Matrix Profile XXII: Exact Discovery of Time Series Motifs under DTW

Sara Alaee, Ryan Mercer, Kaveh Kamgar, Eamonn Keogh

University of California, Riverside

{salae001, rmerc002, kkamg001}@ucr.edu, eamonn@cs.ucr.edu

Dear Reader: This is a greatly expanded version of the ICDM 2020 paper. This means that some parts of it were not peer-reviewed. However, all code and data is publicly available.

Abstract— In recent years, time series motif discovery has emerged as perhaps the most important primitive for many analytical tasks, including clustering, classification, rule discovery, segmentation, and summarization. In parallel, it has long been known that Dynamic Time Warping (DTW) is superior to other similarity measures such as Euclidean Distance under most settings. However, due to the computational complexity of both DTW and motif discovery, virtually no research efforts have been directed at combining these two ideas. The current best mechanisms to address their lethargy appear to be mutually incompatible. In this work, we present the first efficient, scalable and exact method to find time series motifs under DTW. Our method automatically performs the best trade-off of time-to-compute versus tightness-of-lower-bounds for a novel hierarchy of lower bounds that we introduce. As we shall show through extensive experiments, our algorithm prunes up to 99.99% of the DTW computations under realistic settings and is up to three to four orders of magnitude faster than the brute force search. This allows us to discover DTW motifs in massive datasets for the first time. As we will show, in many domains, DTW-based motifs represent semantically meaningful conserved behavior that would escape our attention using all existing Euclidean distance-based methods.

Keywords: *Time Series, Motifs, Dynamic Time Warping*

1 INTRODUCTION

Time series motif discovery— the unearthing of locally conserved behavior in a long time series — has emerged as one of the most important time series primitives in the last decade [1]. In recent years, there has been significant progress in the scalability of motif discovery, but essentially all algorithms use the *Euclidean Distance* (ED) [5][18]. This is somewhat surprising, because in parallel, the community seems to have converged on the understanding that the Dynamic Time Warping (DTW) is superior in most domains, at least for the tasks of clustering, classification, and similarity search [15][22][23][30]. Could DTW also be superior to ED for motif discovery? To preview our answer to this question, consider Fig. 1, which shows the top-1 motif discovered in a household electrical power demand dataset, using the Euclidean distance [19].

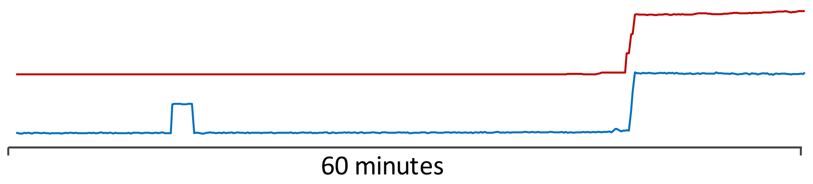


Fig. 1 The top-1 Euclidean distance motif discovered in a one-month long electrical power demand dataset.

We have no obvious reasons to discount this motif. It clearly shows the highly conserved behavior of relatively low demand for power for about ¾ of an hour, followed by a high sustained demand. Note that the Euclidean distance is robust to the short "blip" that appears in just the blue time series (from the duration, shape and watts drawn, this pattern almost certainly represents an electrical kettle).

However, now let us consider Fig. 2, which shows a different pair of subsequences from the same dataset.

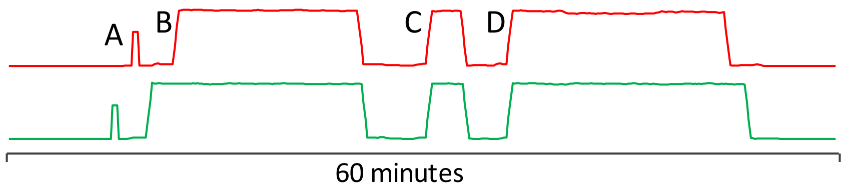


Fig. 2 A pair of subsequences from household electrical power demand data. The pattern corresponds to (A) short run of discharge pump to empty any liquid in the machine, (B) pumping water into reservoir, (C) spraying water over dishes (D) pumping out water.

In retrospect, we would surely have preferred to have discovered this pair of motifs as the top-1 motif. The complexity of the pattern that is conserved points to a common mechanism. In fact, this is the case. This pattern corresponds to a particular program from a dishwasher. Why was this pattern not discovered by the classic motif discovery algorithm? Fig. 3 offers a visual explanation.

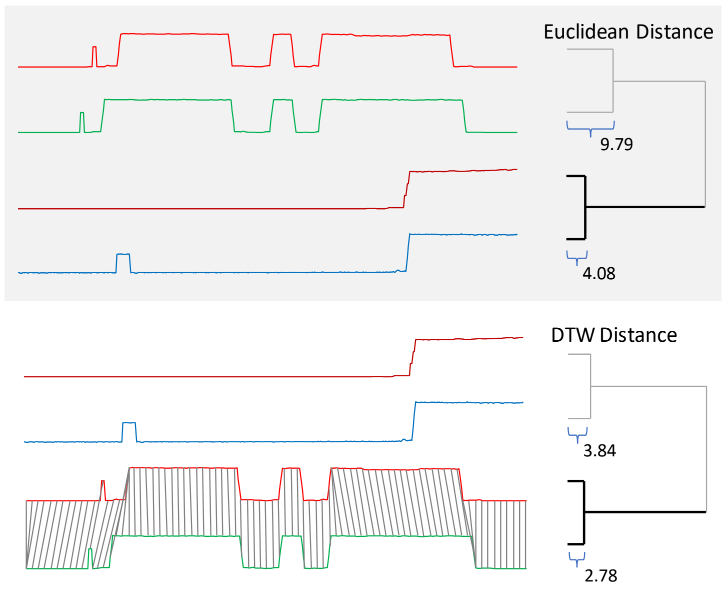


Fig. 3 A pair of subsequences from household electrical power demand data. The pattern corresponds to (A) short run of discharge pump to empty any liquid in the machine, (B) pumping water into reservoir, (C) spraying water over dishes (D) pumping out water.

As shown with the gray hatch-lines between the bottom pair of subsequences in Fig. 3, DTW's ability to non-linearly match features that may be out of phase allows it to report a much smaller distance for subsequences that are semantically similar, but have local regions that are out-of-phase [15][17][31].

As we will show, given the ability to find motifs under DTW, examples like the one above are replete in diverse domains such as industry, medicine, and human/animal behavior. Given that there is a large body of literature on both motif discovery [1][5][16][17][18][28][31][32][37][38] and Dynamic Time Warping (and it variants) [6][11][15][21][22][23][24][26][27][29][30], why are there essentially no DTW-based motif discovery tools?

We believe that the following explains this omission. Both motif discovery and DTW comparisons are famously computationally demanding [1][15]. Recent years have seen significant progress for both, especially the *Matrix Profile* for the former [37], but the main speed-up techniques for each are not obviously combinable.

In this work we introduce a novel algorithm that makes DTW motif discovery tenable for large datasets for the first time. We call our algorithm SWAMP, Scalable Warping Aware Matrix Profile. This is something of a misnomer, since we attempt to *avoid* computing most of the true DTW Matrix Profile by instead computing much cheaper upper/lower bounding Matrix Profiles.

We claim the following contributions for our work:

1. We show, for the first time, that there exists conserved structure in real-world time series that can be found with DTW motifs, but *not* with classic Euclidean distance motifs [18][32]. It was not clear that this had to be the case, as [18] and others had argued for the diminished utility of DTW for *motif discovery* (*all*-to-all search), relative to its known utility for similarity search (*one*-to-all search)[1].
2. We introduce SWAMP, the first *exact* algorithm for DTW motif discovery that significantly outperforms brute force search by two or more orders of magnitude.
3. Our algorithmic approach uses a novel "*adaptive hierarchy of lower bounds*" methodology that may be useful for other problems.

The rest of the paper is organized as follows. In Section 2, we present the formal definitions and background, before outlining our approach in Section 3. Section 4 contains an extensive experimental evaluation. Section 5 provides a case study on using SWAMP in classification. Finally, we offer conclusions and directions for future work in Section **Error! Reference source not found.**.

2 BACKGROUND AND RELATED WORK

2.1 Time Series Notation

We begin by introducing the necessary definitions and fundamental concepts, beginning with the definition of a *Time Series*:

Definition 1: A Time Series **T** = $t_1, t_2, \ldots, t_n$ is a sequence of n real values.

Our distance measures quantify the distance between two time series based on local subsections called *subsequences*:

[1] In brief, the argument is this: Recall that cDTW is constrained by a parameter w, the maximum amount of warping allowed, and that as w approaches zero, cDTW degenerates to the Euclidean distance. It has been shown that the best setting for w decreases as the number of comparisons increase (see Figure 6 of [12]). For similarity search, there are $O(n)$ comparisons, but for motif search there are $O(n^2)$ comparisons, favoring a small value for w, perhaps approaching zero.

Definition 2: A subsequence $\mathbf{T}_{i,L}$ is a contiguous subset of values with length L starting from position i in time series $\mathbf{T}$; the subsequence $\mathbf{T}_{i,L}$ is in form $\mathbf{T}_{i,L} = t_i, t_{i+1}, \ldots, t_{i+L-1}$ where $(1 \leq i \leq n - L + 1)$ and L is a user-defined subsequence length with value in range of $4 \leq L \leq |\mathbf{T}|$.

Here we allow L to be as short as four, although that value is pathologically short for almost any domain [22].

The nearest neighbor of a subsequence is the subsequence that has the smallest distance to it. The closest pairs of these neighbors are called the time series *motifs*.

Definition 3: A motif is the most similar subsequence pair of a time series. Formally, $\mathbf{T}_{a,L}$ and $\mathbf{T}_{b,L}$ is the motif pair iff $dist(\mathbf{T}_{a,L}, \mathbf{T}_{b,L}) \leq dist(\mathbf{T}_{i,L}, \mathbf{T}_{j,L})\ \forall\, i,j \in [1,2,\ldots,n-L+1]$,, where $a \neq b$ and $i \neq j$, and $dist$ is a distance measure.

One can observe that the potential best matches to a subsequence (other than itself) tend to be the subsequences beginning immediately before or after the subsequence. However, we clearly want to exclude such redundant "near self matches". Intuitively, any definition of motif should exclude the possibility of counting such *trivial matches*.

Definition 4: Given a time series $\mathbf{T}$, containing a subsequence $\mathbf{T}_{i,L}$ beginning at position i and a subsequence $\mathbf{T}_{j,L}$ beginning at j, we say that $\mathbf{T}_{j,L}$ is a trivial match to $\mathbf{T}_{i,L}$ if $j \leq i + L - 1$.

Following [3] we use a vector called the *Matrix Profile* (*MP*) to represent the distances between all subsequences and their nearest neighbors.

Definition 5: A Matrix Profile (MP) of time series $\mathbf{T}$ is a vector of distances between each subsequence $\mathbf{T}_{i,L}$ and its nearest neighbor (closest match) in time series $\mathbf{T}$.

The classic Matrix Profile definition assumes Euclidean distance measure which computes the distance between the i^{th} point in one subsequence with the i^{th} point in the other (see Fig. 4.*left*). However, as shown in Fig. 4.*center*, the non-linear DTW alignment allows a more intuitive distance that matches similar shapes even if they are locally out of phase. For brevity, we omit a formal definition of the (increasingly well-known) DTW, instead referring the interested reader to [23][15][22].

Similarity search under DTW can be demanding in terms of CPU time. One way to address this problem is to use a *lower bound* to help prune sequences that could not possibly be a best match [22]. While there exist dozens of lower bounds in the literature, in our work we use a generalization of the LB_{Keogh} [15][22].

Definition 6: The $\mathrm{LB}_{\mathrm{Keogh}}$ lower bound between a time series **Q** and another time series **T**, given a warping window size *w*, is defined as the distance from the closest of the upper and lower envelopes around **Q**, to **T**. Formally:

$$LB_{Keogh}(Q,T) = \sqrt{\sum_{i=1}^{n} \begin{cases} (t_i - U_i)^2 & if\ t_i > U_i \\ (t_i - L_i)^2 & if\ t_i < L_i \\ 0 & otherwise \end{cases}} \qquad \textbf{Eq. 1}$$

Where the upper envelope (U_i) and lower envelope (L_i) of **Q** are defined as:

$$\begin{aligned} U_i &= \max\,(q_{i-w}, q_{i-w+1}, \dots, q_{i+w}) \\ L_i &= \min\,(q_{i-w}, q_{i-w+1}, \dots, q_{i+w}) \end{aligned} \qquad \textbf{Eq. 2}$$

Fig. 4 illustrates this definition.

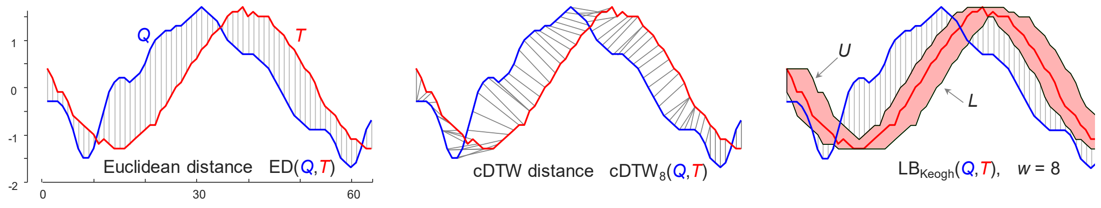


Fig. 4 For two time series Q and T: *left*) Their Euclidean Distance. *center*) Their DTW distance. *right*) Their $\mathrm{LB}_{\mathrm{Keogh}}$ distance.

For computationally demanding tasks, even the lower bound computation may take a lot of time. Thus, we plan to exploit a "spectrum" of lower bounds as we explain in Section 3.1, each of which makes a different compromise of fidelity versus tightness.

To create this spectrum, we exploit our ability to perform various computations on the reduced dimensionality data. More concretely, we can perform downsampling using the *Piecewise Aggregate Approximation* (*PAA*) [34].

Definition 7: The PAA of time series **T** of length *n* can be calculated by dividing **T** into *k* equal-sized windows and computing the mean value of data within each window. The vector of these values is the PAA representation of the time series.

It is convenient to express the compression rate of a PAA approximation as "*D* to 1", or $D{:}\,1$, where $\mathrm{D} = \mathrm{n}/\mathrm{k}$. This notation can be visualized as shown in Fig. 5.

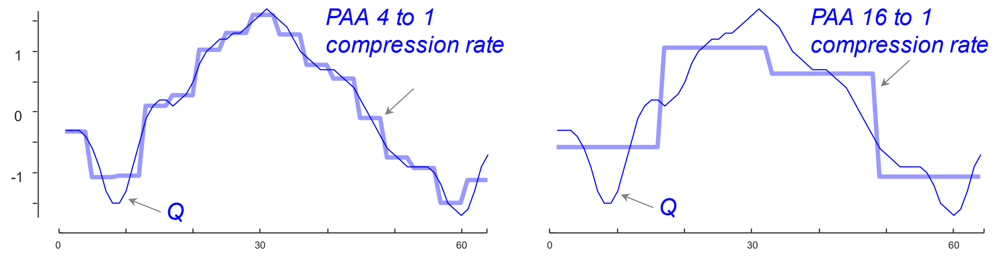


Fig. 5 A time series **Q**, downsampled using PAA to two different compression rates. *left*) 4:1 *right*) 16:1

Given that we can downsample time series, we can also generalize LB_{Keogh} to such downsampled data, with $LB_{Keogh}D{:}1$ ($D \geq 1$):

Definition 8: The downsampled lowerbound $LB_{Keogh}D{:}1(\mathbf{Q},\mathbf{T})$ between a time series **Q** and another time series **T** is defined as the distance from the closest of the downsampled upper and lower envelopes around **Q**, to the downsampled **T**. Formally:

$$LB_{Keogh}D{:}1(Q,T) = \sqrt{\sum_{i=1}^{n} \begin{cases} (t\_D_i - U\_D_i)^2 & if\ t\_D_i > U\_D_i \\ (t\_D_i - L\_D_i)^2 & if\ t\_D_i < L\_D_i \\ 0 & otherwise \end{cases}} \quad \textbf{Eq. 3}$$

Where $T\_D = \mathrm{PAA}\,(\mathrm{T}, \mathrm{D}),\ U\_D = \mathrm{PAA}\,(\mathrm{U}_Q, \mathrm{D}),\ and\ \ L\_D = \mathrm{PAA}\,(\mathrm{L}_Q, \mathrm{D}).$

Fig. 6 illustrates this definition.

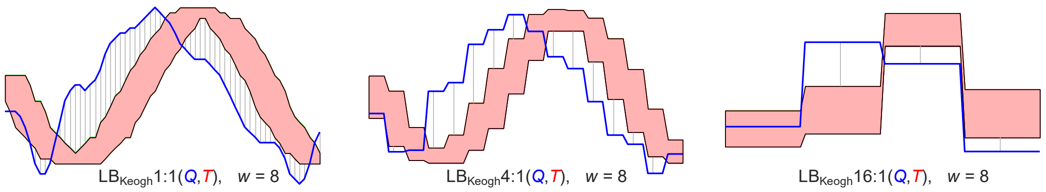


Fig. 6 An illustration of parametrized LB_{Keogh}. Three possible settings that make different trade-offs on the spectrum of time-to-compute vs. tightness of lower bound. The special case of $LB_{Keogh}1{:}1$ is the classic lower bound also shown in Figure 4, and used extensively in the community [9][15][16].

Given these downsampled lower bounds, we can still use the LB_{Keogh} distance, but we need to scale the distance by $\sqrt{n/D}$ to generate a tighter, yet still admissible lower bound. The proof of this variation of the lower bound appears in a slightly different context in [39]. To see why it is needed, refer to Fig. 6.*right*. Here each gray hatch-line represents the aggregate distance for 16 datapoints. If we only counted each line once, we would have a very weak lower bound. It seems that we could scale each line's contribution by 16 (or more generally, *D*), but then we would not have an admissible bound. It can be shown that $\sqrt{n}/D$ is the optimally tight admissible scaling factor [36][39].

To see how the parameterization affects the tightness of the lower bound, we selected 256 random pairs from the electrical demand dataset (see Fig. 14) and computed both their true distance and the lower bound distances at the dimensionalities shown in Fig. 6. The results are shown in Fig. 7.

Note that while our examples use powers of two for both the original and reduced dimensionality, PAA and our parametrized lower bounds are defined in the more general case [34].

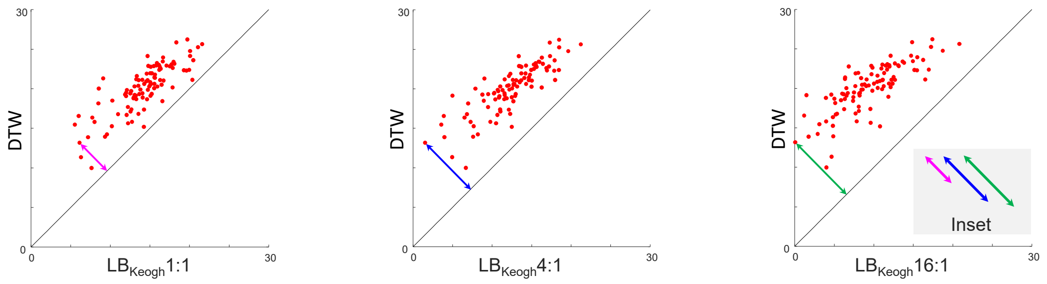


Fig. 7 An illustration of the tightness of the parametrized LB_{Keogh}. The tightness for each pair is inversely proportional to orthogonal distance to the diagonal line. For one randomly selected point, we show how this changes (inset).

2.2 Related Work

There is a huge body of literature on DTW [22] and on motif discovery [1][17][18][31]. However, there are very few papers on the intersection of these ideas.

In [20] the authors introduce "*A fast method for motif discovery in large time series database under dynamic time warping*". However, this method does not produce the top motifs as we have defined them in definition 3. It is perhaps better seen as a clustering algorithm that produces centroids that could be considered "*motifs*". Likewise, Lagun et. al. created an algorithm to explore cursor movement data [16]. The algorithm discovers "common motifs" and does use the DTW distance, but once again, it is better seen as a clustering algorithm that produces centroids that could be considered motifs. These papers speak to the utility of both motif discovery and to the use of DTW. However, these works do not offer us actionable insights for the task-at-hand.

Finally, a recent paper uses DTW and reports results for "*Motif Discovery*" [40]. However, this paper is simply doing what the community commonly calls "*range queries*", not motif discovery.

To the best of our knowledge, there is only one research effort that finds exact motifs under DTW [28]. This method creates a full DTW Matrix Profile but optimizes its creation by exploiting many of the techniques used by the UCR-Suite [22]. These optimizations produce a ten-fold improvement over a naïve brute force implementation. We also avail ourselves of these optimizations; however, we achieve a much more dramatic speed up by not computing the full DTW Matrix Profile, but rather computing as little of it as possible, in order to admissibly discover just the top DTW motifs.

The reader may wonder if we could replace exact DTW with one of the "fast approximations" to it, such as FastDTW [25]. Recent works suggest that these approximations are actually not faster than the carefully optimized exact DTW [35]. Moreover, all such work is empirical, there are no bounds on how bad the approximation can be [35]. Thus, we do not see this as a promising avenue for acceleration.

3 OBSERVATIONS AND ALGORITHMS

Before introducing our algorithm in detail, we will take the time to outline the intuition behind our approach. In Fig. 8 we show a time series and its DTW MP.

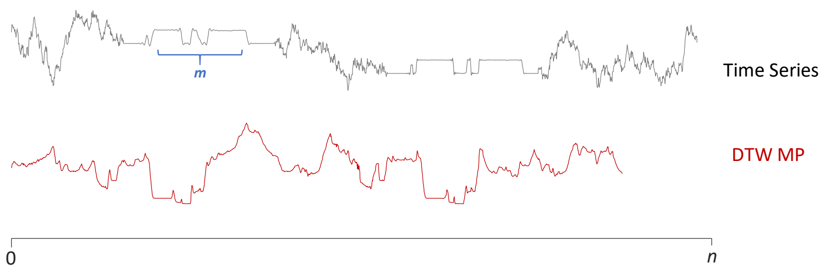


Fig. 8 A time series and its DTW MP. The lowest points of the DTW MP are the locations of the top-1 DTW motifs.

The two lowest points (they must have tied values by definition [37]) correspond to the top-1 DTW motif. Thus, while we have solved our task-at-hand, this brute force computation of the DTW MP required $O(n^2m^2)$ time.

There are some optimizations (which we use) including early abandoning, using the squared distance, etc. (see [19] and [22]). However, these only shave off small constant factors. It is possible to index DTW. However, that only helps to accelerate *future* ad-hoc similarity search queries. Here, the time required to build the index would only dramatically increase the time above.

Note that the Euclidean distance is an *upper* bound for the DTW. Moreover, there are perhaps a few dozen known *lower* bounds to DTW, including the LB_{Keogh} [15]. In Fig. 9 we revisit the data shown in Fig. 8 to include the MPs for these two additional measures. Note that they "squeeze" the DTW MP from above and below.

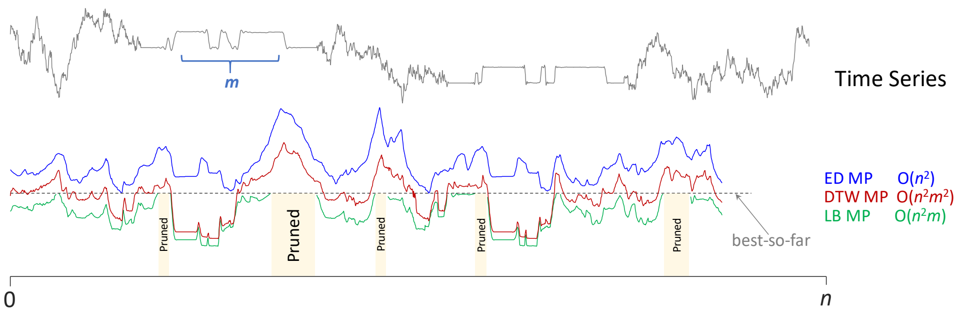


Fig. 9 A time series and its ED MP, its DTW MP and its LB_{Keogh}1:1MP. Note that the lowest values in the ED MP (denoted with the horizontal dashed line) are an upper bound on the values of the top DTW motif.

This figure suggests an immediate improvement to the brute force algorithm. The lowest value of the ED MP is an upper bound on the value of the top-1 DTW motif. Thus, before we compute the DTW MP, we could first compute the ED MP and use its smallest value to initialize the *best-so-far* value for the DTW MP search algorithm. This has two exploitable consequences. It would speed up the brute force algorithm, because the effectiveness of early abandoning is improved if you can find a good *best-so-far* early on. However, there is a much more consequential observation. Any region in the time series for which the lower bound is greater than the best-so-far can be admissibly pruned from the search space.

Note that this pruning can dramatically accelerate our search. For example, suppose that the fraction p of the time series is pruned from consideration as the location of the best motif. We then only have to compute $(1-p)^2$ of the possible pairs of subsequences. Moreover, this ratio can only get better, as we find good matches that further drive the *best-so-far* down.

In fact, as we shall see, on real datasets this pruning can be so effective that instead of doing $O(n^2)$ invocations of DTW, we only need to do a small constant number. This reduces the time complexity to find the top-1 DTW motifs from $O(n^2m^2)$, to the $O(n^2m)$ time required to compute the lower bound. Because m can be in the range, of say, one hundred to ten thousand (see the insect example in Fig. 12), this offers a significant speedup. Nevertheless, it is natural to ask if we can further improve on this.

Let us revisit Fig. 9. Note that in some locations, the lower bound is much greater than the *best-so-far*. This suggests an opportunity. In general, it is often the case that there are multiple lower bounds for a distance measure, which produce different tradeoffs on the spectrum of time-to-compute versus average tightness. Thus, instead of always using the tightest lower bound available to us everywhere, it would be better to use faster "*just tight enough*" lower bounds wherever possible. As we shall see, this is exactly what SWAMP does.

3.1 Creating a Spectrum of Lower Bounds

As noted above, our SWAMP algorithm depends on the availability of multiple lower bounds that make different tradeoffs on the spectrum of tightness versus speed of execution.

It is not meaningful to measure the tightness of lower bounds on a single pair of time series, as the idiosyncrasies of the particular pair of subsequences may favor different lower bounds. Instead, it is common to measure the tightness of a lower bound by averaging over many pairs of randomly chosen time series [23].

$$tightness(A,B) = \frac{LB(A,B)}{DTW(A,B)} \qquad \textbf{Eq. 4}$$

In Fig. 10 we average over six million pairs of random-walk for each setting.

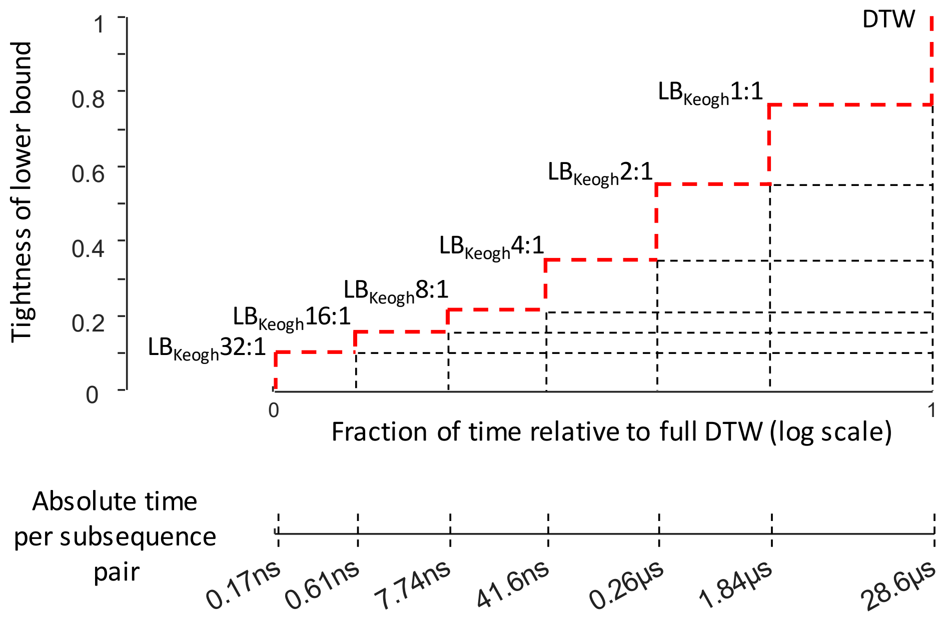


Fig. 10 A spectrum of lower bounds for DTW plotted with time (note the log scale) versus tightness. Recall that DTW is a lower bound to itself, thus occupies the top right corner.

It is important to ward off a possible misunderstanding. If we computed the entire $LB_{Keogh}1:1$ and found that it aggressively pruned off all but one pair of subsequences (the true top-1 motif), then we would achieve a speedup of about 28.6μs/1.84μs = 15.5. This 15-fold speed would be impressive, but it *appears* to be the upper bound on speed-up. However, as hinted at above, we hope to prune off many of the $LB_{Keogh}1:1$ computations themselves, with the much cheaper $LB_{Keogh}2:1$ calculations. Moreover, we plan to do this iteratively, using cheaper (but weaker) lower bounds to prune off as many as possible more expensive (but stronger) lower bounds.

Note that the red dashed line in Fig. 10 forms a Pareto frontier [20]. If there is any lower bound that is above the red line at any point, we should use it. We have investigated the dozens of alternative lower bounds, but we did not discover better performing bounds. There are two main classes of lower bounds:

- Lower bounds such as $LB_{Kim}FL$ that are $O(1)$ should in principle be on the Pareto frontier to the right of $LB_{Keogh}32:1$. However, their $O(1)$time complexity assumes that the two time series are already normalized. If we are forced to normalize, these bounds are pushed to the interior of the frontier (however, as we will later show, the $LB_{Kim}FL$ can be used in a later phase of the algorithm, when the normalization is “free”, because it is computed for another purpose).

- There are at least a dozen lower bounds that are variants of the LB_{Keogh}, including $LB_{Improved}$, $LB_{Enhanced}$, LB_{New}, $LB_{Rotation}$, LB_{Hust}, LB_{En}, etc. [13][22][12]. All of these exploit LB_{Keogh}, plus some additional information to produce a tighter lower bound. However, in all cases we found that the additional time required to exploit the "additional information" did not pay for itself. We would have been better off spending that extra time to compute the LB_{Keogh} at a higher level. However, we note that in some cases a clever implementation insight might fix this overhead.

Note that while we did not discover any other lower bounds to help for the task at hand, this does not say anything about the utility of these bounds for *other* tasks. LB_{Kim}FL has been shown to be useful for in-memory similarity search [22], and the more expensive lower bounds are useful for disk-based indexing [15].

3.2 Introducing SWAMP

For notational simplicity we consider the task of finding the top-1 motif under DTW for a given value of *w*. The generalizations to top-K motifs or range motifs are trivial [37].

We can best think of SWAMP as a two-phase algorithm. In Phase I it uses a single *upper* bound, and an adaptive hierarchy of *lower* bounds to prune off as many of the candidate time series subsequences as possible (*candidate* for being one the best DTW motif pair). Then, in Phase II, any surviving pairs of subsequences are searched with a highly optimized "brute force" search algorithm. The algorithm in Table 3 formalizes SWAMP which includes subroutines that compute Phase I and Phase II.

3.2.1 Phase I of SWAMP

We start by reviewing the two exploitable facts that we previewed in Fig. 9.

- The ED MP is the upper bound for LB_{Keogh}MP.
- The LB_{Keogh}MP is the lower bound for DTW MP.

Based on these observations, we know that any section of LB_{Keogh}MP (i.e. LB_{Keogh}1:1MP) that is greater than the minimum of ED MP (we consider that as the *best-so-far*), could not contain the best motif and can therefore be pruned. We can compute the DTW score for the region suggested by the lowest value of the pruned

LB_{Keogh}1:1MP. If the score is lower than the minimum of ED MP, we can further lower the best-so-far. In this case, we can further reduce the number of DTW tests.

This basic strategy gains speedup, replacing most of the expensive DTW calculations with cheaper lower bound calculations. However, while computing LB_{Keogh}1:1MP is much faster than full DTW, it is still computationally expensive. Nevertheless, as we discussed in the previous section, we may not need to compute the full LB_{Keogh}1:1MP to find the best motifs. Instead, we can apply the above strategy on a hierarchy of cheaper downsampled LB_{Keogh}MP. The algorithm in Table 1 formalizes this process.

Table 1. ComputeDSMP: Hierarchically computes the downsampled lower bound Matrix Profile and prunes off the unpromising locations.

Procedure: *ComputeDSMP(T,L,w)*
Input: time series *T*, subsequence length *L*, warping window size *w*
Output: expanded LB_{Keogh}D:1 values *LBMP*, expanded LB_{Keogh}D:1 indexes *LB_*index, pruned locations of time series *pruned*, candidate motif distance *best-so-far*

```
1   ED_mp ← ComputeMatrixProfile(T,L)                     // Using SCRIMP [25]
2   ED_motif_idx ← argmin(ED_mp)
3   best-so-far ← dtw_distance(ED_motif_idx)
4   D ← L
5   pruned(:) ← false
6   while D>0:                          // iterate over increasing fine approximations
7       [LBMP,LB_index] ← LBKeoghDSMP(T,L,D,pruned)        // See Table 2
8       LB_motif_dist, LB_motif_idx ← min(LBMP)
9       if LB_motif_dist < best-so-far:
10          best-so-far ← LB_motif_dist
11          pruned(LBMP > best-so-far) ← true
12      endif
13      D ← floor(D/2)   // next iteration will be twice as fine
14  endwhile
```

In line 1 we compute the classic Matrix Profile for the time series T with the given subsequence length L. This is needed to provide the upper bound of the distance between the DTW motifs we will discover. Using this Matrix Profile, we find the ED motifs, i.e. the pair of lowest values [37][38]. We then measure the distance between those motifs using the DTW distance rather than the ED distance, in order to initialize the best-so-far distance (lines 2-3).

Starting with a downsampling factor equal to the subsequence length (line 4), we first compute a very cheap lower bound for the entire time series using the algorithm in Table 2. If the DTW distance for the region suggested by the lowest value of this lower bound is smaller than the *best-so-far*, we update the *best-so-far*. For regions where it is too weak to prune, we selectively compute a tighter bound and repeat the same process. The algorithm ends after it has explored the highest resolution (i.e. $D = 1$) (lines 6-12).

Note that when computing lower bounds at any resolution level, we take the pruned-off locations at the lower levels into account, meaning that we do not compute a lower bound for those regions. The lower bound computation process is described Table 2.

Table 2. LBKeoghDSMP: Computes the LBKeoghMP for the downsampled time series.

```
Procedure LB_Keogh DSMP(T,L,D,pruned)
Input: time series T, Subsequence length L, Downsampling factor D, pruned locations of the time series pruned
Output: expanded LB_Keogh D:1 values LBMP, expanded LB_Keogh D:1 indexes LB_index
1  pruned_D ← paa (pruned, T_D)
2  L_D ← L × floor(length(T_D)/length(T))
3  MP_D, LB_index ← LB_Keogh(T_D, L_D, pruned_D)
4  LBMP ← interpolate(MP_D, floor(length(T) / length(T_D)))
5  LBMP ← sqrt(length(T) / length(T_D)) × LBMP
6  T_D ← paa (T, D)
```

Lines 1 and 2 downsample both the time series and the Boolean vector specifying the pruned and non-pruned locations. Line 3 scales downs the subsequence length relative to the downsampling rate. Lines 4-6 compute the downsampled lower bound $LB_{Keogh}D{:}1$, expand it to the size of the complete lower bound and scale up the result by the downsampling factor.

3.2.2 Phase II of SWAMP

Let us review the situation at the end of Phase I. From the original set of $n - L + 1$ candidate time series subsequences that might have contained the top-1 motif, we pruned many (hopefully the vast majority) of them into a much smaller set c, of remaining candidates.

Globally, we know:

1. A *best-so-far* value, which is an upper bound on the value of the top-1 motif. We also know which pair from c is responsible for producing that low value.

Locally, for each subsequence, we know:

2. A DTW lower bound value on its distance to its nearest neighbor.
3. The location of its nearest neighbor in the lower bound space, which may or may not also be its DTW nearest neighbor.

We now need to process the set of candidates c to find the true top-1 motif, or it the current *best-so-far* refers to the top-1 motif, confirm that fact by pruning every other possible candidate.

Note that even if we processed all $O(c^2)$ pairwise comparisons randomly, there is still the possibility of pruning more candidates. In particular, every time the *best-so-far* value decreases, we can use the information in '3' above to prune additional candidates in *c*, whose lower whose lower bounds now exceed the newly decreased *best-so-fa*r value.

As shown in Table 3 we can see this search as a classic nested loop, in which the outer loop considers each candidate in *c*, finding its DTW nearest neighbor (non-trivial match) in *c*.

Given our stated strategy of trying to drive the *best-so-far* down as fast as possible, the optimal ordering for our search is obvious. In the outer loop we should start with a candidate that is one of the true DTW motif pair, and the inner loop we should start with the other subsequence of that motif pair. Clearly, we cannot do this, since that assumes we already know what we are actually trying to compute. However, we can approximate this optimal ordering quite well. On average, the true DTW distance is highly correlated with its lower bound (see Fig. 9). Thus, we should order the outer loop in increasing order of the lower bounds provided by LB_{Keogh}1:1 in the last iteration of Phase I (line 2).

For the inner loop, for the very first iteration we consider the candidate's nearest neighbor in lower bound space and replace it with its immediate neighbor (4-7). After this first comparison, the subsequent iterations can be done in any order. The algorithm in Table 3 formalizes these observations.

Table 3. SWAMP: Discovers the top-1 DTW motifs.

```
Procedure SWAMP(T,L,w)
Input: time series T, Subsequence length L, warping window size w
Output: candidate motif distance best-so-far, motif pair locations motif_pair
1   [LBMP, pruned, best-so-far, LB_index] ← ComputeDSMP(T,L,w)      //Table 1
2   [candids,candids_index] ← sorted(LBMP)                          // begin Phase II
3   for  i=1:length(candids):
4       candid_idx ← candids_index[i]
5       if  pruned[candid_idx]
6           continue
7       endif
```

```
8    neigh_idx ← candid_idx + L: length(candids)
9    swap(neigh_idx[1], neigh_idx[LB_index[candid_idx]])
10   for j=1:length(neigh_idx)
11     if pruned[neigh_idx[j]]
12       continue
13     endif
14     a ← T[candid_idx : candid_idx + L-1]
15     b ← T[neigh_idx[j] : neigh_idx[j] + L-1]
16     if $LB_{Kim}FL(a,b)$ >= best-so-far
17       continue
18     elseif $LB_{Keogh}(a,b)$ >= best-so-far
19         continue
20     endif
21     dist ← dtw_distance(a, b, w, best-so-far)
22     if dist < best-so-far
23       best-so-far ← dist
24       motif_pair ← [candid_idx, neigh_idx]
25       pruned(candid_idx(candids >= best-so-far)) ← true
26     endif
27   endfor
```

Note that we have added four further optimizations into the inner loop. We use a cheap but weak lower bound $LB_{Kim}FL$ to prune some subsequences (line 12). For those pairs that survive, we use a tighter but more expensive lower bound $LB_{Keogh}1:1$ (line 13). Moreover, we use the early abandoning version of LB_{Keogh}, as introduced in [22]. Finally, if all previous attempts at pruning fail, and we are forced to do DTW, we compute the early abandoning version of DTW, which was also introduced in [22] (line 14). If any candidates survive that step, we update the best-so-far and prune the remaining unpromising subsequences (line 15-18).

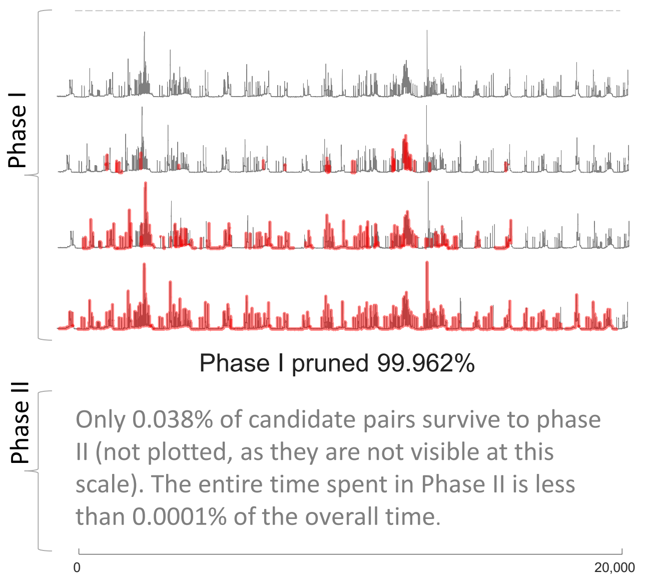
Revisiting $LB_{Keogh}1:1$ in this phase is worth clarifying. We do already know the $LB_{Keogh}1:1$ distance to each candidate's nearest neighbor (from Phase I), but not to all its neighbors. Therefore, it is possible that with another round of lower bound computation for the remaining pairs, we can potentially have more prunings

3.3 A Visual Intuition of SWAMP

We conclude our introduction of SWAMP with a visual intuition and review. In a test that previews the experiment shown later in Fig. 14, we searched for the top DTW motif of length 400, in an electrical demand dataset of length 20,000, using a warping window of 16. We carefully recorded what elements of the SWAMP algorithm are responsible for processing (pruning or computing) what fraction of the candidate pairs of subsequences. Fig. 11 shows the results.

This trace shows that at least in this case, only a vanishing small percentage of candidates survive to Phase II, where increasingly expensive computations are used to prune them, except for just 0.0204% of the candidates, which actually need full DTW.

The figure also shows the utility of our hierarchy of lower bounds approach. For example, $\mathrm{LB}_{\mathrm{Keogh}}$2:1 pruned 65.71% of the candidates. Had we not used our hierarchical approach, then $\mathrm{LB}_{\mathrm{Keogh}}$1:1 would also have pruned them, but would have taken about four times longer. In general, this figure suggests that every element of our algorithm is responsible for some speedup.



Only 0.038% of candidate pairs survive to phase II (not plotted, as they are not visible at this scale). The entire time spent in Phase II is less than 0.0001% of the overall time.

- ED MP is computed (does not prune) taking 0.0055% of overall time
- $\mathrm{LB}_{\mathrm{Keogh}}$20:1 pruned nothing, taking 0.11% of the overall time
- $\mathrm{LB}_{\mathrm{Keogh}}$5:1 pruned 9.76%, taking 3.297% of the overall time
- $\mathrm{LB}_{\mathrm{Keogh}}$2:1 pruned an additional 65.71%, taking 19.79% of the overall time
- $\mathrm{LB}_{\mathrm{Keogh}}$1:1 pruned an additional 24.49%, taking 71.26% of the overall time

- $\mathrm{LB}_{\mathrm{KimFL}}$ pruned 0.074%
- $\mathrm{LB}_{\mathrm{Keogh}}$1:1 pruned 0.024%
- 0.019% are early abandoned during DTW
- Only 0.0204% of all candidate pairwise computations needed full DTW

Fig. 11 For the power demand dataset (see Fig. 14), there are 184,348,801 pairwise subsequences that could be the top motif, which need to be pruned or compared. From top to bottom we see the progress of SWAMP in processing these candidate pairs. Numbers may not sum exactly to 100%, due to rounding for presentation.

3.4 Complexity Analysis

The Euclidean distance MP algorithms such as SCRIMP [38] have identical best-case and worst-case times, independent of the data. In contrast the performance of SWAMP does depend on the data. For example, if given pure random data (not random-walk, which is actually an ideal case) and a large value for w, the lower bounds become very weak. In essence, they report a lower bound of zero for almost all comparisons. In this case, the time complexity of SWAMP is the same as brute force search, $O(n^2m^2)$ time. This is because after no pruning in Phase I, it will be forced to do $O(n^2)$ comparisons that take $O(m^2)$ time.

Of course, such a dataset is unlikely to yield interesting motifs anyway. It is reasonable to ask what the time complexity is for datasets that are likely to yield semantically meaningful motifs, such as those shown in Section 4. For those datasets, we were able to prune at least 99.9% of all DTW calculations in Phase I alone. If we pessimistically assume that only the finest level of pruning (that is, $\text{LB}_{\text{Keogh}}1{:}1$) in Phase I actually did any pruning, then we have reduced the complexity to $O(n^2m)$. This is the same as the best time complexity known for Euclidean distance motif discovery before 2016 [18]. However, let us revisit our pessimistic assumption. Suppose that 99% of the pruning came from a coarser lower bound, let us say $\text{LB}_{\text{Keogh}}\text{D}{:}1$, then the time complexity reduces to $O(n^2m/D^2)$. In real word datasets it is often the case that a coarser lower bound can prune the majority of DTW calculations. For example, for the dataset shown in Figure 16, when D is 4 it still prunes off 82.7% of the DTW calculations. Because m is often approximately D^2, this means that the time complexity can be effectively $O(n^2)$, which is the same time complexity of SCRIMP [38] and the other state-of-the-art Euclidean motif discovery algorithms. This may seem unintuitive but recall that in order to find the closest pair of subsequences, SCRIMP computes the exact distance of *every* pair of subsequences. In contrast, SWAMP tries to compute as *few exact distances as possible*, preferring to prune virtually everything if possible.

In terms of space complexity SWAMP only requires an inconsequential $O(n)$ space overhead.

4 EMPRICAL EVALUATION

We begin by stating our experimental philosophy. We have designed all experiments such that they are easily reproducible. To this end, we have built a webpage that contains all datasets, code and random number seeds used in this work, together with spreadsheets which contain the raw numbers [41]. This philosophy extends to all the examples in the previous section.

4.1 Examples of DTW Motifs

Before conducting more formal experiments, we will take the time to show some examples of DTW motifs we have discovered in various datasets, in order to sharpen the readers' appreciation of the utility of DTW in motif discovery.

Entomologists use an apparatus called an electrical penetration graph (EPG) to study the behavior of sap-sucking insects [33]. It is known anecdotally [33], and by the use of classic motif discovery [18], that some such behaviors are often highly conserved at a time scale of 1 to 5 seconds. However, is there any behavior conserved at a longer time scale? As shown in Fig. 12.*bottom.left*, if we used the Euclidean distance, we might say "no". While the two patterns in the motif are vaguely similar, we might attribute this to random chance.

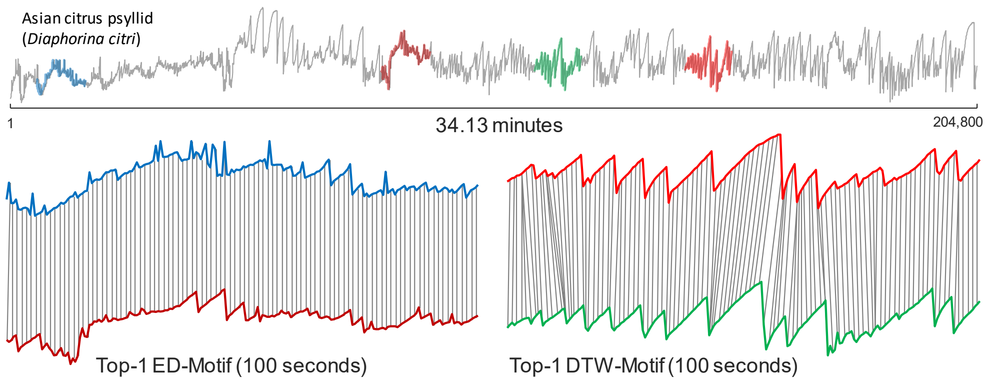


Fig. 12 *top*) About 34 minutes of EPG data collected from an Asian citrus psyllid (ACP) that was feeding on a Troyer citrange (Sweet Orange) [21]. *bottom*) The top-1 ED motif (*left*) and the top-1 DTW motif (*right*).

However, if we simply use the DTW distance, we discover an unexpectedly well-conserved long motif, corresponding to feeding behavior known as phloem ingestion [33].

Exploring such datasets rapidly gives one an appreciation as to how brittle the Euclidean distance can be. Consider the experiment on a different individual from the same insect species show in Fig. 13.

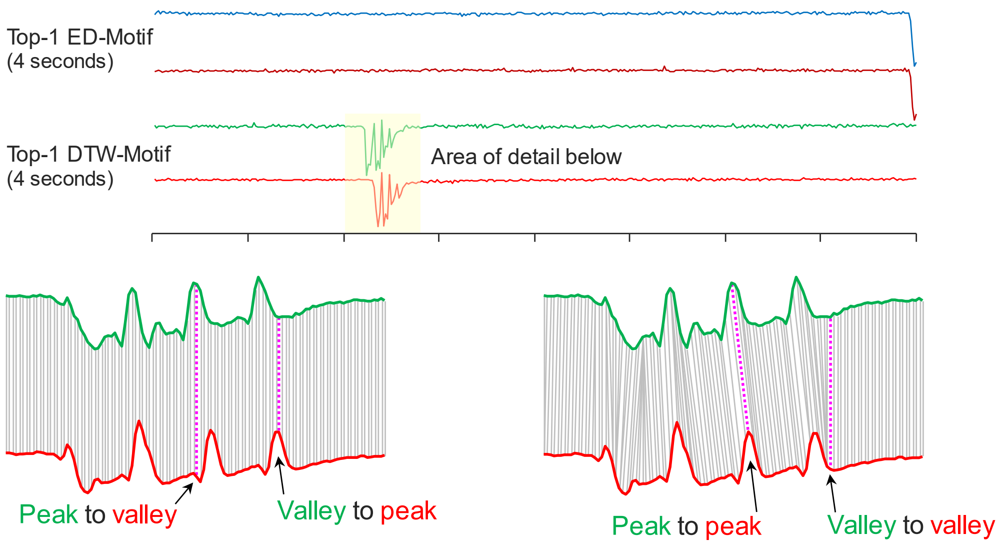


Fig. 13 *top*) The top-1 ED and DTW motifs discovered in seven-hour segment EPG data collected from an ACP [21]. *bottom*) A zoom-in of the DTW motif visually explains why ED has difficulty finding the same motif as DTW.

As before, we cannot directly fault the ED. It does return a pair of subsequences that are similar, although somewhat "boring and degenerate". However, an entomologist would surely prefer to see the DTW motif, which contains examples of a probing behavior [33]. To understand why ED could not discover these, in Fig. 13.*bottom* we show the alignment both methods have on the sections corresponding to the behavior. ED, with its one-to-one alignment, cannot avoid mapping some peaks to valleys, incurring a large distance. In contrast, the flexibility of DTW allows it to map peak-to-peak and valley-to-valley, allowing the discovery of these semantically identical behaviors.

In Fig. 2 we showed a motif we discovered in consumer electrical-demand telemetry. However, for visual clarity we chose a very simple example, the data was from a single outlet that was attached to the dishwasher. In Fig. 14 we consider a much more complex and difficult example; we examine the entire household demand, which includes the combination of refrigeration, cooking devices, laundry machines, entertainment devices, etc.

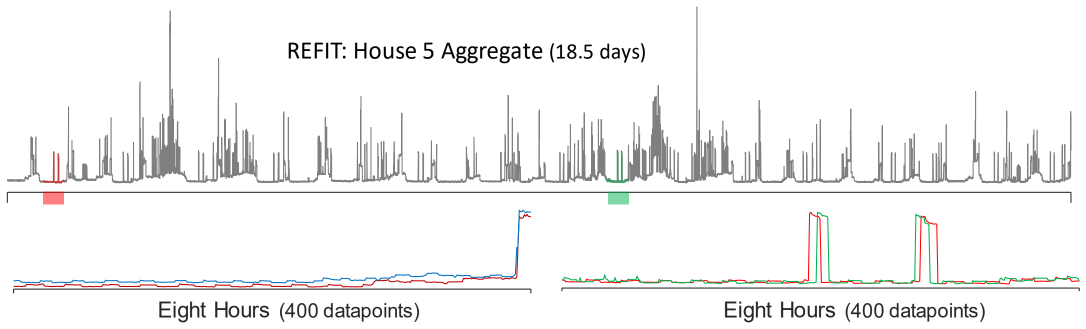


Fig. 14 *top*) The electrical power demand of a UK house over 18.5 days [13]. bottom) The top-1 ED motif (*left*) and the top-1 DTW motif (*right*) for an eight-hour query length.

As before, it is hard to fault the ED motif. It tells us that sometimes there is a relatively long lull in demand, followed by a sharp increase. However, the DTW motif tells us much more. There is a highly conserved pattern, that surely has some semantic meaning. Based on its timing (in the middle of the night) we suspect the following. Most power providers in England use a differential tariff to encourage users to shift power-hungry processes to run during the night, using off-peak electricity [8]. For many people, there are few options, unless they have high thermal mass heaters (i.e. underfloor heating). However, many modern European washing machines support the use of programable timers, so many people run their machines at night. The DTW-motif is surely a wash cycle.

One of the most common uses of motif discovery is in analyzing human behavior. It is natural to ask if DTW motifs are helpful in that context. To avoid the conflict of interest of producing our own new datasets to test this, we simply examined the human behavior datasets in the UCI machine learning repository [4]. Fig. 15 shows one sample experiment.

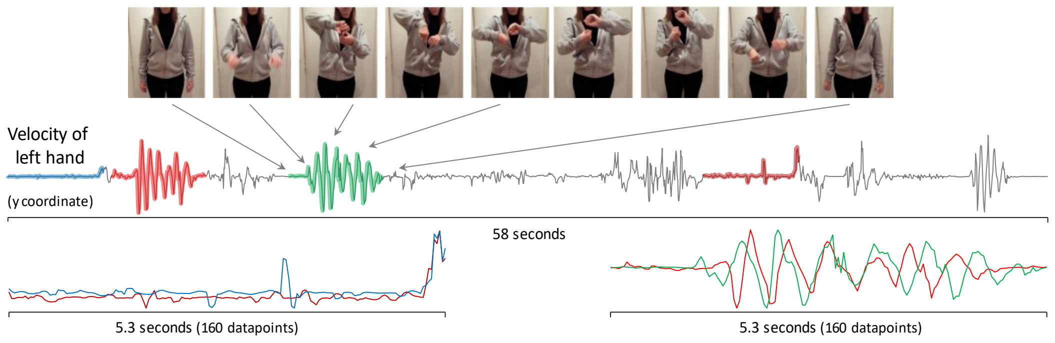


Fig. 15 *top*) A time series created by tracking the left-hand of a volunteer, using the Kinect system [6]. *bottom*) The top-1 ED motif (*left*) and the top-1 DTW motif (*right*) are very different. The ED motif corresponds to the rest position before two different gestures; however, the DTW motif is a repeated gesture.

As this figure hints at, DTW is often able to find repeated structure that defeats the Euclidean distance.

In Fig. 16 we performed a similar experiment using a different sensor (gyroscope), physically mounted on the body. Note that the amount of warping visible in Fig. 16.*bottom.left* is very small, but once again, it is enough to defeat the Euclidean distance.

Note that if we review the motifs discovered by ED in Fig. 1, Fig. 14, Fig. 15 and Fig. 16 they are all very similar, in spite of coming from different domains. Moreover, they are all "simple".

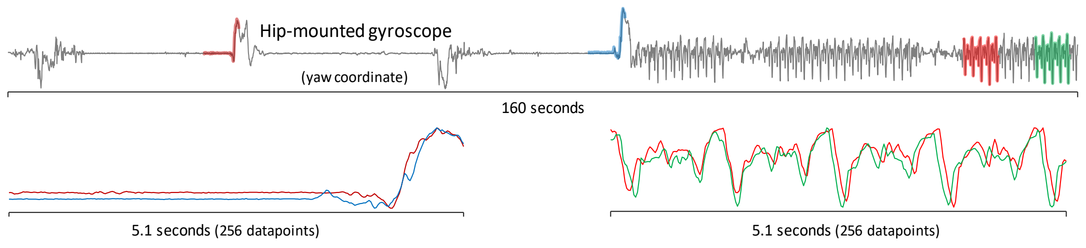


Fig. 16 *top*) A time series created by a hip-worn gyroscope [4]. *bottom*) The top-1 ED motif (*left*) and the top-1 DTW motif (*right*) are very different. The ED motif corresponds to two transitions (`lie-to-sit` and `sit-to-stand`), the DTW motif corresponds to periods of walking-downstairs.

This "simplicity bias" was observed in [5], which suggests a technique to bias the results away from simple motifs. However, it is not clear we can bias towards warped patterns.

4.2 An Example of DTW Motif Join

One of the useful implications of framing our hunt for DTW motifs as a Matrix Profile problem is that we can avail ourselves of the wealth of expanded definitions for the Matrix Profile [5][37][38]. In particular, here we show the utility of conducting time series joins. The Matrix Profile is defined for all types of joins. For example, classic motif discovery can be seen as a self-join. Here we are interested in a full-outer-join, equivalent to simply concatenating the two time series of interest, and running SWAMP on the result.

To demonstrate the utility of motif joins, we consider a dataset of cricket umpire signals. A carefully processed and contrived version of this dataset appears in the UCR Archive [6]. However, the dataset shown in Fig. 17 was recorded in the same session, but in natural uninterrupted sequences. We took the full z-axis right-hand acceleration time series from two participants and joined them. Because this is a full-outer-join, it is possible that subsequences from one person could join with themselves. That is exactly what happened with the ED motif, joining Graeme's sign for "*noball*" with his (only superficially similar) sign for "*six*".

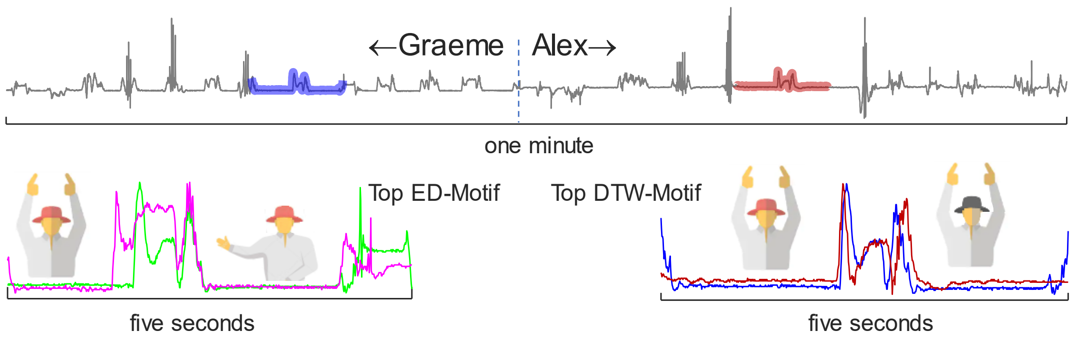


Fig. 17 *top*) A dataset consisting of Graeme signaling, followed by Alex signaling cricket umpire signs. *bottom.left*) The top-1 ED motif joins the sign for "*noball*" with the sign for "*six*". *bottom.right*) the top-1 DTW motif joins two signs for "*six*", one each performed by Alex and Graeme.

In contrast, the DTW motif joins examples of the sign for a "*six*", in spite of the fact that Alex signs it more leisurely than Graeme.

In passing, we foreshadow the scalability results in Section 4.4 by noting that for this experiment, 99.8562% of all candidate motif pairs were pruned in Phase I, and that by the end of Phase II, 99.999692% of all candidate motif pairs were pruned. Thus, only 0.000308% of the possible DTW comparisons needed to be made.

4.3 DTW Motifs Create Testable Hypotheses

We continue our examples with a case study that hints at one of the major uses of motif discovery, finding interesting hypothesis to explore. In Fig. 18 we show the results of DTW motif discovery on a dataset obtained by attaching a sensor to a wild penguin. The discovered motif is highly conserved (except for a little warping that defeats the ED), suggesting that it has some semantic meaning. However, what is that meaning? We do not know. However, if we plot the data with the simultaneous pressure reading, we see that this behavior seems to happen *just* as the swimming bird returns to the surface for a brief gulp of air before diving again. Testing this hypothesis on other datasets reveals it to be almost always true. However, understating the meaning/mechanism is ongoing work.

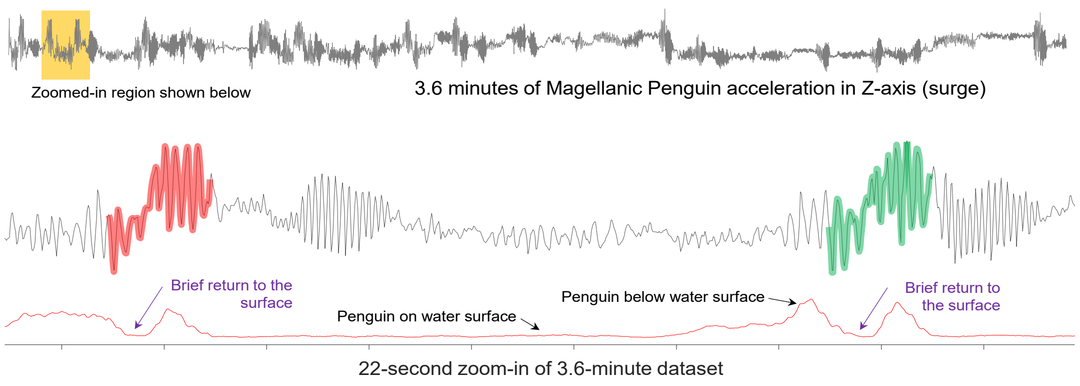


Fig. 18 *top*) Telemetry from a wild penguin hunting at sea. *bottom*) A zoom-in of the region that happens to contain the top-1 DTW motif (the ED motif is shown at [28] and is not obviously interesting). By aligning the motifs with a recording of pressure (red line) we find tentative meaning of the motifs.

4.4 Scalability of SWAMP

To demonstrate the scalability of SWAMP we revisited the experiments shown in Fig. 12, Fig. 14 and Fig. 16. We computed the time needed for brute force search. We measured the time needed for SWAMP. Finally, we also computed the time needed to find the best Euclidean motif, using the highly optimized state-of-the-art SCRIMP algorithm [38]. This comparison is unfair to us, as SCRIMP is returning a different, and much easier-to-compute answer than our algorithm. However, it offers what is surely an upper bound on the speedup that can be obtained. Fig. 19 shows the results.

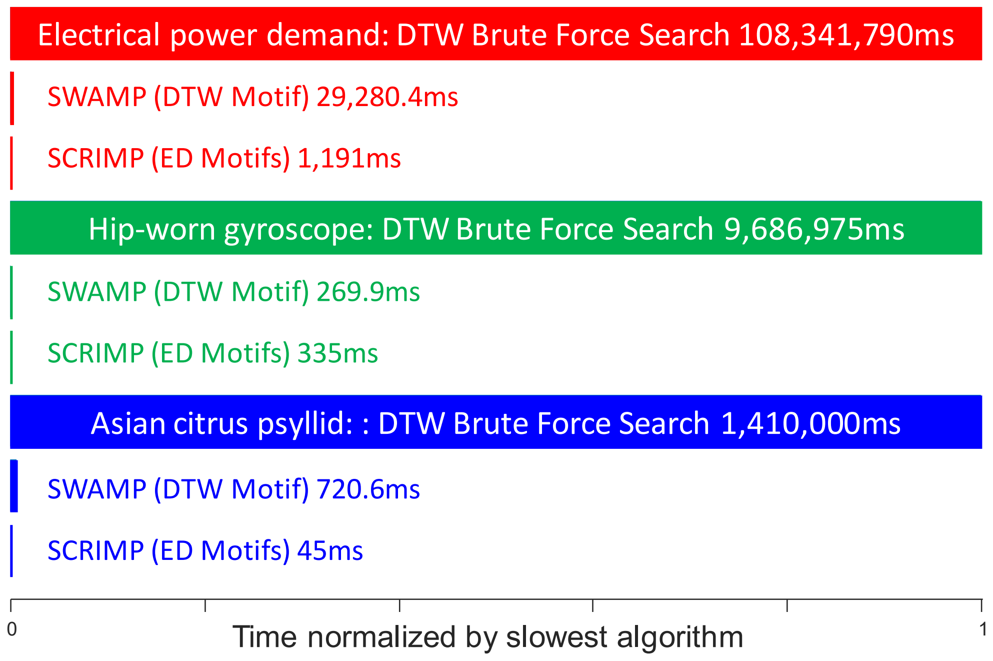


Fig. 19 The times required by three algorithms to find motifs in the three examples shown in Fig. 14, Fig. 16, and Fig. 12. Note that the bars are normalized by slowest performing algorithm, i.e. brute force search.

The results can be summarized as follows. SWAMP is two to three orders of magnitude faster than brute force search, and an order of magnitude slower than the fastest *Euclidean* motif algorithm.

These experiments also offer us a chance to do a lesion study. We spent considerable effort motivating the need for a spectrum of lower bounds in Phase I of our algorithm (Table 1, lines 6-13). Suppose instead we only use the highest resolution, $LB_{Keogh}1:1$. The returned answer would clearly be the same, but how would this affect speed? We tested this, discovering that the time needed increased by 187%, 1,028% and 113% respectively, showing that the "*use the cheapest lower bound you can*" approach really does help.

Finally, we compared to [28], which is the only other exact algorithm for finding DTW motifs. On the three datasets above this algorithm was slower by 17,274%, 185,511% and 13,857% respectively.

Given the utility of our algorithm for several data mining tasks, we chose to conduct additional detailed experiments which we discuss in the following. Fig. 20 shows the time to compute and pruning rate for motif discovery with fixed subsequence length 400, fixed warping window size 16 and increasingly long time series. The time series in question is an extended version of the household electrical power demand dataset we used in Section 4.1.

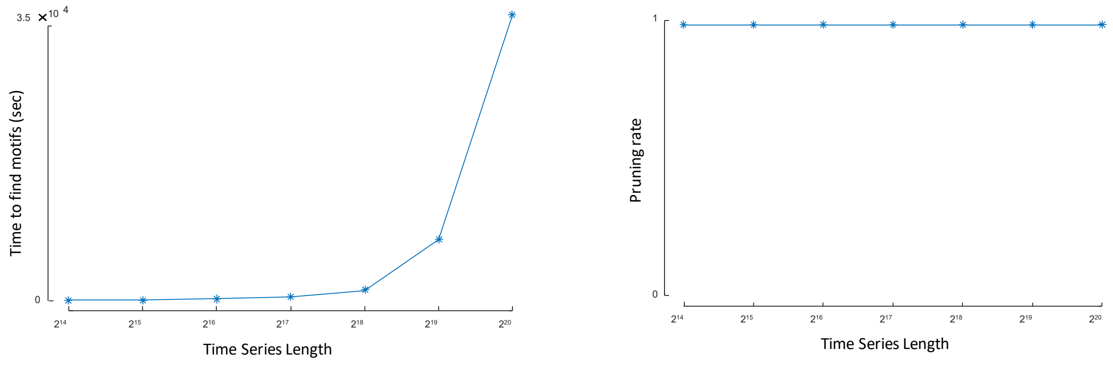


Fig. 20 Time to compute SWAMP increases as we increase the length of the time series. However, since we have pruning rate of almost 100% in every case, the increase would be tolerable for very large datasets.

Clearly, the time to compute SWAMP would increase as we increase the length of the time series. However, as shown in Fig. 20.*right*, the pruning rate remains as high as 1 for all lengths, meaning that we avoid computing most of the true DTW Matrix Profile even for very large datasets. With a dataset as large as 1,000,000 data points, the time to compute SWAMP is as low as eight minutes as shown in Fig. 20.*left*.

4.5 Objective Evidence of the Superiority of DTW over Euclidean Distance

As shown in previous sections, DTW motif discovery can be used to spot conserved patterns in real datasets. However, the comparison to Euclidean distance was mostly anecdotal. In this section we will compare the utility of DTW over Euclidean distance on a large-scale experiment with real and complex data.

In Fig. 21, we show an example of a "sentence" created by concatenating words spelled out by the eye movements of an individual modeling Locked-In Syndrome [9]. The participants learn a code to translate words into a sequence of eye-movements, in which the eye traces along the eight cardinal directions of the compass (and "`blinking`" as a special character). For example, the word "*moth*" (ガ) can be communicated the sequence: "`left right down left upper-right lower-left blinking`". These eye movements can be tracked with an inexpensive apparatus and can then be used to transcribe movement-to-text.

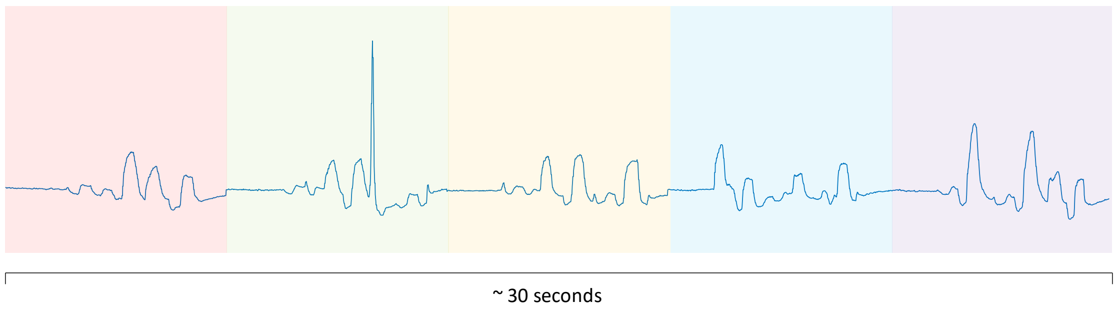


Fig. 21 A time series corresponding to a sentence (in Japanese) spelled out by the eye movements of an individual modeling Locked-In Syndrome. Only vertical axis is shown. Each colored box shows one word. All words have been rescaled to exactly the same length, to facilitate comparison to Euclidean distance.

To compare the accuracy of DTW and Euclidean distance on the task of discovering conserved structure, we propose the following experiment. From a vocabulary of 150 words, each of which was performed three times by a single individual, we randomly create a sentence that has exactly *one* repeated word. As shown in Fig. 21, because the default position of the eye at the beginning and end of word is straight ahead, we can concatenate words without producing any obvious artifacts in-between them.

We generated sentences consisting of 8, 16, 32 and 64 words. We performed one hundred runs of motif discovery on these time series using both ED (using the Matrix

Profile [37][38]) and DTW (using SWAMP with $w = 24$). While all words are of length 600 (possible including a relatively constant prefix and suffix, because the participant was in a relaxed state of looking straight ahead), we wanted to avoid given the algorithm's this exact value, so for every experiment we gave both ED and DTW a random subsequence length between 540 and 660. Fig. 22 shows the motif discovery success rates for different algorithms.

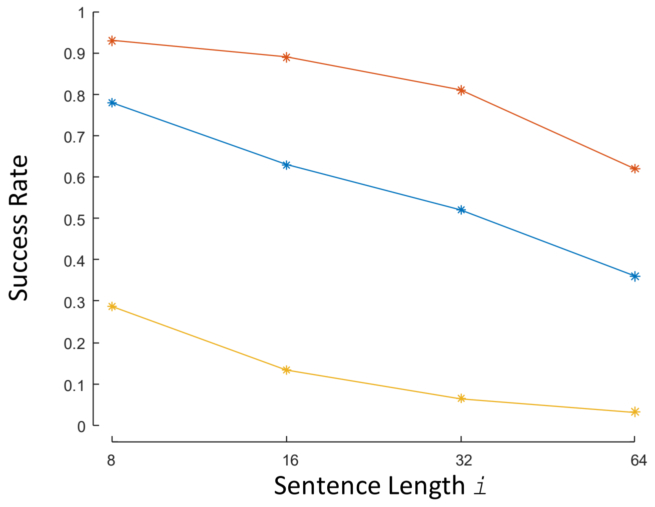


Fig. 22 SWAMP(*red*) with a warping window size 24 performs better than both ED MP(*blue*) and the default rate(*yellow*) in finding the correct motifs on different settings.

Both ED and DTW work much better than the default rate (random guessing in proportion to the prior probability of events). However, DTW is clearly superior to both.

Let use briefly consider the sources of error. Because each time series is generated by concatenating words without any pause or noises in between, we might create "artificial" repeated words. For example, suppose the unique words we embedded happened to include "`wombat`", "`mangos`" and "`batman`". When embedded into a sentence they could form: ...`wom`**`batman`**`gos`**`batman`**..., making an accidental motif of "`batman`", which really has only one occurrence. This issue is more likely in the domain under consideration, which has a cardinality of just nine distinct symbols. Fig. 23 shows one such example of a spurious word we discovered.

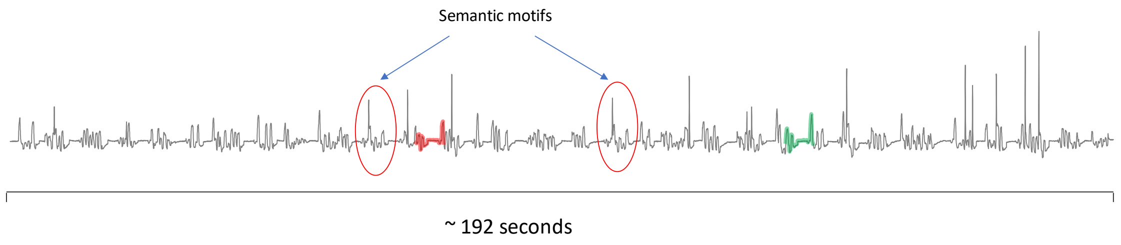


Fig. 23 The suffix and prefix of two words concatenated to each other (*red*) is similar to the suffix and prefix of two other words concatenated to each other (*green*). These two words have been mistaken with the semantic motifs (showed by the red circle).

As shown in Fig. 24, another reason why SWAMP can fail here is simply because we are only using the X-axis time series. This motivates the need for multidimensional DTW motif discovery. The issues in generalizing to multidimensional DTW are subtle (should we allow each dimension to warp independently, or force them to warp in synchronicity? See [27]), however they are orthogonal to SWAMP's speed-up mechanisms.

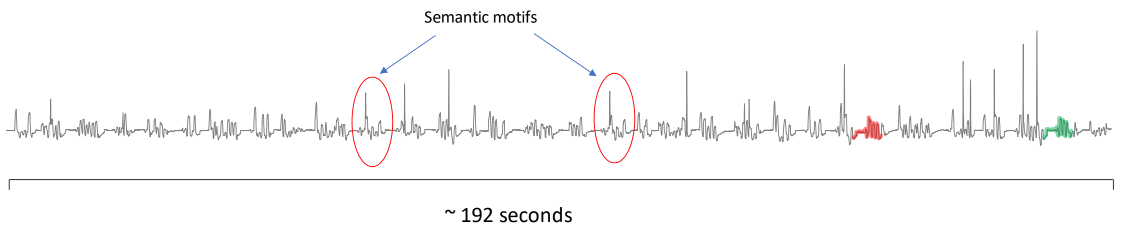


Fig. 24 Two different words that are more similar (*red* and *green*) than the embedded motifs (showed by the red circle) when considering only the vertical axis.

Finally, we should note that even the research effort that produced this dataset, and introduced a custom classification model that included information from the *shape* of the time series, and linguistic information (which we ignore), only manage to achieve a 81.2% classification accuracy, so some error seems intrinsic to this domain.

4.6 Discussion

We conclude with a discussion that will help practitioners decide if they should use DTW or ED when searching for motifs, and also help them decide on a warping window size. Note that the second question really subsumes the first, as in the special case of w = 0, DTW is logically identical to ED.

In some domains there is a *Zeitgeber*[2], an external stimulus that synchronizes processes. In nature, this can be daily, lunar or annual cycles. In culture, this can include the

[2] German for "time-giver", *Zeitgeber* is normally only used for biological processes, here we extend the meaning to social and cultural processes

weekly cycles of the typical nine-to-five constraints of the western workday. For example, Fig. 25.*left* shows pedestrian traffic outside a train station in Melbourne for two randomly chosen weeks. There are differences between two weeks, but they can mostly be explained by changes in *volume* at a given time. For example, a school holiday reducing lunch time traffic on Monday. Likewise, some physical devices (mostly solid state) can produce highly regular outputs. Of course, these distinctions may not be hard and fast. A pacemaker may give an otherwise erratic pulse rate a metronome-like regularity. Such "Zeitgeber time series" will rarely benefit from DTW.

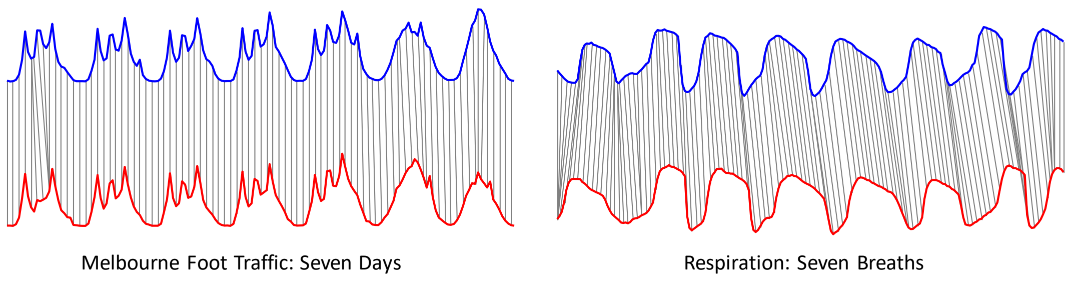


Fig. 25 Two pairs of time series subsequences with seven peaks, aligned with DTW. For the foot traffic dataset, the DTW alignment is *almost* linear, essentially the Euclidean Distance. In contrast, respiration data has highly non-linear alignment.

In contrast, as Fig. 25.*right* hints at, many biological signals can have similar shapes but develop at varying rates of time. This is also true of many physical processes. For example, the motif shown in Fig. 14.*bottom.right* was created by an integrated circuit (IC) controlling a device. One might imagine that the IC would produce perfect timing. However, it is at the mercy of the varying water pressure and water temperature in the house.

4.7 SWAMP Variants

In the last forty years there have been many modifications or extensions of DTW proposed. A twenty year old classic reference lists dozens of variants [26], and the pace of research has greatly accelerated since then [11][15][27][29]. We believe that the SWAMP algorithm can support most or all variants of DTW. Moreover, there may be reasons to use some of these variants, not least because they may be faster. In the following section we discuss this idea.

4.7.1 Itakura Constraint on Warping Path

Virtually all works on DTW define a global constraint which determines how far the warping path is allowed to deviate from the diagonal. Up to this point, we have used the Sakoe-Chiba constraint [21], which is the most commonly used variant by the data mining community [6][15][22][28][29][30]. However, there exists other constraints including the Itakura parallelogram [24]. Fig. 26 illustrates the two schemes.

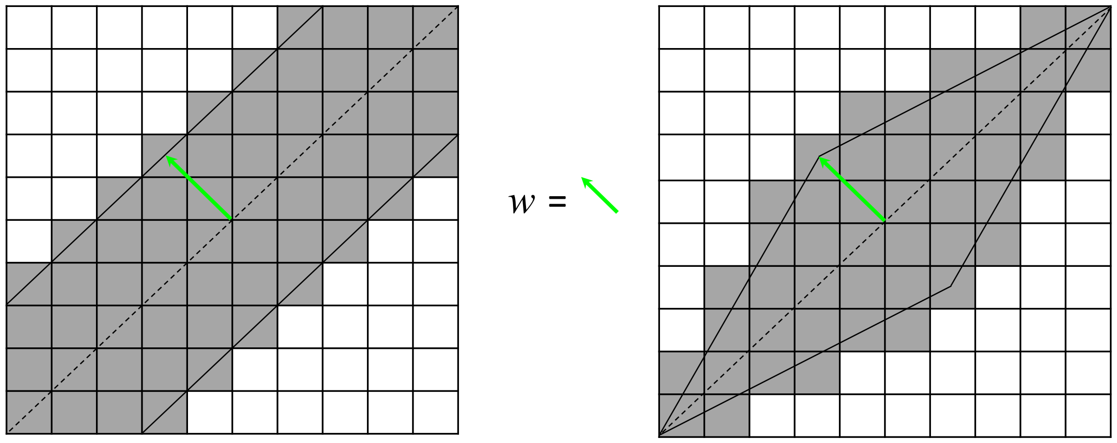


Fig. 26 Global constraints limit the scope of the warping path to the grey areas. The two most common constraints are *left*) Sakoe-Chiba band, and *right*) Itakura parallelogram. *w* is a term defining allowed range of warping for a given point in a sequence.

As we will show below, there are two reasons to expect that the Itakura parallelogram could be faster than Sakoe-Chiba band for motif discovery. However, if we are to advocate the Itakura parallelogram, we must first address the quality of results it can return. It seems to be generally believed by the data mining community that the Itakura constraint is inferior to the Sakoe-Chiba band. For example, a recent head-to-head comparison of the two methods on eighty-five datasets finds "…*although the Itakura parallelogram is generally inferior to the Sakoe-Chiba band*..." The clustering experiment shown in Fig. 27.*top.panel* seems to confirm this.

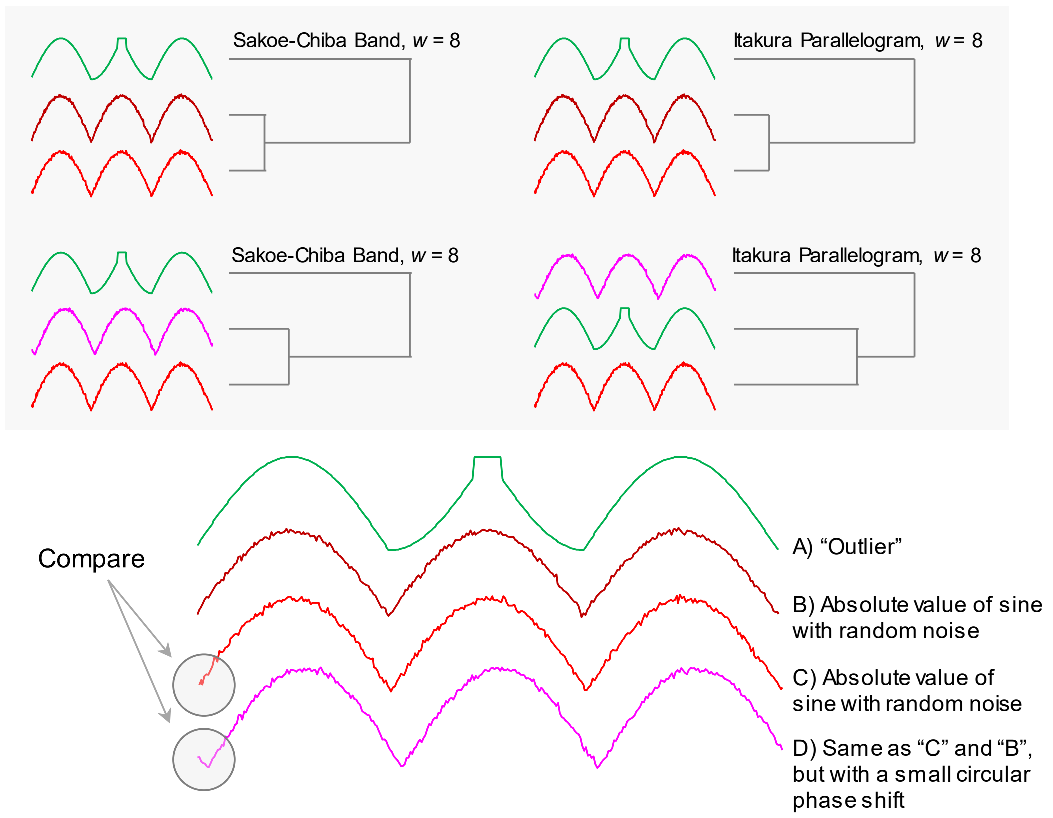


Fig. 27 *top.panel*) Three data objects, A, B and C, clustered using either the Sakoe-Chiba band or the Itakura constraint produce essentially identical results. However, if we replace C with D, which is almost identical, but just slightly circularly shifted (see *bottom.panel*) the Sakoe-Chiba band is largely unaffected, but the Itakura constraint produces an unintuitive clustering, grouping C and A together, and considering D as the outlier.

However, these results are based solely upon data from the UCR archive [6]. Most of these datasets consist of individual exemplars extracted from a longer time series. For example, individual heartbeats extracted from an electrocardiogram. Sometimes, these individual heartbeats, gait cycles or gestures are not perfectly extracted, and may have small artifacts at their beginning or end. This is modeled by the data object D shown in Fig. 27.

This issue is compounded by the fact that most heartbeats extraction algorithms, and gait-cycle extraction algorithms tend to define the beginning point of a cycle at the most dynamic locations of the cycle. For heartbeats this is the peak of the R-wave, and for gait it is (typically) the heel strike. Both warping methods can tolerate differences between two time series that happen towards the middle of the time series, but as we get closer to either of the ends, the narrowing apex of the parallelogram mean that any differences are more keenly felt by the Itakura approach. The issues caused by these small artifacts at the Prefix and Suffix of a time series have been noted before in a classification context [29], and techniques have been suggested to solve this problem. However, this issue is largely irrelevant if we generalize from the (somewhat unnatural) UCR-contrived classification setting and consider subsequence similarity search or motif discovery. In such case we are implicitly or explicitly "sliding the subsequences against each other" and reporting the smallest distance. Thus, global misalignment of patterns (as opposed to local misalignments addressed by DTW itself) are not an issue.

To summarize, while the community may be correct to (slightly) prefer Sakoe-Chiba band over the Itakura constraint for *classification* of extracted time series snippets [11] this preference does not seem to have implications for *motif discovery* from long streams. Moreover, as we explain below, the Itakura approach can produce tighter lower bounds, hence speeding up our algorithm.

4.7.2 Exploiting the Itakura Speed up Part I: Tighter Lower Bounds

Recall that we defined the upper and lower envelopes enclosing a time series **Q** in Eq. 2.**Error! Reference source not found.** In this equation *w* is the band or the maximum allowed range of warping. In the case of Sakoe-Chiba, *w* is independent of the index *i* of the time series. However, for Itakura it is a function of *i*. Fig. 28 shows the envelopes created for the time series **Q** using the two schemes.

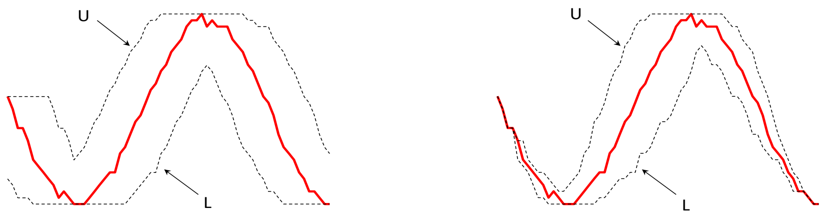


Fig. 28 An illustration of the envelopes U and L, created for time series **Q** (shown in red), using *left*) the Sakoe-Chiba band and *right*) the Itakura parallelogram.

We defined the lower bounding function between the time series **T** and **Q**, i.e. LB$_{\text{Keogh}}$(**T**,**Q**), in Eq. 1. The example in Fig. 4 shows the lower bound generated using the Sakoe-Chiba band. Fig. 29 illustrates the lower bounds generated using the Sakoe-Chiba and Itakura for the same time series in Fig. 4.

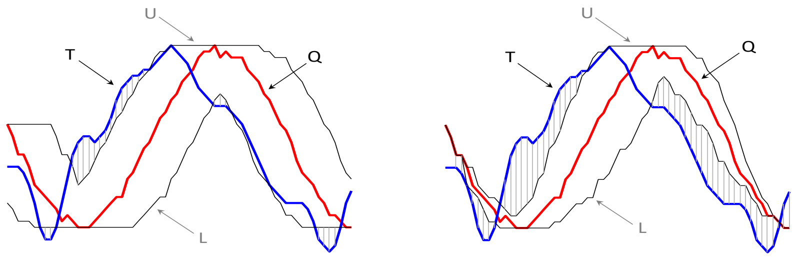


Fig. 29 An illustration of the lower bounding function $LB_{Keogh}(\mathbf{Q},\mathbf{T})$. Time series **Q** (shown in red), is enclosed in the bounding envelope of U and L using *left*) the Sakoe-Chiba band and *right*) the Itakura parallelogram.

Since the tightness of the bounds is proportional to the area filled with the gray hatch lines, we can see that in this example the Itakura parallelogram provides a tighter bound than the Sakoe-Chiba band.

The reader will appreciate that with equal *w* in these two cases, the parallelogram always produces a tighter lower bound. It is suggestive of a significant speed up. However, it is not necessary the case. Recall that for the SWAMP algorithm, the speed depends upon both tightness of lower bounds and the value *best-so-far*. If the *best-so-far* is small, the algorithm can prune more efficiently. While the Itakura parallelogram has a tighter lower bound, the motif distance under the Itakura parallelogram can be higher, if there is significant warping near the beginning or end of the motif. If that is the case, the final *best-so-far* will not be as low as in the Sakoe-Chiba case. It is not obvious how these competing factors will affect the final result, but a brief review of some previous results gives us hope. Consider again the DTW motifs discovered in Fig. 2, Fig. 13, Fig. 14 and Fig. 15. In each case the warping variability is concentrated in the center of the subsequence. This means that the Itakura and Sakoe-Chiba distances will be almost identical, but as noted above, the Itakura parallelogram will have a much tighter lower bound. To compare the speed-up using the Itakura band against the Sakoe-Chiba for our algorithm, we repeated our experiments for three datasets described in Fig. 19, this time using Itakura constraint. The discovered motifs in every case remained essentially the same. However, the time needed to compute SWAMP changed as follows: On the three datasets in Fig. 14 and Fig. 16 and Fig. 12 the Itakura parallelogram was faster by 80%, 9% and 80%.

4.7.3 Exploiting the Itakura Speed up Part II: Earlier Early-Abandoning

While the results in the previous section suggest that Itakura is more efficient than Sakoe-Chiba in some datasets, there is still another observation that we can exploit.

Normally when we want to compute the lower bound the LB_{Keogh} lower bound, we do it in a left-to-right order. However, if we want to early abandon as early as possible, we should do it in order of largest (in expectation) value first. This way, the incrementality computed error grows as fast as possible, and as soon as it exceeds the *best-so-far* we can abandon.

For the Sakoe-Chiba, every point on the subsequences has the same expected contribution, hence our use of simple left to right. However, as Fig. 29 shows, Itakura parallelogram tends to have most of its lower bound contribution at either end, where the envelopes are thinner. This suggests following heuristic: sort the indices in the ascending order of their distance between lower and upper envelopes and compute the lower bound in that order. This means visiting the endpoints of the sequence first and then moving towards the middle points. To be clear, instead of scanning the indices of Eq. 3 in the order 1, 2, 3,.. ,L-1, L, we visit them in the order 1, L, 2, L-1, 3, L-2,.., L/2. As before, as soon as the distance between the endpoints of two sequences is higher than the *best-so-far*, we can stop the lower bound calculation as shown in Fig. 30.

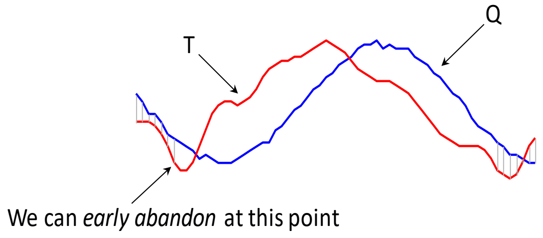


Fig. 30 An illustration of early abandoning of LB_{Keogh} using Itakura constraint. We have a *best-so-far* value of bsf. After incrementally summing the first fourteen (of sixty four) individual contributions to the lower bound (seven on each endpoint) we have exceeded bsf, thus it is pointless to continue the calculation [14].

To see what difference this optimization makes, without regard to implementation dependent details of a language, we revisited Phase II of the experiment shown in Fig. 14. We measured at what point the early abandoning could actually abandon for all comparisons in Phase II. Fig. 31 shows the results.

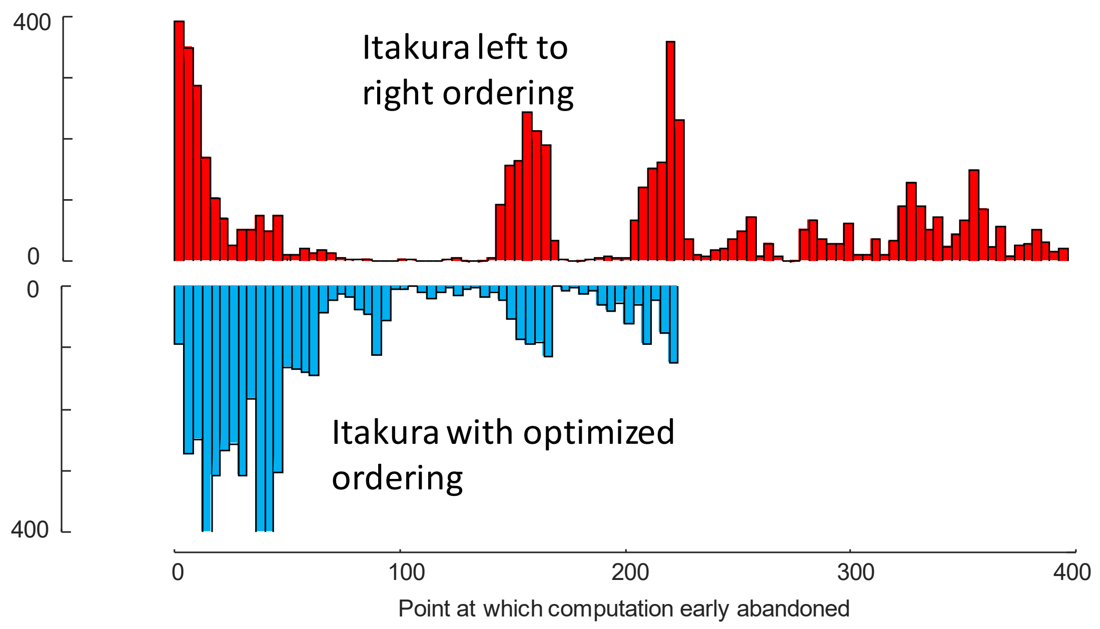


Fig. 31 The distribution of early abandoning offsets for all comparisons in Phase II of the experiment in Fig. 14. The distribution is spread out over the whole range of values with the left to right ordering (*red*), while it is mostly skewed to the beginning offsets with the optimized ordering (*blue*).

Note that in the act of sorting the indices we have been able to shift the early abandoning to the most beginning offsets, cutting the number of all point-wise comparisons to almost half the length of the sequence, as shown in Fig. 31. However, without the optimized ordering, the distribution would be spread out over the whole range of values from zero to the full length of the sequence.

We refer the interested readers to the companion website [41] where we have made available all the source codes for this algorithm to download and execute.

5 A CASE STUDY IN USING SWAMP TO SUPPORT A CLASSIFICATION TASK

We conclude this work by showing how SWAMP can be used to help build a time series classification algorithm. There are literally hundreds of time series classification algorithms in the literature [1]. However, the vast majority of them only consider the UCR archive datasets or similar data sources, which have had individual examples carefully extracted, normalized and processed. As [7] and others have recently noted, these works largely bypass the real difficulty of creating a practical time series classifier. We argue that the key question is how we can extract high quality exemplars from noisy and weakly labeled data, and estimate a distance threshold. The latter issue is typically glossed over by researchers that rely on the UCR archive datasets to motivate and test their contributions. For example, one of the most famous datasets in the UCR archive is Gun/Point, which tasks us with the problem of discriminating between aiming a gun, and merely pointing with a finger. It is obvious that in any practical situation, most of the time, an individual is doing *neither* action. This suggests that there should be a third, highly polymorphic class, `neither`, with a very high prior probability. It is not clear how most proposed methods would general to this more realistic setting.

Here we present an end-to-end example of how SWAMP can be used to build a classifier. This is meant to be a demonstration; a more rigorous evaluation is beyond the scope of this paper. For simplicity, we will confine our attention to a *single* time series, the Z-axis gyroscope on the right wrist. However, generalization to multidimensional data is straightforward.

We consider the OPPORTUNITY Activity Recognition Dataset [2]. This activity recognition environment and scenario has been designed to generate many activity primitives in a realistic setting. Subjects operated in a room simulating a studio flat with furniture and kitchen. The subjects were monitored using body-worn sensors. In addition, various items and utensils also had sensors, some binary (such as door open/closed) and some real-valued acceleration values (including the cup). It can be helpful to transform these real-valued data into binary data, equivalent to "*cup was moved*".

The data creators "*instructed users to follow a high-level script but leaving them free interpretation as how to achieve the high-level goals*." In one experiment they asked the user to perform a drill comprising of the following actions:

1. Open then close the fridge
2. Open then close the dishwasher
3. Open then close 3 drawers (at different heights)
4. Open then close door 1
5. Open then close door 2
6. Toggle the lights on then off
7. Clean the table
8. Drink while standing
9. Drink while seated

As Fig. 32 shows, even if we know this script, it can be difficult to semantically segment this data. Fortunately, we can use the sensors on the implements to at least weakly label this data. For example, if there is significant acceleration of the cup, this is suggestive of either the `Drink-while-standing` or `Drink-while-seated` class. However, there are two reasons why this does not completely solve our problem at hand. First, it is possible that the cup could move during other behaviors, especially `Clean-the-table`. Second, even if we are sure that the motion of the utensils is associated with a particular class, we cannot be sure exactly what part of the behavior is conserved. For example, it may be the case that some conserved motion *before* the cup is lifted is conserved (as the user reaches for it). It may also be the case that some motion during drinking is *not* well conserved. For example, perhaps the path from the table to the user's lip is well conserved, but once at the lip, the level of the liquid in the cup specifies the amount of rotation needed to imbibe, thus that part of the behavior is not well conserved. To be clear, this is pure speculation on our part. However, most classification tasks surely have similar uncertainties.

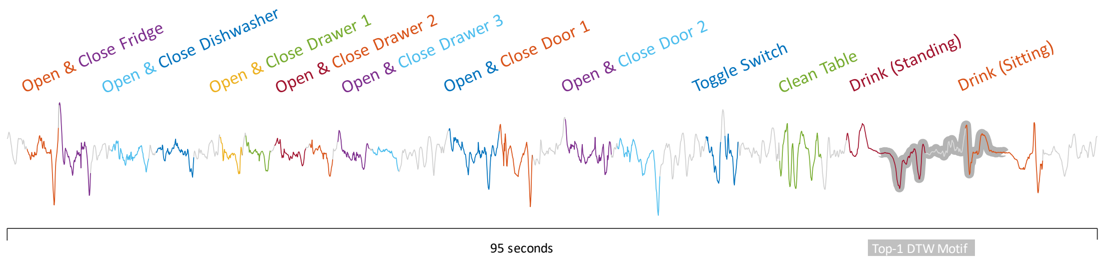


Fig. 32 A sequence of nine activities performed by the user in Opportunity Activity dataset.

This motivates our approach. Even in this most familiar domain, we cannot tell from prior knowledge what part of a behavior is conserved. However, SWAMP can find the most conserved patterns in the time series. If these motifs happen to approximately line up with a weak label for a behavior, we can assume that the motif is likely to be a good prototype to recognize future occurrences of that behavior. Moreover, even if we have only two weak labels, we have at least a starting point to produce a threshold. The motif distance reflects the smallest match between two instances; as such it is clearly a lower bound to a good threshold value, but clearly it is at least the correct order of magnitude.

Our experiments consist of two parts. The first part demonstrates use of SWAMP for intra-subject activity classification. We chose the motion data corresponding to the Right Wrist Inertial Measurement (RLA) Gyroscopic Z. Training was done on the first ten repetitions of the activity set. The first test was done on the remaining ten repetitions in the same Drill 1 session. Fig. 33 shows the top motifs discovered from the training data. The input time series was of length 28,330 with a subsequence length of 300 (chosen to reflect the approximate length of for `Drink-while-standing`) and a warping window of 4.

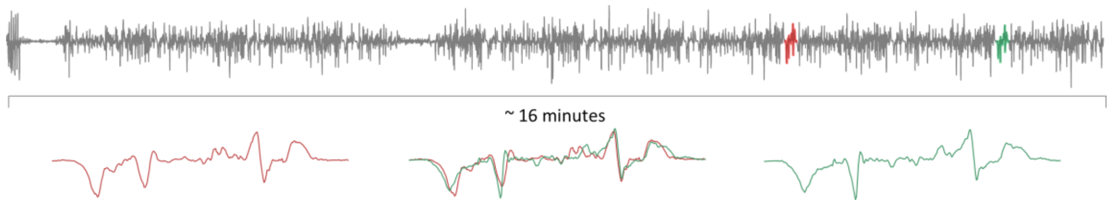


Fig. 33 About sixteen minutes of the activity sequence associated with the first Drill session of the Opportunity dataset. The top DTW motifs was the pair [20102, 25575] and corresponds to `Drink-while-standing`, with a distance of 4.41. The best Euclidean distance motif was the pair [3867, 20098] with a distance of 6.93, which also corresponds to `Drink-while-standing`.

To classify the test data, we computed the DTW distance between the first top motif in the train data and each test subsequence as shown in Fig. 34. We then validated the matches against the weak labels associated with the time series as shown in Fig. 38. The weak labels were pre-processed before the classification. First, we z-normalized Gyroscope Z-axis. The results were set to binary values by using a threshold value of 2.0. There are two events of drinking from cup in the dataset, `Drinking-while-standing` and `Drinking-while-seated`. We manually removed activity for `Drinking-while-seated` which is visually straightforward. Finally, we activated all points within 2 seconds of activity (i.e. 60 samples). All locations where the distance is minimum on Fig. 34 correspond to the activity of `Drink-while-standing` which have been correctly classified using DTW as shown in Fig. 38. Here the default rate is just 9.8%.

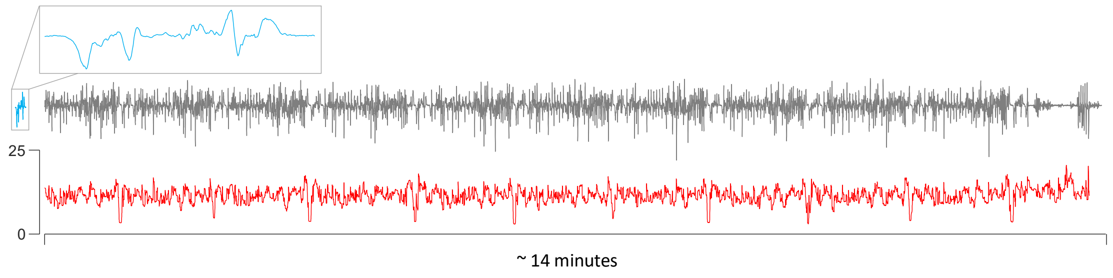


Fig. 34 DTW query distances (*red*) between the training set's first top motif (*blue*) and the first test set (*grey*).

This test demonstrates that even though an activity may be the top-1 motif for both DTW and ED, DTW outperforms ED when querying for all instances of such an activity. If there is a phase variation in this activity, ED will have difficulty, especially if the motif is complex. In this first test, ED performs closely to DTW, but we speculate that intra-subject phase variations are minimal when performing a monotonized task like repeating a set of activities twenty times.

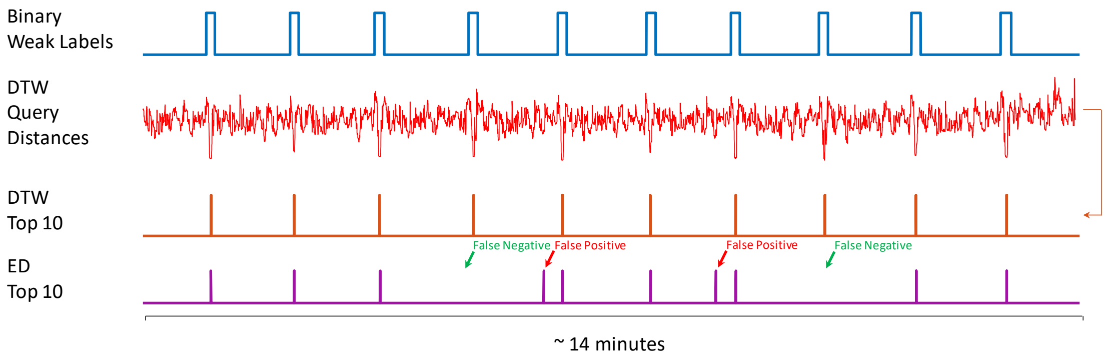


Fig. 35 The intra-subject variability of the `Drink-while-standing` activity is well captured by both ED and DTW. However, DTW outperforms ED with no false positives. The reason that ED has some FP so close to TP in some cases is that it is matching with `Drink-while-sitting` rather than `Drink-while standing`.

The second experiment uses the same training data as the first (i.e. first ten activity set repetitions of the right wrist time series in Drill 1), but now we consider inter-subject variability. The test data from the Drill 2 session also corresponds to the right wrist motion. Fig. 36 shows the test data and the DTW distances between the first top-1 motif in the train data and each test subsequence. Fig. 37 illustrates the classification results for this time series.

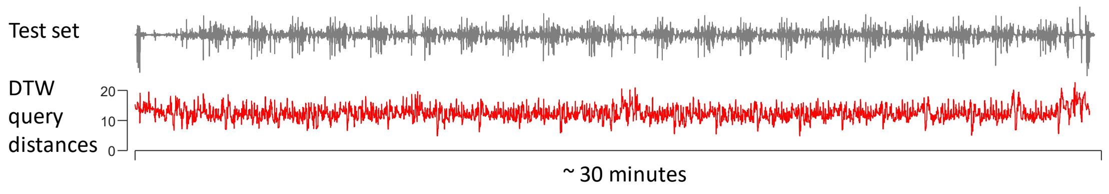


Fig. 36 DTW query distances (*red*) between the training set's first top motif and the second test set (*grey*).

As Fig. 37 shows, the inter-subject variability of the `Drink-while-standing` activity is well captured by DTW. Out of twenty events on this time series, seventeen events have been correctly classified (i.e. true positives) while three events have been missed (i.e. false negatives). Compare it to the results from ED where only nine have been correctly classified. Note that here the default rate is just 11%. This test demonstrates that DTW queries using a motif of motion data can effectively be used as an activity classifier despite inter-subject variability.

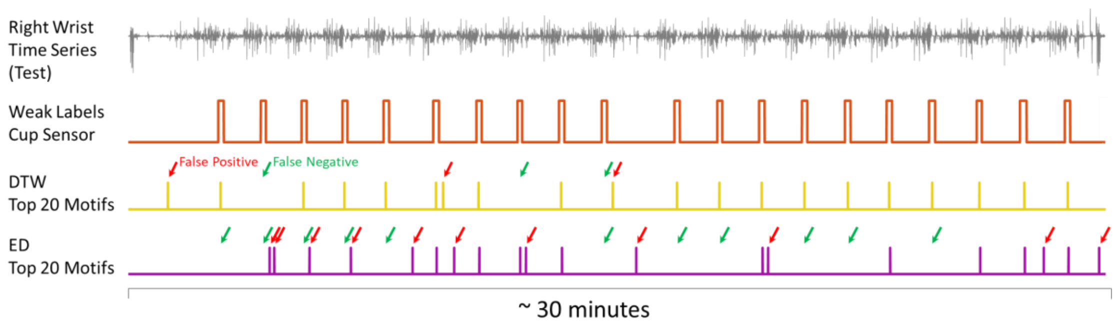


Fig. 37 DTW correctly finds seventeen instances of the `Drink-while-standing` activity out of twenty events. However, ED discovered only nine correct instances.

Without motif discovery, a reasonable idea would have been to assume that the beginning of the cup activation indicated the beginning of the pattern to extract in the Gyroscopic Z axis. We also tried this, using the same length query. Since there are twenty such locations in the training data, we tried all of them. The average result is 15.7 true positives and 4.3 false positives. Recall that using SWAMP gave use seventeen true positives and three false positives. This suggests that the simple heuristic of using the most conserved pattern is a promising idea.

While the above results are limited, they clearly show the utility of using motifs as a starting point in any attempt to build a time series classifier from the ground up.

6 CONCLUSION AND FUTURE WORK

We have introduced SWAMP, the first practical tool to find DTW-based motifs in large datasets. Moreover, we have shown that on many real datasets, DTW returns more meaningful motifs.

SWAMP offers many avenues for improvement. The time the classic Matrix Profile algorithms takes depends only on *n*, thus it is possible to know how long it will take to find the motifs and build a perfectly accurate progress bar for an interactive tool. In contrast, the time required for SWAMP also depends on *L*, *w* and the data itself. We would like to be able to tell the user (at least approximately) how long our algorithm will take to finish.

The time for brute force DTW motif discovery is completely dominated by the cost of computing DTW. However, SWAMP is dominated by the time computation of the lower bounds. There has been little effort to optimize the time needed to compute these bounds, because they are most commonly used for disk-based indexing, which is itself dominated by I/O costs. Thus, we suspect that lower bounds computation may be amiable to many further algorithmic and implementation optimizations. Such optimizations could be trivially plugged into SWAMP to improve its performance. We also plan to explore the possibly of framing SWAMP as an *anytime* algorithm [38].

ACKNOWLEDGEMENTS

We thank all the creators of the data sets used in this work.

REFERENCES

[1] A. J. Bagnall, et. al. The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances. Data Mining and Knowledge Discovery. 31(3): 606-660 (2017).

[2] R. Chavarriaga, et. al. The Opportunity challenge: A benchmark database for on-body sensor-based activity recognition. Pattern Recognition Letters, 2013

[3] B. Chiu, E. Keogh, and S. Lonardi, Probabilistic discovery of time series Motifs, Proc. of SIGKDD'03, pp. 493–498, 2003.

[4] D. Dua, and C. Graff. (2019). The UCI Machine Learning Repository. http://archive.ics.uci.edu/ml

[5] H. A. Dau, and E. J. Keogh. Matrix Profile V: A Generic Technique to Incorporate Domain Knowledge into Motif Discovery. KDD 2017: 125-134.

[6] The UCR Time Series Classification Archive. https://www.cs.ucr.edu/~eamonn/time_series_data_2018

[7] H.A. Dau, et. al. The UCR Time Series Archive. IEEE/CAA Journal of Automatica Sinica. vol.6 no.6, November 2019.

[8] Economy 7 (Wikipedia). http://en.wikipedia.org/wiki/Economy_7

[9] F. Fang and T. Shinozaki. Electrooculography-based continuous eye-writing recognition system for efficient assistive communication systems. PloS one, 13(2)

[10] R. A. Feitosa, et. al. Multidimensional Representations for the Gesture Phase Segmentation Problem - An Exploratory Study using Multilayer Perceptrons. ICAART (2) 2018: 347-354.

[11] Z. Geler, V. Kurbalija, M. Ivanović, M. Radovanović and W. Dai. Dynamic Time Warping: Itakura vs Sakoe-Chiba. 2019 IEEE International Symposium on INnovations in Intelligent SysTems and Applications (INISTA), Sofia, Bulgaria, 2019, pp. 1-6.

[12] X. Gong, et al. Fast similarity search of multi-dimensional time series via segment rotation. DSAA 2015.

[13] L. Junkui, W. Yuanzhen, and L. Xinping. LB HUST: A symmetrical boundary distance for clustering time series. (ICIT'06). IEEE, 2006.

[14] E. Keogh, L. Wei, X. Xi, M. Vlachos, S.H. Lee, and P. Protopapas. 2009. Supporting exact indexing of arbitrarily rotated shapes and periodic time series under Euclidean and warping distance measures. VLDB J. 18, 3, 611-630.

[15] E. Keogh, and C. A. Ratanamahatana. Exact indexing of dynamic time warping. KAIS 7.3 (2005): 358-386.

[16] D. Lagun, et. al. Discovering common Motifs in cursor movement data for improving web search. WSDM, 2014.

[17] D. Minnen, et. al. Discovering Multivariate Motifs using Subsequence Density Estimation and Greedy Mixture Learning, 22nd Conf. on Artificial Intelligence (AAAI'07), 2007.

[18] A. Mueen, et. al. Exact Discovery of Time Series Motifs. SDM 2009: 473-484.

[19] D. Murray, et. al. A data management platform for personalized real-time energy feedback. In Proceedings of the 8th, EEDAL'15.

[20] Pareto Efficiency (Wikipedia). http://en.wikipedia.org/wiki/Pareto_efficiency

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

[22] T. Rakthanmanon, et. al. Addressing Big Data Time Series: Mining Trillions of Time Series Subsequences Under Dynamic Time Warping. TKDD 7(3): 10:1-10:31 (2013).

[23] C. A. Ratanamahatana and E. Keogh. (2005). Three Myths about Dynamic Time Warping. (SDM '05), pp. 506-510.

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

[25] S. Salvador and P. Chan, FastDTW: Toward Accurate Dynamic Time Warping in Linear Time and Space. Intelligent Data Analysis, vol. 11, no. 5, 2007, pp. 561-580.

[26] D. Sankoff and J. Kruskal (2000). Time Warps, String Edits, and Macromolecules – The Theory and Practice of Sequence Comparison (updated reprint of a 1983 book). ISBN 1-57586-217-4.

[27] M. Shokoohi-Yekta, J. Wang, and E. Keogh. On the Non-Trivial Generalization of Dynamic Time Warping to the Multi-Dimensional Case. SDM 2015: 289-297

[28] D. F. Silva, and G. E. A. P. A. Batista. Elastic Time Series Motifs and Discords. (2018) ICMLA, 2018, pp. 237-242.

[29] D. F. Silva, G. E. A. P. A. Batista, E. J. Keogh. Prefix and Suffix Invariant Dynamic Time Warping. ICDM 2016: 1209-1214

[30] C. Tan, F. Petitjean, and G. Webb. A new framework and method to lower bound DTW. SDM 2019: 522-530.

[31] Y. Tanaka, K. Iwamoto, and K. Uehara, Discovery of time-series Motif from multi-dimensional data based on MDL principle, Machine Learning, 58(2-3):269– 300, 2005.

[32] C. D. Truong, and D. T. Anh. A fast method for Motif discovery in large time series database under dynamic time warping. In: Proceedings of the 6th KSE, 9-11, pp. 155–168. (2014).

[33] D.S. Willet, et. al. 2016. Machine learning for characterization of insect vector feeding. PLoS computational biology, 12(11).

[34] L. Wei. SAX and PAA for when N/n is not an integer. Documentation at http://cs.ucr.edu/~eamonn/SAX.htm

[35] R. Wu, and E. Keogh. FastDTW is approximate and Generally Slower than the Algorithm it Approximates. CoRR abs/2003.11246 (2020)

[36] B. K. Yi, and C. Faloutsos (2000). Fast Time Sequence Indexing for Arbitrary Lp Norms. VLDB. Cairo, Egypt. pp 385-394.

[37] Y. Zhu, et. al. Matrix Profile II: Exploiting a Novel Algorithm and GPUs to break the one Hundred Million Barrier for Time Series Motifs and Joins. ICDM. IEEE, 2016.

[38] Y. Zhu, et. al. Matrix Profile XI: SCRIMP++: Time Series Motif Discovery at Interactive Speed. ICDM. 2018.

[39] Y. Zhu, and D. Shasha. Warping Indexes with Envelope Transforms for Query by Humming. SIGMOD 2003: 181-192

[40] A. Ziehn, et. al. Time Series Similarity Search for Streaming Data in Distributed Systems. EDBT/ICDT 2019 Conference, 2019.

[41] Supporting webpage: https://sites.google.com/site/dtwmotifdiscovery/

REPRODUCIBILITY

We have taken extraordinary steps to make sure that every experiment (including the figures and samples that proceed the official experimental Section) are easy to reproduce. To this end:

- For experiments that have a stochastic element, we initialize with the same random number generator seed before each iteration. This ensures that a reader can exactly reproduce our output, independent of their platform.
- Every data used in each figure or table is explicitly labeled with the name of the figure/table and archived at [41] in a universally readable ASCII plain text format, in addition to the. mat format that we use internally.
- We have created a presentation that gives additional information about anything we did to create our final figures. For example, purely for aesthetic reasons, we "flipped" one of the dendrograms shown in Figure 3 upside down (without changing its topology or distances). The presentation reconciles the slight differences between the output of the code, and the final figures.
- In addition to the main code, we have included all the minor code, including the code to produce dendrograms etc.

For many experiments we choose to use time series and query lengths that are powers of two. This is not required for SWAMP but is a consideration for future researchers who may try to improve on our results with either DFT or DWT methods, both of which have their best cases when the data lengths are powers of two.

As noted in the paper but reiterated here, in many works, the size of the warping window is often given as a percentage of the length of the time series [15][23], in this work we give it as an absolute number. One reason for this is because a given percentage may not evenly divide a time series length, and different rounding policies may affect the results.

Were warranted, we presented some details in the paper very tersely. For example, we noted in the main text:

Finally, we compared to [28], which is the only other exact algorithm for finding DTW motifs. On the three datasets above this algorithm was 17,274.76%, 185,511.97% and 13,857.22% respectively.

The details are a little sparse in that text. However:

- The differences are so large that we hope the reader will understand our decision not to spend too much of the page limits here.
- The full detailed results are available at [41], together with the full code and data needed to reproduce the results.

Here we note that this comparison was completely fair. We used the exact same computer, same datasets, and same implementations of all common subroutines, including the various lower bounds, ED and DTW comparison algorithms, etc. Moreover, we further optimized the original algorithm extensively. The original algorithm finds both discords and motifs under DTW, but we made it faster by removing the need to find discords, and only requiring it to find the top-1 motif.

Likewise, our comparison to brute-force search was rigorously fair. There are many ways to make a DTW-based algorithm perform poorly. For example, one could implement the rival method using the recursive version of DTW instead of the iterative version. The recursive version of DTW is one to two orders of magnitude slower than the iterative version. However, here we again used the exact same computer, same datasets, and most importantly same implementations of all common subroutines, including the various lower bounds, ED and DTW comparison algorithms.

A REPRODUCIBILITY “ROSETTA STONE”

As noted above, we have made all our code publicly available in perpetuity [41]. However, a reader may wish to implement and test our ideas on another platform. If we both agree on all distance measures, including the Euclidean distance, cDTW distance and parametrized lower bounds, then we can be virtually assured that all other steps will be in agreement. It may seem unlikely that we could disagree on such matters. However, our experience suggests otherwise. For example, we have seen the w parameter in cDTW interpreted as the total freedom to wander off the diagonal. In essence, that (mis)understanding will give only half the w value that we mean to communicate (and is more commonly understood [22]). Likewise, by default, some DTW programs normalize the distance by the path length. This makes only a very subtle difference when w is small, nevertheless it could cause our lower bounds to no longer be admissible. Thus, in order to make sure we agree on all measures in Table 4 we will create a pair of time series that the interested reader can literally cut-and-paste into their framework and compare results on all measures.

Note that after we z-normalized these time series, we rounded them to have just two significant digits, in order to further facilitate a detailed forensic tracing of the computation. However, this rounding means that the two time series are no longer exactly z-normalized. All subsequent analysis assumes the exact values in Table 4.

In Fig. 38 we show a visual intuition for the various measures that are key to this work. The Euclidean distance ED(Q,T) is 7.88098.

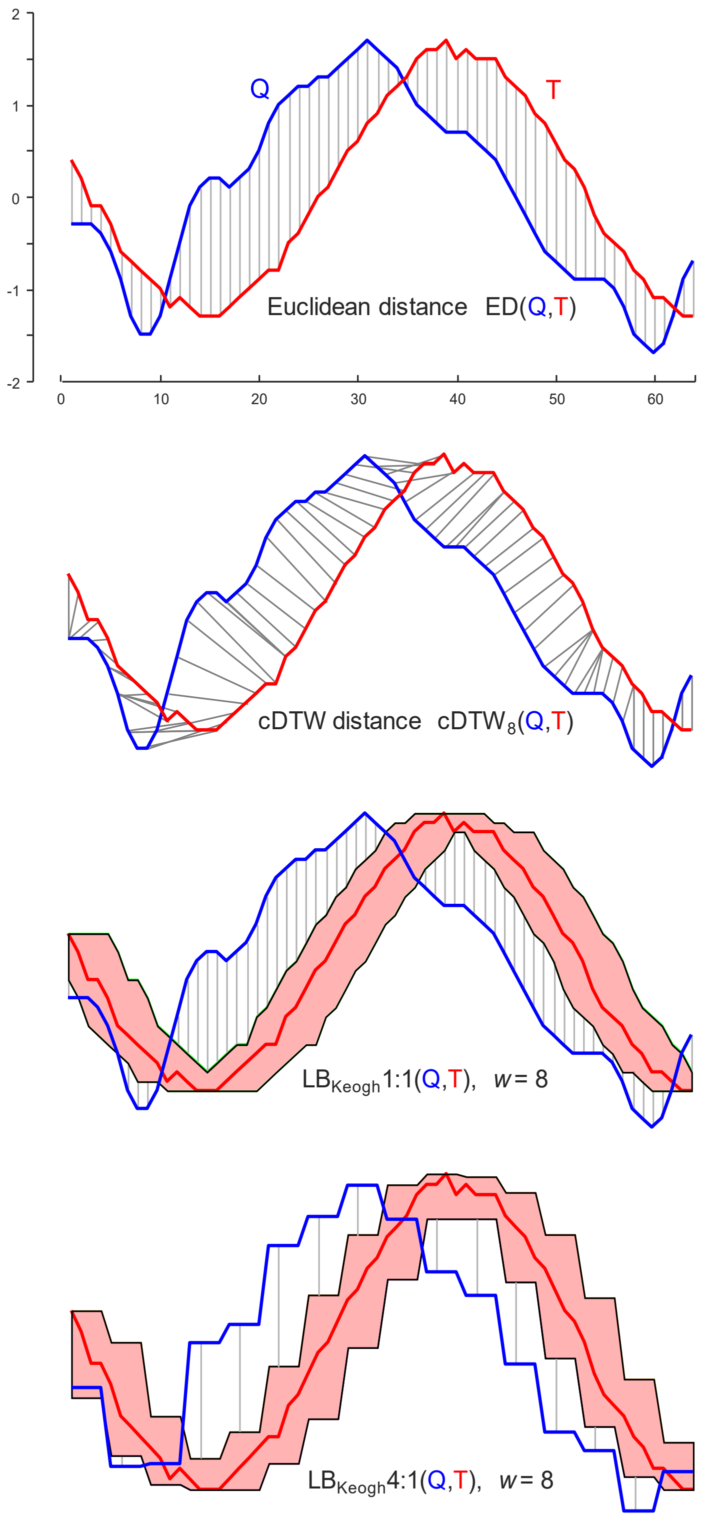


Fig. 38 *top to bottom*) For the two time series listed in Table 1, a visual intuition that shows: the Euclidean distance, the cDTW, the classic LB_{Keogh} lower bound, and the reduced dimensionality LB_{Keogh} lower bound.

Recall that in our implementation we perform the optimization of not using the squared root function (see section 4.1.1 of [22]). However, we ignore that optimization here. Using a value of eight for the warping parameter *w*, cDTW(Q,T) is 2.4240. The value of Keogh's classic lower bound, in our notation $LB_{Keogh}1:1(Q,T)$, is 1.5865. It important to recall that this function is not symmetric, in general $LB_{Keogh}1:1(Q,T) \neq LB_{Keogh}1:1(T,Q)$. Finally, Figure 21 illustrates the fourfold reduced lower bound, $LB_{Keogh}4:1(Q,T)$, which has a value of 0.4999.

Note that $LB_{Keogh}4:1(Q,T) \leq LB_{Keogh}1:1(Q,T) \leq cDTW(Q,T) \leq ED(Q,T)$ as we should expect.

Table 4. A pair of calibration time series

T	Q
0.40	-0.30
0.20	-0.30
-0.10	-0.30
-0.10	-0.40
-0.30	-0.60
-0.60	-0.90
-0.70	-1.30
-0.80	-1.50
-0.90	-1.50
-1.00	-1.30
-1.20	-0.90
-1.10	-0.50
-1.20	-0.10
-1.30	0.10
-1.30	0.20
-1.30	0.20
-1.20	0.10
-1.10	0.20
-1.00	0.30
-0.90	0.50
-0.80	0.80
-0.80	1.00
-0.50	1.10
-0.40	1.20
-0.20	1.20
0.00	1.30
0.10	1.30
0.30	1.40
0.50	1.50
0.60	1.60
0.80	1.70
0.90	1.60
1.10	1.50
1.20	1.40
1.30	1.20
1.50	1.00
1.60	0.90
1.60	0.80
1.70	0.70
1.50	0.70
1.60	0.70
1.50	0.60
1.50	0.50
1.50	0.40
1.30	0.20
1.20	0.00
1.10	-0.20
0.90	-0.40
0.80	-0.60
0.60	-0.70
0.40	-0.80
0.30	-0.90
0.10	-0.90
-0.20	-0.90
-0.40	-0.90
-0.50	-1.00
-0.60	-1.20
-0.80	-1.50
-0.90	-1.60
-1.10	-1.70
-1.10	-1.60
-1.20	-1.30
-1.30	-0.90
-1.30	-0.70
Mean = -0.001562, STD = 1.0015055	Mean = 0.003125, STD = 1.002848