AWarp: Fast Warping Distance for Sparse Time Series

Abdullah Mueen, Nikan Chavoshi, Noor Abu-El-Rub, Hossein Hamooni, Amanda Minnich
Department of Computer Science, University of New Mexico

***Abstract*— Dynamic Time Warping (DTW) distance has been effectively used in mining time series data in a multitude of domains. However, in its original formulation DTW is extremely inefficient in comparing long sparse time series, containing mostly zeros and some unevenly spaced non-zero observations. Original DTW distance does not take advantage of this sparsity, leading to redundant calculations and a prohibitively large computational cost for long time series.**

We derive a new time warping similarity measure (AWarp) for sparse time series that works on the run-length encoded representation of sparse time series. The complexity of AWarp is quadratic on the *number of observations* as opposed to the *range of time* of the time series. Therefore, AWarp can be several orders of magnitude faster than DTW on sparse time series. AWarp is *exact* for binary-valued time series and a *close approximation* of the original DTW distance for any-valued series. We discuss useful variants of AWarp: bounded (both upper and lower), constrained, and multidimensional. We show applications of AWarp to three data mining tasks including clustering, classification, and outlier detection, which are otherwise not feasible using classic DTW, while producing equivalent results. Potential areas of application include bot detection, human activity classification, and unusual review pattern mining.

I. Introduction

Time warping naturally appears in many domains, especially in the activities of humans and animals. For example, humans can produce the same motion or speech at a different pace and acceleration and have it still be recognizable. Time warping is also present in discrete action sequences. For example, Figure 1 shows the 24-hour time series of the front door statuses of two single-resident apartments. Each day shows a warped version of the unique schedule of the resident in that apartment. A simple hierarchical clustering of the data shows that the daily patterns of a person can be clustered well if we use Dynamic Time Warping (DTW) distance instead of the widely used Euclidean distance.

Dynamic Time Warping (DTW) is a distance measure that has been used in dozens of research works on mining equally sampled time series data [13]. However, new sensor technologies (both soft and hard) can capture a sequence of discrete events that forms a sparse time series (as in Figure 1). In its original form, DTW distance does not take advantage of this sparsity. For example, Twitter records discrete activities of more than 700 million users at a resolution of milliseconds. Comparing the activities of two users for a day at this resolution requires $86,400,000^2$ computations, which amounts to more than a day in an off-the-shelf machine. The number of activities performed by average users are on the order tens or hundreds. Clearly, the current amount of computation required to calculate DTW distance is excessive.

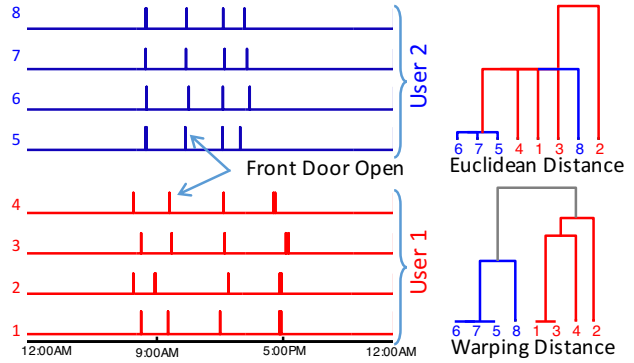

Fig. 1. (left) Day-long signals generated from the front doors of two single-resident apartments of two users. (right) Euclidean distance cannot capture the difference between the two users, while DTW distance can.

We develop a time warping distance measure, AWarp, for sparse time series data that works on run-length encoded time series. Run-length encoded time series are much shorter than their versions before encoding; for example, in Figure 1 the run-length encoded time series for instance 7 will have only eight numbers, as opposed to 86,400 observations for a day. AWarp is exact for binary-valued time series and closely approximates the DTW distance for any-valued time series. AWarp is extendable to constrained warping and multidimensional warping. We have shown three applications of AWarp in the important areas of bot discovery, human activity classification, and unusual review pattern discovery.

We give necessary background (Section II) on sparse time series and their various representations, and on Dynamic Time Warping. Next we describe the core AWarp algorithm and its variants in Section IV. We show performance analysis of the algorithm in Section V and demonstrate potential applications in Section VI. We conclude in Section VII.

II. Encoding Sparse Time Series

We first define time series and dynamic time warping distance (DTW). We then discuss sparse time series and run-length encoding and show a motivating example.

A. Definition

A *time series* is defined as a vector $T =< v_1, v_2, \ldots, v_n >$ of observations made at equal intervals. Most distance measures and mining algorithms are invariant to the absolute start time and sampling interval of the time series [11][25].

For two series $x = x_1, x_2, \ldots, x_n$ and $y = y_1, y_2, \ldots, y_m$ of length n and m, where $n > m$ without losing generality, the classic Dynamic Time Warping distance is defined as below.

2374-8486/16 $31.00 © 2016 IEEE
DOI 10.1109/ICDM.2016.21

$$DTW(x,y) = D(n,m)$$

$$D(i,j) = (x_i - y_j)^2 + \min \begin{cases} D(i-1,j) \\ D(i,j-1) \\ D(i-1,j-1) \end{cases}$$

$$D(0,0) = 0, \forall_{ij} D(i,0) = D(0,j) = \infty$$

We intentionally skip taking the square root of $D(n,m)$, as it does not change the relative ordering of pairs and makes it efficient for speedup techniques. A dynamic programming algorithm to populate the DTW matrix and calculate the DTW distance is well known. An example DTW matrix for two time series is given in Figure 2(a).

Constrained DTW distance is a variant that limits the the allowed time gap between two aligned observations. In effect, the DTW matrix is populated partially around the diagonal (readers can find details about DTW in many online resources such as Wikipedia and also in [13]).

B. Sparse Time Series and Representations

A time series is simply a sequence of observations made in temporal order. The phenomena that we observe can be continuous or discrete in time. For example, the temperature of a sea surface at specific point on earth is a continuous phenomenon. In contrast, the activities of a user on social media are discrete because the user can be inactive at times. When observing a discrete phenomena, a sparse time series is produced, which is the focus of this work.

A sparse time series has many more zero-valued observations than non-zero observations. We define the *sparsity factor*, s, of a time series as the ratio between the length of the time series and the number of non-zero observations. The higher the sparsity factor, the more sparse a time series is. Representing a sparse time series in the traditional vector format wastes significant amount of space. For example, the REFIT [15] datasets are stored in this format. A more optimal way to store sparse time series is as a sequence of time-value pairs.

Time-value Sequence: Each observation is stored as a (t,v) pair and a sparse time series is an ordered set $T_v = \{(t_i, v_i) | t_i < t_{i+1}, i = 1 \ldots n-1\}$. For example, the CASAS datasets [10] are represented in this format. This is the most common representation of sparse time series. **Example:** The time series $T =< 7,0,0,9,6,0,0,0,1 >$ can be represented equivalently as $T_v = \{(1,7),(4,9),(5,6),(9,1)\}$ if the start time is 1.

In this paper, we use a well known compression technique, *run-length encoding* [2], to represent sparse time series. We differ from the classic run-length encoding as we only encode the runs of zeros and leave the runs of non-zero observations as they are.

Length-Encoded series: Let us assume we have a time series T. A length-encoded time series is T_e where we replace a run of k zeros in T with a (k). Here we use the parenthesis to represent the duration of zeros. **Example:** For the same sparse time series, $T =< 7,0,0,9,6,0,0,0,1 >$, the length-encoded series is $T_e =< 7,(2),9,6,(3),1 >$.

We can also define length-encoded series in a rather complex way from the time-value sequence T_v as $T_e =< v_1, (t_2 - t_1 + 1), v_2, (t_3 - t_2 + 1), \ldots, v_{n-1}, (t_n - t_{n-1} + 1), v_n >$. In other words, we insert the duration between each pair of observations in between the observations to create a length-encoded series. From now on, we use simply use encoded series to denote length-encoded series.

Note that a time series of four observations, such as T_v, needs eight integers for storage in the time-value sequence representation. In traditional representation, T could require any number of integers larger or equal to eight to store the series because the lengths of the runs of zeros can arbitrarily vary in size. In an encoded series, T_e needs at most eight integers. Thus, for a fixed sparsity factor, the encoded series require the lowest amount of space.

Run-length encoding compresses a run of zeros by the length of the run. There is no better compression than just one number. In that sense, run-length encoded series are also *fully encoded series*. We can also define partially encoded series, which will be useful to calculate multidimensional DTW distance.

Partially encoded series: Given an encoded series T_e, a partially encoded series T_{pe} is an equivalent series where one or more of the runs of zeros are split into parts. **Example:** $T_{pe} =< 7,(2),9,6,(2),(1),1 >$ is a partially encoded series of T_e from the previous example. If we keep *splitting* the runs of zeros in a partially encoded series, we reach the same length as the traditional series, with zero being represented by (1) and no more possible splits.

If a time series starts with a run of zeros, we treat the first zero as an observation and encode the rest of the run. This ensures that an encoded series always starts with an observation, and not with a run of zeroes. Similarly, we ensure that the series ends with an observation. Since T_e and T_{pe} are equivalent, their DTW distances to any other series remain identical. The conversion between the three representations of sparse time series can be performed in time linear to the length of the time series.

C. Motivating Example

We now present an example to motivate AWarp. In Figure 2(a), we show two toy time series x and y of lengths 14 and 11, respectively. The DTW distance between the two time series is 1. The DTW matrix is a 14×11 matrix as shown in Figure 2(a). If we encode the time series x and y, the two time series shrink to X (length 8) and Y (length 5), respectively. The AWarp matrix calculated on these encoded time series is only of size 8×5 (shown in (b)). The AWarp distance is the same as the DTW distance, 1. The computation in each boxed sub-matrix of the DTW matrix is replaced by a one cell in the AWarp matrix. The value in the bottom-right corner of a sub-matrix is identical to the corresponding cell in the AWarp matrix. Note that a sub-matrix is not always a constant matrix with identical values. Some of the sub-matrices are monotonically increasing sequences. To complete the example, we also show the constrained AWarp matrix for a constraint window of size

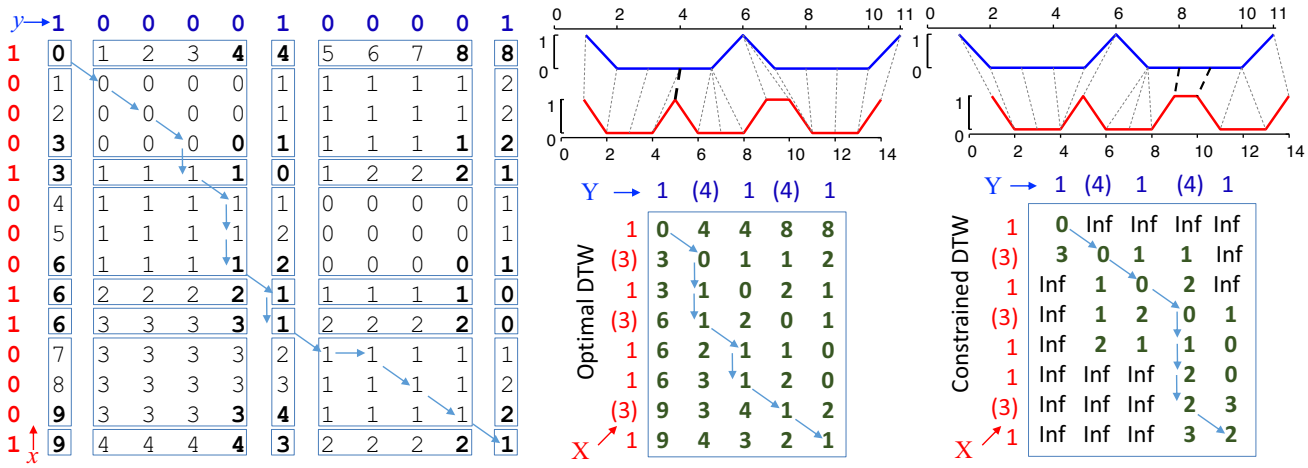

Fig. 2. (a) Two sparse time series x and y and their DTW matrix. (b) The AWarp matrix for their encoded versions, X and Y. (c) The AWarp matrix for a constraint window of size 5.

5 in Figure 2(c). The constrained warping distance is always larger than the optimal DTW distance. In this example, the constrained AWarp distance is 2, which is exactly the same as the constrained DTW distance under the same constraint window.

III. Related Work

Dynamic time warping is a long-studied algorithm in many research communities, including signal processing [7], speech recognition [19][12], data mining [14], and image processing [18]. We adopt warping distance for sparse time series. Although many human activity datasets are publicly available, warping-invariant mining has not been applied to sparse time series from discrete human activities (to the best of our knowledge). Our work is the first to exploit sparsity for time efficiency in warping-invariant mining.

Some works exploit other forms of sparsity in DTW calculations [4][24]. In [4], the authors reduce space complexity by approximating the distance; however, there is no reduction in time complexity. In contrast, our method reduces both time and space complexity with negligible difference in accuracy. In [24], the authors have not used the sparsity of the time series or the sparsity of the DTW matrix, rather sparsity is used when combining features that are independently calculated without using DTW. We claim our work as the first to calculate warping similarity on an encoded representation of sparse time series data.

A significant body of research exists on efficient DTW calculation [9][20][21]. In all of these work, calculation of one global DTW distance has a worst-case time complexity of $O(n^2)$, where n is the length of the time series and w is the warping window. AWarp has a worst-case complexity of $O(m^2)$, where m is the number of non-zero observations. This makes a significant difference in performance for sparse time series.

DTW-based similarity search in streaming or database settings has been made efficient by indexing [13], hybrid bounding [16], admissible pruning [6], and filter-and-refine [5] approaches. These approaches are equally applicable for sparse time series and can use AWarp, instead of DTW, for un-pruned distance comparisons. We leave it as a future work to adopt these techniques to perform similarity search under AWarp. In [8], the authors have shown that locally-relevant constraints learned from salient features of the comparing time series are better than a fixed constraint for the entire time series. We will evaluate this approach on constrained AWarp in future.

IV. AWarp Distance Measure

We start by describing the AWarp algorithm for simple binary valued series. We then relax this simplification and discuss the general case of any-valued time series. Finally we show the constrained and multidimensional versions of AWarp.

A. Binary-valued Series

Algorithm 2 is the AWarp distance function for run-length encoded time series. The inputs to the algorithm are two run-length encoded time series. The algorithm fills in a matrix D of size $l_x \times l_y$ in the same way as the DTW algorithm. Here l_x and l_y are the lengths of the two encoded series x and y, respectively. The algorithm has two loops in lines 4 and 5 that go over all the cells of the AWarp matrix. The algorithm calculates three costs for a cell based on three other cells: (*diagonal*, *left*, and *top*) relative to the cell being populated. Finally, in line 11, the algorithm takes the minimum of the costs as per the definition of DTW.

While calculating the cost of a pair of values x_i and y_j, Algorithm 1 treats various mutually exclusive cases differently based on the values of x_i and y_j (i.e. a real observation or a run of zeros), and the direction of the cell (i.e. $D_{i-1,j-1}, D_{i-1,j}$ or $D_{i,j-1}$) to which the cost will be added to. The following facts describe the cases in $UBCosts$, one by one.

Observation 1: AWarp (Algorithm 2) is identical to DTW for any traditional time series, although it is designed for encoded series.

It is a trivial observation. If x and y are traditional vectors, there is no run of zeros in x and y by definition. Therefore, the $UBCosts$ algorithm must always execute the first case in line 1, which is the squared error between the values, as in the definition of DTW.

Observation 2: AWarp distance of encoded binary-valued series is identical to the DTW distance of their traditional representations.

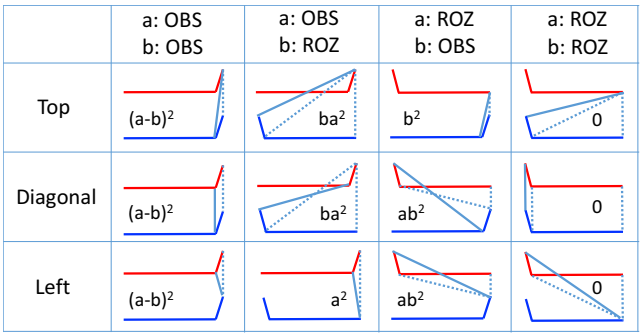

Fig. 3. Twelve cases covered by the Algorithm 1. OBS: observation, ROZ: run of zeros.

Algorithm 1 describes the cases we need to treat separately for binary-valued encoded series. The case in line 1 is the trivial case when both of the inputs a and b are real observations. The value v is simply the squared error. In line 2, we have one observation (a=1) and one run of zeros (b). There can be two inner cases: the run of zeros has already been aligned ($left$) or it is being aligned for the first time ($right$ or $diagonal$). If the run of zeros is being aligned for the first time, we have no choice other than aligning all of the zeros with some real observation(s). In the case of a binary-valued series, the real observation(s) are always identical and their values are one, no matter where they are located. Thus the term ba^2 aligns the zeros. If the run of zeros has already been aligned to previous value(s) of the real observation a, we just align a with the last zero of the run, hence the term $a^2 = 1$. The case in line 4 is the mirror of the case in line 2. The default case in line 6 is triggered when both a and b are runs of zeros, which can only result into a distance of zero. In Figure 3, we show twelve cases, which are all of the possible cases in binary-valued time series, and we illustrate how $UBCosts$ calculates the optimal alignment. The solid lines (aligning the red and blue time series) represent the so-far-alignment, and the dotted lines show the new alignment for which $UBCosts$ is calculating the cost.

As shown in Figure 2, if we take the DTW matrix of the traditional binary-valued time series and remove the rows and columns corresponding to zeros that are followed by other zeros, we obtain the matrix calculated by the AWarp algorithm.

B. Any-valued Series

As we have described the exactness of AWarp in case of binary-valued time series, the natural question is if the exactness holds for any-valued time series. The answer is no.

Algorithm 1 $UBCosts(a, b, c)$

Require: $a \leftarrow$ an observation, $b \leftarrow$ another observation, $c \leftarrow$ a case identifier
Ensure: Output the distance value v between a and b
1: case: a and b are observations: $v \leftarrow (a-b)^2$
2: case: a is an observation and b is a run of zeros:
3: **if** $c = left$ $v \leftarrow a^2$ **else** $v \leftarrow ba^2$
4: case: a is a run of zeros and b is an observation:
5: **if** $c = top$ $v \leftarrow b^2$ **else** $v \leftarrow ab^2$
6: case $default$: $v \leftarrow 0$
7: **return** v

Algorithm 2 $AWarp(x, y)$

Require: $x, y \leftarrow$ two encoded time series for comparison
Ensure: Output warping distance between x and y
1: $l_x \leftarrow length(x), l_y \leftarrow length(y)$
2: $D(0 : l_x, 0 : l_y) \leftarrow \infty$
3: $D_{0,0} \leftarrow 0$
4: **for** $i \leftarrow 1$ to l_x **do**
5: **for** $j \leftarrow 1$ to l_y **do**
6: $a_d \leftarrow D_{i-1,j-1} + UBCosts(x_i, y_j, diagonal)$
7: $a_l \leftarrow D_{i,j-1} + UBCosts(x_i, y_j, top)$
8: $a_t \leftarrow D_{i-1,j} + UBCosts(x_i, y_j, left)$
9: $D_{i,j} \leftarrow min(a_d, a_l, a_t)$
10: **return** D_{l_x,l_y}

Observation 3: AWarp on any-valued encoded series approximates the DTW distance between their traditional representations.

We first discuss why AWarp is not exact for any-valued time series. Although the encoded representation is not lossy, the optimal alignment, which is similar to classic DTW, is not possible for any-valued encoded series. This is because run-length encoding treats all zeros as identical, while an optimal warping alignment may treat zeros in the same run differently.

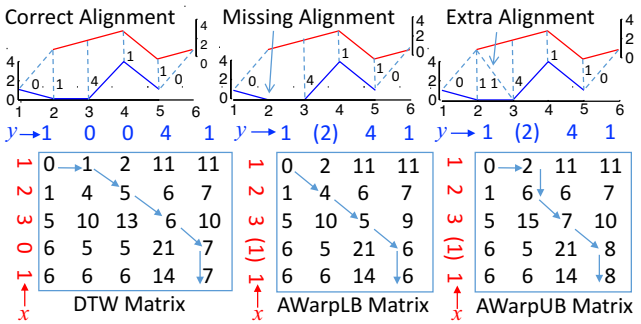

Fig. 4. An example demonstrating that optimal alignment in the encoded representation is not possible.

Example: In Figure 4, two time series $x =< 1, 2, 3, 0, 1 >$ and $y =< 1, 0, 0, 4, 1 >$ are shown in red and blue, respectively. Note that these time series contain various positive observations as opposed to just one. The optimal DTW aligns

the first zero of y with the first one of x and the second zero of y is aligned with the two of x. Such a scenario of aligning part of a run of zeros to one observation and the remaining part of the run to another observation is not possible in the encoded representation, where we treat all the zeros as one entity. If we encode x and y and calculate the AWarp distance, the $UBCosts$ function aligns the run of two zeros of y to the first one of x. Therefore, AWarp accumulates a higher distance than the optimal DTW and forms an upper-bounding function of the DTW distance measure. Similarly, if in the $UBCosts$ algorithm, we skipped aligning the run of two zeros of y with the first one of x, AWarp would have accumulated a smaller distance than the optimal DTW and formed a lower-bounding function of the DTW distance.

Algorithm 3 $LBCosts(a, b, c)$

Require: $a \leftarrow$ an observation, $b \leftarrow$ another observation, $c \leftarrow$ a case identifier

Ensure: Output the distance value v between a and b

1: case: a and b are observations: $v \leftarrow (a-b)^2$
2: case: a is an observation and b is a run of zeros:
3: **if** $c = top$ $v \leftarrow ba^2$ **else** $v \leftarrow a^2$
4: case: a is a run of zeros and b is an observation:
5: **if** $c = left$ $v \leftarrow ab^2$ **else** $v \leftarrow b^2$
6: case $default$: $v \leftarrow 0$
7: **return** v

We define the lower-bounding cases in Algorithm 3, where the term ba^2 is applied to only the top case and the term ab^2 is applied to only the $left$ case. The difference between the $UBCosts$ and $LBCosts$ is that the $diagonal$ cost in the former is always equal or larger (ab^2 or ba^2) than the latter (b^2 or a^2). From now on, we will use AWarp_UB and AWarp interchangeably to refer to Algorithm 2 and AWarp_LB to the refer to the same algorithm where $UBCosts$ are replaced with $LBCosts$.

At this point, the most important question is: *how good are these bounding functions?* To test them, we generate a comprehensive set of synthetic datasets in the following way. Each dataset has a sparsity factor from the following: 2, 4, 8, 12, 16, 24, 32. Each dataset is associated with a distribution (uniform, normal, binomial and exponential) to generate random numbers from. To generate a dataset, we create 1000 pairs of zero vectors of length 128. We insert random values between one and five in the zero vectors at random locations drawn from the associated distribution. The number of values that are inserted depends on the associated sparsity factor.

For each pair of time series in a dataset, we calculate the upper bound (i.e. AWarp), the lower bound as described above, and the DTW distance in the traditional representation. We calculate the percentage of exact and approximate matches (up to 5% error) between the bounds and DTW distances. The results are shown in Figure 5. AWarp_UB, approximately 90% of the times, is within 5% of the true distance value. The accuracy converges to 100% as data becomes sparser. These results empirically support that AWarp distance for sparse time series in the encoded form is almost identical to the DTW distance in the traditional form.

The cup-shapes of the approximate matches in Figure 5 can be explained. For low sparsity factor, the number and length of the runs of zeros are smaller than that when sparsity factor is high. Thus, for low sparsity factor, high accuracy is achieved by exploiting the observation 1.

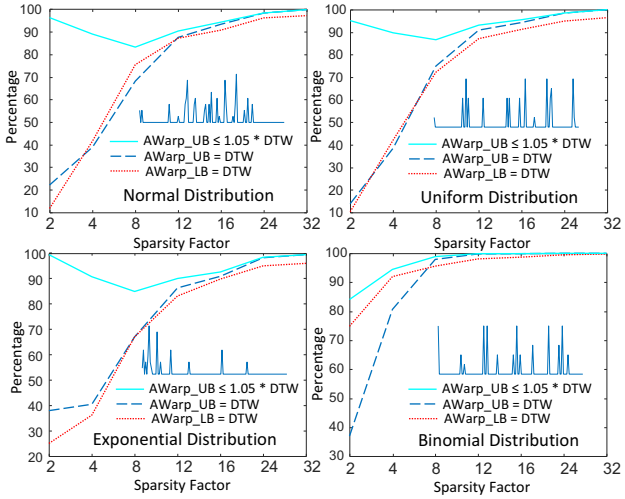

Fig. 5. AWarp_LB and AWarp_UB on encoded series with respect to DTW on vector representation. On average, 90% of the times the upper bound is within 5% of the true distance. Sample time series are shown inside.

Although AWarp is not exactly identical to DTW, there is a simple way to test if AWarp distance is exact. we can calculate Awarp_LB and check if it is equal to AWarp. If they are the same, the distance must be exactly equal to the DTW distance. Thus, we can validate the exactness without calculating the expensive DTW distance by just two AWarp calculations on encoded series, and use AWarp as a pre-processing step ahead of the exact DTW calculation on sparse data.

C. Invariance to Partial Encoding

As mentioned before, a partially-encoded series is a longer version of an encoded series where a run of zeros can follow another run of zeros. Let us informally define $order$ of partially encoded series as the number of zeros that have been encoded.

Observation 4: AWarp is invariant to the order of partial encoding.

Let us first give an example. If $x =< 7, (2), 9, 6, (3), 1 >$ is an encoded series and $x' =< 7, (2), 9, 6, (2), (1), 1 >$ is a partially encoded series of x, then the above fact ensures AWarp(x,y) = AWarp(x',y). This observation can be easily explained by the $UBCosts$ algorithm, which solely depends on the two values, a and b, and is not impacted by prior or later values in the series. Since x and x' are equivalent series, the distance values must be identical. Optimality in substructures is a classic property of dynamic programming. This fact is simply an alternative description of the optimal

substructure of the AWarp algorithm that we will exploit in the multidimensional version.

AWarp(x',y') is always closer to the DTW distance on traditional representations than AWarp(x,y), where x' and y' are partial encodings of x and y, respectively. The reason is that the more runs of zeros are split, the closer the partial encoding is to the traditional representation. To test this statement, we define an operation, *split*, on an encoded series that splits every run of two or more zeros into half. If we iteratively split an encoded series, the series is eventually converted to the traditional version. The impact of such iterative splits on exactness is shown in the Figure 6(right). As we split more, the error decreases and the exactness increases.

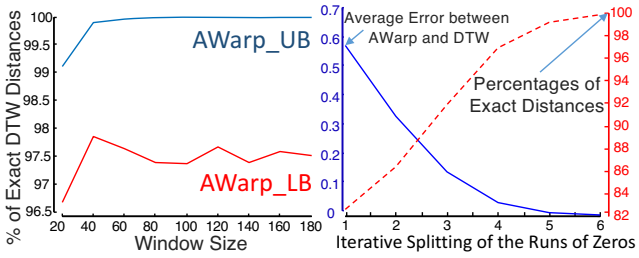

Fig. 6. (left) The exactness of constrained AWarp_LB and AWarp_UB for various windows. (right) The error and exactness of partially encoded representation as we split runs of zeros into halves iteratively.

D. Multidimensional Warping

We have so far discussed the one dimensional algorithms for calculating AWarp. We consider the multidimensional extension of AWarp using approaches similar to those developed for traditional DTW in [23]. There are three general ways to extend DTW to multidimensional time series:

Independent: Calculate the individual optimal distances and sum them after normalization by the path length.

Aggregate: Sum up the individual dimensions into one superposed time series and encode them to calculate the AWarp distance using Algorithm 2.

Dependent: Calculate the global optimal distance assuming that all of the observations at a timestamp must be aligned together to the observations of another timestamp.

Extending AWarp to multidimensional-encoded time series is trivial for the independent scenario. In the aggregate scenario, we sum up the individual dimensions. A simple way to sum two encoded sparse time series is to convert them to traditional time series, add the series, and encode them back to obtain the aggregated time series. It is even more simple to aggregate two time-value sequences. We concatenate the two sequences, sort the concatenated sequence based on time, and add observations which appear at the same time. The time cost is linear in both the cases.

In the dependent scenario, it is non-trivial to calculate the global optimal distance. The recursive step of the dependent version of multidimensional warping distance is given below.

$$D(i,j) = \sum_{k=1}^{d}(X_{ik} - Y_{jk})^2 + \min \begin{cases} D(i-1,j) \\ D(i,j-1) \\ D(i-1,j-1) \end{cases}$$

The above definition of the multidimensional DTW does not work on encoded series directly. For example, if a two-dimensional series is $(x_1, x_2) = (< 1,0,0,-1,0,0,0,1 >, < 1,0,0,0,0,1,0,1 >)$, then the encoded representation is $(x_1, x_2) = (< 1,(2),-1,(3),1 >, < 1,(4),1,(1),1 >)$. Clearly, the locations of real observations are not aligned in x_1 and x_2. In order to convert them to a workable representation, we partially encode x and y in a way that runs of zeros always end at an observation in one of the dimensions. For example, $(x'_1, x'_2) = (< 1,(2),-1,(1),(1),(1),1 >, < 1,(2),(1),(1),1,(1),1 >)$ is an equivalent representation of x and y where the values are time aligned. On sequences of different lengths, aligning them requires managing the ends carefully. In [3], we provide an algorithm which shows how to align two dimensions for multidimensional AWarp. When there are more than two dimensions, the process will be to align pairs of dimensions until no change is needed.

The AWarp algorithm will need to calculate the sum of $UBCosts$ over all of the dimensions in lines 8-10 to accommodate the recursion specified above. We skip the details due to lack of space and will explain in detail in an extended version of this paper. In [23], the authors have shown that a combination of the dependent and independent algorithms can beat both of them individually. We will consider such extensions for multidimensional AWarp in future.

E. Constrained Warping

It is widely accepted that constraining the warping between two time series in a user-given window not only helps data mining algorithms to run more quickly, but also enforces physical laws in the matching process [19][13][16]. Figure 2(right) shows an example of a constrained (Sakoe-Chiba band) AWarp matrix. The constrained AWarp algorithm for encoded time series is shown in Algorithm 4. This algorithm is identical to Algorithm 2 except the lines 6-9. In line 6, the absolute difference between the timestamps of x_i and y_j is calculated. We assume that the timestamp of every observation in the encoded series is available to us. It takes linear time to calculate these absolute timestamps if we know t_0, and the overhead is minimal compared to the overall computational cost.

The condition on line 7 ensures that if $tx_i > ty_j + w$ then $tx_{i-1} > ty_j + w$ must be true to set a cell to infinity. If $tx_i > ty_j + w$ and $tx_{i-1} < ty_j + w$, then x_i is a run of zeros, which contains the timestamp $ty_j + w$ (boundary of the Sakoe-Chiba band). As mentioned before, AWarp cannot align a run of zeros in parts, therefore, when a run of zeros contains the boundary of Sakoe-Chiba band, we extend the band until the next observation after the run of zeros. This forces us to calculate some extra cells that would have been infinity if we used the traditional representation. However, constrained

AWarp ensures that no cell within the band is skipped, as Line 7 also checks the mirror case for $ty_j > tx_i + w$.

In Figure 6(left), we show the correctness of the $AWarp\_LB$ and $AWarp\_UB$ algorithms as we increase constraint window size. We generate a time series of length 200 with 50% sparsity and normally distributed observations. We calculate 10,000 random distances using Algorithm 4 and check what percentage of the distances match the exact constrained DTW distance. We find that the accuracy increases as the window grows. $AWarp\_UB$ converges quickly to 100%, while $AWarp\_LB$ show some variance. Note that the exactness is always above 96.5% for $AWarp\_LB$ and above 99% for $AWarp\_UB$.

Algorithm 4 $Constrained\_AWarp(x, y, w)$

Require: $x \leftarrow$ a sequence of timestamps, $y \leftarrow$ another sequence of timestamps
Ensure: Output warping distance between the two sequences x and y

1: $l_x \leftarrow length(x), l_y \leftarrow length(y)$
2: $D(0 : l_x, 0 : l_y) \leftarrow \infty$
3: $D_{0,0} \leftarrow 0$
4: **for** $i \leftarrow 1$ to l_x **do**
5: **for** $j \leftarrow 1$ to l_y **do**
6: $gap \leftarrow |tx_i - ty_j|$
7: **if** $gap > w$ **and** $(ty_{j-1} - tx_i > w$ **or** $tx_{i-1} - ty_j > w)$ **then**
8: $D_{i,j} \leftarrow \infty$
9: **else**
10: $a_d \leftarrow D_{i-1,j-1} + UBCosts(x_i, y_j, diagonal)$
11: $a_l \leftarrow D_{i,j-1} + UBCosts(x_i, y_j, left)$
12: $a_t \leftarrow D_{i-1,j} + UBCosts(x_i, y_j, top)$
13: $D_{i,j} \leftarrow min(a_d, a_l, a_t)$
14: **return** D_{l_x,l_y}

V. Experiments

Reproducibility Statement: We share *everything* related to this paper in our anonymous repository [3]. We share code for AWarp in two languages (C++ and MATLAB), presentation slides, datasets, experimental results, additional experiments, and additional data.

Dataset	Instances	Length	Resolution	Duration
TA	4,170	36,799	1 Second	One Day
AR	3,755	1,334	1 Day	Years
HA	1,628	288	5 Minutes	One Day
PW	3,089	288	5 Minutes	One Day

TABLE I. Dataset summary

Datasets: We use four real datasets from diverse domains to demonstrate the scalability of AWarp. The datasets are: Twitter user activity time series (TA), app review time series (AR), human activity time series (HA) and power usage time series (PW). In Table I, we briefly describe the datasets. The resolutions of the datasets are very carefully chosen to be relevant for the respective domains. In human behavioral activity and electric power usage, a resolution of five minutes is reasonable. In online reviewing activity, a resolution of a day is enough. In Twitter activity time series, a resolution of a second is required because many actions in Twitter only need mouse clicks (e.g. follow, retweet). Detailed descriptions of the datasets are given in the subsequent application sections.

1) Speedups: We generate 100,000 pairs of sparse time series for various sparsity factors and lengths where the activities are uniformly distributed. We calculate the average speedup achieved by AWarp over DTW for these pairs and show the results in Figure 7.

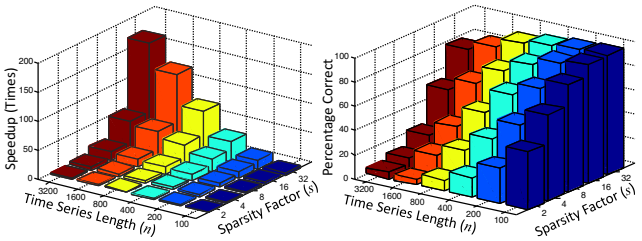

Fig. 7. Speed and accuracy with respect to the sparsity and size of the datasets.

As data becomes more sparse, speedup increases. As data gets larger, the speedup increases even more.This is an incredible feature of AWarp that can *enable applications of warping distance to datasets where DTW cannot run on the uncompressed sparse time series.*

2) Tractability: A valid question at this point is: are the sizes and sparsity factors of real datasets large enough to require a method like AWarp? We first validate the major motivation of AWarp. We test the speed of AWarp by comparing the running time of AWarp in the encoded representation with that of DTW in the traditional representation. The gain in speed naturally depends on the resolution of the time series. The higher the resolution, the more sparse the data becomes and the more speedup we gain. We use reasonable resolutions for all of our datasets as shown in the Table I.

We perform *all-pair distance calculations* on each of the datasets using DTW and AWarp. All-pair distance calculations is a basic operation for many data mining task including: hierarchical clustering, outlier detection, and nearest neighbor classification. We record the speedup and the respective sparsity factors for four real datasets in Table V-.2. The sparsity factors in our real datasets are large enough to extract at least 2×, and up to 557×, speedup. In each of these domains, the data owners (e.g. Twitter, Google Play) have several orders of magnitude more data than what we use for this experiment. AWarp will be very useful at that scale for performing many basic data mining tasks under warping similarity. We describe four such data mining tasks in the next section.

3) Comparison with a Baseline: As described earlier, the purpose of AWarp is to calculate the warping similarity of sparse time series much more quickly than the classic dynamic time warping algorithm while retaining the accuracy of a warping distance measure. There are other methods (e.g.

Dataset	s	DTW	AWarp	SpeedUp
TwitterActivity	746	180 hrs	0.3 hr	557×
AppReviews	3	46 hrs	21 hrs	2×
HumanActivity	42	907 Sec	34 Sec	27×
PowerUsage	28	1170 Sec	40 Sec	29×

TABLE II. Speedup achieved on real datasets.

FastDTW) that achieve the same for arbitrary time series data, as opposed to sparse time series. We compare AWarp to FastDTW [20] on 1000 pairs of sparse time series for different values of the *radius* parameter. We measure total execution times and percentages of exact distances produced by FastDTW and show the results in Figure 8. On the same chart, we point to the worst and median accuracy achieved by AWarp (implemented in MATLAB) and the corresponding execution time for various sparsity factors. Note that AWarp has no input parameters. Also note that FastDTW does not vary on sparsity. For completeness, we point to the timings of two classic DTW implementations. FastDTW (Python) is completely dominated by our implementations. We show a hypothetical 10× accelerated curve for FastDTW, which is also dominated by our implementations of AWarp and DTW.

Dozens of techniques are available to speedup similarity search [13], subsequence search [17], and indexing time series [22] data. These techniques are equally applicable to sparse time series and can benefit from AWarp's speedup just by replacing DTW with AWarp when calculating true distances to eliminate false positives. Comparing AWarp, DTW, and FastDTW in searching or indexing algorithms is out of scope of this work.

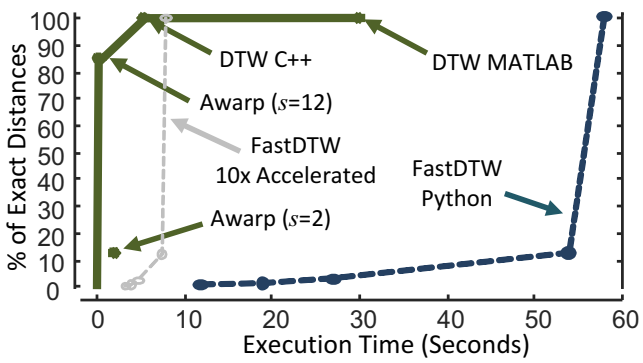

Fig. 8. Speed accuracy trade-off for various methods and implementations.

VI. Data Mining Applications

AWarp is a distance measure that nearly optimally aligns two discrete time series much more quickly than DTW aligns them in their traditional representation. However, this work needs to be justified by showing the utility of this speedup in real data mining tasks. In this section, we show four cases of important data mining tasks that *require time warping* and could not have been performed using time warping distance functions without the speedup provided by AWarp.

A. Bot Discovery in Twitter

We evaluate the performance of AWarp for clustering the Twitter activities of thousands of users. We assemble a dataset of every activity, including tweet, retweet, follow, unfollow, delete, and many others from 4,170 randomly chosen users for a day. We form activity time series for each of the users at a resolution of seconds (the data is available at milliseconds resolution).

Activity time series can be very useful for finding surprisingly correlated user groups that are mostly bot operated. To find such correlated user groups, we hierarchically cluster the users based on their AWarp distances. We use the single linkage technique and a threshold of 1 to create the clusters.

We find ten clusters that are very dense groups of ten or more users with highly synchronous activities. Several of these clusters can be further merged to form four semantically coherent clusters. One of the clusters *was* spreading pornographic content and is now mostly suspended by Twitter. Another cluster is spreading news, videos, and images about Selena Gomez (`wedselena13`, `wedselena`, `wedselena12`). The remaining two clusters were spreading identical content in two specific languages: Portuguese (`patetamos`, `IndiretasMusica`, `LoucoDeVodka`) and Malaysian (`elzmn01`, `_ItSy4mimi`, `zazaizzaty96`).

We show some of the activity time series from the cluster of Portuguese language in Figure 9(left). The time series show arbitrary shifts in tweet timestamps because of queuing delay, transmission delay, tweet registration delay, geographically separated data centers, and many other reasons. Such unstructured delay between synchronous tweets breaks Euclidean distance- and lagged Euclidean distance-based methods and prevents this bot group from being detected and suspended. Since AWarp is two orders of magnitude faster on Twitter data, we could perform the clustering under warping distance and discover such a cluster.

B. Behavioral Classification

We evaluate the classification performance of AWarp in a real-world setting. We use two human activity datasets (HH102 and HH104) from the WSU CASAS repository [10]. Each dataset is from a single-resident apartment recording the activities (e.g. door open, light on, etc.) of the resident. The datasets are partially annotated by labeling the beginning and end of some day-to-day activities, such as toilet, dress, sleep, cook, leave_home, etc. Instead of using the annotations to classify the activities, we ask an alternate question: *can we identify a person based on the status (e.g. opened or closed) of the front door of his apartment?* We pick the daily time series of the front door of the two apartments for over two years and create a balanced two-class classification problem of 1,628 instances of daily time series of length 288 (i.e. one observation every five minutes). A sample of the dataset is shown in the Figure 1.

We use a 1-NN classifier under Euclidean distance, DTW distance (global and constrained), and our proposed AWarp

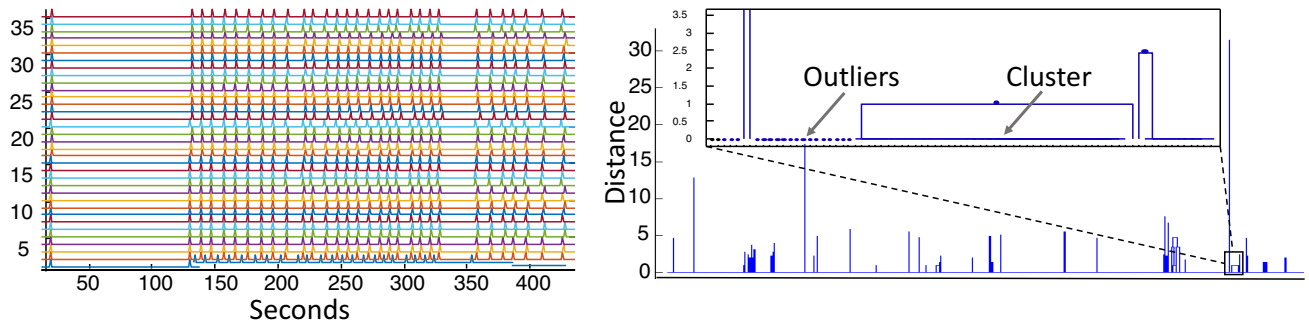

Fig. 9. (Left) Time series of a cluster of 35 bots. Each spike is one tweet. Note the warping in time axis. (Right) Dendrogram of the Twitter accounts using constrained (60 sec) AWarp. Most of the random users are outliers and several clusters of bots are formed.

distance (global and constrained). We evaluate the leave-one-out accuracy for each of these classifiers (see Table VI-B).

Euclidean	DTW	DTW_100	AWarp	AWarp_100
59.89%	62.71%	78.19%	76.78%	**78.50%**

TABLE III. Accuracies of different distance functions

It is interesting to note that there is a big gap between the accuracy of global DTW distance (62.71%) and the accuracy of the global AWarp distance (78.19%). Although global DTW finds *the* optimal alignment between the two series, AWarp penalizes a run of zeros being aligned with some real observations more than DTW does. The difference goes away when we use constrained versions of both of the measures with 100-minutes widows. Because long runs of zeros are broken into at most 100 minute runs, the difference between the global versions is reduced.

Irrespective of the difference noted above, a 1-NN classification using AWarp is **26**× faster than the DTW based classifier. This is a substantial difference for large datasets. We estimate that if we use all of the fourteen CASAS datasets of single-resident apartments, it would take 50 minutes to perform these experiments using AWarp, versus 23 hours using a DTW-based classifier.

C. Power Usage Classification

We also evaluate the performance of AWarp on a dataset of the power usage of appliances from two different houses. This dataset has been collected from [15]. Instead of considering all the appliances, we first consider only the power usage of the dishwasher appliance. Typically a dishwasher consumes more than 2000 watts at regular operation. We discretize the power usage time series to on-off time series at a resolution of five minutes. In total we have 500 days of on-off time series for the dishwashers. The two classes have 214 and 286 instances of days. These data are very sparse because dishwashers are not often in use. We consider classifying households by using their dishwashing pattern.

We use a 1-NN classifier under Euclidean distance, DTW distance (global and constrained), and our proposed AWarp distance (global and constrained). We evaluate the leave-one-out accuracy for each of these classifiers and report the results in Table VI-C.

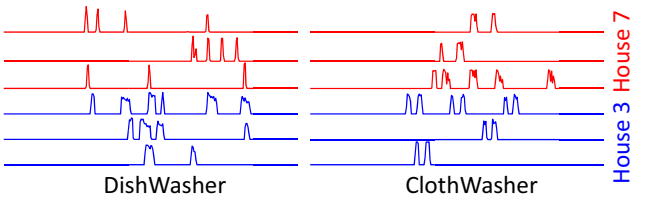

Fig. 10. Multidimensional power usage data from two households. Each time series is 1 day long at 5 minutes resolution starting at midnight. There is neither a fixed schedule nor a fixed load to these appliances.

	Eucl.	DTW	DTW_{1h}	AWarp	$AWarp_{1h}$
DW	79.56%	82.16%	76.95%	**83.57%**	77.15%
CW	81.96%	**87.58%**	82.77%	85.37%	81.16%
Both	82.16%	**88.98%**	85.77%	87.58%	71.34%

TABLE IV. Accuracy of different distance functions.

We also evaluate the classification accuracy of the same two houses based on the power usage of washing machines. We finally evaluate the accuracy considering both of the appliances together using the multidimensional extension of AWarp. In all three cases, global DTW or AWarp has the highest accuracy compared to constrained DTW, constrained AWarp and Euclidean distances. To perform a leave-one-out cross-validation, DTW took 4.5 hours while AWarp took 9 minutes with a tiny reduction in accuracy of 1.4%.

D. Unusual Review Pattern Discovery

We collect a dataset of app reviews from the Google Play Marketplace. This dataset contains the review time series for 3,755 mobile apps. To form review time series, we collect the number of reviews an app receives in a day since the beginning of data availability. The time series are therefore of varying lengths, with an average length of 1,334 days.

We perform discord discovery [26] on these data to identify the most anomalous review time series. The discord is the object in a dataset whose nearest neighbor is the farthest among all other nearest neighbors. We use AWarp as a distance measure to identify the discord. We find a pair of apps that are "far" from every other app while they are reasonably similar to each other. These apps are `com.facebook.katana` and `com.supercell.clashofclans`, which are two of the most popular apps in the Google Play Marketplace [1]. These

apps have received more than 20 million reviews each and they receive several thousands of reviews every day, which is much greater than the average number of reviews an app receives in the store.

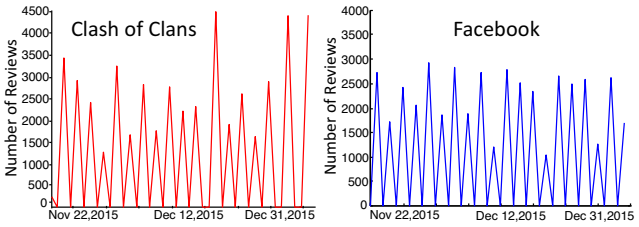

Fig. 11. Review time series found as outliers illustrate the capacity hit and subsequent two day cycle in the data collection system.

However, the success of AWarp is not catching the popular apps, which can easily be found in Wikipedia, but in efficiently identifying anomalous patterns. The patterns that cause AWarp to detect these two apps as outliers are shown in Figure 11. These pattern show that the apps receive thousands of reviews in one day and do not receive any on another day, which is an impossible scenario. The data collection system has a dynamic limit on the number of reviews it can collect and the system works in a two-day cycle. If an app is highly popular, the number of reviews it receives in a day exceeds the dynamic limit. For the two outlier apps, the limit is exceeded every day and the collection system gets reviews written in one day every two days, which is why the pattern appears. Thus, the outliers represent the overloaded scenarios of the data collection system.

VII. Conclusion

The goal of our work is to develop a time warping distance measure for sparse time series to exploit sparsity for efficiency. We develop AWarp, which is orders of magnitude faster than DTW and calculates a close approximation of DTW, if not a more accurate measure in some cases, such as in human activity datasets. We show applications of AWarp to four domains where DTW is unusable and AWarp can produce interesting results. We discover new bot behavior in Twitter, and we classify human activity much more quickly than with DTW-based classifiers.

VIII. Funding Statement

This work was supported by the NSF CCF Grant No. 1527127 and the NSF Graduate Research Fellowship under Grant No. DGE-0237002.

References

[1] List of most downloaded android applications. https://en.wikipedia.org/wiki/List_of_most_downloaded_Android_applications.

[2] Run-length encoding. https://en.wikipedia.org/wiki/Run-length_encoding.

[3] Supporting webpage containing paper, slides, code, and sample datasets. http://www.cs.unm.edu/~mueen/Projects/AWarp/.

[4] G. Al-Naymat, S. Chawla, and J. Taheri. SparseDTW: A Novel Approach to Speed up Dynamic Time Warping. page 17, Jan. 2012.

[5] I. Assent, M. Wichterich, R. Krieger, H. Kremer, and T. Seidl. Anticipatory DTW for Efficient Similarity Search in Time Series Databases. *Journal Proceedings of the VLDB Endowment*, 2(1):826–837, Aug. 2009.

[6] N. Begum, L. Ulanova, J. Wang, and E. Keogh. Accelerating Dynamic Time Warping Clustering with a Novel Admissible Pruning Strategy. In *Proceedings of the 21th ACM SIGKDD Conference - KDD '15*, pages 49–58, New York, New York, USA, Aug. 2015. ACM Press.

[7] N. Boulgouris, K. Plataniotis, and D. Hatzinakos. Gait recognition using dynamic time warping. In *IEEE 6th Workshop on Multimedia Signal Processing, 2004.*, pages 263–266. IEEE, 2004.

[8] K. S. Candan, R. Rossini, M. L. Sapino, and X. Wang. sDTW: Computing DTW Distances using Locally Relevant Constraints based on Salient Feature Alignments. *PVLDB*, 5(11):1519–1530, 2012.

[9] S. Chu, E. Keogh, D. Hart, and M. Pazzani. *Iterative Deepening Dynamic Time Warping for Time Series*, chapter 12, pages 195–212. 2002.

[10] D. J. Cook, A. S. Crandall, B. L. Thomas, and N. C. Krishnan. CASAS: A Smart Home in a Box. *Computer*, 46(7):62–69, July 2013.

[11] C. Faloutsos and C. Faloutsos. Fast subsequence matching in time-series databases. *ACM SIGMOD Record*, 23(2):419–429, 1994.

[12] H. Hamooni and A. Mueen. Dual-domain Hierarchical Classification of Phonetic Time Series. In *ICDM 2014*, ICDM, 2014.

[13] E. Keogh. Exact indexing of dynamic time warping. In *Proceedings of the 28th international conference on Very Large Data Bases*, VLDB '02, pages 406–417, 2002.

[14] E. J. Keogh and M. J. Pazzani. Scaling up dynamic time warping for datamining applications. In *Proceedings of the sixth ACM SIGKDD Conference - KDD '00*, pages 285–289, New York, New York, USA, Aug. 2000. ACM Press.

[15] D. Murray and L. Stankovic. Refit: Electrical load measurements. http://www.refitsmarthomes.org/.

[16] T. Rakthanmanon, B. Campana, A. Mueen, G. Batista, B. Westover, Q. Zhu, J. Zakaria, and E. Keogh. Searching and mining trillions of time series subsequences under dynamic time warping. In *Proceedings of the 18th ACM SIGKDD Conference - KDD '12*, page 262, New York, New York, USA, Aug. 2012. ACM Press.

[17] T. Rakthanmanon, B. Campana, A. Mueen, G. Batista, B. Westover, Q. Zhu, J. Zakaria, and E. Keogh. Searching and mining trillions of time series subsequences under dynamic time warping. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 262–270, 2012.

[18] T. M. Rath and R. Manmatha. Word image matching using dynamic time warping. In *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on*, volume 2, pages II—-521. IEEE, 2003.

[19] H. Sakoe and S. Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 26(1):43–49, Feb. 1978.

[20] S. Salvador and P. Chan. Toward Accurate Dynamic Time Warping in Linear Time and Space. *Intell. Data Anal.*, 11(5):561–580, Oct. 2007.

[21] D. Sart, A. Mueen, W. Najjar, V. Niennattrakul, and E. Keogh. Accelerating Dynamic Time Warping Subsequnce Search with GPUs and FPGAs. ICDM 2010. In *Proceedings - IEEE International Conference on Data Mining, ICDM*, pages 1001–1006, 2010.

[22] J. Shieh. iSAX : Indexing and Mining Terabyte Sized Time Series. In *Work*, volume KDD '08, pages 623–631, 2008.

[23] M. Shokoohi-Yekta, J. Wang, and E. Keogh. *On the Non-Trivial Generalization of Dynamic Time Warping to the Multi-Dimensional Case*, chapter 33, pages 289–297.

[24] L. N. Tan, A. Alwan, G. Kossan, M. L. Cody, and C. E. Taylor. Dynamic time warping and sparse representation classification for birdsong phrase classification using limited training data. *The Journal of the Acoustical Society of America*, 137(3):1069–80, Mar. 2015.

[25] X. Wang, A. Mueen, H. Ding, G. Trajcevski, P. Scheuermann, and E. Keogh. Experimental comparison of representation methods and distance measures for time series data. *Data Mining and Knowledge Discovery*, 26(2):275–309, 2013.

[26] D. Yankov, E. Keogh, J. Medina, B. Chiu, and V. Zordan. Detecting time series motifs under uniform scaling. In *Proceedings of the 13th ACM SIGKDD Conference - KDD 07*, KDD '07, page 844, 2007.