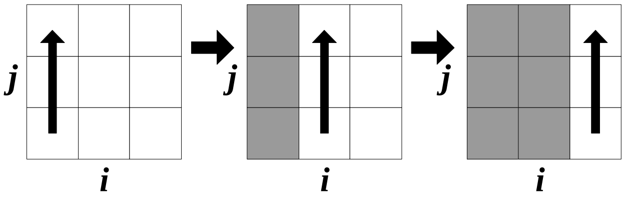

Figure 3. The order that the cost matrix is filled.

After the entire matrix is filled, a warp path must be found from *D*(1, 1) to *D*(|*X*|, |*Y*|). The warp path is actually calculated in reverse order starting at *D*(|*X*|, |*Y*|). A greedy search is performed that evaluates cells to the left, down, and diagonally to the bottom-left. Whichever of these three adjacent cells has the smallest value is added to the beginning of the warp path found so far, and the search continues from that cell. The search stops when *D*(1, 1) is reached.

Complexity of DTW. Time and Space complexity of the DTW is easy 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*| by |*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 DTW algorithm is possible by only keeping the current and previous columns in memory as the cost matrix is filled from left to right (see Figure 3). By only retaining two columns at any one time, the optimal warp distance between the two time series can be determined. However it is not possible to reconstruct the warp path between these two time series because the information required to calculate the warp path is thrown away with the 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 [14] or merge time series together [1][3] 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 methods to speed up dynamic time warping. The methods used make DTW faster fall into three categories:

1) *Constraints* – Limit the number of cells that are evaluated in the cost matrix.

2) *Data 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-Chuba Band [13] and the Itakura Parallelogram [4], which are shown in Figure 4.

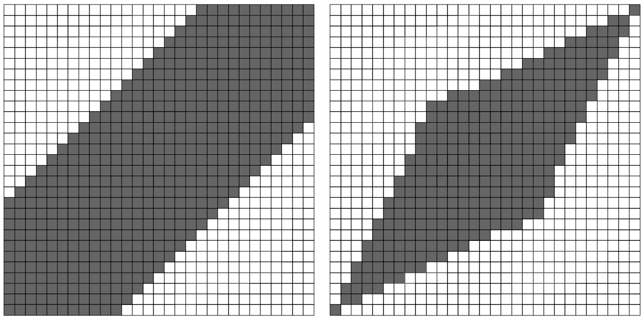

Figure 4. Two constraints: Sakoe-Chuba Band (left) and an Itakura Parallelogram (right), both have a width of 5.

The shaded areas in Figure 4 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, the DTW algorithm finds the 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 the DTW algorithm is still O(N^2) if the size of the input window is a function of the length of the input time series. Constraints work well in domains where the optimal warp path is expected to be close to a linear warp and passes through the cost matrix diagonally in a relatively straight line. Constraints work poorly if time series are of events that start and stop at radically different times because the warp path can stray very far from a linear warp and nearly the entire cost matrix must be evaluated to find the optimal warp path.

Data abstraction speeds up the DTW algorithm by running DTW on a reduced representation of the data [2][9]. The left side of Figure 5 shows a full-resolution cost matrix for which a minimum-distance warp path must be found. Rather than running the DTW algorithm 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 the full resolution cost matrix.

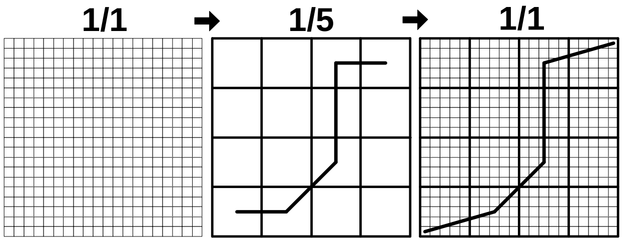

Figure 5. Speeding up DTW by data abstraction.

The result is that DTW is sped up by a large constant factor, but the algorithm still runs in O(N^2) time and space. Obviously, the warp distance that is calculated between the two time series becomes increasingly inaccurate as the level of abstraction increases. Projecting the lower resolution warp path to the full resolution usually creates a warp path that is far from optimal because even *IF* the optimal warp path actually 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 uses lower-bounding functions to prune out the number of times DTW needs to be run for certain tasks such as clustering a set of time series or finding the time series that is most similar to a given time series [6][10]. Indexing significantly speeds up many DTW applications by reducing the number of times DTW is run, but does not speed up the actual DTW algorithm.

Our FastDTW algorithm uses ideas from both the constraints and data abstraction categories. Using a combination of both overcomes many limitations of using either method individually, and yields an algorithm that is O(N) in both time and space.

3. APPROACH

The multilevel approach that FastDTW uses is inspired by the multilevel approach used for graph bisection [5]. 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 the 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

The FastDTW algorithm 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 that warp path as an initial guess for a higher resolution's minimum-distance warp path.

3) ***Refinement*** – Refine the warp path projected from a lower resolution through local adjustments of the warp path.

Coarsening reduces the size (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 several times to produce many different resolutions of the time series. *Projection* takes a warp path calculated at a lower resolution and determines what cells in the next higher resolution time series the warp path passes through. 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 solution refinement to find a warp path at the higher resolution. *Refinement* finds the optimal warp path *in the neighborhood* of the projected path, where the size of the neighborhood is controlled by the *radius* parameter.

Standard dynamic time warping (DTW) is an O(N^2) algorithm 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 the 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 size of the input time series, the 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 Figures 1 and 2. The standard DTW algorithm is run to find the optimal warp path for the lowest resolution time series. This lowest resolution warp path is shown in the left of Figure 6. After the warp path is found for the lowest resolution, it is projected to the next higher resolution. In Figure 6, the projection of the warp path from a resolution of 1/8 is shown as the heavily shaded cells at 1/4 resolution.

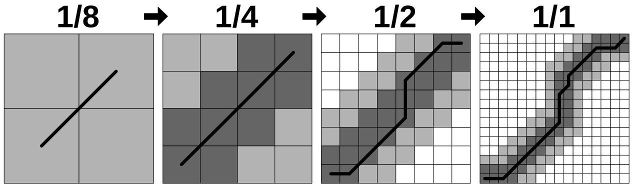

Figure 6. The four different resolutions evaluated during a complete run of the FastDTW algorithm.

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 the optimal warp path *through the area of the warp path that was projected from the lower resolution*. However, the entire optimal warp path may not be contained within projected path. To increase the chances of finding the 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 Figure 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 Figure 6. The projection is expanded by the *radius* and refined one last time. This refined warp path is the output of the algorithm.

Notice that the warp path found by the FastDTW algorithm in Figure 6 is the optimal warp path that was found by the standard DTW in Figure 2. However, FastDTW only evaluated the shaded cells, while DTW evaluates all of the cells in the cost matrix. FastDTW evaluated 4+16+44+100=164 cells at all resolutions, while DTW evaluates all 235 (16^2) cells. This increase in efficiency is not very significant for his 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 DTW always evaluates N^2 cells (if both time series are of length N). FastDTW scales linearly because the width of the path through the matrix that is being evaluated is constant at all resolutions.

The example in Figure 6 finds the optimal warp path, but the FastDTW algorithm is not guaranteed to always find a warp path that is optimal. However, 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 the DTW algorithm (optimal but $O(N^2)$). The accuracy of FastDTW using different settings for the *radius* parameter will be demonstrated in Section 4.

The pseudocode for the FastDTW algorithm is shown Figure 7. The input to the algorithm is two time series, and the *radius* parameter. The output of FastDTW is a warp path and the distance between the two time series along that warp path. Line 2 determines the minimum length of a time series at the lowest resolution. This size is dependent on the *radius* parameter and determines the smallest possible resolution size for which decreasing the resolution further would be pointless because full dynamic time warping would need to be calculated at more than one resolution.

FastDTW has a straightforward recursive implementation. The base case is when one of the input time series has a length less than *minTSsize*. For the base case, the algorithm simply returns the result of the standard DTW algorithm. The recursive case has three main steps. First, two new lower-resolution time series are created that have half as many points as the input time series (***coarsening***). This is performed by lines 17-18 in Figure 7. Next, a low resolution path is found for the coarsened time series (lines 20-21) and ***projected*** to a higher resolution (lines 23-25). This projected path is also expanded by *radius* cells to create a search window that will be passed to a constrained version of the DTW algorithm that only evaluates the cells in the search window (line 27). The constrained DTW algorithm ***refines*** the warp path that was projected form the lower resolution. The result of this refinement is then returned.

```
                 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)
 5| {
 6|   // Base Case: for a very small time series run
 7|   //   the full DTW algorithm.
 8|   RETURN DTW(X, Y)
 9| }
10| ELSE
11| {
12|   // Recursive Case: Project the warp path from
13|   //   a coarser resolution onto the current
14|   //   current resolution.  Run DTW only along
15|   //   the projected path (and also ‘radius’ cells
16|   //   from the  projected path).
17|   TimeSeries shrunkX = X.reduceByHalf()
18|   TimeSeries shrunkY = Y.reduceByHalf()
19|
20|   WarpPath lowResPath =
21|       FastDTW(shrunkX,shrunkY, radius)
22|
23|   SearchWindow window =
24|       ExpandedResWindow(lowResPath, X, Y,
25|                              radius)
26|
27|   RETURN DTW(X, Y, window)
28| }
```

Figure 7. The FastDTW algorithm.

The execution of the FastDTW algorithm repeatedly runs lines 17-18 in recursive calls to lower resolutions are made by line 21. This creates multiple resolutions until the base case is reached (line 8). The base case is executed only a single time, and afterwards lines 23-27 are executed for each recursive call (or resolution) on the stack.

Next, we will provide a theoretical analysis of FastDTW based on time and space complexity.

Time Complexity of FastDTW. To simplify the calculations we will assume that the two full-resolution 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 any other cells within *radius* (denoted as *r* in the rest of this analysis to save space) cells away from the projected path. The worst case, a straight diagonal projected warp path is depicted in Figure 8.

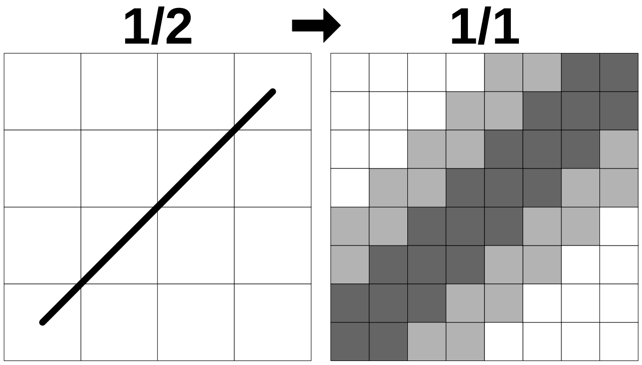

Figure 8. Maximum (worst-case) number of cells evaluated for a *radius* of 1.

The lightly shaded cells in Figure 8 are the 2*Nr* cells on each side of the projected path (heavily shaded cells), which itself has 3*N* cells. The projected path therefore has the following maximum number of cells at a resolution with two time series containing *N* points:

$$3N + 2(2Nr) = N(4r+3) \qquad [1]$$

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

$$\left\{\frac{N}{2^{res}}\right\}_{res=0}^{res=\infty} = N, \frac{N}{2}, \frac{N}{2^2}, \frac{N}{2^3}, \frac{N}{2^4}, \cdots \qquad [2]$$

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

$$\sum_{res=0}^{\infty} \frac{N}{2^{res}}(4r+3) = N(4r+3) + \frac{N}{2}(4r+3) + \frac{N}{2^2}(4r+3) + \cdots \qquad [3]$$

The series in Equation 3 is very similar to the series

$$\sum_{res=0}^{\infty} \frac{1}{2^{res}} = 1 + \frac{1}{2} + \frac{1}{2^2} + \frac{1}{2^3} + \frac{1}{2^4} + \cdots = 2 \qquad [4]$$

Multiplying Equation 4 by Equation 1 yields

$$N(4r+3) + \frac{N}{2}(4r+3) + \frac{N}{2^2}(4r+3) + \cdots = 2N(4r+3) \qquad [5]$$

Since the sequence in Equation 5 is identical to the sequence in Equation 3, the number of cells evaluated at all resolutions is

Total number of cells filled = $2N(4r+3)$ [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 needed to create the resolutions is proportional to the number of points in all of the resolutions, which is the series in Equation 2. The solution of Equation 2 is obtained by multiplying Equation 4 by *N*, which yields 2*N*. Since multiple resolutions of both time series must be created, 2*N* is multiplied by two to get the final time complexity.

Time to create all resolutions = $4N$ [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 2*N* in the worst case (*N* is the best case for a diagonal line). Multiplying Equation 4 by 2*N* gives the worst-case length of all warp paths added together from every resolution:

Time to trace warp paths = $4N$ [8]

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

FastDTW time complexity = $N(8r+14)$ [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 Equation 2 without the first them, which is 2*N*-*N*=*N*. For both input time series the space complexity is

Space of resolutions (other that full resolution) = $2N$ [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 Equation 1

Space of cost matrix = $N(4r+3)$ [11]

The space complexity of storing the warp path is equal to the longest warp path that can exist at full resolution. If the warp path traces the perimeter of the cost matrix, then the length of that path will be

Space complexity of storing the warp path = $2N$ [12]

And adding Equations 10, 11, and 12 gives the total worst-cast space complexity of

FastDTW space complexity = $N(4r+7)$ [13]

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

4. EMPIRICAL EVALUATION

The goal of this evaluation is demonstrate the efficiency and accuracy of the FastDTW algorithm on a wide range of time series data sets. To ensure reproducibility, all datasets and algorithms used in this evaluation can be found online at “http://cs.fit.edu/~pkc/FastDTW/”. This evaluation will first demonstrate the accuracy of the FastDTW algorithm and will then empirically verify its linear time complexity.

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. The error of an approximate DTW algorithm, such as our FastDTW algorithm, is calculated by the following equation:

$$\textbf{\textit{Error of a warp path}} = \frac{approxDist - optimalDist}{optimalDist} \times 100 \quad [14]$$

If the DTW 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 the standard DTW algorithm. The error of a warp path will always be ≥0% (because *optimalDist* is never larger than *approxDist*) and can exceed 100% if the distance of the approximate warp path is more than double the optimal distance.

The FastDTW algorithm is evaluated against two other existing approximate DTW algorithms: Sakoe-Chuba bands and data abstraction. Sakoe-Chuba bands (see left side of Figure 4) constrain the DTW algorithm to only evaluate a specified *radius* away from a linear warp within the cost matrix. Itakura Parallelograms (see right side of Figure 4) are not evaluated because, for a given *radius*, a band will always find a warp path equal to or better than that of the parallelogram. This is because the parallelogram constraint is a subset of the band constraint. The data abstraction DTW algorithm used in this evaluation first samples the data, and then runs the standard DTW algorithm to find a warp path on the sampled data. This warp path is then projected to the full resolution as previously shown in Figure 5.

The *radius* parameter performs a similar function for all three algorithms. It expands the region of the cost matrix searched from an initial "guess". For bands, the initial guess is a linear warp. For data abstraction, it is the projected warp path from the sampled data, and for FastDTW it is the projected warp path from the previous resolution. Each algorithm will be run with multiple *radius* parameters on a wide range of data sets.

All three algorithms (FastDTW, bands, and data abstraction) are only being evaluated based on accuracy in this section. However, care has been taken to ensure that the time each algorithm requires to execute is similar for the same *radius*. The data abstraction algorithm is made O(N) by sampling the data down to $\sqrt{N}$ points before performing quadratic time warping (O($\sqrt{N}^{\,2}$) = O(N)). All three algorithms evaluate roughly the same number of cells in the cost matrix for any particular *radius*. FastDTW has some overhead for evaluating previous resolutions, and data abstraction has overhead for running standard DTW on the sampled time series. However, all three algorithms are linear with respect to the length of the input time series, and the number of cells evaluated for a given *radius* does not differ by more than a power of two of for any pair of algorithms.

The time series data sets used to evaluate the accuracy of the FastDTW algorithm include very similar data sets that are from the same domain, and dissimilar data sets 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. Dynamic time warping is most frequently used to compare the similarity between time series, so it is likely that the majority of time series that are compared are similar and from the same domain. However, very dissimilar time series are also evaluated to ensure that the approximate FastDTW algorithm works well when warping two time series that do not share common features. The accuracy of each DTW algorithm is measured on three groups of data:

1) *Random* – 990 time warps between 45 time series from different domains (eeg, random walk, earthquake, speech, tide, etc.). The average length is 1128 points.

2) *Trace* - 10,900 time warps between 200 time series data sets. The Gun domain contains 4 classes that simulate instrumentation failure in a nuclear power plant. All time series have a length of 275 points.

3) *Gun* – 10,900 time warps between 200 time series data sets. The Gun domain contains 2 classes, with 100 time series of a gun being drawn from a holster and 100 time series of a gun being pointed. All time series have a length of 151 points.

All data sets used in this evaluation were obtained from the UCR Time Series Data Mining Archive and are publicly available [7]. Each algorithm and group of data is also run multiple times with the following settings for the *radius* parameter: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 20, and 30. For a given algorithm, group of data, and *radius*, the average error of all possible warp paths between time series in the group are recorded.

4.1.2 Results and Analysis

The FastDTW algorithm is very accurate for all three groups of data that it was tested on. FastDTW has an error of only 19.2% to 0.0%, 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. Average error of three the 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%

Table 1 shows the average error for all three algorithms over all three test cases, when run with the *radius* set to 0, 1, 10, 20, and 30. FastDTW has a small amount of error for all *radius* settings, and begins to approach 0% error when *radius* is set at or above 10. Data abstraction is inaccurate for small *radius* values, but begins to be reasonably accurate when run with larger *radius* settings. The band algorithm is very inaccurate for all *radius* settings except for 30.

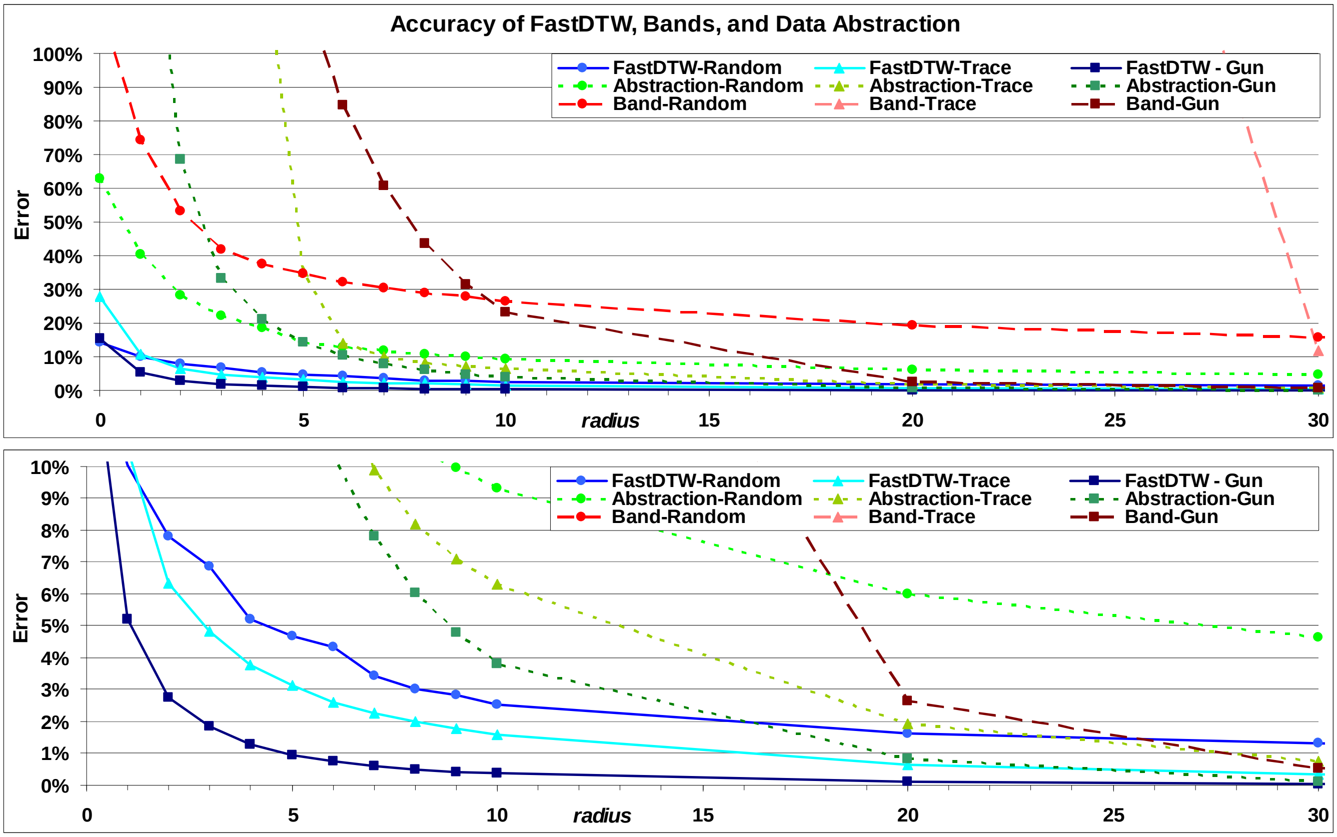

Figure 9. Accuracy of FastDTW compared to Bands and Data Abstraction. The top figure's *y*-axis is 0%-100% and the bottom figure's *y*-axis is 0%-10%.

Data abstraction is inaccurate (500-1000% error) for small *radius* settings because it 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 (which is not part of the original data abstraction algorithm, it is introduced in this paper), can make it rather accurate 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.

Bands can only have good results if a near-optimal warp path is entirely contained within *radius* cells from a linear warp. When bands are used with a *radius* of 0, and the two time series are of equal length, it generalizes to Euclidean distance...which is a notoriously inaccurate similarity measure for time series [15]. A slight misalignment between the two time series being warped can cause a very large amount of error in the warp path.

The accuracy of each algorithm on the different groups of data is displayed in Figure 9.

In Figure 9, the *x*-axis is the radius parameter used, and the *y*-axis is the error of the tested algorithm. Each of the 9 lines is a combination between the three algorithms and the three groups of data sets. The FastDTW algorithm curves are solid lines, data abstraction curves are dotted lines, and band curves are dashed lines. The three groups of data can be identified by the shape of the markers on the curves. Round markers are used on curves using Random data, triangle markers are for the Trace data, and square markers are for the Gun data.

The three solid lines at the bottom of Figure 9 are the error curves for FastDTW on all three groups of data. The error is small for all three lines, meaning that the accuracy FastDTW is not effected very much by the characteristics or similarity of the input time series. FastDTW is significantly more accurate than the other two methods when the *radius* parameter is set to small values. When the *radius* parameter is larger, the abstraction method begins to approach the accuracy of FastDTW. However, FastDTW was always at least 2-3 times more accurate than abstraction in our experiments.

The three dotted Abstraction lines all have large errors for small *radius* values, but converge to less than 5% error on all data sets as the *radius* is increased to 30. This is due to the previously stated problem of the projected warp path being close to a near-optimal solution, but not taking local variations of the warp path into account. Abstraction does perform reasonably well if the *radius* is increased to at least 10. The ability of data 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 run-time of the original data abstraction algorithm is the same as 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 used in this evaluation.

The three dashed Band lines all have errors greater than 100% (as high as 7225%) for small *radius* values, and converge very slowly to 0% error as the *radius* increases. Band performs best on the random data because if two time series have almost nothing in common, an arbitrary warp path probably has a warp path distance that is not significantly much different from the minimum-distance or maximum-distance warp paths. The other two groups of data are data sets in a similar domain, which means that the optimal warp distance can be very small. Due to the way that error is calculated in Equation 14, if the optimal warp distance is very small, then the potential error can be very large because the optimal warp distance is the denominator of a fraction. 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 surprising since the time series seem to be reasonably in phase with each other (near a linear warp).

4.2 Efficiency

4.2.1 Procedures and Criteria

The efficiency of the FastDTW algorithm will be measured in seconds, with respect to the length of the input time series, and compared to the standard DTW algorithm. The FastDTW algorithm 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 data sets of a single period of a sine wave with Gaussian noise inserted. Only the lengths of the time series are significant because the shape of the time series has little significance on the run-time of either algorithm. The lengths of the time series evaluated vary from 10 to 150,000.

The standard DTW algorithm used in this evaluation is the linear-space implementation that only retains the last two columns of the cost matrix. If the standard DTW implementation is used, the test machine runs out of memory when the length of the time series exceeds ~3,000. The FastDTW algorithm is implemented as described in this paper except that the cost matrix is filled using secondary storage if the lengths of the time series grow 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 standard DTW algorithm for all but the smallest time series. FastDTW is 50 to 150 times faster than standard DTW (using *radius* values of 0 and 100 respectively) when the time series have lengths of 150,000 points. A sample of the results of the FastDTW algorithm can be seen in Table 2.

Table 2. Execution time (in seconds) of DTW and FastDTW on time series of four different lengths.

	Length of Time Series			
	100	*1,000*	*10,000*	*100,000*
DTW	0.02	0.92	57.45	7969.59
FastDTW (*radius*=0)	0.01	0.02	0.38	67.94
FastDTW (*radius*=100)	0.02	0.06	8.42	207.19

In Table 2, FastDTW and DTW have similar execution times for the 100 point time series. For the larger 10,000 and 100,000 point time series FastDTW runs much more quickly than DTW. But execution time for the 1,000 point time series is both faster and slower than DTW, depending on the *radius* parameter. The exact length at which FastDTW runs quicker than DTW depends on the *radius* parameter. Figure 10 shows the critical region where one algorithm is faster than the other depending on the *radius* parameter.

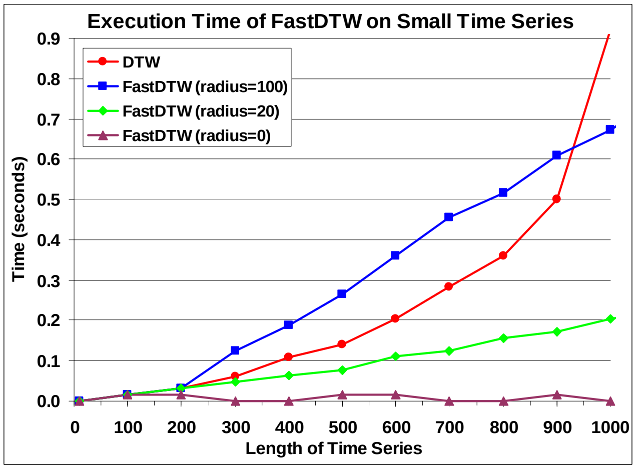

Figure 10. The efficiency of FastDTW and DTW on small time series.

The FastDTW algorithm, with a *radius* of 100, takes longer to run than DTW until the size of the time series exceeds approximately 900 points. However, with a *radius* of 0 or 20, the DTW algorithm is never faster than the FastDTW algorithm for small time series, and once the length of the time series exceed 200-300 points, FastDTW becomes the more efficient algorithm. For small time series it makes more sense to use the DTW algorithm rather than FastDTW. The FastDTW algorithm is not significantly faster (and possibly a little slower) than DTW for small time series, and DTW is guaranteed to always find the optimal warp path. However, for large time series, the quadratic time complexity becomes prohibitive.

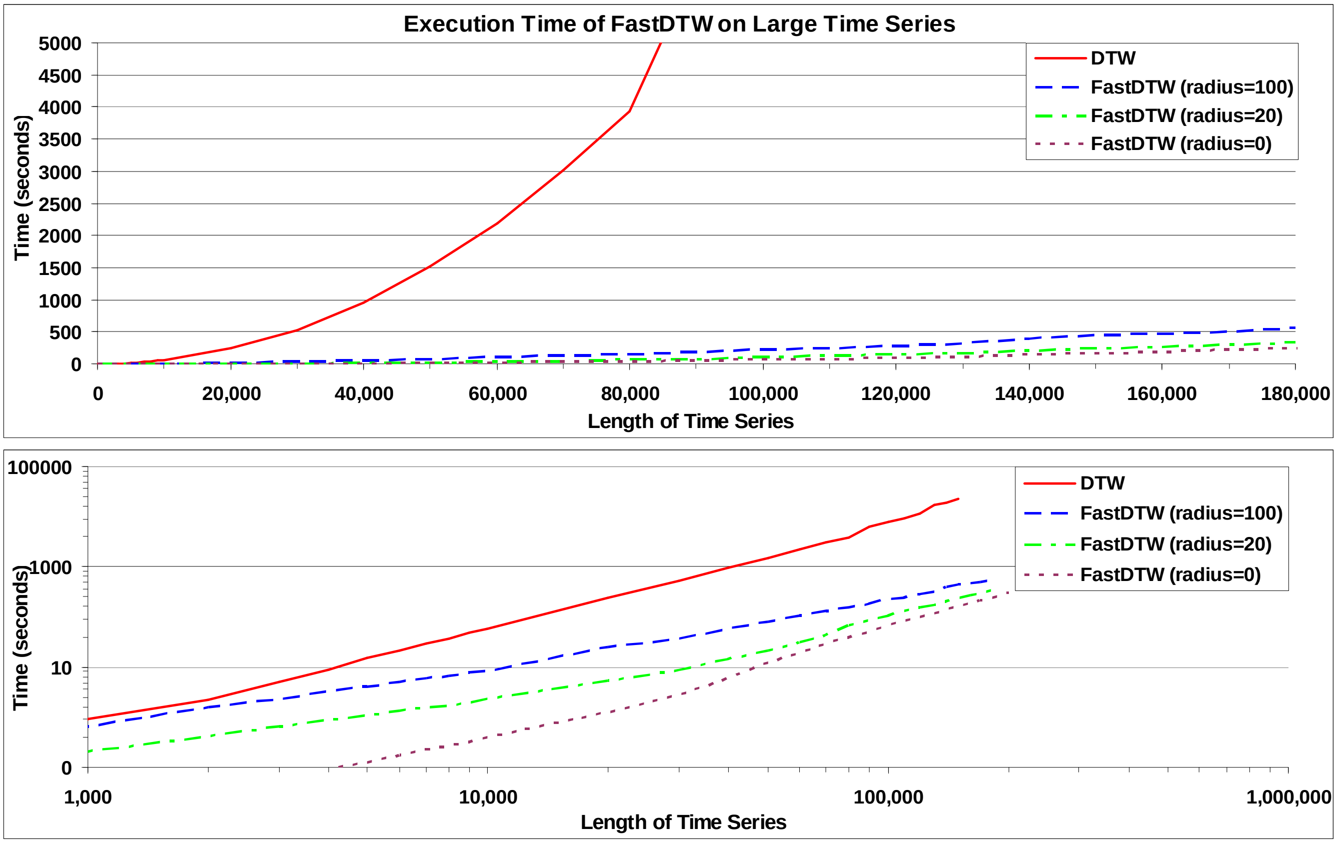

Figure 11. The eficiency of FastDTW and DTW on large time series. The top figure is scaled normally, and the bottom figure has log-log scaling.

The full results, using *radius* values of 0, 20, and 100 on time series ranging in length from 10 to 200,000 are shown in Figure 11. In Figure 11, the *y*-axis is the execution time and the *x*-axis is the length of the time series. The two graphs in Figure 11 are two views of the same data. The top graph is scaled normally, and the bottom graph has log-log scaling. Looking at the top graph, it is immediately obvious that the time complexity of DTW is much greater than that of FastDTW. DTW has an exponential curve, while all three FastDTW curves are approximately straight lines. In the log-scaled graph at the bottom of Figure 11, the three curves of FastDTW can be viewed more easily. The *radius* parameter increases the execution of FastDTW by a constant factor, which is why the three FastDTW lines seem to be converging on the log-scaled graph as length of the time series increases. The constant factor difference between them gets less significant as the length of the time series increases.

In Section 3.1, we proved theoretically that the FastDTW algorithm was O(*N*). Using the empirical data in Figure 11, the equation of the FastDTW curve with a radius of 100 is

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

This coefficient of the squared term is very small, and it seems like the linear term is the most significant term in the equation...which would empirically prove that the FastDTW algorithm is O(*N*). However, since the values for *x* are so large, the squared term actually dominates the equation when *x*>100,000. The reason for this slight sub-linearity in the algorithm occurs when the number of cells being filled in the search window will not fit into main memory, and must be saved to the disk. Writing the cells to the disk can be performed in linear time. However, when reading the cells from the random-access file to construct a warp path, reading individual 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. In other words, 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 the FastDTW algorithm, a linear and accurate approximation of dynamic time warping (DTW). FastDTW uses a multilevel approach that recursively projects a warp path from a coarser resolution to the current resolution and refines it. While the quadratic time and space complexity of DTW has limited its use to only the smallest time series data sets, FastDTW can be run on much larger data sets. FastDTW 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 showed that FastDTW has a linear time and space complexity. Expirical results have also shown 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 lowers the error to under 1%. FastDTW's accuracy was compared to two existing methods, Data Abstraction and Sakoe-Chiba Bands, and was found to be far more accurate than either approach when using small *radius* values. FastDTW's solutions also always approached zero error (optimal warp path) with smaller *radius* values than the other two methods. An additional contribution of this paper is demonstrating how to apply the refinement portion of the FastDTW algorithm to the Data Abstraction approximate DTW algorithm. Doing so increased the accuracy of Data Abstraction by more than 100-fold in our evaluation with a *radius* of only 10.

The main limitation of the FastDTW algorithm is that it is an approximate algorithm and is not guaranteed to find the optimal solution (although it very often does). If for some reason a problem requires optimal warp paths to be found. Future work will look into increasing the accuracy of FastDTW. Possibilities to increase the accuracy of FastDTW include changing the step size (magnitute of the resolution change) between resolutions and evaluating search algorithms to guide search during the refinement step rather than simple expanding the search window in both directions.

6. REFERENCES.

[1] Abdulla, W., D. Chow, and G. Sin, Cross-words reference template for DTW-based speech recognition systems, in Proc. IEEE TENCON, Bangalore, India, 2003.

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

[3] Gupta, L., D. Molfese, R. Tammana & P. Simos. Nonlinear Alignment and Averaging for Estimating the Evoked Potential. In *IEEE Transactions on Biomedical Engineering*, vol. 43, no. 4, pp. 346-356, 1996.

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

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

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

[7] Keogh, E. 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, 2002

[8] Keogh, E. & M. Pazzani. Derivative Dynamic Time Warping. In *Proc. of the First Intl. SIAM Intl. Conf. on Data Mining*, Chicago, Illinois, 2001.

[9] Keogh, E. & 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*, pp.285-289. Boston, Massachuseetts, 2000.

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

[11] Kruskall, J. & 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

[12] Ratanamahatana, C. & 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.

[13] Sakoe, H. & S. Chiba. (1978) Dynamic programming algorithm optimization for spoken word recognition. IEEE Trans. Acoustics, Speech, and Signal Proc., Vol. ASSP-26.

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

[15] Vlachos, M., G. Kollios, & D. Gunopulos. Discovering Similar Multidimensional Trajectories. In *Proc. 18th Intl. Conf. on Data Engineering*, pp. 673-684, San Jose, California, 2002.