The Swiss Army Knife of Time Series Data Mining: Ten Useful Things you can do with the Matrix Profile and Ten Lines of Code

Yan Zhu, Shaghayegh Gharghabi, Diego Furtado Silva[1], Hoang Anh Dau, Chin-Chia Michael Yeh, Nader Shakibay Senobari, Abdulaziz Almaslukh, Kaveh Kamgar, Zachary Zimmerman, Gareth Funning, Abdullah Mueen[2], Eamonn Keogh

University of California, Riverside
[1] *Federal University of São Carlos*
[2] *University of New Mexico*
yzhu015@ucr.edu, eamonn@cs.ucr.edu

Abstract— The recently introduced data structure, the Matrix Profile, annotates a time series by recording the location *of* and distance *to* the nearest neighbor of every subsequence. This information trivially provides answers to queries for both *time series motifs* and *time series discords*, perhaps two of the most frequently used primitives in time series data mining. One attractive feature of the Matrix Profile is that it completely divorces the high-level details of the analytics performed, from the computational "heavy lifting." The Matrix Profile can be computed using the appropriate computational paradigm for the task at hand: CPU, GPU, FPGA, distributed computing, anytime computation, incremental computation, and so forth. However, all the details of such computation can be hidden from the analyst who only needs to think about her analytical need. In this work, we expand on this philosophy and ask the following question: If we assume that we get the Matrix Profile for free, what interesting analytics can we do, writing at most ten lines of code? As we will show, the answer is surprisingly large and diverse. Our aim here is not to establish or compete with state-of-the-art results, but merely to show that we can both reproduce the results of many existing algorithms and find novel regularities in time series data collections with very little effort.

***Keywords**: Time Series, Joins, Motif Discovery, Anomaly Detection*

1 Introduction

The recently introduced time series data structure, the Matrix Profile, annotates a time series by recording the location *of* and distance *to* the nearest neighbor of every subsequence [52][57]. This means that it encodes all the information needed to

answer both *time series motif* and *time series discord* queries, perhaps two of the most frequently used primitives in time series data mining [24][28][48][52][57]. Both of these primitives can be discovered in other ways; however, the Matrix Profile can be computed very efficiently, regardless of the length of the subsequences considered (i.e. the dimensionality). This is a useful property because all other algorithms that compute these primitives suffer greatly from the curse of dimensionality [24][28][48]. For example, before the invention of the Matrix Profile, no one attempted to discover motifs longer than 900 datapoints long [24][28][48]. In contrast, [52] demonstrates a successful experiment in bioinformatics that requires finding motifs of length 60,000. Similarly, before the Matrix Profile, the longest dataset searched for *exact* motifs was a million datapoints long [24][28][48], but [57] increases that record one-hundred fold.

While the scalability of the Matrix Profile is an attractive and enabling property, it is not its most interesting feature. The original Matrix Profile paper concludes with the sentence, "*There are many avenues for future work, and we suspect that the research community will find many uses for the matrix profile*." [52]. Recently this claim has been borne out in a series of papers to show that the Matrix Profile can be used to support a host of analytic tasks including: semantic segmentation [13], the discovery of evolving patterns (time series chains) [55], and finding predictive patterns in weakly labeled data [51]. It is the extraordinary *generality* of the Matrix Profile that is its most important and useful feature. To support this somewhat subjective claim, in this work we make a more concrete claim. Given just the Matrix Profile, and at most ten lines of additional code (in a high-level language, here we use Matlab), one can perform a host of analytic tasks, as well as reproduce the results of much more complicated algorithms.

Philosophically, we would like the community to regard the Matrix Profile much like most programmers regard the `sort` subroutine in their favorite language. A casual programmer does not care or need to know how it is implemented (quick-sort, merge-sort, heap-sort etc.), she regards it as computationally "free[1]," and she uses it to solve

[1] We will revisit the idea of computationally "free" for the Matrix Profile in Section 4. For the case of sorting numbers, most invocations of sorting are on less than one million numbers, and it is possible to sort a million 32-bit numbers on a modern machine in 20 milliseconds with essentially no space overhead. Thus, for most

many problems. In a similar spirit, a data analyst does not need to know how the Matrix Profile is computed (It could be by STAMP [52], STOMP [57], STOMP*I* [53], SCRIMP++ [56], etc.), she can typically regard it as computationally "free" and use it to solve many time series data mining problems. We regard this simple abstraction as *game changing*. Analysts are much more likely to try out a new idea if they could get the first results in a few minutes, including both coding time and computational time. A tentative idea that take hours or days to produce may never get past the idea stage.

The rest of this paper is organized as follows. After a brief review of the Matrix Profile in Section 2, in Section 3 we will show ten case studies that support our claim that many interesting problems can be solved using the Matrix Profile and at most ten lines of additional code. In Section 4, we will offer conclusions and directions for future work.

2 General Related Work and Background

In the following section we briefly review the notion and definitions necessary to understand the Matrix Profile [50][51][52][53][55][56][57]. Readers familiar with this material can skip ahead to Section 3.

2.1 Definitions and Notation

We begin by defining the data type of interest, *time series*:

Definition 1: A *time series* $T \in \mathbb{R}^n$ is a sequence of real-valued numbers $t_i \in \mathbb{R}$: $T = [t_1, t_2, ..., t_n]$ where n is the length of T.

We are not interested in the *global*, but the *local* properties of a time series. A local region of a time series is called a *subsequence*:

Definition 2: A *subsequence* $T_{i,m} \in \mathbb{R}^m$ of time series T is a continuous subset of the values from T of length m starting from position i. Formally, $T_{i,m} = [t_i, t_{i+1},\ldots, t_{i+m-1}]$, where $1 \leq i \leq n-m+1$, and m is a user-defined subsequence length.

We can extract all the subsequences from a given time series by sliding a window of size m across the time series. This is called an *all-subsequence set*:

---

applications/users, it makes sense to think of sorting as a no-cost resource. Clearly, sorting can be a bottleneck for some applications, but these are rare enough that we think our claim self-evident.

Definition 3: An *all-subsequence set* $\boldsymbol{A}$ of a time series T is an ordered set of all the subsequences of T: $\boldsymbol{A} = \{T_{1,m}, T_{2,m}, \ldots, T_{n-m+1,m}\}$. We use $\boldsymbol{A}_i$ to denote $T_{i,m}$.

Note the all-subsequence set is defined purely for notational purposes. In our implementation, we do not actually *extract* the subsequences in this form as it would require significant time overhead, and explode the memory requirements.

We can take any subsequence from a time series and compute its distance to all the sequences in an all-subsequence set. We store these distance values in a vector called the *distance profile*:

Definition 4: A *distance profile D* is a vector of the Euclidean distances between a given query subsequence and every subsequence in the all-subsequence set.

We assume the distance is measured as the Euclidean distance between the z-normalized subsequences [4].

The first four definitions are illustrated in **Fig. 1**.

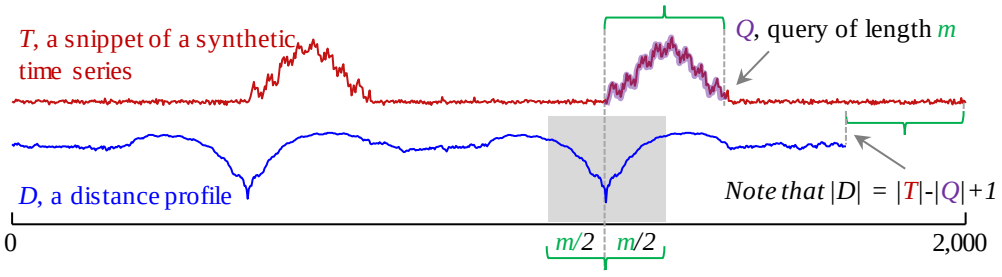

Fig. 1 A subsequence Q extracted from a time series T is used as a query to every subsequence in T. The vector of all distances is a distance profile.

Note the query subsequence and the all-subsequence set may or may not belong to the same time series. By definition, if the query subsequence and the all-subsequence set belong to the same time series, the distance profile must be zero at the location of the query, and close to zero just before and after (assuming only that the time series is somewhat smooth). Such matches are called *trivial matches* in the literature [28], and are avoided by ignoring an exclusion zone (shown as a gray region) of $m/2$ before and after the location of the query. Practically, we set the distance values in the exclusion zone to infinity.

The minimum value of a distance profile indicates the nearest neighbor (i.e., *1NN*) of the given query subsequence within the all-subsequence set. We are interested in

finding the nearest neighbor of *every* subsequence; this constitutes a *similarity join set*:

Definition 5: *Similarity join set:* given two all-subsequence sets $\boldsymbol{A}$ and $\boldsymbol{B}$, a similarity join set $\boldsymbol{J_{AB}}$ of $\boldsymbol{A}$ and $\boldsymbol{B}$ is a set containing pairs of each subsequence in $\boldsymbol{A}$ with its nearest neighbor in $\boldsymbol{B}$: $\boldsymbol{J_{AB}}=\{\langle\, \boldsymbol{A}[i], \boldsymbol{B}[j] \,\rangle \mid \theta_{1nn}\,(\boldsymbol{A}[i], \boldsymbol{B}[j])\}$. Here $\theta_{1nn}\,(\boldsymbol{A}[i], \boldsymbol{B}[j])$ is a Boolean function which returns "true" only if $\boldsymbol{B}[j]$ is the nearest neighbor of $\boldsymbol{A}[i]$ in the set $\boldsymbol{B}$. We denote the similarity join set formally as $\boldsymbol{J_{AB}} = \boldsymbol{A} \bowtie_{\theta 1nn} \boldsymbol{B}$.

We use two vectors, the *matrix profile* and the *matrix profile index*, to store the nearest neighbor information of a similarity join set. The *matrix profile* stores the distances between all the subsequences and their nearest neighbors:

Definition 6: A *matrix profile* P_{AB} is a vector of the Euclidean distances between each pair in $\boldsymbol{J_{AB}}$, where the i^{th} element of P_{AB} is the distance between $\boldsymbol{A}_i$ and its nearest neighbor in $\boldsymbol{B}$.

We call this vector a matrix profile since it could be computed by using the full distance matrix of all pairs of subsequences in time series T, and evaluating the minimum value of each row (although this method is naïve and space-inefficient). **Fig. 2** shows the matrix profile of our running example.

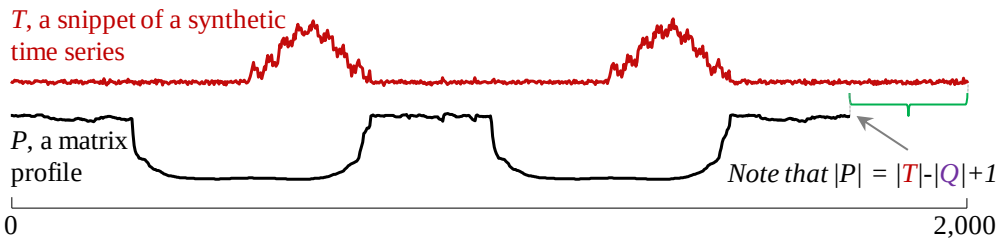

Fig. 2 A time series T, and its self-join matrix profile P.

The i^{th} element in the matrix profile P indicates the Euclidean distance from subsequence $T_{i,m}$ to its nearest neighbor in time series T. However, it does not indicate the location of that nearest neighbor. This information is recorded in a companion data structure called the *matrix profile index*:

Definition 7: A *matrix profile index* I_{AB} of a similarity join set $\boldsymbol{J_{AB}}$ is a vector of integers where the ith element of I_{AB} is j if $\{\langle \boldsymbol{A}_i, \boldsymbol{B}_j \rangle\} \in \boldsymbol{J_{AB}}$

By storing the neighboring information in this manner, we can efficiently retrieve the nearest neighbor of query $\boldsymbol{A}_i$ by accessing the i^{th} element in the matrix profile index.

In general, the function which computes the similarity join set of two input time series is not symmetric: $\boldsymbol{J_{AB}} \neq \boldsymbol{J_{BA}}$, $P_{AB} \neq P_{BA}$, and $I_{AB} \neq I_{BA}$, except in the special case where $\boldsymbol{A}=\boldsymbol{B}$.

We can regard the matrix profile as a *meta* time series annotating the time series T if the matrix profile is generated by joining T with itself (i.e., $\boldsymbol{A}=\boldsymbol{B}$). This profile has a host of interesting and exploitable properties. For example, the highest point on the profile corresponds to the time series discord [8], the (tying) lowest points correspond to the locations of the best time series motif pair [28], and the variance can be seen as a measure of the T's complexity.

We call this special case of similarity join set (**Definition 5**) a *self-similarity join set*, the corresponding matrix profile a *self-similarity join matrix profile*, and the corresponding matrix profile index a *self-similarity join matrix profile index*.

> **Definition 8:** A *self-similarity join set* $\boldsymbol{J_{AA}}$ is the similarity join of an all-subsequence set $\boldsymbol{A}$ with itself**.** We denote this formally as $\boldsymbol{J_{AA}} = \boldsymbol{A} \bowtie_{\theta 1\text{nn}} \boldsymbol{A}$. We denote the corresponding self-similarity join matrix profile as P_{AA}, and the corresponding self-similarity join matrix profile index as I_{AA}.

For clarity of presentation, we have confined this work to the single dimensional case; however, nothing about our work intrinsically precludes generalizations to multidimensional data. In the multidimensional data, we would still have a single *matrix profile*, and a single *matrix profile index*; the *only* change needed is to replace the one-dimensional Euclidean Distance with the *b*-dimensional Euclidean Distance, where *b* is the number of dimensions the user wants to consider.

2.2 Summary of the Previous Section

Since the previous section was rather dense, here we summarize the main takeaway points. We can create two meta time series, the *matrix profile* and the *matrix profile index*, to annotate a time series A with the distance *to* and location *of* all its subsequences' nearest neighbors in itself or another time series B. These two data objects explicitly contain the answers to the time series data mining tasks of motif

discovery and discord discovery [53]. Moreover, as we will show below, we can easily perform many other kinds of analytics using the *matrix profile* and the *matrix profile index* as primitives.

To make the contributions of this work more concrete, we will occasionally show the actual code we use to solve various problems. The two basic tools that perform the key operations explained above are:

```
    [MP, MPindex] = computeMatrixProfile(T,m);          % Def 8
[JMP, JMPindex] = computeMatrixProfileJoin(A,B,m); % Def 6-7
```

Once again, a key assumption of this work is that these operations can be computed very fast, by any one of half a dozen algorithms optimized for various computational paradigms. Thus we simply take these operations as given, and see what we can do with them with just a tiny amount of extra coding effort.

3 Ten Useful Things you can do with the Matrix Profile and Ten Lines of Code

In this section, we show the eponymous *ten useful things you can do with the matrix profile and ten lines of code*. In every case we make the data available [59]. The code to compute the matrix profile can be found at [40] and the remaining code is placed inline in this work. Note that we see these as ten *demonstrations*. We do not expand any section with the rigor one might expect if it were a single idea being presented in a paper.

3.1 Discovering Motifs Under Uniform Scaling

The utility for motif discovery under uniform-scaling invariance was first considered in [48]. We revisit the motivation with a simple and visually compelling example. We took two exemplars from the same class from the MALLAT dataset [9] and imbedded them into a random walk dataset. As **Fig. 3**.*top* shows, even without the color-coded clue brushed onto the data by the Matrix Profile discovery tool [40], the repeated pattern is visually obvious.

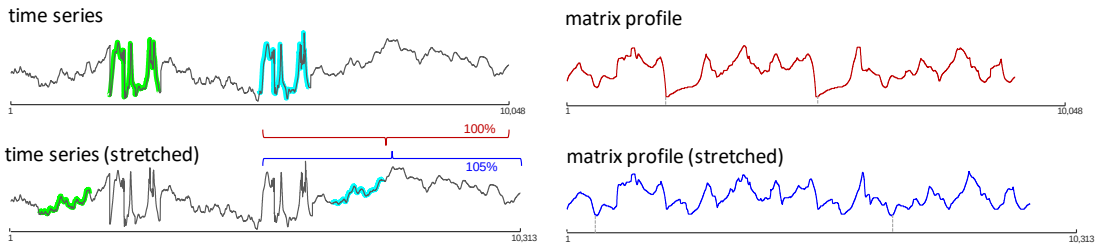

Fig. 3 *top.left*) A random walk time series with two exemplars from the MALLAT dataset imbedded at locations 2001 and 5025. The color highlighting indicates the top-1 motif, which unsurprisingly are exactly the imbedded patterns. *top.right*) The matrix profile corresponding to the random walk time series. The minimum values correspond to the top-1 motif in the time series. *bottom.left*) The same time series, but with the second half linearly stretched by 5%. This causes the top-motif to change to snippets of random walk. *bottom.right*) The matrix profile corresponding to the stretched time series. We can see that the minimum points changed.

We then took the second half of the time series and linearly stretched it by 5%. By any standard, such a change is a trivial difference and essentially visually imperceptible. Nevertheless, as **Fig. 3**.*bottom* shows, the pair of imbedded patterns are no longer the top-1 motif, an unexpected and disquieting result. Before we show how to address this within this paper's "the Matrix Profile plus ten-lines-of-code framework," we note the following facts that mitigate the issue.

- For the rescaled version, the pair of imbedded patterns was the *second*-best motif and only *just* nudged out by the spurious random walk pair.
- If, instead of searching with a motif length of 1,024, the original length of the imbedded pattern, we had searched for a shorter length, say 500, then the best motif would have been a subsequence of the imbedded pattern. The user could then have examined the shorter motif and realized it could be extended significantly while maintaining its similarity.
- We deliberately chose this dataset, from the 85 in the UCR archive, knowing it would be very sensitive to changes in linear scaling. This is because *complex* time series (see [3]) with very sharp rises and falls are particularly sensitive to having features out of phase. For most datasets, motif discovery is much more robust to small amounts of uniform scaling.

Despite all these mitigating facts, **Fig. 3**.*bottom* clearly shows that there may be some situations in which there is a need to find motifs with invariance to uniform scaling. To the best of our knowledge, there is only one research effort that has addressed this. However, this method is *approximate*, requires many parameters to be set, and is only

able to support a limited range of scaling [48]. In contrast, we can easily and *exactly* solve this problem under our simple assumptions.

For the moment, assume that we know the scaling factor we want to be invariant to happens to be 1.64. We can take the dataset *T* and copy a stretched version of it into *T*2, simply by using:

```
T2 = T(1: 100/164: end); % Unofficial Matlab way to resample
```

If we now call:

```
[JMP, JMPindex] = computeMatrixProfileJoin(T,T2,500);
```

Now, the resulting Matrix Profile will discover the motifs with the appropriate uniform scaling invariance. In fact, we did exactly this on a 6,106,456 length trace of household electrical demand to discover the motif shown in **Fig. 4**.

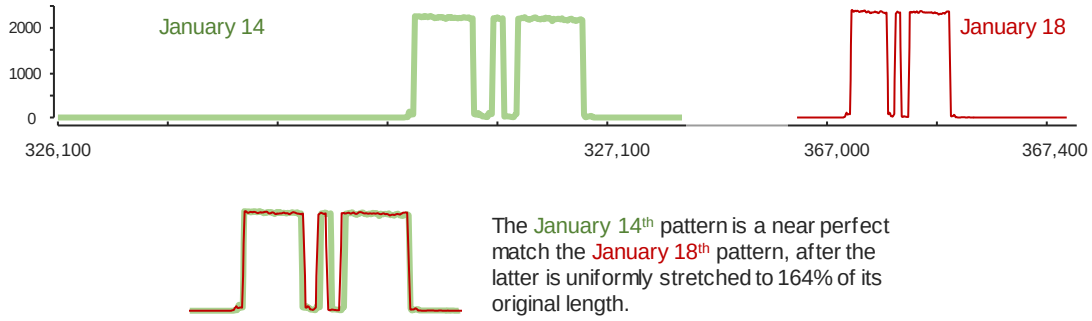

Fig. 4 *top*) Two non-contagious snippets from the ElectriSense dataset [16]. While semantically similar, they have a very large Euclidian distance because they are of different lengths. *bottom*) After stretching the January 18th pattern by 164%, the two patterns are almost identical**.**

The motif pattern appears to be the three elements of a dishwasher cycle (clean, rinse, dry), which can take different amounts of time due to the use of the optional *half-load* feature [23]. In this case, we knew from some first principle physics how to set the scaling factor, but that may not always be the case. Given our assumptions, we can simply iterate over all possible scaling factors in a given range. For example, to discover motifs that are similar after scaling one pattern by 150% to 180%, we can use the following code snippet.

```
minJMP=inf(1,length(T)), minScale=ones(1,length(T));
minJMPindex=zeros(1,length(T));
for scale_factor = 150 : 180
  T2 = T(1: 100/scale_factor: end);
  [JMP, JMPindex] = computeMatrixProfileJoin(T,T2,500);
  locs = JMP < minJMP;
  minJMP(locs)=JMP(locs),  minScale(locs)=scale_factor/100;
  minJMPindex(locs)=JMPindex(locs);
end
```

This example perfectly elucidates the philosophy driving this paper. For many time series data mining tasks, we may not need to spend significant human time designing, implementing and tuning new algorithms. The Matrix Profile and ten lines of code may be sufficient.

3.2 Discovering Time Series Semordnilaps

Consider the sentence fragment we discovered in Wikipedia, "… *the longest-lived Tasmanian devil recorded was Coolah* ..." [46]. This snippet contains a Semordnilap pair [33], the mirrored words "lived" and "devil". Semordnilaps are easy to find in arbitrary text strings, and indeed have an important role in molecular biology. For example, many restriction enzymes recognize specific palindromic sequences and cut them. As a concrete example, the restriction enzyme EcoRI recognizes the following palindromic pair, "GAATTC" and "CTTAAG" [21].

Because the original definition of time series motifs was directly inspired by the analogy to DNA, it is natural to ask if there is a natural time series analogy to semordnilaps, and if so, can they be efficiently discovered? From the previous example, the reader will readily see that this is trivial, we can simply use:

```
 T2 = fliplr(T);  % returns T reversed
[JMP, JMPindex] = computeMatrixProfileJoin(T,T2,m);
```

The only question remaining is: are there *natural* domains that contain time series semordnilaps? The answer is affirmative.

To demonstrate the utility of Semordnilap discovery, we consider Joseph Haydn's Symphony No. 47 in G major, written in 1772. In particular, we examined a performance by the Tafelmusik Orchestra, directed by Bruno Weil in 1993 [30]. The

performance is twenty-one minutes and two seconds long. As shown in **Fig. 5**.*top*, we converted it to Mel-frequency cepstral coefficients (MFCC) using windows with 0.5 second and 50% of overlap (standard music processing settings). We set *m* to 150, or 37.5 seconds.

At 14 minutes and 53 seconds, there is a Semordnilap of a passage we encountered at 14 minutes and 16 seconds.

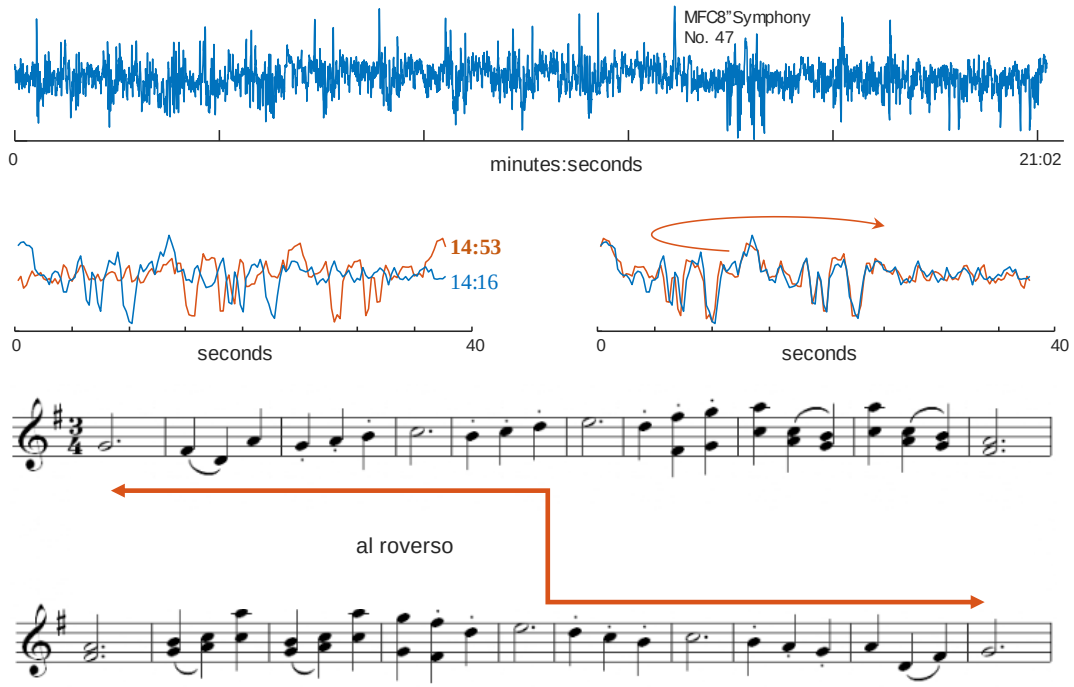

Fig. 5 *top*) Haydn's Symphony No. 47 converted to MFCC. *center.left*) Two snippets found by Semordnilap discovery appear unrelated until we flip one backwards in time (*center.right*). *bottom*) The sheet music for the relevant section explains this unexpected discovery.

Fig. 5.*bottom* explains the presence of such a perfectly conserved Semordnilap. As noted in [7], "*The most extraordinary of all canonic movements from this time is of course from Symphony No. 47. Here Haydn writes out only one reprise of a two-reprise form, and the performer must play the music 'backward' the second time around*".

While this example is clearly contrived, there may be Semordnilaps waiting to be discovered in dance, travel trajectories, medical data, industrial processes, and a host of domains that have yet to occur to us.

3.3 Discovering Time Series Reverse Complements

Our success in finding Semordnilaps immediately suggests another specialized type pattern we could search for. Are there examples of patterns which repeat, but in which one pattern is the inverse of the other? That is to say, unlike Semordnilaps, which are "flipped" in the *time* axis, are there patterns that are flipped upside-down in the *value* axis? We call such patterns Time Series Reverse Complements (TSRCs).

For example, El Nino Southern Oscillation (ENSO) is a phenomenon that is characterized by intermittent negative correlations between the surface temperatures observed in the Central and Eastern Ocean [19]. However, there are much more quotidian examples. Consider the two-minute snippet of time series shown in **Fig. 6**. It shows the y-axis from a hip-worn accelerometer from the USC-HAD Database [54]. As shown in **Fig. 6**.*bottom.left*, the best motif of length twenty seconds is not well conserved, and almost looks like two random subsequences. This is unsurprising, apart from dance or athletic performances, we would not expect human behavior to faithfully repeat over such an extended time scale. However, we also searched for the best TSRC pattern of the same length, and as shown in **Fig. 6**.*bottom.center* and **Fig. 6**.*bottom.right* it is stunningly well conserved.

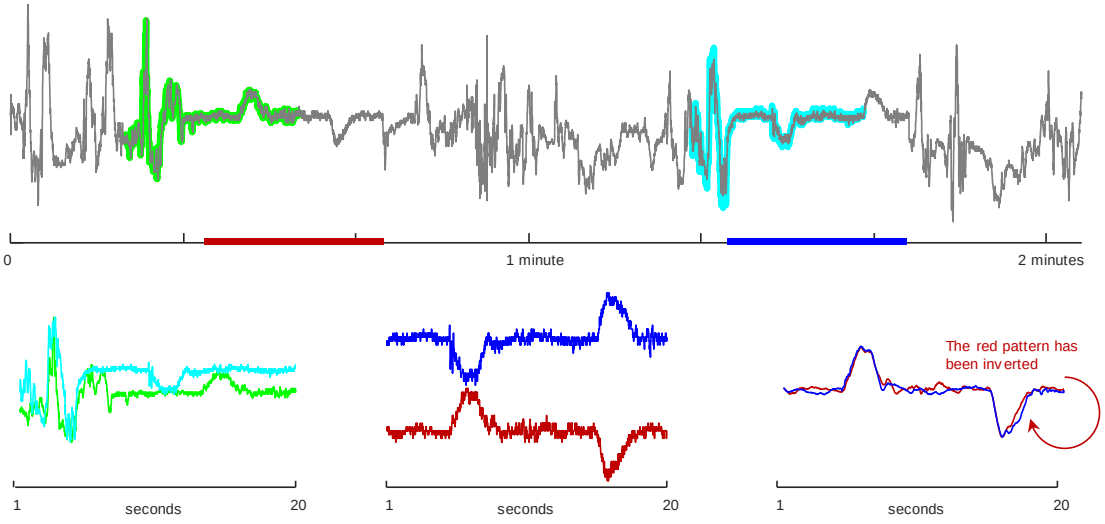

Fig. 6 *top*) Approximately two minutes from a dataset from a hip-worn accelerometer of quotidian activity. *bottom.left*) The best motif of length twenty seconds is not well conserved, however, if we generalize the search to consider TSRC motifs (*bottom.center*) we find a highly conserved pattern. To better see how well conserved it is, in (*bottom.right*) we show the patterns with one element inverted, and both patterns smoothed. However, we note that we discovered this pattern in the original noisy space.

What is the mechanism that produced this pattern? At about twenty-two seconds into the recording, the user stepped into an elevator. The first bump is the "jolt" of the elevator ascending, followed by the "dip-and-recover" as the elevator decelerated the desired floor. After about one minute, the user took a return trip, descending the same number of floors.

The reader will readily appreciate that discovering TSRCs with the matrix profile is trivial, we simply used:

```
 T2 = T*(-1);  % returns T flipped upside down
[JMP, JMPindex] = computeMatrixProfileJoin(T,T2,m);
```

Note that in this case, the discovered TSRC *also* happens to be a Semordnilap. However, this need not be the case in general.

3.4 Segmenting Repetitive Exercises

In recent years, there have been dozens of papers published on the task of segmenting repetitive exercises – such as weight training and calisthenics – via body worn sensors. See [26] and the references therein and thereof. As [26] forcefully argues, this problem is more difficult than it may seem at first glance. Many of the proposed methods use Hidden Markov Models, a powerful technique, but one that typically requires a lot of training data and careful parameter tuning. While we do not claim to be able to reproduce all the features of systems such as RecoFit [26], we note that at least in some cases, the Matrix Profile and a single line of extra code can segment repetitive exercises with high accuracy. Consider the following two lines of code.

```
[MP, MPindex] = computeMatrixProfile(T,m);
regions_of_repetition = MP < 2/3 * (min(MP)+max(MP));
```

We tested this code snippet on the Pamap Dataset [34], a dataset frequently used by the relevant community. **Fig. 7** shows the result.

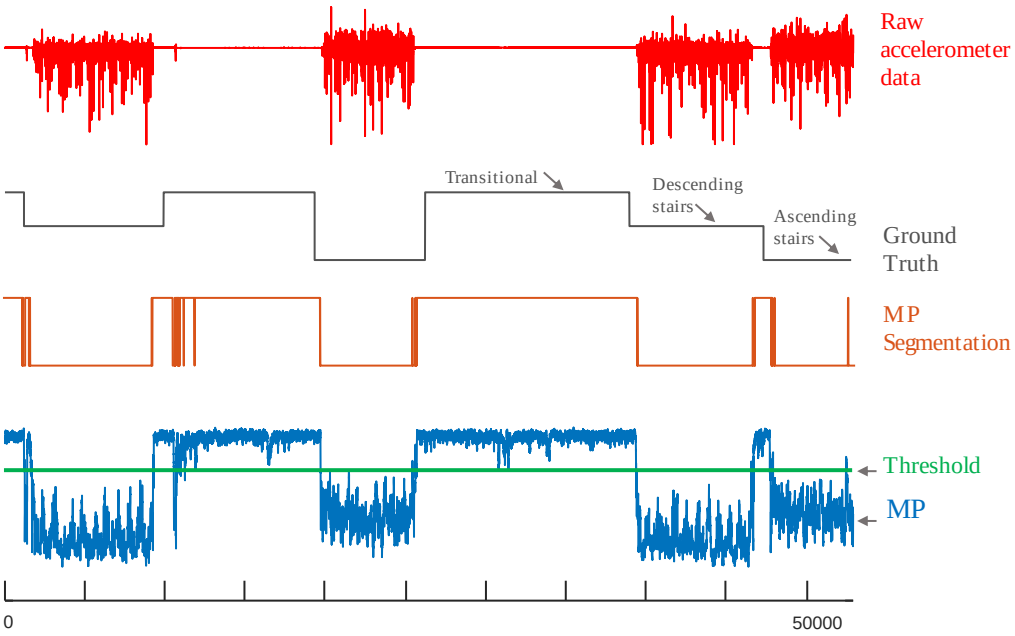

Fig. 7 *top to bottom*) A snippet of accelerometer data from Pamap Dataset-Subject 1, shoe-Acc X-axis, with its ground truth segmentation, into Ascending stairs, Descending stairs and Transitional activities. The MP segmentation we predicted largely agrees, and was computed simply by thresholding the Matrix Profile. After casting the ground truth segmentation in a Boolean vector of {Transitional | other} we find out predicted segmentation agrees with it 93% of the time.

Why does this simple idea work so well? Note that activities such as ascending stairs and descending stairs correspond to very well-conserved, periodic movements of the person, so such data would have a low matrix profile value. In contrast, the transitional activities are more at random, generating very noisy patterns with high matrix profile values. Therefore, in this dataset, a single threshold is enough for us to segment the activities.

3.5 Robust Distance Functions

Distance functions are at the heart of much of data mining, especially *time series* data mining [3]. We can characterize distance functions by the invariances they achieve. For example (here we illustrate with *text*, the discrete analogue of time series):

- Euclidian distance is invariant to noisy data, and able to discover the similarity between `cat` and `rat`.
- Dynamic Time Warping is invariant to local misalignments in the data and differing data lengths, and able to discover the similarity between `concat` and `cooncat`.

- Cross Correlation is invariant to phase alignment, and can discover the similarity between `cathouse` and `housecat`.
- Longest Common Subsequence is invariant to minor insertions/deletion in the data and able to discover the similarity between `genome` and `gene`.

While there are many such distance measures to handle various distortions in *short* time series, *long* time series provide greater challenges. Consider the following phrases:

```
A = we can sequence the genome of the cat
B = the cats genome was sequenced in 2014
C = xe hes jlvoeqee kjsw eaqwe oqawe acea
```

Here the hamming distance (the discrete analogue of Euclidean Distance) between `A` and `B` is 31, but the distance between `A` and `C` is only 26. This is an unintuitive result, given that we immediately see the common structure in `A` and `B`. What we want is a distance measure that can reward `A` and `B` for sharing many subsequences, even if they are out of order. This issue also occurs for time series. To see this, we consider pairs of ten-second snippets extracted from four individuals experiencing cardiac issues. As shown in **Fig. 8**.*left*, we clustered them using a Euclidean distance average-linkage hierarchical clustering.

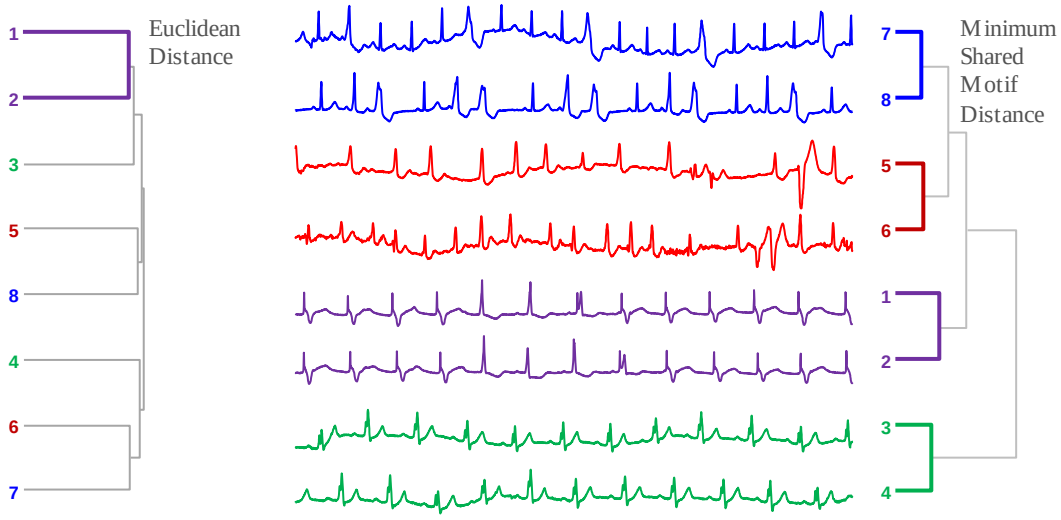

Fig. 8 *left*) Eight ten-second snippets of time series, from four individuals, clustered using Euclidean distance single-linkage hierarchical clustering. *right*) The same snippets clustered using a Matrix Profile based distance measure.

Here the disappointing results of Euclidean distance could be mitigated by very careful beat extraction and alignment. However, we want to be able to use distance

functions with minimum human effort and knowledge. There are some distance functions that can achieve the required invariances. Their names, bag-of-patterns [25], bag-of-words [45], and so on suggest both their source of inspiration and their approach. While these methods may produce better results for our task at hand, they all have at least three parameters and require significant implementation effort. In the spirit of this work, can we reproduce at least *some* of their effort with the MP and a few lines of code? To answer this question, consider the following lines of code:

```
[JMP, dummy] = computeMatrixProilfeJoin(A,B,m);
MSMD = min(JMP);
```

Using this MSMD distance measure, we produced the clustering shown in **Fig. 8**.*right*. Note that MSMD is symmetric: we can reverse the order of A and B in the pseudo code and obtain the same result. Assume S_A is a set of all subsequences extracted from time series *A*, and S_B is a set of all subsequences extracted from time series *B*, then MSMD is simply the minimum among all pairwise distances between subsequences from S_A and subsequences from S_B.

We can further test the utility of the MSMD distance measure, using classification. Almost all time series classification comparisons are based on the UCR archive [3][11]. However most of the datasets in that archive are extracted from larger datasets with an extraction tool based on the Euclidian distance. Given that, it is hardly surprising that Euclidian distance (and DTW, which subsumes Euclidian distance as a special case) will be hard to beat [11]. However, the newest release of the archive contains three related datasets that were processed in a different way. The source dataset is data from fifty-two pigs having three vital signs monitored, before and after an induced injury [15]. The data are vital signs measured at high frequency (250Hz) using a bed-side hemodynamic monitoring system, much like a setup that one might expect to see in a modern ICU. The collected measurements are arterial blood pressure, central venous pressure, and airway pressure. Critically for our purposes, the authors note "*Unlike in the* (pre-2018) *UCR data sets, the vital signs are not temporally aligned: The starting point of observation is effectively arbitrary*". We compared MSMD, Euclidian distance and DTW on the three pig datasets. We used the predefined train/test splits, learning MSMD's best value for *m*, and DTW's best

value for *w* (the warping window width) with cross validation on just the training data. Table 1 summarizes the results.

Table 1: A comparison of the holdout error rates of one-nearest neighbor with three distance measures. In each dataset, the default rate is 0.980, because each of the 52 pigs is equally likely.

Dataset	MSMD (*m*)	Euclidian distance	DTW (*w*)
PigAirwayPressure	0.134 (425)	0.944	0.903 (1)
PigArtPressure	0.000 (140)	0.875	0.802 (1)
PigCVP	0.105 (200)	0.918	0.841 (11)

The results show that while both Euclidian distance and DTW struggle to beat the default rate, the MSMD can achieve a very low error rate. It is possible that we could improve DTW by using endpoint invariance DTW, and we could improve Euclidian distance by doing circular shift Euclidian distance. However, these results strongly support our basic claim: you can get good results with the Matrix Profile and a handful of lines of code.

3.6 Meter-Swapping Detection

Electricity theft is a multi-billion-dollar problem worldwide [39]. There are dozens of ways to steal power, but some modern wireless meters offer a surprisingly easy method with little chance of detection [20]. Suppose customer **A** is a heavy consumer of electricity; perhaps he has several electric cars, or a machine shop, or a marijuana nursery in his garage. Further suppose that he notes that one of his neighbors, customer **B**, an elderly widow living alone, consumes very little power. It is possible for **A** to surreptitiously switch his meter with **B**, and thus only have to pay for her meager consumption, while she unwittingly gets lumbered with paying for his extravagant consumption. This crime is called *meter-swapping*, and has become increasingly prevalent as power companies have reduced meter reading staff in favor of wireless meter reading [39].

It might be imagined that this would be easy for the power company to detect, as there would be a significant change in the average power consumed by two houses. However, as **Fig. 9**.*top* hints at, power consumption is often bursty anyway. For example, families take vacations, welcome a new baby, or have children return from college for a few weeks.

Our intuition to solve this problem is to note that while *volume* of consumption is not a good feature, some households may have a unique "shape" of the consumption over a day. Note that we do not expect *all* days to be conserved and unique, it is sufficient for our purposes that the household *occasionally* produces a well-conserved pattern, perhaps correspond to a low-power use on the Sabbath for an orthodox family, or (as in one of the authors' experience) an all-day obligation to wash and dry the soccer kits for the entire team once every seven weeks.

We consider a dataset of household electrical power demand collected from twenty houses in the UK in 2013 [29]. To simulate a meter-swapping event, we randomly chose two of these time series, and swapped their traces starting at November 10th. As we can see in **Fig. 9**.*top* this change is not readily visually obvious.

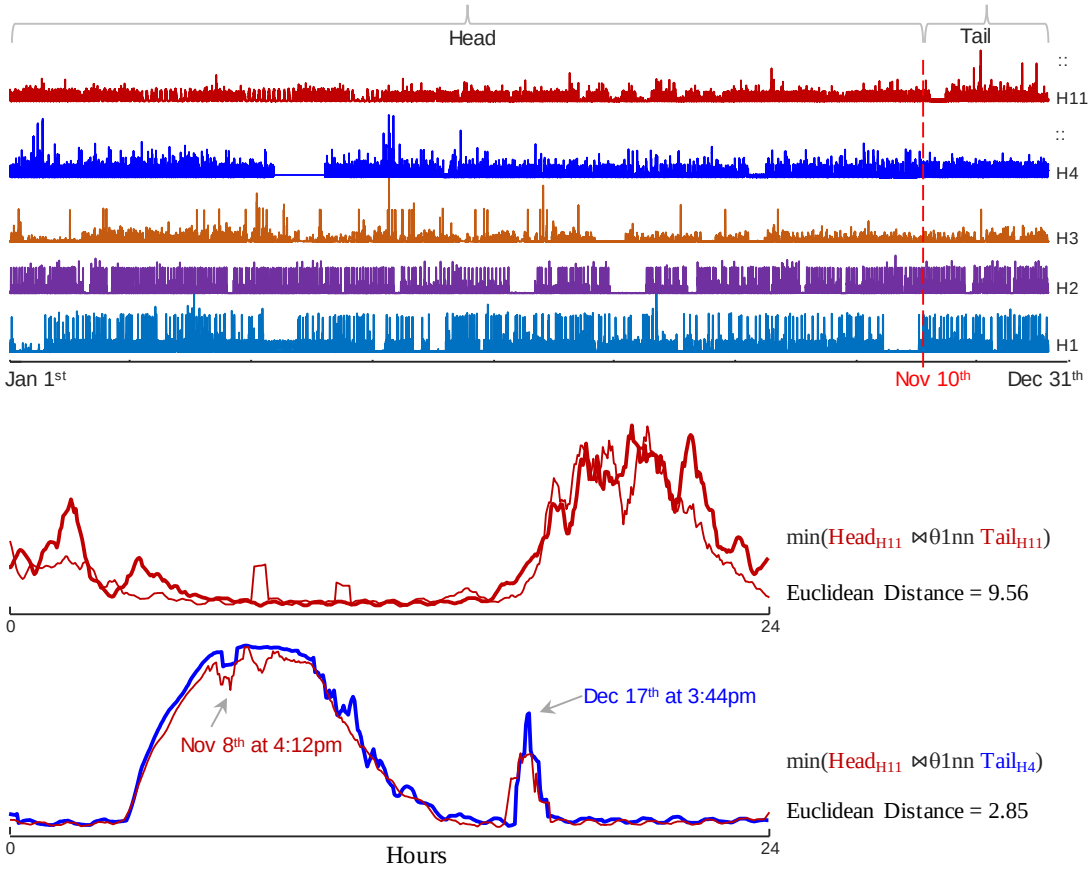

Fig. 9 *top*) Five time series from the set of 20 we consider for this demonstration, spanning from January 1st to December 23rd. A randomly chosen pair of time series had their "tails" (the region after November 10th) swapped to simulate a meter-swapping event. *middle*) If we join the head and tail of H11, the 1st motif pair has a mutual distance of 9.56, slightly lower than the mean of the motif distance for all 20 houses. *bottom*) If we join the head and tail pair from any of the 400 such combinations, the 1st motif pair from the join of $Head_{H11}$ and $Tail_{H4}$ produce the smallest mutual distance, of just 2.85; the motif patterns look strikingly similar.

To find the swapped time series pair, we propose the following simple algorithm. We divide all the time series into two sections: the "Head," prior to November 10th and the "Tail," subsequent to November 10th. We join all possible combinations of Heads and Tails, and record the pair H_i, H_j that minimizes the following score:

$$\text{Swap-Score}(i,j) = \min(\text{Head}_{Hi} \bowtie\theta 1\text{nn Tail}_{Hj}) / (\min(\text{Head}_{Hi} \bowtie\theta 1\text{nn Tail}_{Hi}) + \text{epsilon})$$

In our simple experiment, this score was minimized by i = H11 and j = H4, which as it happens, *are* our swapped pair. As **Fig. 9**.*bottom* shows, the motif spanning these two apparently distinct traces time series is *suspiciously* similar, perhaps similar enough to warrant a visit by a meter reader/fraud prevention officer.

As before, the code to do this is trivial given the Matrix Profile:

```
for i=1:5
  [MP,MPindex] = computeMatrixProfile(Head(i),m);
  minMP = min(MP) + eps % eps is Matlab's built-in epsilon
  for j=i+1:5   % Produce all pairs of Heads and Tails
    [JMP,JMPindex]=computeMatrixProfileJoin(Head(i),Tail(j),m);
    Score = min(JMP) / minMP ;
    <trivial code to maintain the minimum Score so far>
  end
end
```

Note that in our simple example we assumed we knew the date of the swap, removing that assumption would simply require expanding our search space.

3.7 Shapelet discovery

Time series shapelets are time series subsequences that best represent a class [49]. The Matrix Profile can help us quickly identify good shapelet candidates. This idea was mentioned in passing in [52] but was not fully developed and evaluated due to lack of space.

We demonstrate our approach with the GunPoint dataset. This dataset has two classes, Gun and NoGun (NoGun is also known as *Point*, hence the name GunPoint). As shown in **Fig. 10**, we construct time series T_A by concatenating all the instances of the

Gun class, and construct time series T_B by concatenating all the instances of the NoGun class. We insert an *NaN* value in between every two concatenated instances to avoid introducing artificial patterns that are not present in the original time series. We then compute two matrix profiles, P_{BB} and P_{BA}. For simplicity, we use a subsequence length of 38, which is the length of the best shapelet reported for this dataset [49].

Intuitively, P_{BB} will be smaller than P_{BA} because we would expect subsequences within the same class to be more similar than those of different classes. The difference between P_{BA} and P_{BB} (we denote it as $P = P_{BA} - P_{BB}$), as shown in **Fig. 10**.*bottom*, generally agrees with this intuition.

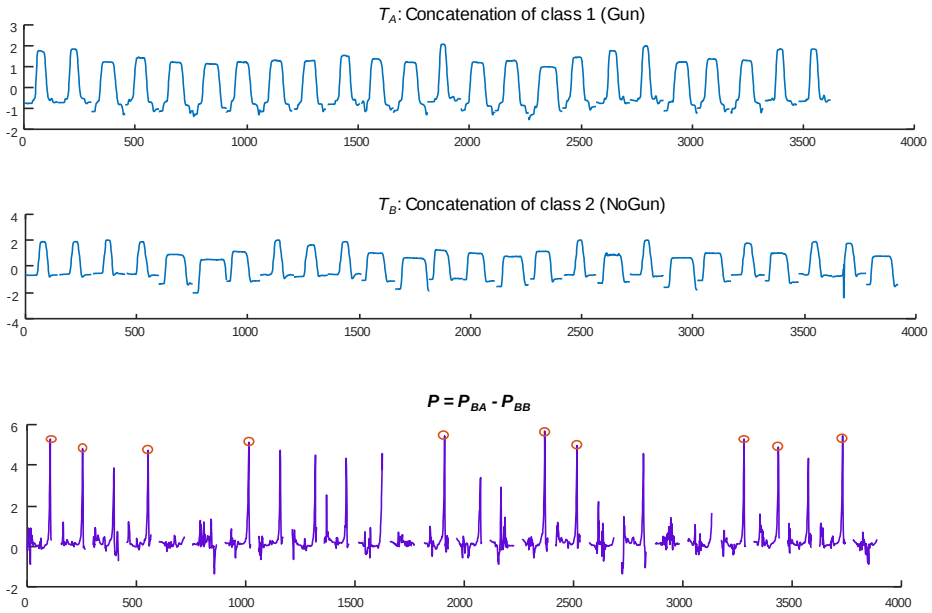

Fig. 10 *top and middle*) Two time series *A* and *B* formed by concatenating instances of each class of GunPoint dataset. *bottom*) The difference between P_{BA} and P_{BB}. The top-10 peak values (highlighted with red circles) are suggestive of good shapelet candidates.

We propose the peak values in *P* are indicators of good shapelet candidates, because they suggest patterns that are well conserved in their own class but are very different from their closest match in the other class. We pick the top-10 candidates from T_B (analogously, we can find the top-10 candidates from T_A if we consider the difference between P_{AB} and P_{AA}). The code snippet is as follows.

```
[PBB, dummy] = computeMatrixProfileJoin(B, B, m);
[PBA, dummy] = computeMatrixProfileJoin(B, A, m);
MPdiff = PBA - PBB;
indicesOfTopShapelet = topTen(MPdiff); % trivial code to implement topTen (extracting the top 10 peaks from the matrix profile) omitted
```

In **Fig. 11**.*left*, we can see that all the top-10 shapelets give very high classification accuracy on both the train and test data. Among them, we choose the one that renders the highest classification accuracy on the training set (the 6th shapelet) and show it in **Fig. 11**.*right*. This shapelet gives 93.33% accuracy on the test data, which is higher than the 91.33% accuracy of One-Nearest-Neighbor with DTW distance measure, with a bonus advantage of significantly less classification time. The shapelet learned reflects a distinct characteristic of the class that it represents (NoGun), as discussed by Ye and Keogh [49]: "*the NoGun class* "*has a "dip" where the actor put her hand down by her side, and inertia carries her hand a little too far and she is forced a correct it…a phenomenon known as 'overshoot'"*. In contrast, in the opposite Gun class, the actor carries a gun. She needs to put the gun back in the holster and then bring her hand to a complete rest position, generating a different pattern.

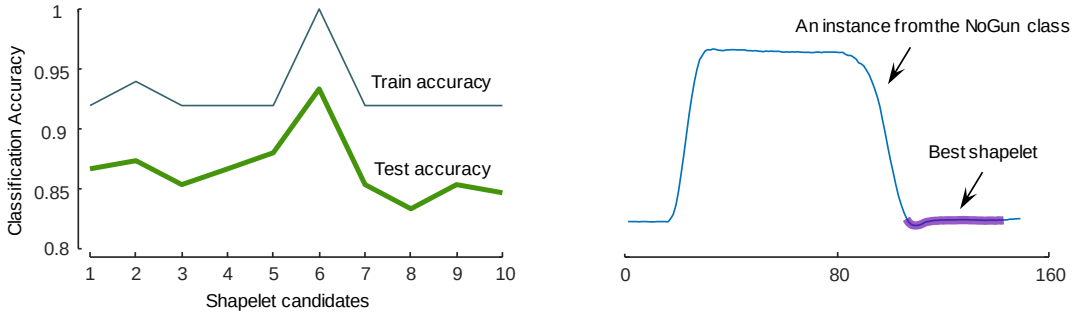

Fig. 11 *left*) Classification accuracy of ten top-ten shapelet candidates. All the candidates render high classification accuracy on both train and test data. *right*) The best shapelet found in training. Classification with this shapelet on test data gives 93.33% accuracy, higher than the 91.33% accuracy of One-Nearest-Neighbor DTW. This 93.33% accuracy is also the best accuracy achieved with the classic shapelet search approach [49].

In hindsight, this shapelet also achieves the same classification accuracy on the test data as the original shapelet algorithm [49]. However, in contrast to the classic shapelet algorithm, which exhaustively evaluates the classification power of every possible shapelet candidate in the dataset, the MP readily provides the top shapelet candidates for free.

3.8 Detecting and Locating Low Frequency Earthquakes

Low frequency earthquakes (LFEs), which recur episodically, could "*potentially contribute to seismic hazard forecasting by providing a new means to monitor slow slip at depth*" [37]. As such, detecting and locating LFEs are of great importance to the seismology community.

The waveforms of a recurring LFE recorded at the same seismic station are normally very similar to each other, as they reflect the unique signature of the wave reflecting and refracting through the local substrate. Thus, we can detect them by extracting the top motifs from the Matrix Profile of the continuous seismograph recording time series (e.g., [57]). However, as indicated in [57], this can result in a lot of false positives since the sensor recording of a single seismic station often contains many repeating sensor artifacts or instrument noise. Though such false positives are easy to filter out by human eye [57], this becomes untenable when the data is long enough to contain hundreds or thousands of false positive events. **Fig. 12** shows the matrix profile corresponding to a 24-hour seismic recording of the FROB station near the central San Andreas fault at Parkfield, CA on Oct. 9th, 2007. The data is sampled at 20Hz (1.728 million datapoints in total). The matrix profile contains hundreds of deep valleys, but only less than 10% of them are corresponding to true LFEs.

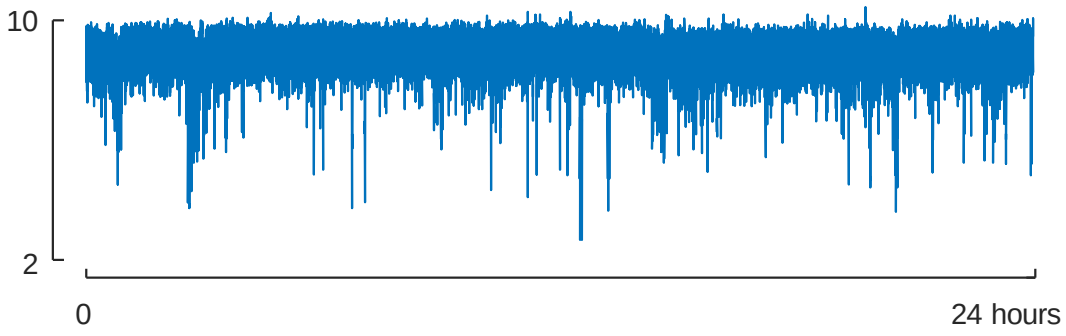

Fig. 12 The matrix profile of the FROB station on Oct 9th, 2007 contains a lot of deep valleys, a vast majority of which are false positives.

Is there a way to automatically filter out all the undesirable events (i.e. false positives)? Note that sensor artifacts or instrument noise are local (i.e., they will only be detected by a single seismic station), while recurring LFEs can be detected by multiple stations at similar times. Correspondingly, we would expect the matrix profiles of multiple stations to show low values at the time when an LFE occurs. In

contrast, when a false positive event occurs, only *one* of the matrix profiles would show low values and the others will show high values. This renders a simple solution to our problem: all we need to do is get the element-wise *maximum* of the matrix profiles corresponding to multiple nearby stations.

In **Fig. 13**.*top* we zoom in a 15-second snippet of the matrix profile shown in **Fig. 12** at around 3am when an LFE occurs, and compare it with the same snippet taken from the matrix profile of a nearby seismic station JCNB (shown in **Fig. 13**.*bottom*). As expected, both snippets contain a valley.

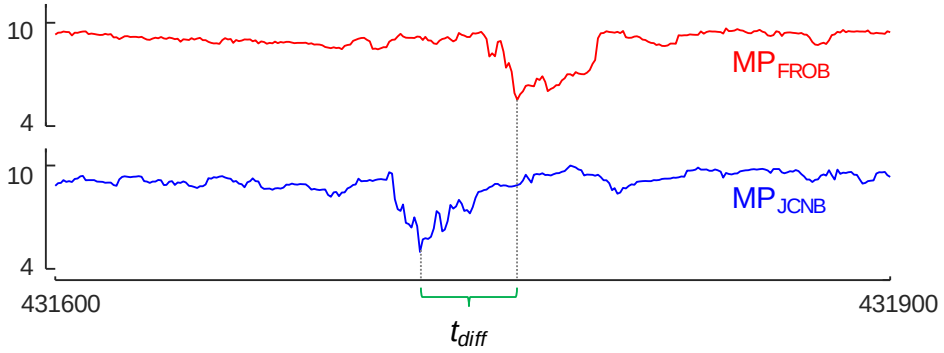

Fig. 13 *top*) A 15-second snippet of the matrix profile shown **Fig. 12** at around 3am. *bottom*) The same snippet of the matrix profile corresponding to the seismic recording of the nearby station JCNB. Both snippets contain a deep valley, but they are a little bit misaligned as the two stations receive the earthquake signal at slightly different times.

However, note that the two valleys are slightly misaligned. This is because the source of the LFE locates slightly closer to the JCNB station than to the FROB station, and earthquakes travel at a finite speed. Thus, if we simply take the element-wise maximum of the two matrix profiles, the valley will become shallow. Fortunately, this misalignment (denoted as t_{diff} in **Fig. 13**.*bottom*) has physical limits: the two stations are about 10 km away, and the velocity of seismic waves near the surface of the earth is around 3-4 km/s, so t_{diff} cannot be more than 5 seconds (i.e., 100 data points). As such, we slightly adjust our "element-wise maximum" strategy to the following: we match the i^{th} element of MP_{FROB} ($1 \leq i \leq |MP_{FROB}|$) to the minimum element within the range $[\max(i\text{-}100, 1), \min(i\text{+}100, |MP_{JCNB}|)]$ of MP_{JCNB}, then take their maximum. The pseudo-code is as follows.

```
[MPfrob,dummy]=computeMatrixProfile(DATAfrob,m); %FROB station
[MPjcnb,dummy]=computeMatrixProfile(DATAjcnb,m); %JCNB station
Lfrob = length(MPfrob), Ljcnb = length(MPjcnb);
for i = 1 : Lfrob
   [minVal,minIdx]=min(MPjcnb(max(i-100,1):min(i+100,Ljcnb)));
   MPfrob(i) = max(MPfrob(i),minVal), Index(i) = minIdx;
end
```

Fig. 14.*top* shows the resulting matrix profile `MPfrob`, which is much "cleaner" than the one shown in **Fig. 12**. We presented the top 10 motifs extracted from this matrix profile to a seismologist [36] (the top 3 are shown in **Fig. 14**.*bottom*), and he verified that they are all true LFEs.

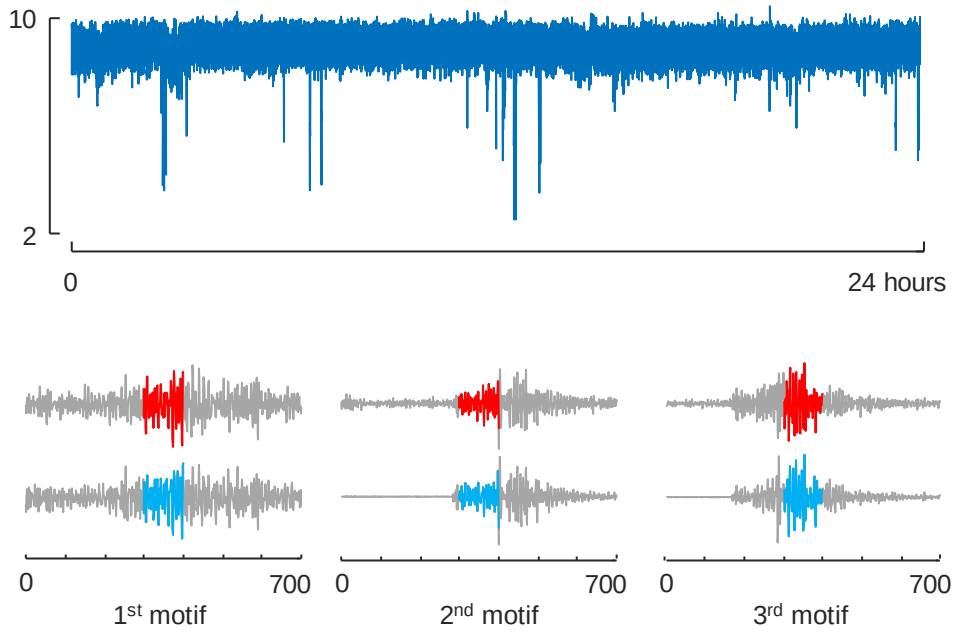

Fig. 14 *top*) The deep valleys in the resulting matrix profile all correspond to true events (compare to **Fig. 12**). *bottom*) The top 3 motifs extracted from the resulting matrix profile.

Besides *detecting* true LFEs, note that our simple strategy also provides extra implications for *locating* the LFE*s*. The time difference t_{diff} shown in **Fig. 13** can be found from the `Index` vector in our pseudocode, and if we know such time difference between 3 pairs of nearby seismic stations in the area, the exact location of the source of the LFE can be calculated. We reserve detailed analysis and further demonstration of such considerations for future work.

3.9 Automatically clustering time series motifs

Building on our previous example we consider applications of the Matrix Profile to clustering of seismic data. Seismic waveform clustering has been applied to earthquake relocation [41][44][35], repeating earthquake source identification [12][31][43][32][5][37][47] and volcano monitoring [38][22][38][2], helping to improve earthquake and volcanic hazard assessments. The seismology community has adopted various methods to cluster the seismic waveforms (time series subsequences corresponding to a seismic event) [2][41]. However, these methods take discrete, phase-aligned seismic waveforms of the same length as their input; extracting such waveforms from a long continuous seismic recording requires a lot of human effort. Here we introduce a simple method based on the Matrix Profile and ten lines of code, that can automatically discover earthquake pattern clusters from the continuous seismic recording.

To allow verification of the correctness of our result, we constructed a seismic time series by embedding twelve earthquake patterns into a 1,000-second-long snippet of seismic background noise, as shown in **Fig. 15**.*a*. The 12 embedded patterns are generated by four different earthquake sources (patterns of the same source are marked with the same color). The patterns corresponding to the same source normally look very similar to each other, while those corresponding to different sources are dissimilar. Our goal is to automatically discover the four natural clusters within the time series.

Before introducing our proposed solution, we would like to first dismiss some apparent solutions. Given the problem setting, the reader might consider finding the top-*k* motifs [28] here. Note that the top-*k* motifs normally refer to the top-*k* most similar *pairs* of subsequences in the time series. However, from **Fig. 15**.*a* we can see that a natural motif cluster can contain more than two occurrences of similar subsequences (e.g., the three red patterns are mutually similar); the classic top-*k* motif definition would separate them into different motif clusters, which is undesirable. The reader might also consider finding the *range* motifs [28] instead of top-*k* motifs. However, discovering range motifs requires setting a threshold parameter *r*: the maximum distance between any two subsequences in a motif cluster must not be

larger than 2*r*. We argue that such threshold is very difficult to set and needs very careful tuning. For example, if two subsequences have a Euclidean distance of three, are they similar enough to be considered as a motif? The answer is not that obvious even for a domain expert who knows the data well.

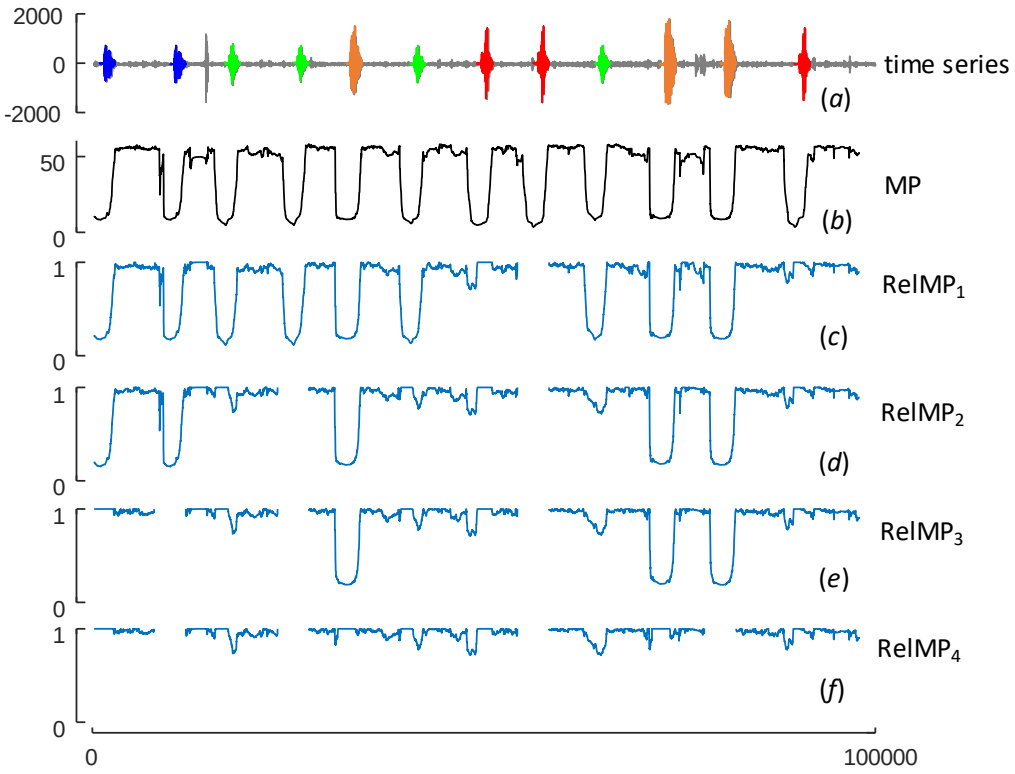

Fig. 15 *a*) A seismic time series with 12 earthquake patterns. These earthquakes are generated by four different sources. Patterns corresponding to the same source are marked in the same color. *b*) The matrix profile of the seismic time series. *c*) The relative matrix profile after the 1st motif pattern is removed. The deep valleys corresponding to the three red patterns disappeared. *d*) The relative matrix profile after the 2nd motif pattern is removed. The deep valleys corresponding to the four green patterns disappeared. *e*) The relative matrix profile after the 3rd motif pattern is removed. The deep valleys corresponding to the two blue patterns disappeared. *f*) The relative matrix profile after the 4th motif pattern is removed. The deep valleys corresponding to the three orange patterns disappeared.

Our solution can automatically find the natural number of motif clusters in the data (i.e., we do not need to specify how many clusters we would like to find), and requires only the setting of a much less critical parameter. The code is as follows:

```
[MP, dummy] = computeMatrixProfile(T, m);
RelMP = MP, i = 1, DissMP = inf(1, length(MP));
while i == 1 || min(RelMP) < 0.2
    [minVal(i), minIdx(i)] = min(RelMP);
    DissmissRange = T(minIdx-m+1 : minIdx+2*m-2);
    [JMP, dummy] = computeMatrixProfileJoin(T,DissmissRange,m);
    DissMP = min(DissMP,JMP); %dismiss all motifs discovered so far
    RelMP = MP ./ DissMP;
    i = i + 1;
end
```

We first compute the matrix profile `MP` corresponding to the input time series `T`, as shown in **Fig. 15**.*b*. We can see deep valleys in the vicinity of all the embedded earthquake patterns, as they all have close matches from the same source. We use the following iterative process to find the motif clusters one by one:

1. We find the minimum value in the current relative matrix profile `RelMP` (in the first iteration, we set `RelMP = MP`). This corresponds to a motif pattern (**Fig. 16**. shows the motif pattern discovered at each iteration).
2. We wish to avoid finding the same motif pattern in the next iteration. As such, we specify a `DissmissRange` which is a section of time series `T` that includes the current discovered motif pattern and its trivial matches, then compute the AB-join matrix profile `JMP` between the original time series `T` and `DissmissRange`. `JMP` measures how similar each subsequence is to the *current* discovered motif pattern.
3. We use a vector `DissMP` to store the distance between every subsequence and its closest match among *all* the motif patterns discovered so far. `DissMP` is initialized as infinity, and once we have computed `JMP`, we update `DissMP` with the element-wise minimum of `DissMP` and `JMP`.
4. We evaluate a "relative" matrix profile `RelMP` by dividing the original matrix profile `MP` with `DissMP`. Our intuition is that, if a subsequence has a very close nearest neighbor, but is very different from any of the discovered motifs

(and their trivial matches), then its `RelMP` value should be low. Note that the values in `RelMP` are always between 0 and 1.

5. If `min(RelMP) < 0.2`, go to step 1 and start the next iteration. Otherwise terminate the process.

From **Fig. 15**.*b-f*, we can see how `RelMP` changes through this iterative process (we use $\texttt{RelMP}_i$ to denote the status of `RelMP` at the end of the i^{th} iteration). After each iteration, several deep valleys corresponding to the earthquake patterns in the same color disappeared from `RelMP`. The process terminates after the 4th iteration, when there are no more valleys apparent in `RelMP`.

The reader might wonder how we define "apparent valleys" here. We set a termination threshold as `min(RelMP) = 1/5`. Recall that `RelMP(j)` measures the relative ratio between `MP(j)`, the distance from the j^{th} subsequence to its nearest neighbor and `DissMP(j)`, the distance from the j^{th} subsequence to its closest match among all the discovered motif patterns. If the j^{th} subsequence belongs to a new cluster, then it should be much more similar to its nearest neighbor than any of the discovered patterns. As such, we require that `MP(j)` cannot be more than 1/5 of `DissMP(j)`.

From **Fig. 16,** we can see that the discovered motifs at different iterations correspond to different earthquake sources (different colors), and the process terminates right after we have discovered all the four embedded earthquake clusters.

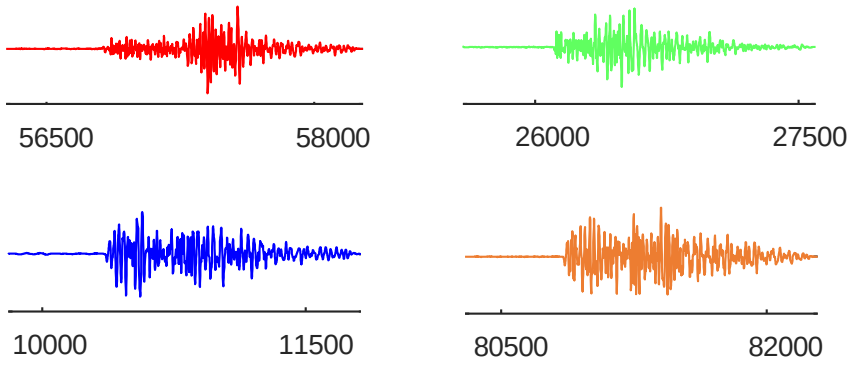

Fig. 16 *top.left*) The 1st motif pattern discovered, corresponding to the minimum point of the matrix profile MP in **Fig. 15**.*b*. *top.right*) The 2nd motif pattern discovered, corresponding to the minimum point of the relative matrix profile RelMP_1 in **Fig. 15**.*c*. *bottom.left*) The 3rd motif pattern discovered, corresponding to the minimum point of the relative matrix profile RelMP_2 in **Fig. 15**.*d*. *bottom.right*) The 4th motif pattern discovered, corresponding to the minimum point of the relative matrix profile RelMP_3 in **Fig. 15**.*e*.

3.10 Quantifying Parkinson Disease

Parkinson Disease (PD) is a neuro-degenerative disease which affects gait and mobility. To assess the severity of the disease, clinicians use the Hoehn and Yahr scale [1]. The original scale published in 1967 [18] ranges from 1 to 5, with the scores 1.5 and 2.5 added in a later revision to allow doctors to describe the progression of the disease.

In a recent paper, the authors propose exploiting the "recurrence" of gait time series as a method to automatically score patients on the Hoehn and Yahr scale [1]. They use a mathematically sophisticated definition of recurrence based on an embedding in a phase space, showing that various heuristic complexity measures of the recurrence quantification analysis correlate to the Hoehn and Yahr scale.

The phase "recurrence" in the title of this paper caught our attention. Time series motifs are simple *recurring* subsequences. It is natural to ask if the Matrix Profile could be used for this task. Our intuition here is simple. Imagine a person could walk with a near perfectly repeated gait cycle. If we computed the matrix profile of their gait telemetry, we would expect the matrix profile values to be very small, as every subsequence would have a near perfect match somewhere. In contrast, if a person has an irregularity to their gait (caused by a tremor, a hesitation, or stumble), we would expect that these movements will add unique elements to each gait. As such, these unique gaits cycle will not find such close matches among their neighbors and therefore the matrix profile will be higher. In **Fig. 17** we find some evidence to support the idea that people with more advanced PD have less well conserved gaits, at least on two randomly selected individuals.

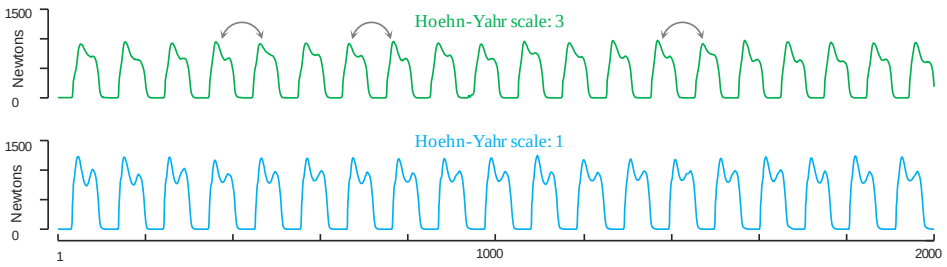

Fig. 17 A comparison of gait force profiles from two individuals walking for 20 seconds. One individual (top) was assessed by clinicians as being '3' on the Hoehn and Yahr scale. His gait cycles are not repeated perfectly, we have highlighted some of the most least conserved adjacent cycles. In contrast, another individual (bottom) who was assessed as '1' on the Hoehn and Yahr scale has a more conserved gait.

To see if this observation is more generally true, we experimented with the Parkinson Disease (PD) dataset provided by Hausdorff group [17], which is publicly available in "PhysioBank Database" [14]. The data consists of gait force profiles of 93 patients with idiopathic PD (disease severity levels 2, 2.5 and 3 in terms of Hoehn-Yahr scale) and 73 healthy control individuals (who we would expect to score at level 1). The data consists of vertical ground reaction force of subjects when walking normally at their own pace. The ground reaction force is the force exerted by the ground on an object in contact with it, in this case, the object is the foot.

We propose using the median of MP of sensor data as a proxy for the severity level of PD. We use the median rather than the maximum or sum, as the latter two functions would be brittle to a single unusual step. By visual inspection it appears that, for all participants, the 20th step is vastly different for the rest, presumably as the patient reached the end of the apparatus and turned around.

We can compute the score for each patient with just:

```
for i=1:size(1,patients)
   [MP,MPindex] = computeMatrixProfile(patients(i),100);
   HoehnYahrProxy = median(MP);
end
```

In **Fig. 18** we show how well this proxy score models the Hoehn-Yahr scale.

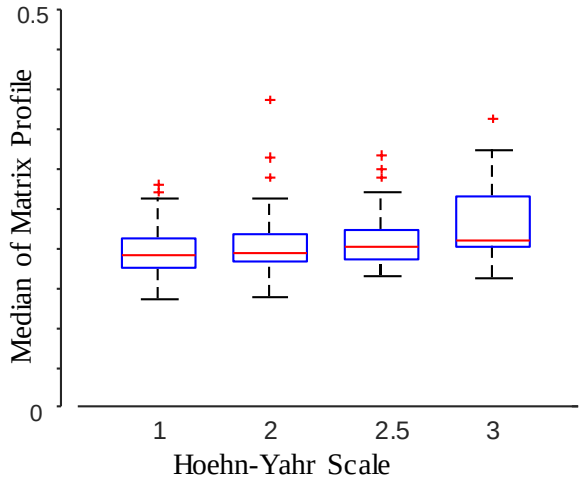

Fig. 18 The median of the Matrix Profile vs. the Hoehn-Yahr Scale (red horizontal bar) plotted within classic box and whisker plots. Note that the median does increase with the severalty of the Hoehn-Yahr scale.

It is important for us to disclaim that we are not making any medical claims for this experiment (we do not have such expertise), and we have not performed the type of statistical test that would pass muster in medical journal. Our point here, as always, is simply that the Matrix Profile and a few lines of code allow you to quickly test ideas that may be fruitful.

3.11 Scalability

Because the time taken for the Matrix Profile depends only on *n*, and not on the motif length *m* or the structure of the data, we can summarize the time complexly for *all* experiments with a single table as shown in Table 2. This is another very useful property of the Matrix Profile, which stands in contrast to almost all other methods. For example, *Quick-Motif* may be able to process a million datapoints on smooth data in about four minutes on a standard desktop [24]. But for noisier data, the time required could balloon by a factor of ten or more [57].

Table 2: Time taken to compute a Matrix Profile for datasets ranging from 2^{18} to 2^{23} using three computational paradigms. Note that for small datasets a standard desktop can beat HPC due to setup costs, but that advantage disappears as we see larger datasets.

Instance Type / Input Size	Experiments of that size	Desktop CPU *Seconds*	c5.18xlarge (72 cores) 3.06 USD/hr *Seconds*	p3.2xlarge (1 Tesla V100) 3.06 USD/hr *Seconds*
2^{18}	3.2, 3.3, 3.4, 3.7, 3.9, 3.10	6.4	7	0.28
2^{19}	3.5	25.3	14	0.68
2^{20}	3.6	99.9	32	2.00
2^{21}	3.8	397.8	76	7.00
2^{22}		1584.8	252	25.80
2^{23}	3.1	6333.3	933	96.80

The three computational approaches considered were: a standard desktop, a 72-core c5 18xlarge spot instance (Intel Skylake architecture), and an Amazon Web Service p3.2xlarge (1 Tesla V100) which cost about 3.06 USD/hour at the time of writing.

Note that the time taken for the desktop version does get prohibitive for large datasets. However, note that the if the data discussed in Section 3.1 was sampled at 1 Hz, then the 6,106,456 datapoints would represent about 70 days. Thus, even if we use a standard desktop, we can compute the Matrix Profile about 1,000 times faster than real-time for this problem. Moreover, recall that this is for the fully converged Matrix Profile. As [56] shows, we can typically closely approximate in the Matrix Profile in about $1/100^{th}$ of the time to complete convergence.

4 Conclusion

We have presented ten time series data mining case studies in which interesting and actionable results can be obtained given just a handful of lines of code. Moreover, these results can be obtained essentially instantaneously, given our assumption that the computation of the Matrix Profile is free. That last assumption needs a little qualification. The existing Matrix Profile algorithms, STAMP [52], and STOMP [57] are effectively instantaneously for many end users, who have only tens of thousands of data points. For example, the Parkinson Disease, music processing, and classification (shapelet) examples all have this property. For users with say one-hundred thousand to one million datapoints, they may need to wait minutes for their results. Our meter-swapping example is of this scale. Note, however, that the minutes of computation time are dwarfed by the full year of time it took to collect the data. For users with datasets in the many millions, computation time becomes more of an issue, but even here there is hope on the horizon. In a recently published paper we introduced a novel algorithm called SCRIMP++ [56] that can approximate the Matrix Profile even faster than STAMP, allowing a user with say a million datapoints to obtain a high quality approximate Matrix Profile in "interactive" time (a handful of seconds), and in an upcoming work we will show that a GPU cluster optimized algorithm called SCAMP can scale up to billion length time series. Moreover, we suspect that others in the community will find ways to accelerate the Matrix Profile that did not occur to us.

Acknowledgements

We gratefully acknowledge funding from NSF IIS-1161997 II, NASA award NNX15AM66H, USGS G16AP00034, MERL Labs and Samsung, and all the data donors.

References

[1] Afsar, O., Tirnakli, U., and Marwan, N. *Recurrence Quantification Analysis at work: Quasi-periodicity based interpretation of gait force profiles for patients with Parkinson disease*. Scientific Reports 8.1 (2018): 9102.

[2] Bardainne, T., Gaillot, P., Dubos-Sallée, N., Blanco, J., and Sénéchal, G. *Characterization of seismic waveforms and classification of seismic events using chirplet atomic decomposition. Example from the Lacq gas field (Western Pyrenees, France).* Geophysical Journal International, 166(2): 699-71

[3] Batista, G.E.A.P.A., Keogh, E.J., Tataw, O.M. and De Souza, V.M.A. *CID: an efficient complexity-invariant distance for time series.* Data Min. Knowl. Discov. 28.3 (2014): 634-669.

[4] Bayardo, R. J., Ma, Y., and Srikant, R. *Scaling Up All Pairs Similarity Search*. WWW 2007, pp 131-140.

[5] Beeler, N.M., Lockner, D.L., and Hickman, S.H. *A simple stick-slip and creep-slip model for repeating earthquakes and its implication for microearthquakes at Parkfield*. Bulletin of the Seismological Society of America, 91.6(2001): 1797-1804.

[6] Begum, N., and Keogh, E. *Rare Time Series Motif Discovery from Unbounded Streams*. PVLDB 8.2 (2014): 149-160.

[7] Bonds, M.E., *Haydn's' Cours complet de la composition' and the Sturm und Drang*. Haydn studies, pp.152-76, 1998.

[8] Chandola, V., Cheboli, D., and Kumar, V. *Detecting Anomalies in a Time Series Database*. UMN TR09-004.

[9] Chen, Y. et al. *The UCR Time Series Classification Archive.* http://www.cs.ucr.edu/~eamonn/time_series_data/

[10] Convolution - Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Convolution. Accessed: 2016-01-19.

[11] Ding, H., Trajcevski, G., Scheuermann, P., Wang, X., and Keogh, E. J. *Querying and Mining of Time Series Data: Experimental Comparison of Representations and Distance Measures*. PVLDB 1(2): 1542-1552. 2008.

[12] Geller, R.J. and Mueller, C.S. *Four similar earthquakes in central California*. Geophysical Research Letters, 7.10(1980): 821–824

[13] Gharghabi, S., Ding, Y., Yeh, C.C.M., Kamgar, K., Ulanova, L. and Keogh, E. *Matrix Profile VIII: Domain Agnostic Online Semantic Segmentation at Superhuman Performance Levels*. IEEE ICDM 2017: 117-126.

[14] Goldberger, A.L. et al. *PhysioBank, PhysioToolkit, and PhysioNet: components of a new research resource for complex physiologic signals*. Circulation 101.23 (2000): e215-e220

[15] Guillame-Bert, M. and Dubrawski, A., 2017. *Classification of time sequences using graphs of temporal constraints*. The Journal of Machine Learning Research, 18(1), pp.4370-4403.

[16] Gupta, S., Reynolds, M.S. and Patel, S.N. *ElectriSense: single-point sensing using EMI for electrical event detection and classification in the home.* In Proceedings of the 12th ACM international conference on ubiquitous computing, pp. 139-148, 2010.

[17] Hausdorff, J.M., Ladin, Z., and Wei, J.Y. *Footswitch system for measurement of the temporal parameters of gait*. Journal of biomechanics 28.3 (1995): 347–351.

[18] Hoehn, M.M and Yahr, M.D. *Parkinsonism: onset, progression and mortality*. Neurology 17.5 (1967): 427–442.

[19] Kao, H.Y. and Yu, J.Y. *Contrasting eastern-Pacific and central-Pacific types of ENSO*. Journal of Climate, 22.3(2009): 615-632.

[20] Kate, P.G. and Rana, J.R. *ZIGBEE based monitoring theft detection and automatic electricity meter reading.* 2015 International Conference on Energy Systems and Applications : 258-262.

[21] Kurpiewski, M.R., Engler, L.E., Wozniak, L.A., Kobylanska, A., Koziolkiewicz, M., Stec, W.J. and Jen-Jacobson, L. *Mechanisms of coupling between DNA recognition specificity and catalysis in EcoRI endonuclease*. Structure, 12.10(2004): 1775-1788.

[22] Lahr, J.C., Chouet B.A., Stephens C. D., Powers J. A., and Page R.A. *Earthquake classification, location, and error analysis in a volcanic environment: Implications for the magmatic system of the 1989–1990 eruptions at Redoubt Volcano*, Alaska, J. Volcanol. Geotherm. Res., 62 (1994): 137–152

[23] LG Dishwasher Owners Manual. URL retrieved on June 21st, 2017. http://www.lg.com/us/support/products/documents/Owners%20Manual.pdf

[24] Li, Y., Yiu, M.L. and Gong, Z. *Quick-motif: An efficient and scalable framework for exact motif discovery.* ICDE 2015: 579-590.

[25] Lin, J., Khade, R. and Li, Y. *Rotation-invariant similarity in time series using bag-of-patterns representation*. Journal of Intelligent Information Systems. 39. 2 (2012): 287-315.

[26] Morris, D., Saponas, T.S., Guillory, A. and Kelner, I. *RecoFit: using a wearable sensor to find, recognize, and count repetitive exercises*. ACM CHI 2014: 3225-3234.

[27] Mueen, A., Hamooni, H., and Estrada, T. *Time Series Join on Subsequence Correlation*. IEEE ICDM 2014, pp. 450-459.

[28] Mueen, A., Keogh, E., Zhu, Q., Cash, S. and Westover, B. *Exact Discovery of Time Series Motif*. SDM 2009.

[29] Murray, D., Liao, J., Stankovic, L., Stankovic, V., Hauxwell-Baldwin, R., Wilson, C., Coleman, M., Kane, T. and Firth, S. *A data management platform for personalised real-time energy feedback*. In Proceedings of the 8th International Conference on Energy Efficiency in Domestic Appliances and Lighting Lighting (EEDAL), 2015: 1–15.

[30] Music performance of Joseph Haydn's Symphony No. 47 in G major, by the Tafelmusik Orchestra. URL Retrieved July 4th 2017. www.youtube.com/watch?v=yeB_Ohpsm64

[31] Nadeau, R.M., Foxall, W., and McEvilly, T.V. *Clustering and periodic recurrence of microearthquakes on the San Andreas Fault at Parkfield, California*. Science, 267.5197(1995): 503–507

[32] Nadeau, R.M., and McEvilly, T.V. *Fault slip rates at depth from recurrence intervals of repeating microearthquakes*. Science, 285.5428(1999): 718–721.

[33] Puder, J. *Seventeen Synonyms of Semordnilap*. Word Ways, 33.1(2010): 9.

[34] Reiss, A. and Stricker, D., *Introducing a new benchmarked dataset for activity monitoring*. In 16th International Symposium on Wearable Computers (ISWC), 2012, pp 108–109.

[35] Richards-Dinger, K.B. and Shearer, P.M. *Earthquake locations in southern California obtained using source-specific station terms*. Journal of Geophysical Research: Solid Earth, 105.B5 (2000): 10939-10960

[36] Shakibay-Senobari, N. Personal Correspondence. June 14, 2018.

[37] Shelly, D.R., Beroza, G.C., Ide, S., and Nakamula, S. *Low-frequency earthquakes in Shikoku, Japan, and their relationship to episodic tremor and slip*. Nature 442.7099(2006): 188-191

[38] Sherburn, S., Scott, B.J., Nishi, Y., and Sugihara, M. *Seismicity at White Island volcano, New Zealand: a revised classification and inferences about source mechanism*, J. Volc. Geotherm. Res., 83 (1998): 287–312

[39] Sreenivasan, G. *Power Theft*. NewDelhi: PHI Learning Pvt. Ltd. 2016.

[40] The UCR Matrix Profile Page. URL retrieved July 9th, 2017. www.cs.ucr.edu/~eamonn/MatrixProfile.html

[41] Trugman, D.T., and Shearer, P. M. GrowClust: A hierarchical clustering algorithm for relative earthquake relocation, with application to the Spanish Springs and Sheldon, Nevada, earthquake sequences. Seismological Research Letters, 88.2A (2017): 379–391.

[42] Ueno, K., Xi, X., Keogh, E. J., and Lee, D.-J. *Anytime Classification Using the Nearest Neighbor Algorithm with Applications to Stream Mining*. ICDM 2006.

[43] Vidale, J.E., ElIsworth, W.L., Cole, A. and Marone, C. *Variations in rupture process with recurrence interval in a repeated small earthquake*. Nature, 368.6472(1994): 624-629.

[44] Waldhauser F. and Ellsworth W.L. *A double-difference earthquake location algorithm: Method and application to the northern Hayward fault*, Bull. Seism. Soc. Am., 90 (2000): 1353-1368

[45] Wang, J., Liu, P., She, M.F., Nahavandi, S. and Kouzani, A. *Bag-of-words representation for biomedical time series classification.* Biomedical Signal Processing and Control, 8.6 (2013): 634-644.

[46] Wikipedia entry for Tasmanian devil. Retrieved June 12th 2017. https://en.wikipedia.org/wiki/Tasmanian_devil

[47] Wisely, B.A., Schmidt, D.A., and Weldon II, R.J. *Compilation of Surface Creep on California Faults and Comparison of WGCEP 2007 Deformation Model to Pacific-North American Plate Motion* (No. 2007-1437-P). Geological Survey (US), 2008.

[48] Yankov, D., Keogh, E., Medina J., Chiu, B. and Zordan, V. *Detecting time series motifs under uniform scaling*. ACM SIGKDD 2007: 844-853

[49] Ye, L., and Keogh, E. *Time Series Shapelets: A New Primitive for Data Mining*. ACM SIGKDD 2009: 947-956.

[50] Yeh, C.C. M., Herle, H. V., and Keogh, E. *Matrix Profile III: The Matrix Profile Allows Visualization of Salient Subsequences in Massive Time Series*. To be appeared in IEEE ICDM 2016.

[51] Yeh, C.C.M., Kavantzas, N. and Keogh, E. *Matrix profile IV: using weakly labeled time series to predict outcomes. Proceedings of the VLDB Endowment*, 10.12 (2017): 1802-1812.

[52] Yeh, C.C. M., Zhu, Y., Ulanova, L., Begum, N., Ding, Y., Dau, H. A., Silva, D. F., Mueen, A., and Keogh, E. *Matrix Profile I: All Pairs Similarity Joins for Time Series: A Unifying View that Includes Motifs, Discords and Shapelets*. To be appeared in IEEE ICDM 2016: 1317-1322.

[53] Yeh, C.C.M., Zhu, Y., Ulanova, L., Begum, N., Ding, Y., Dau, H.A., Zimmerman, Z., Silva, D.F., Mueen, A. and Keogh, E., 2018. *Time series joins, motifs, discords and shapelets: a unifying view that exploits the matrix profile*. Data Mining and Knowledge Discovery 32(1): 83-123.

[54] Zhang, M. and Sawchuk, A. USC-HAD: a daily activity dataset for ubiquitous activity recognition using wearable sensors. UbiComp 2012: 1036-1043.

[55] Zhu, Y., Imamura, M., Nikovski, D., and Keogh, E. *Matrix Profile VII: Time Series Chains: A New Primitive for Time Series Data Mining*. IEEE ICDM 2017: 695-704.

[56] Zhu, Y., Yeh, C.C.M., Zimmerman, Z., Kamgar, K., and Keogh, E. *Matrix Profile XI: SCRIMP++: Time Series Motif Discovery at Interactive Speeds*. IEEE ICDM 2018 (to appear).

[57] Zhu, Y., Zimmerman, Z., Senobari, N. S., Yeh, C.-C. M., Funning, G., Mueen, A., Brisk, P., and Keogh, E. *Matrix Profile II: Exploiting a Novel Algorithm and GPUs to Break the One Hundred Million Barrier for Time Series Motifs and Joins*. IEEE ICDM 2016: 739-748.

[58] Zilberstein, S. and Russell, S. *Approximate Reasoning Using Anytime Algorithms*. In Imprecise and Approximate Computation, Kluwer Academic Publishers, 1995.

[59] Supporting Webpage: https://sites.google.com/site/matrixprofiletopten/