UNIVERSITY OF CALIFORNIA
RIVERSIDE

Novel Multiresolution Pattern Detection Techniques in Large Scale Time Series Data

A Dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy

in

Computer Science

by

Maryam Shahcheraghi

September 2024

Dissertation Committee:
- Dr. Eamonn Keogh, Chairperson
- Dr. Vagelis Papalexakis
- Dr. Yue Dong
- Dr. Silas Richelson

The Dissertation of Maryam Shahcheraghi is approved:

_____

_____

_____

_____

Committee Chairperson

University of California, Riverside

Acknowledgement

I've been eagerly counting down the days until I write my final PhD thesis. It's been a long and challenging journey for me, as it has been for many other students in their PhD program. From graduate school to marriage, immigration, navigating Covid, and becoming a mom, I've learned that life is a continuous journey.

During this remarkable journey, I had the privilege of working under the supervision of Prof. Eamonn Keogh – the best supervisor I could have imagined. He's not only an expert in his field but also supportive and meticulous, paying attention even to the smallest details.

I'd like to express my gratitude to all my mentors and lab mates who have supported me throughout this journey. Special thanks to Dr. Zach Zimmermann for his unwavering guidance over the years. Additionally, I'd like to extend my appreciation to my senior lab mates: Dr. Ryan Mercer, Dr. Renjie Wu, and Dr. Audrey Der.

I am also grateful to my committee members, both during the proposal and final dissertation stages: Prof. Vagelis Papalexakis, Prof. Yue Dong, and Prof. Silas Richelson.

To my beloved parents, Mojgan and Mohammad, Maman and Baba, I couldn't have reached this point without your endless love, care, sacrifices, and support. I am also thankful to my in-laws, brothers, and sisters-in-law for their tremendous support and encouragement. I am truly blessed to have you all in my life.

And to my love, my best friend, Yasin – you have been there for me whenever I needed you, offering support and encouragement. There were so many moments when I felt like

giving up, but you were always there to lift my spirits with your sympathy, advice, and wise comments even on coding or foundational ideas.

Lastly, to my little angel, Mana – you have been my sweet companion throughout this journey. Even during times of stress or exhaustion, your presence reminds me of the importance of love and family. I strive to show you my love, even when I am tired or frustrated, because you mean the world to me.

In the name of the Source of Wisdom

ABSTRACT OF THE DISSERTATION

Novel Multiresolution Pattern Detection Techniques in Large Scale Time Series Data

by

Maryam Shahcheraghi

Doctor of Philosophy, Graduate Program in Computer Science
University of California, Riverside, September 2024
Dr. Eamonn Keogh, Chairperson

Time series data are one the most important data type in data science field. Analyzing the local patterns in time series data can be leveraged in diverse domains such as astronomy, biology, entomology, economics, etc. Time series analytical reports can help the experts in these field to not only explain the behavior of their system, but to make predictions and even get an insight into what has been unknown for them. For instance, the repetition of a behavior in a different scale or speed might have been unseen from a sight of the experts in a large scale time series record. Anomalies can be found in time series data fairly easily. Approximately conserved patterns in time series, motifs, are another term of our interest.

Time series similarity matrices (informally, recurrence plots or dot-plots), are useful tools for time series data mining. They can be used to guide data exploration, and various useful features can be derived from them and then fed into downstream analytics. However, time series similarity matrices suffer from very poor scalability, taxing both time and memory requirements. In this dissertation, we introduce novel ideas that allow us to scale the largest time series similarity matrices that can be examined by several orders of magnitude. The

first idea is a novel algorithm to compute the matrices in a way that removes dependency on the subsequence length. This algorithm is so fast that it allows us to now address datasets where the memory limitations begin to dominate. Our second novel contribution is a multiscale algorithm that computes an approximation of the matrix appropriate for the limitations of the user's memory/screen-resolution, then performs a local, just-in-time recomputation of any region that the user wishes to zoom-in on. Given that this largely removes time and space barriers, human visual attention then becomes the bottleneck. We further introduce algorithms that search massive matrices with quadrillions of cells and then prioritize regions for later examination by either humans or algorithms. We demonstrate the utility of our ideas for data exploration, segmentation, and classification in domains as diverse as astronomy, bioinformatics, entomology, and wildlife monitoring. Moreover, as noted approximately repeated subsequences in a longer time series, i.e. *time series motifs*, are important primitive in time series data mining. Motifs are used in dozens of downstream tasks, including classification, clustering, summarization, rule discovery, segmentation etc. Time series motif discovery is notoriously computationally expensive task. Some motif discovery algorithms are fast in the *best* case, but in other datasets, even if both the data and motif lengths are held the same, both their time and space complexity can explode. The Matrix Profile has the nice property that its time and space complexity are independent of the data. Moreover, the Matrix Profile is fast enough for datasets in the with say one million datapoints, which covers a large fraction of user cases. However, there are situations where we may wish to consider datasets which are much larger. We introduce the first lower bound for the Matrix Profile and an algorithm that exploits that lower bound to

allow orders of magnitude speeds on most real-world datasets. We demonstrate the utility of our ideas with the largest and most ambitious motif discovery experiments ever attempted.

Table of Contents

Table of Figures

List of Tables

1 INTRODUCTION

There are many tasks that researchers routinely perform on time series, including classification, clustering, segmentation, anomaly detection, etc. However, given a new dataset, the first task is typically to simply gain an understanding of that data, in particular the relationship among the subsequences within it [50][54]. One way to do this is to use a *recurrence plot.* Given a long time series and a user-specified subsequence length, it is possible to construct a similarity matrix with colors (or shades of gray) representing the distance between all possible pairs of subsequences. Variants of these plots are also called dot plots, self-similarity matrices, similarity plots, time series similarity matrices, etc. [11][28]. For concreteness we will call the variant of interest here *time series similarity matrix plots* or just *Mplots*. Mplots have many uses in data mining. They can be used for visual exploration of data, or various features can be extracted from them, and then fed into other algorithms. For example, the *Matrix Profile* is an increasingly popular time series analytical tool that is directly extracted from a Mplot, by recording the smallest (off-the-diagonal) value in each column [50][54][56].

A simple Mplot makes comparisons *within* time series **A**, as shown in Fig.1, and we can also use this representation to compare and contrast *between* two time series **A** and **B**, with a variant we call an AB-Mplot. Such plots allow us to understand where two time series are similar and different.

Mplots (under the different names noted above) are used in astronomy [35], economics [43], music [15], physiology [46][47], neuroscience [25], earth sciences [3], medicine [1][53] and engineering [28]. As noted in a founding paper on the topic, "*information*

obtained from recurrence plots is often surprising, and not easily obtainable by other methods" [11].

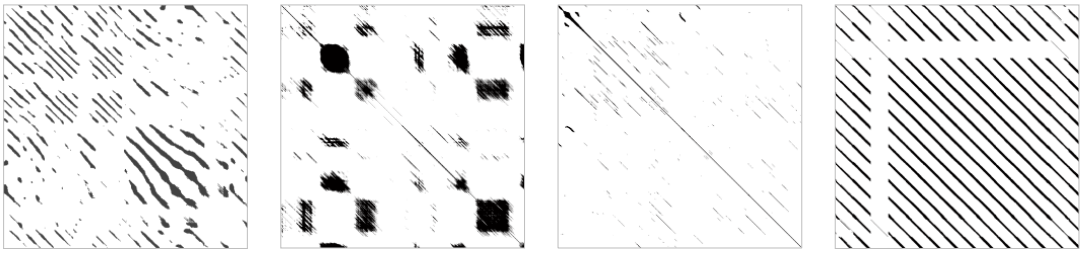

Fig.1 Examples of Mplots hint at the diversity and utility of this data structure. Like many plots in the paper, these figures suffer somewhat from the size of reproduction. We encourage the interested reader to visit [41] which has larger figures and videos. Here we show binarized Mplots, but more generally Mplots allow a spectrum of colors to indicate degree of similarity.

In spite of this ubiquity, it is surprising that they are not used *more* often in the data mining community. We believe that this is because of the following three bottlenecks:

- **CPU**: Classic Mplots require processing that is quadratic in the length of the time series, and linear in the length of the subsequences. This seems to have limited their use to time series with a length of about 20,000 [11][28][26][35].
- **Memory**: If a researcher has spent significant resources to obtain a long time series of say length 100,000, she may well be willing to wait hours or even days to compute a Mplot, in order to glean information from her dataset. However, she is unlikely to have the requisite 80 gigabytes of main memory to work with.
- **Human Visual Attention/Screen Resolution:** Even if a user could somehow bypass the two difficulties above, this would eventually lead to the situation where her ability to visually scan the Mplot, and the ability of a standard screen to display such a huge matrix, become bottlenecks. For example, in Section 3.3.5 we compute a Mplot that if

printed out on the scale used in the figures in this paper, would cover a soccer field. Clearly such Mplots would defy any attempt at human visual inspection.

In this work we introduce techniques to solve all the above issues. We begin by showing how we can reduce the amortized time to compute a single cell of a Mplot to just O(1), not the current O(m), where m is the subsequence length. Because m can be >1,000, this means we can compute Mplot up to three orders of magnitude faster than is currently possible. Moreover, as we will show, for truly ambitious datasets, we have ported our ideas to GPUs, to allow us to compute a Mplot with quadrillions of cells.

We further show how we can address the memory bottleneck by the introduction of a multiscale algorithm that computes an approximation of the matrix appropriate for the limitations of the user's screen/memory, then performs a local, just-in-time recomputation of any region that the user wishes to zoom-in on. Finally, we show that for truly massive Mplots, we can create algorithms that can build the matrices "patchwise" and search each patch for features that a user may wish to have drawn to her attention. This removes human visual attention as a bottleneck for Mplots.

The next two chapters cover the technical information and experimental results for Mplot. Chapter 2 starts with the necessary definitions and notations, along with a review of related work. It then provides a detailed explanation of the SPLAT algorithm. Chapter 3 offers guidance on how to interpret Mplot, followed by a discussion on reverse engineering to identify interesting patterns within it. Finally, it presents the experimental results for Mplot, including both anecdotal evidence and speedup results.

2 INTRODUCING MPLOTS: SCALING TIME SERIES RECURRENCE PLOTS TO MASSIVE DATASETS

This chapter provides technical background needed to understand Mplots. We will start with the key definitions and notations. Then, we will explain the SPLAT algorithm in detail. Additionally, we will review the related work to place our contribution in context.

2.1 Definitions And Notation

Our data type of interest is *time series*.

Definition 1: A *time series* $\mathbf{T} = t_1, t_2, \ldots, t_n$ is a sequence of real-valued numbers.

For the task-at-hand, we are not interested in global properties of a time series but rather the relationships between small regions of the time series called *subsequences*.

Definition 2: A *subsequence* $\mathbf{T}^{(i,m)}$ is a contiguous subset of values from **T** starting at index i with length m.

We can measure the distance between any two time series *subsequences* of equal length using a distance measure. In this work, we use the ubiquitous z-normalized Euclidean distance [50]. Note the subsequence length here takes on a similar role to the *embedding dimension* in discrete dot plots [28][26]. However, the implications of changing lengths are more complex in our case. Making the embedding dimension larger can only make a dot plot sparser and decrease the length of "runs" in the plots. However, because we are working in the z-normalized space, longer subsequences can have a *lower* Euclidean distance, and therefore produce longer runs. We will return to the observation later in this work.

If we need to measure the distance between a short time series and every subsequence from a long time series, we can produce a *distance profile*.

Definition 3: A *distance profile* $\mathbf{DP}_{\mathrm{AB}}^{(j,m)}$ is the vector of distances between each subsequence in a reference time series $\mathbf{T}_{\mathrm{A}}$ and a query subsequence $\mathbf{T}_{\mathrm{B}}^{(j,m)}$.

The distance can be computed very efficiently using the MASS algorithm [32]. However, if we are limited by time, we can perform the classic trick of computing the distance profile on a downsampled version of **A**, using a similarly downsampled version of **B**. We propose to use Piecewise Aggregate Approximation (PAA) to downsample the data [22]. If we wish to downsample a time series by a factor of *d*, we indicate this by PAA(**A**,*d*). As Fig.2 shows, on many datasets it is possible to significantly downsample the data, while retaining the essential features.

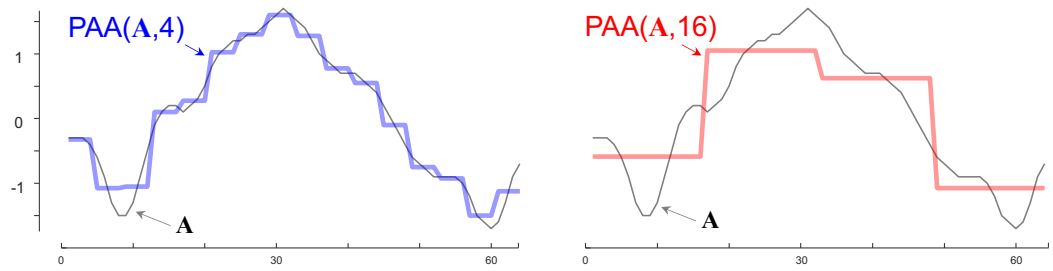

Fig.2 A time series A with n = 64 downsampled with PAA. *left*) Downsampled 1 in 4, *right*) Downsampled 1 in 16. While the 1 in 16 plot has lost significant detail, the 1 in 4 downsampling does preserve the basic shape of the time series.

Note that downsampling may be a particular attractive strategy here, as the memory and time savings are *quadratic* in the downsampling rate. Although the Mplot has been informally introduced and compared to recurrence plots (dot plots), for concreteness we define it here.

Definition 4: A *Mplot* is the visualized matrix of distance profiles. Each row *j* of this matrix is $\mathbf{DP}_{\mathrm{AB}}^{(j,m)}$.

A Mplot is, by its nature, a dense *real-valued* matrix. However, for better visualization (especially for a figure in a paper) we often binarized its values to either see the highest or lowest values. Binarizing Mplots helps to only display the values in the desired range. So, a user who is interested in subsequences with high similarity (motifs)/ low similarity (discords) can set a threshold to only see the values in that range.

When **A** = **B**, this definition is logically equivalent to a self-join of **A**. The popular Matrix Profile is simply the vector of length |**A**| that contains the minimum (non-diagonal) value in each column [50][54][56]. The state-of-the-art Matrix Profile algorithms (SCRIMP, SCAMP) can compute this incrementally, without ever having to have the entire matrix in main memory at one time. When **A** ≠ **B**, this definition is logically equivalent to an **AB**-join. Such joins are frequently used in recurrence plots to visualize the differences between two DNA sequences, but surprisingly, to the best of our knowledge, there are very few uses of real-valued **AB**-join time series.

With these definitions and notations in place, we can now review the related work to better understand the context and significance of our contributions.

2.2 Related Work

As noted in the introduction, the basic idea of creating a matrix to represent the similarity of subsequences has many names, and the literature is not consistent in naming conventions. It is important that we differentiate Mplots from *true* recurrence plots. Mplots are superficially similar to recurrence plots (dot plots) which are often used in

bioinformatics, linguistics, etc. [19]. Moreover, many of the uses of recurrence plots are also uses of Mplots. However, it is worth explicitly pointing out some of the differences between them:

- Dot plots are discrete. Every cell in the matrix is binary. In contrast Mplots must be real-valued, as we may be interested in relative degrees of similarity.
- As a consequence of the binary nature of dot plots, they are normally extremely sparse, with typical densities less than 0.000001. This means that space complexity is rarely an issue for dot plots (by exploiting sparse matrix support in many programming languages).
- Each cell in a dot plot is the result of an equality test comparing two scalers, such as 'T'='A'? In contrast each cell in a Mplot is the result of a distance comparison between two vectors, which can have a length of over 1,000. Moreover, these vectors need to be normalized before being compared (surprisingly, normalization generally takes longer than the distance computation [37]). This means that Mplots may take orders of magnitude longer to be computed.
- Dot plots are only useful for finding similarity (i.e., *conservation*). In contrast, with Mplots we may wish to compare two datasets where we expect conservation, *and/or* violations of conservation (i.e., *dissimilarity*).

Because of these many differences between Mplots and recurrence plots, little of the vast literature on efficient construction of the latter is helpful in scaling up the former. Nevertheless, most of the utility of visualizing recurrence plots also applies to Mplots.

There are many creative ways to visualize time series, see [13] and the references therein. However, Mplots are particularly direct and intuitive. Moreover, unlike say Viztrees [24], they preserve the temporal information context. For example, if we examine a year's worth of transaction time series and our eye is drawn to a motif that occurs at about 12% across and 40% down, we can use our intuition to guess that these events might correspond to Valentine's Day and Mother's Day[1], two days with similar spending patterns on flowers and restaurants.

Having reviewed the related work, we now turn our attention to the SPLAT algorithm, which serves as the foundation of our approach. We will also explain the main bottlenecks and how they are addressed in SPLAT.

2.3 Algorithms That Scale up Mplots

In this section we introduce three novel ideas that allow us to scale up the largest size of Mplot that can be considered by several orders of magnitude. We begin by addressing the CPU bottleneck.

2.3.1 Removing the CPU Bottleneck

It is clear that a Mplot's time complexity must be at least $O(n^2)$ (This is *not* the case for true dot plots, which can be constrained to be arbitrarily sparse, and use various hashing-based optimizations). However, the time complexity is actually $O(m \times n^2)$ [28][35]. As we show in Section 3.3, m can be in the thousands. Moreover, this complexity hides some constant factors. As we are working with z-normalized time series, the time taken to perform the z-normalization is actually greater than the time needed for the Euclidean

[1] Here we assume the USA Mother's Day

distance calculation [37]. In this section we show that we can completely remove the dependence on m and make Mplot's a true O(n^2) algorithm with tiny constant factor.

The idea of making the time complexity of a Mplot independent of the m value is similar to the Matrix Profile algorithms proposed in [54][56]. Let us consider the formula for calculating a cell of distance profile ($d^{(i,j)}$).

$$d^{(i,j)} = \sqrt{2m(1 - \frac{QT^{(i,j)} - m\mu^i\mu^j}{m\sigma^i\sigma^j})} \quad \textbf{(1)}$$

Where, $d^{(i,j)}$ is assumed to be the Euclidean distance of z-normalized subsequences. $QT^{(i,j)}$, is the dot product of corresponding subsequences. μ^i and σ^i are the mean and standard deviation of $T^{(i,m)}$, respectively.

In Table 1 we introduce an algorithm that exploits these observations. We call our algorithm SPLAT, Scalable Processing of LArger Time series.

The SPLAT algorithm starts by initializing the Mplot matrix in line 1. The matrix row and column count equal the number of subsequences in T_A and T_B, respectively. Line 2 precomputes the mean and standard deviation of each subsequence of input time series. By updating the QT values in line 6, the distance profile of the reference time series and the query is calculated as shown in line 7. Finally, with line 8, the Mplot matrix value of each cell is updated.

Table 1. The SPLAT Algorithm to compute Mplots.

Function: SPLAT(T_A, T_B, m)
Input: Reference time series T_A, Query time series T_B, Subsequence length m
Output: Distance matrix Mplot

```
1  Mplot = nan(length(T_A)-m+1, length(T_B)-m+1)
2  μ, σ ← computeMeanStd(T_A, T_B, m)                    //see [37]
3  for diag = 1 to Mplot_column_count
4     for row = 1 to Mplot_row_count – diag + 1
5        col ← row + diag – 1
6        QT_row,col ← ComputeDotProduct(T_A^(col,m), T_B^(row,m))    //see [54]
7        d ← CalculateDistanceProfile(T_A^(col,m), T_B^(row,m))      //see (1)
8        Mplot(row, row + diag - 1) = d
9  return Mplot
```

The SPLAT algorithm defined here is a general case where two distinct time series are compared (AB-join), however if we set both input time series as T_A, this algorithm computes the special case of self-join similarity. For the self-join case, we can trivially make the algorithm twice as fast by exploiting the symmetry about the diagonal.

By taking advantage of the techniques in [37][54] and [56] in addition to the mean and standard deviation, the dot product can also be calculated in O(1). So, a time complexity of O(n^2) is achieved which is the minimum required to compute all the values in a Mplot with $n \times n$ cells.

The SPLAT algorithm can efficiently compute large Mplots, but we may task it with a long time series that would take longer to compute than the user's patience allows. To address this issue, we can create a *contract algorithm* version of SPLAT, parameterized by the maximum amount of time the user is willing to wait [55]. For example, in SPLAT(**A**,**B**,*m*,4), the user is requesting the best approximation that can be computed in four seconds or less.

To achieve this user-requested time limit, we will approximate the time series with PAA (Recall Fig.2). We will use the absolute *minimum* amount of downsampling to achieve this user-requested acceleration. This is easy to implement. Suppose we have previously performed a calibration run with a Mplot with |**A**| =10,000, and found it took *S* seconds. We can then predict that building a Mplot for size time series **T**, of length n', will take *SplatTimePredict*(**T**, n') = $S * (n'/|\mathbf{A}|)^2$.

If this is within our time budget, there is nothing to do. If this takes longer than our user supplied time budget, we then reduce **T** to create $\mathbf{T}_{PAA}$ = PAA(**T**,*p*), where *p* = *getPaaFactor* = $n'/|\mathbf{A}|$, ensuring that this approximation will take exactly *S* seconds.

Although the minimum possible time complexity has now been achieved with these ideas, we will run into issues with memory usage for a long time series. A Mplot needs all its cell values in memory, unlike say the state-of-the-art Matrix Profile algorithms [54][56], which only require keeping the minimum of each column of Mplot. Thus, memory becomes the next bottleneck. Our proposed solution to this issue is described in Section 2.3.2.

2.3.2 Removing the Memory Bottleneck

The ideas in the previous section greatly reduce the *time* needed to compute large Mplots, however as we consider ever larger Mplots we bump into a new hurdle, *main memory*.

The reader may wonder why we should use the time and memory resources required to compute large matrices, when none of the available screens are able to display them at native resolution. Note that the highest resolution in a commercially available system is currently 8K (7680×4320 pixels). In fact, there is a reason to compute a Mplot at a resolution greater than can be (currently) displayed. We propose to create *multi-resolution*

approach, which allows a user to initially see an approximation of a massive Mplot, and interactively zoom-in on any areas that catch her eye as requiring a more detailed inspection. When the zoomed-in patch is requested, one of two things happens:

- If the zoomed-in patch was precomputed at the required resolution, we can simply fetch it from memory.
- If the zoomed-in patch was not precomputed at a fine enough level, it is recalculated, on-demand, at the finer resolution required.

Note that this style of user interaction echoes the widely known visual information seeking mantra given by Ben Shneiderman: *Overview first, zoom and filter, then details-on-demand* [42].

Assume that the entire area of an 8K screen is to be used to show a Mplot. Using the SPLAT algorithm, we could *exactly* compute an AB-join of two time series of length 7,680 and 4,320 in well under one second on a standard desktop (by way of contrast, if $m = 512$, existing brute-force algorithms take about 840 seconds). This is effectively real-time or interactive for our purposes. We set one second as being the limit for any refresh interaction with our system.

With this in mind, we propose a *multi-resolution* approach to allow Mplots to handle long time series called *MultiResSPLAT*. The basic intuition is as follows:

- *MultiResSPLAT* accepts a threshold for user patience for screen refreshes, i.e., one second.

- The *SplatTimePredict* function predicts how long it would take the Mplot matrix to be computed.
- If the predicted time exceeds the user's patience, the tool downsamples the time series by a factor of p such that the computation time is less than that threshold. The factor p is computed by *getPaaFactor*. This matrix, computed on downsampled data, is shown as the Mplot.
- The user may be satisfied with the approximate Mplot. However, if she wishes to zoom-in to inspect any region in more detail, we recursively repeat this process for that local patch of the matrix.
- Likewise, if the user is currently viewing a zoomed-in region of a Mplot, and she wishes to pan her view, we will not have the new patch computed at the current resolution, so we again compute it on-demand, at the highest resolution allowed by its size and the threshold for user patience.

Table 2. The MultiResSPLAT Algorithm.

Function: MultiResSPLAT(T_A, T_B, m)
Input: Reference time series T_A, Query time series T_B, Subsequence length m,
Output: Distance matrix Mplot

```
1  user_patience = t
2  estimated_time ← SplatTimePredict(T_A, T_B, m)
3  if estimated_time > user_patience
4     p ← getPaaFactor(T_A)
5     T'_A ← paa(T_A, p)
6     T'_B ← paa(T_B, p)
7     m' ← floor(m/p)
8     Mplot ← SPLAT(T'_A, T'_B, m')        // see Table 1
9  else
10    Mplot ← SPLAT(T_A, T_B, m)           // see Table 1
11 return Mplot
```

In Table 2 we formalize these ideas, beginning with the main MultiResSPLAT algorithm. In line 1 the user patience threshold is set to *t* seconds. With line 2, we estimate the SPLAT time on the input time series. By comparing the estimated time and *t* in line 3, the algorithm decides whether a downsampling is required or not. If downsampling is needed, the PAA factor, *p*, will be calculated as shown in line 4. Then the new downsampled time series (T'_A, T'_B) and the reduced subsequence length (m') are set within lines 5 to 7. Finally, the SPLAT algorithm is applied on the downsampled time series of interest as shown in lines 8 and 10.

In Table 3 we show how we can use the MultiResSPLAT algorithm recursively, to allow zooming-in on a region of an approximately computed Mplot, to show that region in a larger size that is more finely approximated. For clarity, Table 3 outlines the algorithm for one single zoom-in. However, it can be trivially extended to allow iterative zooming-in, where a user "drills down" to an event that catches her eye.

Table 3. The MultiResSPLATZoom Algorithm.

Function: MultiResSPLATZoom(T_A, T_B, *m, plt*) **Input:** Reference time series T_A, Query time series T_B, Subsequence length *m*, Existing Mplot *plt* **Output:** Distance matrix Mplot	
1	[*lx*,*rx*, *dy*, *uy*] ← getUserRequestedPatch(*plt*)
2	seg_A ← findExactLocationOnTimeseries(T_A, *lx*, *rx*)
3	seg_B ← findExactLocationOnTimeseries(T_B, *dy*, *uy*)
4	**return** MultiResSPLAT($T_A(\text{seg}_\text{A})$, $T_B(\text{seg}_B)$, *m*) //see Table 2

The algorithm starts by obtaining the user-requested patch from an existing Mplot (*plt*) in line 1. This request normally comes from a classic rectangular selection tool. As the user selects a rectangle region on *plt*, the four corners of the selected area are returned as *lx*, *rx*, *uy*, *dy*, which are left/right *x* and up/down *y* values, respectively. Then in lines 2 and 3, the coordinates are mapped to the exact locations in both input time series T_A (seg_A) and T_B

(seg_B). Finally, the MultiResSPLAT is called on the new subsets of the input time series and the new zoomed-in Mplot is returned with line 4.

We omit the details of the panning function, which is similar. Note that the experience of using these tools is completely transparent to the user. She can pan and zoom at will and have essentially the same experience as if the system had precomputed and stored a massive matrix. Using MultiResSPLAT, memory usage can be improved by orders of magnitude. Assume we need to run SPLAT on a time series of length 1,000,000 and return a matrix with a trillion cells. In MultiResSPLAT the computed matrix size is always below a threshold, say 7,680 and 4,320, which reduces the memory footprint by a factor of ~31,000.

2.3.3 Removing the Human Visual Attention Bottleneck

In the previous two sections we mitigated both memory and time limitations to create large Mplots. However, this reveals two new related bottlenecks, human visual attention and screen resolution. It is reasonable to ask why we should bother to compute a matrix of size, say 50,000 by 50,000 if we are going to display it on a mere, say 2,000 by 2,000 pixel patch of the screen. The results in the last section partly answer this question, a downscaled approximation of a large Mplot is often good enough to allow a user to spot a tentative, but possibly "blurred" pattern, which she can then explore by zooming-in. However, for truly massive Mplots, the downscaled approximation may obscure patterns. There is an obvious solution, to compute the Mplot *patchwise*, and then show the user the full-scale piecewise patches consecutively. However, that simply shifts the bottleneck to human visual attention, which is an even more precious resource.

Note that if the user is interested in visually searching for features or patterns that can be *objectively* ranked, we can use our piecewise strategy to search for such features, and only save the top-*k* patches for later "offline" visual inspection. Assume for the moment that such a target feature, $T_{feature}$, exists. In Table 4 we show how we can use the *PiecewiseSPLAT* algorithm to find the patch that contains the top-1 $T_{feature}$.

Table 4. The PiecewiseSPLAT Algorithm.

Function: PieceWiseSPLAT(T_A, T_B, *m, p, ov*)
Input: Reference time series T_A, Query time series T_B, Feature we wish to find $T_{feature}$, Subsequence length *m*, Patch size *p*, Overlap size *ov*,
Output: The patch with top-1 $T_{feature}$ best_patch

```
1 best_patch = Nan
2 for row = 1 : T_B_Length – p ; row + p - ov
3    for col = 1 : T_A_Length – p ; col + p - ov
4      T'_A, T'_B = T_A(col:col+p), T_B(row:row+p)
5      curr_patch ← SPLAT(T'_A, T'_B, m)          //see Table 1
6      best_patch ← T_feature(best_patch, curr_patch)
7 return best_patch
```

The top-1 patch is initially set to Null in line 1. Given a patch size of *p*, the reference and query time series are examined piece by piece within lines 2 to 5. Each patch is then compared to the best patch so far in line 6, w.r.t. $T_{feature}$. The best patch is updated only when the examined score is greater than our best-so-far. Line 7 returns the best patch of the Mplot with regard to the user desired $T_{feature}$.

Note that there is some computational overhead, in that the patches must slightly overlap. This is because some features that we may wish to search for may span a region of pixels, and we do not want to miss a feature that is close to the edge of a patch. Note however that this overlap must be of the order *m*, which is typically in the range of 8 to 258. Whereas the patch size might be in the range of 20,000 × 20,000 (the best size depends on the main memory available) so the computational overhead of the overlap is inconsequential.

Thus far we have glossed over the nature of $T_{feature}$. Here we can leverage decades of research. There are dozens of algorithms for extracting features from Mplots, some generic, and some domain specific. Some examples include:

- **Bioacoustics**: Malige et al. use Mplots [26], to analyze humpback whale communications, and explicitly define specialized features on the matrix such as *song* and *theme*.
- **Astronomy**: Phillipson uses Mplot to investigate stochastic light curves of Active Galactic Nuclei [35], and define a feature called *optical quasi-periodic oscillation* that can be computed from the plots.

In addition to these domain specific features, there are hundreds of generic features that a user may wish to search for, including Recurrence rate (RR), Determinism (DET), Laminarity (LAM), Ratio (RATIO), Trapping time (TT), Divergence (DIV), Entropy (ENTR), etc.[28]. Note that not all proposed features can be computed piecewise using the algorithm in Table 4 (some features require random access to all parts of the matrix), but the vast majority can.

For concreteness, in Section 3.3 we will show how this strategy can be used to solve two problems in which we can define simple and intuitive features that allow us to find targeted events in a time series that would be difficult to discover using any other method.

2.3.4 Parameter-Free Mplots: 3D Mplots, Mplot Movies and Multifocal Mplots

Given that we can now compute Mplots orders of magnitude faster, it is natural to ask if there are ways to exploit this alacrity to somehow improve Mplots or provide new services.

Here we briefly discuss three such examples, although we suspect that the community may discover many more.

A recent paper motivates the issue we address, noting that "(Mplots) *cannot handle the variability of discriminative region scales and lengths of sequences*" [52]. The issue at hand is unique to Mplots and does not happen for true dot-plots. Suppose we build a dot-plot for long string of natural language with $m = 3$. The plot will reveal a repeated "word" of length three, such as ..b**ing**e watch**ing**. If there is repeated structure longer than three, such as …notwithst**anding** her dem**anding**nesses…, this will also be revealed in the dot-plot, as a "streak" with a length of four, because each of the consecutive substrings in the motif, "**and**", "**ndi**" "**din**" and "**ing**" have a match in the same order.

Surprisingly for the corresponding situation with real valued time series, we cannot make the same claim. It is possible that two subsequences match well, but their sub-subsequences do not. This is because we are working with z-normalized time series. For example, consider the two time series A = [1 0 1 9] and B = [0 1 0 9]. Their z-normalized Euclidean Distance is very small, just 0.382. However, consider their subsequences A' = [1 0 1] and B' = [0 1 0], in spite of being shorter, their z-normalized Euclidean Distance is 2.829, an order of magnitude larger.

The practical upshot of this is that an Mplot created with a user defined parameter m, we cannot guarantee that this will reveal similarities of subsequences with lengths greatly different to m. Before continuing, we should note (as most of the examples in this paper show) that in general Mplots *are* very forgiving to the choice of m, for almost all datasets and applications. For example, almost all *atomic* human gestures, dance moves, ASL

words, sport performances (i.e., a tennis serve), happen over a time range of about 1/5th to 2 seconds. Using a *m* value at the shorter end of that range will tend to reveal all such conserved behaviors. However, there are some domains that can have conserved behaviors over an even wider range. An example familiar to the current authors is the behavior of sap feeding insects [5], [7], [17], [48], which have conserved behaviors that vary in performance length of at least two orders of magnitude.

We proposed to address this issue in one of three ways:

- We can produce Mplot *movies*, by creating a Mplot for all possible values of *m* and writing each consecutive Mplot to a frame of a video. These videos are reminiscent of a video showing a microscope focusing, the image is initially "blurred", but later comes into focus. Critically, different parts of the Mplot video can come into focus at different times, suggesting a time series that has multiscale structures.
- We can create 3D Mplots, by stacking the (sparsified) frames in the Y-axis. These 3D scatterplots can be rotated and viewed from various angles.

Static examples of these two ideas are shown in Fig.3.

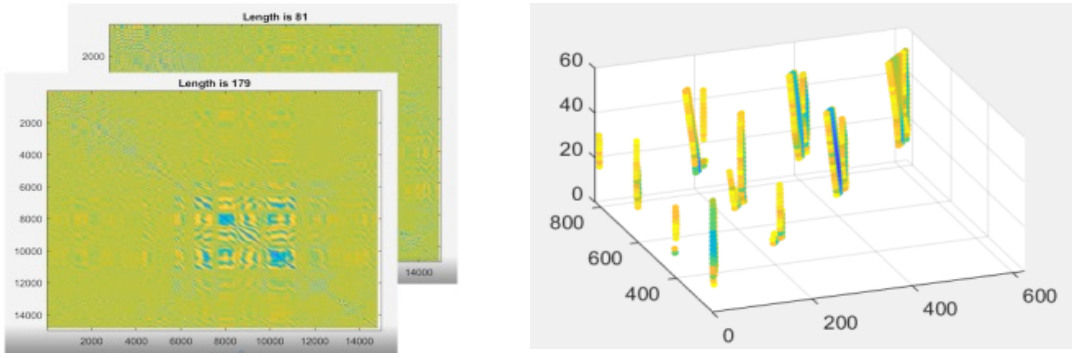

Fig.3 *left*) Screen grabs from a Mplot video. *right*) A 3D Mplot shows how motifs change as a function of the subsequence length.

While these two Mplot variants are compelling and useful, they do not lend themselves to evaluation in a paper. We will therefore not further evaluate or discuss them. However, we invite the reader to visit [41] to see a gallery of them used in various domains.

The final variant of Mplot that we introduce are multifocal Mplots, which do lend themselves to the static format of a paper. Our idea is inspired by focus stacking, a technique that allows photographers to create a single image where objects on various focal planes are all in focus. The technique involves photographing the same composition multiple times with various focal points. These images are then composited to create a single image in which everything in the photo is in focus. This is a perfectly analog to the task at hand, the notion of "focus" here means an appropriate choice of m. Since there is no single choice of m for all parts of the time series, we can simply compute all m and composite the final result, into a single image. Fig.4 shows an example of a multifocal Mplot on some insect electrical penetration graph (EPG) telemetry.

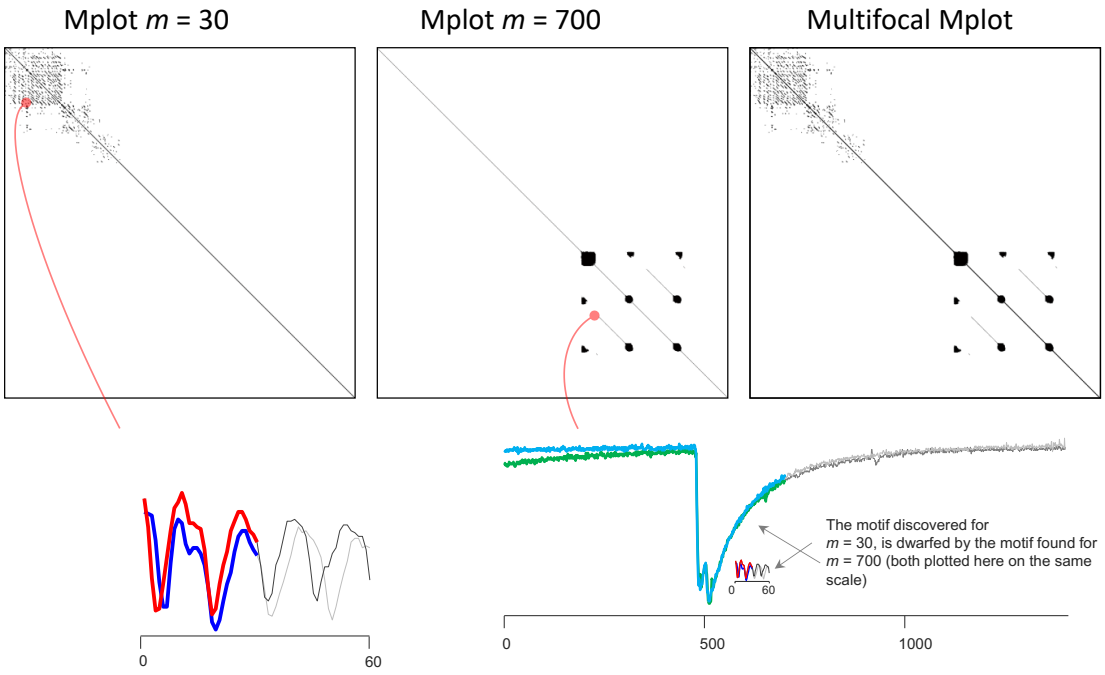

Fig.4 *left*) A Mplot with m = 30 discovers conserved periodic behavior corresponding to xylem ingestion [17] but fails to discover conserved behavior at longer time frames. *center*) A Mplot with m = 700 discovers conserved periodic behavior corresponding to intercellular passage but cannot represent the shorter xylem ingestion behavior. *right*) A multifocal Mplot can simultaneously represent conserved behavior at both scales.

Because our problems are such a perfect analogue for *focus stacking*, we do not need to create any new software to create a multifocal Mplots, we can simply use off-the-shelf image processing software and input a Mplot movie, including Photoshop's built-in focus stacking tool.

Note that all three of these techniques remove the need for a user to set the Mplot's single parameter, the subsequence length m, thus make Mplots essentially parameter-free.

2.3.5 Pooling SPLAT

If we create a Mplot that is larger than the screen resolution available, the operating system will rescale the image for display. The algorithms used for this, Nearest Neighbor, Bilinear, Lanczos, etc. are optimized for natural images but may be poor choices for Mplots. In

particular they may obscure fine details. For example, consider Fig.5.*left* which shows a Mplot which is being downscaled with nine pixels mapping to one. Most rescaling algorithms reduce to *averaging* in such cases, and the black pixel indicating a motif is obscured.

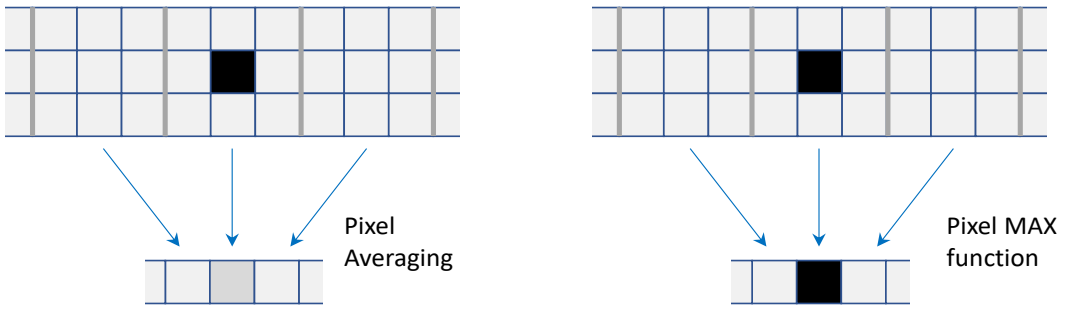

Fig.5 *left*) A naïve averaging of pixels can "blur" out features when downscaling. *right*) In contrast, while a MAX aggregation may create some small amount of spatial uncertainty, it preserves the strength ("color") of the discovered motif.

To mitigate this issue, we propose to take explicit control of how Mplot images are resized. Instead of simply averaging the pixels, we allow arbitrary aggregation functions. For example, to help highlight motifs we can use a MAX function as shown in Fig.5.*right*, and to preserve discords (anomalies/differences) we use a MIN function. Slightly more exotic functions can be defined to attempt to preserve both discords and motifs at the same time. In Table 5 the general algorithm is outlined.

Table 5. The PoolingSPLAT Algorithm.

Function: PoolingSPLAT(T_A, T_B, *m, w, h*)
Input: Reference time series T_A, Query time series T_B, Subsequence length *m*, Mplot output size *s*, Aggregate function *f*
Output: pooled_Mplot

```
1  pooled_Mplot = nan(s,s)
2  n_rowslice = number_of_T_B_subsequence /s
3  n_colslice = number_of_T_A_subsequence/s
4  for col = 1 to Mplot_column_count
5     for row = 1 to Mplot_row_count
6        d ← compute_Mplot_value(row,col)              //see Table 1
7        row', col' ← row/ n_rowslice, col/ n_colslice
8        pooled_Mplot(row' , col') ← f(d,pooled_Mplot(row', col'))
9  return pooled_Mplot
```

In line 1, a fixed size output Mplot is defined independent of the input time series length. This fixed size depends on the desired resolution of the output plot. For example, on an 8K monitor, a user may request an output of 4320×4320. In lines 2 and 3 we compute how many cells from the original Mplot will be assigned to each cell in the pooled Mplot. With lines 4 to 6 distance computation is done as in Table 1. As indicated in line 7, the location for mapping the current value in the pooled Mplot is found. We use standard image resizing algorithms to avoid aliasing artifacts. Then line 8 compares the current distance value, with the existing value in the pooled Mplot and updates that location with respect to the desired aggregate function's output. Finally, line 9 returns the fixed size pooled Mplot.

In this chapter, we have established a comprehensive understanding of how SPLAT operates and the improvements it offers. This foundational knowledge sets the stage for exploring practical applications and evaluating performance. In the next chapter, we will delve into practical examples and present experimental results to demonstrate the efficacy and efficiency of Mplot.

3 EXPLORING MPLOTS: PRACTICAL EXAMPLES AND EXPERIMENTAL RESULTS

In this chapter, we move from theory to practical application. This section focuses on analyzing real-world data through Mplots, exploring their ability to reveal complex patterns and relationships. Through case studies and experiments, we aim to demonstrate the utility of Mplots in advancing knowledge and understanding across diverse fields.

3.1 Interpreting Mplots

There are many useful guides to interpreting recurrence plots/dot plots available [28]. We will not duplicate those efforts here. However, as we noted in Section 2.2, there are several differences between *true* recurrence plots and Mplots, and some of those differences effect the interpretation of plots. In Fig.6 we show some examples of patterns that are unique to Mplots. When discussing the time series that created these patterns, we use the familiar expository trick of using *text* as a proxy for time series, and *hamming distance* as a proxy for Euclidean distance.

In a dot plot with $m = 4$, a recurring pattern of say CATA would produce a single point on the plot. In a dot plot with $m = 3$, the recurring pattern of CATA would produce a *two* consecutive points "smeared" in diagonal line, and so on.

In principle Mplots are similar, and a motif that was exactly m datapoints long could produce a single dot (U2). However, even if the natural motifs in the time series are exactly m datapoints long, the use of parameter m would tend not to produce a single point, but a smeared line. The reason is that if two subsequences beginning at locations i and k, are a close match, then we will still have a reasonably close match for i and $k\pm1$, i and $k\pm2$, etc. This is not true for the discrete strings of dot plots. Therefore, if we see a diagonal streak

on a Mplot built with parameter *m*, whose length in the x-axis is *d,* we should interpret this as the existence of a motif of length a little greater than *d*. Thus, the pink pattern seen beginning at Q2 suggests the existence of a motif of length five or six, not just four. This suggests a general strategy for setting the value of *m*. We should set it to be a little *less than* the length of the motifs we want or expect to find.

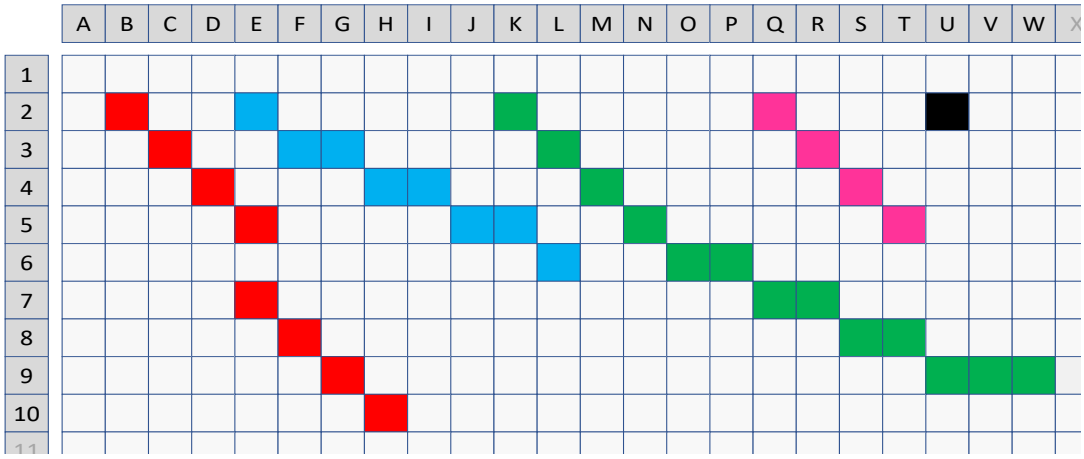

Fig.6 Some examples of patterns we may see on a Mplot. Here we assume *m* = 4 was used to create this plot

One of the patterns that are unique to Mplot is the green curved line shown in beginning at K2. This suggests that there is a motif, but the second occurrence begins to slow down. Intuitively this would be like CATA and CATTAAAAA. Naturally, the pattern can curve in the opposite direction if the second occurrence is *speeding up* instead. We call instances of such patterns "chirps". If we see a streak that curves in both directions in a serpentine fashion this is suggestive of a pair of subsequences that match after allowing one to locally "warp" in order to match the other [37]. This is an important benefit, as finding motifs with invariance to warping (i.e. Dynamic Time Warping [37]) which is known to be very computationally demanding [2].

The blue streak beginning at E2 shows a straight line streak that is not parallel to the diagonal, indicating a motif where one occurrence is a linearly rescaled version of the other, something like TAG and TTAAGG. As we will later show, we can use the observed angle of this streak to predict the amount of rescaling and then exploit this fact.

Finally, the red streak beginning at B2 suggests a motif of about length eight in which the second occurrence has some spurious sub-patterns inserted at about the midway point, something like TAG**X**CAT and TAGCAT (alternatively, we can see the first occurrence as *missing* some sub-patterns).

We have shown these examples on binarized toy examples, however more generally, using real-valued Mplots, the colors or shades of gray offer further information about the degree of pattern conservation. In our experimental section we show examples of such patterns discovered in real datasets.

3.2 Interpreting Mplots: REVERSE ENGINEERED

In the previous section we showed how to interpret some of the basic patterns and regularities that we regularly encounter in a Mplot. However, it is also possible to reverse engineer this process. We can *imagine* a hypothetical structure in a time series that might be of interest, and then further imagine how that structure would manifest itself locally on a Mplot. Moreover, we may be able to write a simple function to search for this local manifestation using the piecewise Mplot function in Table 4. To make this clear, we will consider a concrete example here.

Finding motifs is generally easy using Mplots (or the Matrix Profile [50][54]). However, it can be very difficult to find motifs under certain circumstances, in particular, it can be hard to find *rare* motifs, if:

- There is a much more common motif or motif(s).
- The rare motif is less well conserved than the common motif or motif(s).

Note that this case is common in real world data. For example, we may have a handful of examples of abnormal heartbeats in an ECG that contains thousands of better conserved normal beats.

Let us think about what a Mplot would look like in such cases. If we had a repeating *common* motif, we would expect to see many more or less solid lines, more or less parallel to the diagonal. This is a very common type of Mplot. However, some such Mplots also have "cross shaped" structures that have very low pixel density within the arms of the cross. In Fig.7 we show two synthetic examples, and Fig.1.*right* showed a natural example.

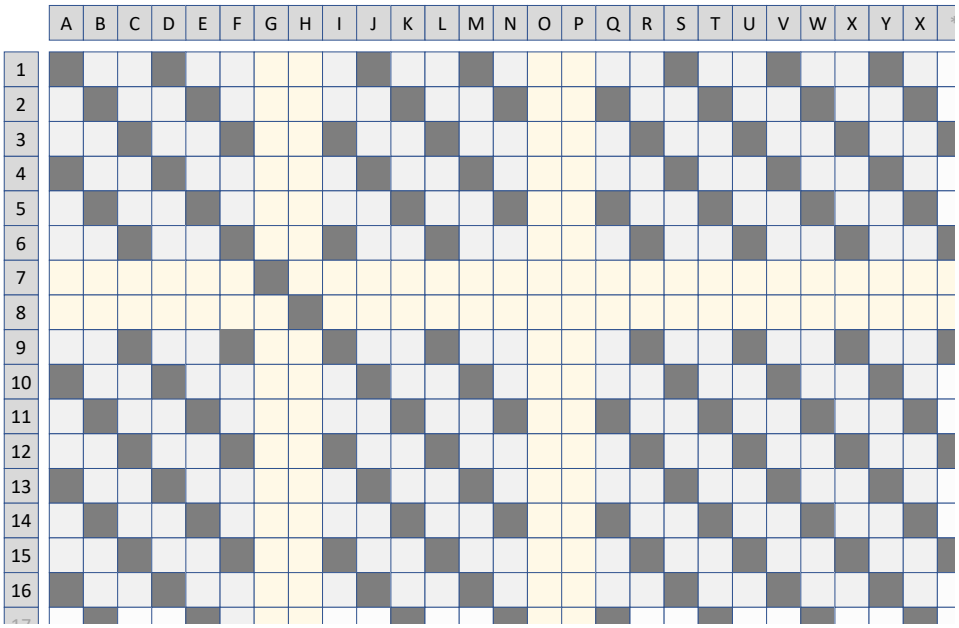

Fig.7 A hypothetical Mplot. Note that there are two "crosses" formed by the sparse rows {7,8} and the sparse columns {G,H} and {O,P}.

The reader will note that there are two slight variations of this pattern shown in Fig.7. In the intersection shown at {7,8},{O,P} the *center* of the cross is also sparse. These are what we should expect from if either or both of the subsequences corresponding to {7,8} or {O,P} are noisy or unique (i.e. *discords*). If *either* of the subsequences is unique, it will be far from everything (except itself), thus its entire row (or column) will be sparse, including when that row (or column) intersects with another sparse column (or row).

However, in contrast, consider the intersection shown at {7,8},{G,H}. Here, while the main arms of the cross are mostly empty, there is a diagonal line that runs through the intersection. This is exactly what we should expect, if the pair of subsequences at {7,8} and {G,H} are a rare motif. This is because a rare pattern will be different to the common patterns, which are by definition almost everywhere; Thus, giving us a mostly sparse row

(or column). However, in the infrequent places that the rare pattern encounters another example of the same rare pattern, it will produce a streak of black pixels.

Having given the intuition as to how a rare motif can manifest itself, we can write a simple function that can test for such patches in a massive Mplot. In Table 6 we outline such an algorithm. The intuition is to look for white rows and columns, which indicates the existence of subsequences with the minimum similarity to the majority of subsequences. We then aim to find a straight black line(s) within the intersection of those white rows and columns. This is the sign of a similarity that rarely happens in the input data.

Table 6. The Rare Motif Algorithm.

Function: findRareMotifs(*T*, *m, p, ov*)
Input: Reference time series *T*, Subsequence length *m*, Patch size *p*, Overlap size *ov*
Output: Top k patches including the rare motifs

```
1   best_patches = [];
2   candr = [];                    // Mplot rows with less similarity to others
3   candc = [];                    // Mplot columns with less similarity to others
4   for row = 1: T_Length – p; row + p - ov
5      for col = row : T _Length – p ; col + p - ov
6         T_A, T_B = T(col:col+p-1), T(row:row+p-1)
7         Mplot = SPLAT(T_A, T_B, m)          // see Table 1
8         candr.append(rows in Mplot | sum(rows) < mean(rows)) //rows with less black pixels
9         candc.append(cols in Mplot | sum(cols) < mean(cols))) //columns with less black pixels
10  for patch in intersection (candr, candc)
11     hlines = HoughTransform(patch)
12     if length(hlines) > 0
13        best_patches.append(patch)
14  return sorted(best_patches, black_pixel_count, 'descending')
```

In line 1, we define an empty list to store the possible best patches. Lines 2 and 3 introduces the list of candidate rows and columns where the locations with less similarity to other locations are stored in. Starting from line 4 the Mplot is computed patchwise. Lines 8 and

9 look for rows and columns in Mplot with the highest probability to include the rare motifs. Since rare motifs do not match to most subsequences, we expect to see a row (or column) of low values in that location. In a binarized matrix that can be seen as a white row (or column). In line 10 and 11 we go over the intersection of candidate rows and columns and look for a high value, indicating a high similarity to another subsequence(s). This is visualized as a straight line in a Mplot. This line can be angled by some value or can be divided into parts, especially if the rare motif is less well conserved than the common motifs (which as we will later see, is empirically often the case). We use the Hough Transform tool to find these lines [10]. If such a line exists, line 13 stores it as one of the best patches. Finally in line 14 we sort the best patches such that a white cross of Mplot with a black line (more black pixels) is prioritized over a white cross with a few random black pixels.

In Section 3.3.5 we will show a real word example of using this idea to search for rare motifs in a vast collection of insect data. We believe that this basic idea could be used to find other structures, including variations of Time Series Chains [21],Time Series Shapelets [50], Time Series Novelets [30], etc. More exciting is the possibility of that this framework will be used to discover structures that did not occur to the current authors.

3.3 Experimental Evaluation

To ensure that our experiments are reproducible, we have built a website [41] that contains all the data/code used in this work. All experiments were conducted on an Intel® Core i7-9700CPU at 2.80GHz with 16 GB of main memory, unless otherwise stated. As noted above, the format of this publication does not lend itself well to Mplots. We encourage the

reader to visit [41] where we have large format images and videos that exploit and demonstrate our ideas.

To help the reader gain some intuition for the utility and generality of Mplots we begin with some anecdotal examples before considering more qualitative experiments.

3.3.1 Hunting for Exoplanets

Exoplanets can be discovered by examining the time series of flux (light intensity) of a star. When a planet passes between the star and the observatory on Earth (or orbiting Earth), its shadow causes a slight dimming of the flux. In some cases, as in Fig.8.*top.right*, the effect can be quite dramatic. This is true if the planet is very large (Jupiter-sized), with a short orbital period, and the data is relatively noise-free. These ideal cases are visually apparent and/or can be easily discovered with Fourier techniques. However, if the planet is small (Mercury-sized), with a longer orbital period, and the data is noisy, this is a much more difficult problem.

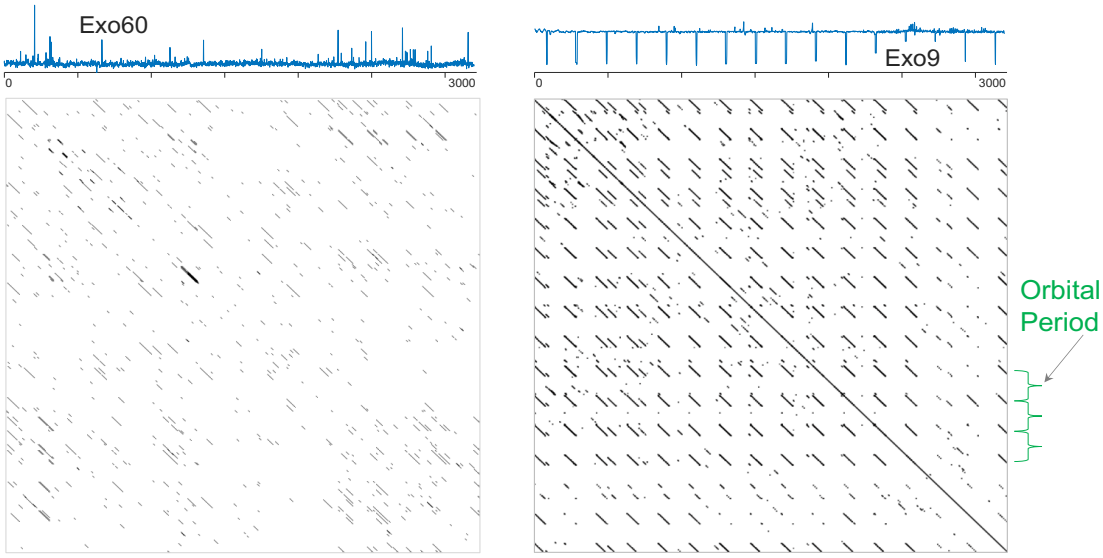

Fig.8 *top.left*) A star-light curve from a star believed not to have an exoplanet. *top.right*) A star light curve from a star known to have an exoplanet. *bottom.left*) The Mplot of the planetless star is relatively featureless. *bottom.right*) The Mplot of Exo9 reveals not only the existence of an exoplanet but tells us its orbital period.

As Fig.8 hints at, we believe that Mplot may be a useful tool to examine these difficult cases, as the evenly spaced diagonal lines not only offer evidence for an exoplanet, but their spacing tells us the period. Note that it is possible that some lines could be missing due to noise (cloud cover, sensor noise, etc.). Consider Fig.9, does it show an Exoplanet?

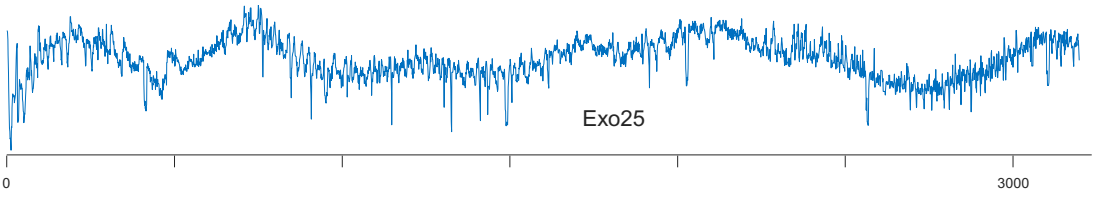

Fig.9 The star light curve for Exo25. Does it suggest the existence of an Exoplanet?

In an attempt to answer this question, we built a Mplot in Fig.10, using the same parameters as in Fig.8.

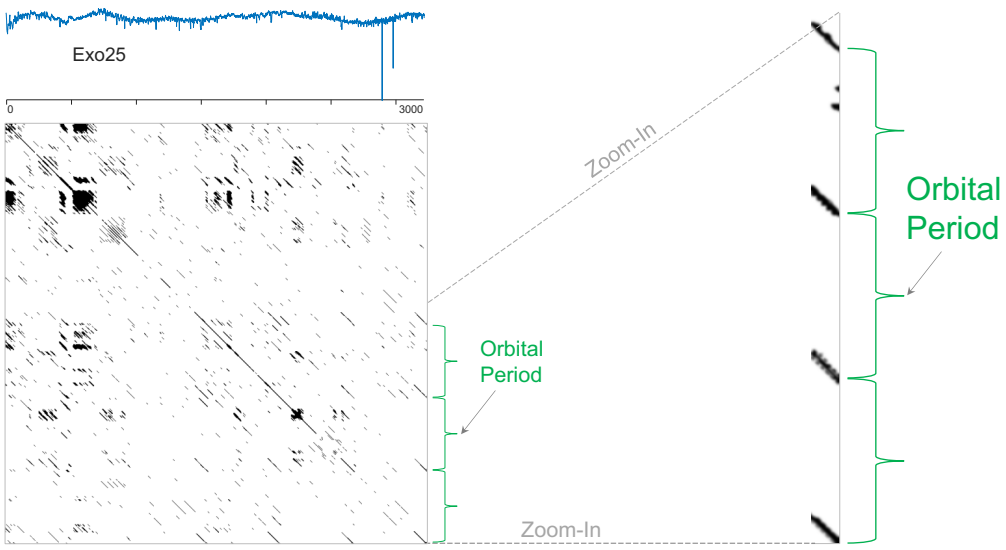

Fig.10 The star-light curve for Exo25 with its Mplot (*m*=100). While there is noise reflecting the original data's noise, there is the unmistakable signature of an exoplanet with an orbital period about three times longer than Exo9 (Cf. Fig.8.*bottom.right*).

A visual inspection offers strong evidence for the existent of an exoplanet. As the call-out in Fig.8.*right* shows, we can clearly see four periods. The much weaker, barely visible fifth period is presumably explained by the noise in the original figure. In [41] we have a gallery of additional exoplanets discovered with this technique.

To be clear, we are not advocating Mplot as a tool for hunting exoplanets. This is an important problem, and it is worth creating bespoke tools that consider the many physical constraints in this domain. This example merely serves to show that Mplots can reveal structure that is not readily apparent in raw time series.

3.3.2 Mplot Filtering

Our ability to create massive Mplots presents both opportunities and problems. One problem is that Mplots can be very "busy", and as we noted earlier, human visual attention is a precious resource. One solution to this issue is to apply filters of various kinds to emphasize patterns that we may be interested in. This can be done in many ways, most of

which are trivial to implement. For example, a traffic manager might choose to highlight motifs that happen within five days of a holiday, or on rainy days (using out-of-band data), etc.

In this section we show a novel filtering strategy that corresponds to a high-level and subtle semantic question; *"Show me patterns common between two sequences, but absent from one or more other sequences.*"

First, a quick review. Recall that Mplots are conceptual precursors to Matrix Profiles [50][54]. In particular, a self-join Matrix Profile can be created by collapsing an $n \times n$ similarity matrix using the smallest value of each column (excluding values on the diagonal). There is a similar correspondence for the AB-join Matrix Profile which is either the row or column collapsed-min of the Mplot between two different time series.

The Contrast Profile [29] is a recent tool for discovering *contrasting patterns* across time series, that is, behaviors that are repeated within one time series but are absent from another. Since the Contrast Profile is defined "lego-like", by combining several Matrix Profiles, this suggests that its definition could be retroactively generalized to Mplots.

The Contrast Profile is defined as the difference between AB-join and self-join Matrix Profiles:

$$\mathbf{CP} = \mathbf{MP_{AB}} - \mathbf{MP_{AA}}$$

We adapt this to create the semantic definition we desire:

$$\mathbf{ContrastMplot} = \mathbf{MP_{habituated}} - \mathbf{MP_{targeted}}$$

The Mplots cannot be directly subtracted due to dimensionality incompatibilities, however this equation serves as a reference when reasoning about how to complete the desired operation. The motivating question: "*Which behaviors are common between two sequences but absent from one or more other sequences?*" hints at a methodology. When thinking about this on a pair-wise basis, we would like to focus on self-join subsequence pairs with high similarity but suppress those which are similar in the "habituating" sequence.

We can achieve this with one Mplot and two AB-join Matrix Profiles. Given two target time series T_A and T_B, and one or more habituating time series T_C we generate a Mplot_{AB} between T_A and T_B, then compute two Matrix Profiles MP_{AC} and MP_{BC}. We habituate through the following indexed definition:

$$\textbf{ContrastMplot}^{(\text{i,j})} = \min(MP_{AC}^{i}, MP_{BC}^{j}) - Mplot_{AB}^{(i,j)}$$

It may be unintuitive to consider why we are combining elements from two different structures. In a Mplot, we are interested in the pair-wise structure across the entire matrix, however when habituating, we are only interested in whether a low distance nearest neighbor exists. Thus, we can collapse the habituating similarity matrix into a Matrix Profile.

We will perform a demonstration using a time series representation of mitochondrial DNA. The conversion from DNA to time series is done with this classic transformation.

$T_1 = 0$, for i = 1 to length(DNAstring)

if $DNAstring_i$ = **A**, then $T_i+1 = T_i + 1$

if $DNAstring_i$ = **C**, then $T_i+1 = T_i - 1$

if $DNAstring_i$ = **G**, then $T_i+1 = T_i - 2$

if $DNAstring_i$ = **T**, then $T_i+1 = T_i + 2$

The two closest species to humans are Chimpanzees (*Pan troglodytes*) and Bonobos (*Pan paniscus*). Chimps and Bonobos are more similar to each other than to humans (Green et al. 2008), so we will investigate whether there exist DNA subsequences shared between them, but which is absent from humans.

We structure the problem by setting Bonobos to T_A, Chimpanzees to T_B, and humans to T_C. One type of DNA mutation is subsequence reversal. The Contrast-Mplot can reveal this by simply concatenating the reversed Bonobo sequence to itself before processing.

In the ContrastMplot shown in Fig.11, the black streaks represent sequences which are conserved between Bonobos and Chimps, and also dissimilar to humans. White represents subsequences pairs between Bonobos and Chimps where either subsequence is conserved at least as well in humans. The dominant visual feature is the patchy diagonal which lies along the reference 1:1 diagonal (blue). This is expected since most of the DNA sequences between the two species are conserved in order. What is more interesting are the *off-diagonal* visual features. Features occurring above the reference diagonal in the reversed region (purple) indicate subsequences which occur earlier in the Bonobos relative to Chimpanzees. One such feature is highlighted in red. Additionally, this feature occurs in

the *reversed* Bonobo region, suggesting that the original DNA was transposed relative to the Chimp's sequence.

Using the BLAST [38] we have identified that the subsequence in question occurs within the COX2 gene, which is known to be closely conserved between Bonobos and Chimps, but divergent in humans [20]. While our demonstration focused on DNA, we anticipate that Contrast-Mplots will have broader applicability to domains where we want to visually reason about shared and unshared patterns in sets of data.

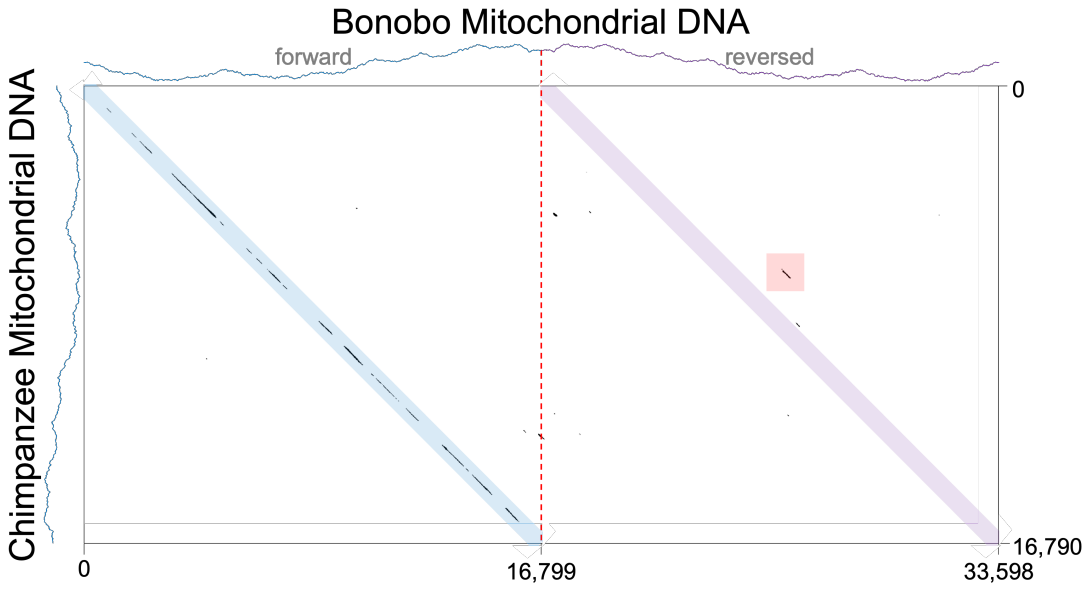

Fig.11 A Contrast-Mplot revealing mitochondrial DNA subsequences are shared between Bonobos and Chimps, but absent from Humans. The region highlighted in red indicates a reversed and offset Bonobo subsequence relative to the Chimp sequence.

3.3.3 Finding Rescaled Motifs using PiecewiseSPLAT

As we noted in Section 2.3.3 we can use PiecewiseSPLAT to find arbitrary features/structures/regularities in massive Mplots that could not fit in main memory. However, for concreteness here we will consider a structure with a direct and immediate visual interpretation, *scaled motifs*; subsequences of different lengths that would have a

small Euclidean distance *if* they were scaled to the same length. If the difference in scale is *very* small, say <8%, then the simple Matrix Profile will probably work s. If the difference in scale is relatively small, say <8 to 20%, then there are a handful of techniques to address such cases [49]. However, here we are interested in motifs that may dramatically differ in scale, say up to 300%.

To discover such rescaled motifs, we can search Mplots with PiecewiseSPLAT. Fig.12 illustrates the main insight.

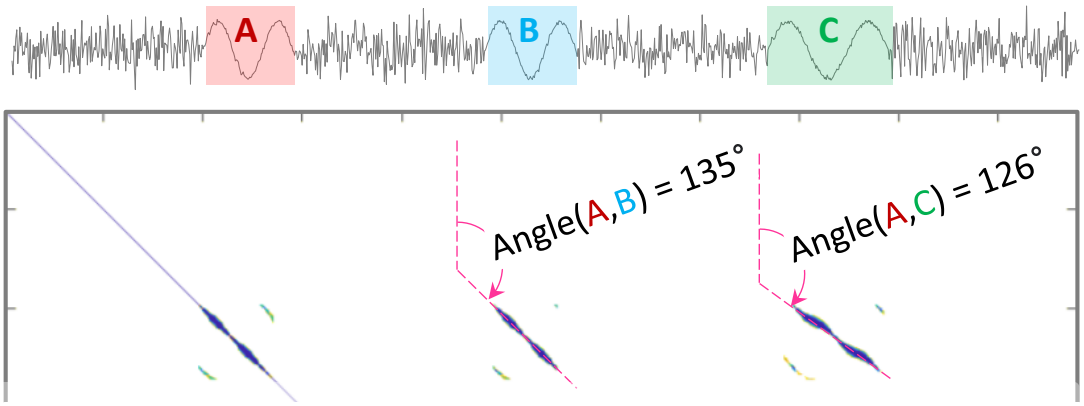

Fig.12 *top*) A toy time series with three sine-wave patterns embedded. Note that instance C is about 37% longer than the other two instances A and B. *bottom*) The corresponding Mplot shows that the difference in lengths manifests as a difference in angle.

Suppose we have two occurrences of a motif, **A** and **B**, of length L, and we create a Mplot with m set to a number less than L. We would expect to see a "streak" of length about L-m $\times \sqrt{2}$, parallel to the diagonal (or 135° to vertical).

However, if we have two motifs that differ in length, as with **A** and **C**, we should expect a similar streak, but at non-zero angle relative to the diagonal. The relationship between the scaling factor and the angle is given by:

$$\text{ScalingFactor}(A,C) = \frac{1}{\tan(\text{Angle}(A,C) - 90°)}$$

Thus, we can reduce the rescaled motif discovery problem to the task of finding lines in an image, and that problem is easily solved by the classic Hough [10]. There is a minor caveat; while the start point and angle of the discovered line reveal the location and scaling factor respectively, they may be a little "blurry", so we need to run a localized brute-force search on the identified area to refine the best motif.

To hint at the utility of this idea, consider Fig.13.

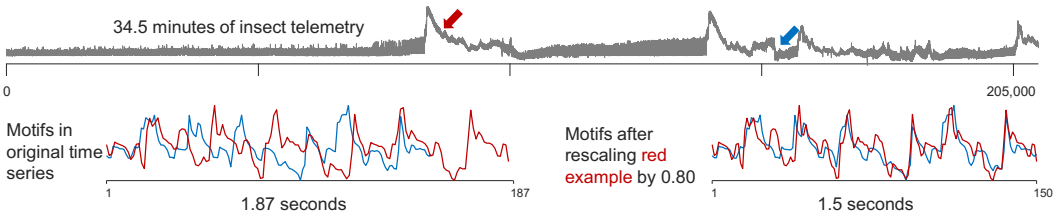

Fig.13 *top*) Telemetry from an insect pest feeding on a plant. *bottom*) A multi-scale motif discovered in the data can only be seen as conserved after one instance is rescaled by a factor of 1.25.

Here we see a motif discovered in telemetry from an insect. Because the two instances of this motif differ in length by a factor of 1.25, classic methods cannot find them (Yeh et al. 2016).

3.3.4 Hunting for Chiroptera with PiecewiseSPLAT

In the previous section we showed that PiecewiseSPLAT could allow us to find motifs with *invariance* to scaling. However sometimes we may explicitly desire to discover *only* those motifs that exhibit scaling.

For example, suppose a biodiversity survey needs to examine audio recorded at night to look for examples of bats. Existing bat classifiers have only been tested on a handful of the 1,400 known species [45]. We would like to have a *general* method to capture any species

of bat. The problem is compound by the fact that many birds and insects also sing at night, not to mention inevitable human noise pollution.

A well-known fact about bats may be useful. Bats use echolocation to find prey, producing bursts of sound and analyzing the returning echoes build a picture of the external world. Critically, the rate at which the bat emits sounds is not constant but changes, as [38] notes "*Over the course of an attack, bats increase call production rate*". It is important to note that this change in call production rate is not an accidental side-effect of the bat's call, but an intrinsic part of the bat's hunting strategy, trading off the energetic cost of producing sounds with the finer spatial resolution of rapid bursts [38].

This suggests an exploitable idea, we might expect that these changes in call rate would produce Mplot structures *not* parallel to the diagonal, as discussed in Section 3.1. Consider Fig.14.*right*.

Fig.14 Five randomly chosen six-second snippets of animals that both fly and produce sound at night. The four leftmost examples are all birds. The rightmost example is a bat, which is unique here in having "stripes" that are not perfectly parallel to the diagonal.

These Mplot snippets are diverse but note that the bird examples all have structure that is parallel to the diagonal. In contrast, the bat call is unique in that it has lines that are at an angle to the diagonal, telling us that the bat produced the motif twice, at two different speeds.

Birds are only using sound to communicate[2], bats are using sound for a completely different purpose, and occasionally producing this unique feature.

To test our hypothesis, we embedded a twenty-second snippet of bat hunting audio into a one-hour audio file containing diverse bird songs. We searched for lines that had an angle of at least ±9.5° to the diagonal, indicating a rescaling factor of 1.40. As shown in Fig.15

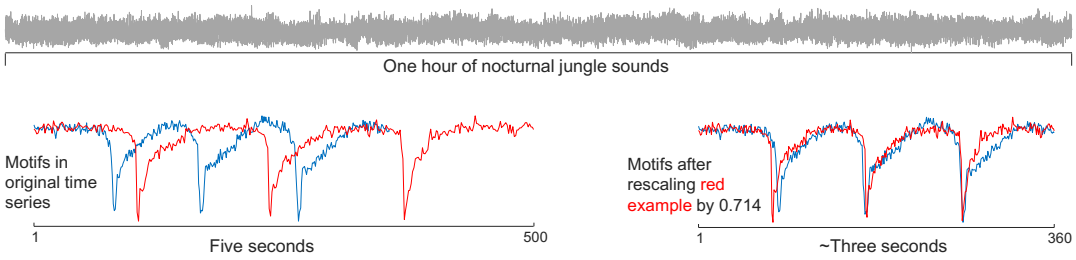

Fig.15 *top*) A one-hour dataset containing bird sounds, and a total of 20 seconds of bat sound. *bottom*) If we use PiecewiseSPLAT to search for motifs that have at least 1.35 rescaling, the top-1 motif is a bat vocalization.

The top-1 motif was indeed a bat vocalization. This experiment took 81 minutes, which is just slightly slower than real-time. Note that for the classic Matrix Profile, the top-10 motifs are all bird (occasionally possibly *insect*) sounds. This example hints at the utility of Mplots, with only the vaguest of domain knowledge we can search large complex datasets for behaviors of interest that can be described in high-level abstract terms.

3.3.5 Searching Massive Mplots

Recall that in Section 6.2 we discussed the possibility of "reverse engineering" the interpretation of Mplots. We noted that it may be possible to think of some structure we would like to find, hypothesize what the structure would look like on a Mplot, then build a

[2] A few birds such as oilbirds/swiftlets *do* use a weak form of echolocation.

simple image processing filter to search for this structure. Here we show a complete worked example of this idea.

Sap feeding insects in the order Hemiptera feed by removing plant sap from transport vessels, such as phloem and xylem elements [5][48]. This behavior is typically not destructive by itself but can spread pathogens from plant to plant. One of the most studied insects is the Asian citrus psyllid (*Diaphorina citri*), which is responsible for billions of dollars in crop losses each year. The primary tool used to study these insects is the electrical penetration graph (EPG), which as shown in Fig.16, produces a complex and noisy time series that reflects the behavior of the insect's straw-like mouthparts as they navigate within the plant tissues.

As shown in Fig.16.*A* One of the most common behaviors seen is *xylem ingestion*. Psyllids spend approximately 22% of their lives engaged in this behavior, with bouts of *xylem ingestion* lasting an average of about forty minutes [17]. It is known that it is rare to observe a *perfect* run of *xylem ingestion* lasting tens of minutes, the behavior is occasionally interrupted by noise. In the EPG literature, "noise" is often used somewhat informally. The device must be very sensitive to record such tiny insects, and as such it is very sensitive to ambient interference (some researchers place the entire apparatus in a Faraday cage in an attempt to mitigate electronic noise interference [33]). However, some authors use "noise" to simply mean any behavior that is not stereotypically part of a known behavioral waveform.

Based on a hunch from an experienced entomologist, we wondered if some of these sections attributed to "noise" could be behaviors that are less well conserved than the

typical *xylem ingestion* waveform. To test this idea, we implemented the image processing filter in Table 6, and searched a 2.7 hour long recording.

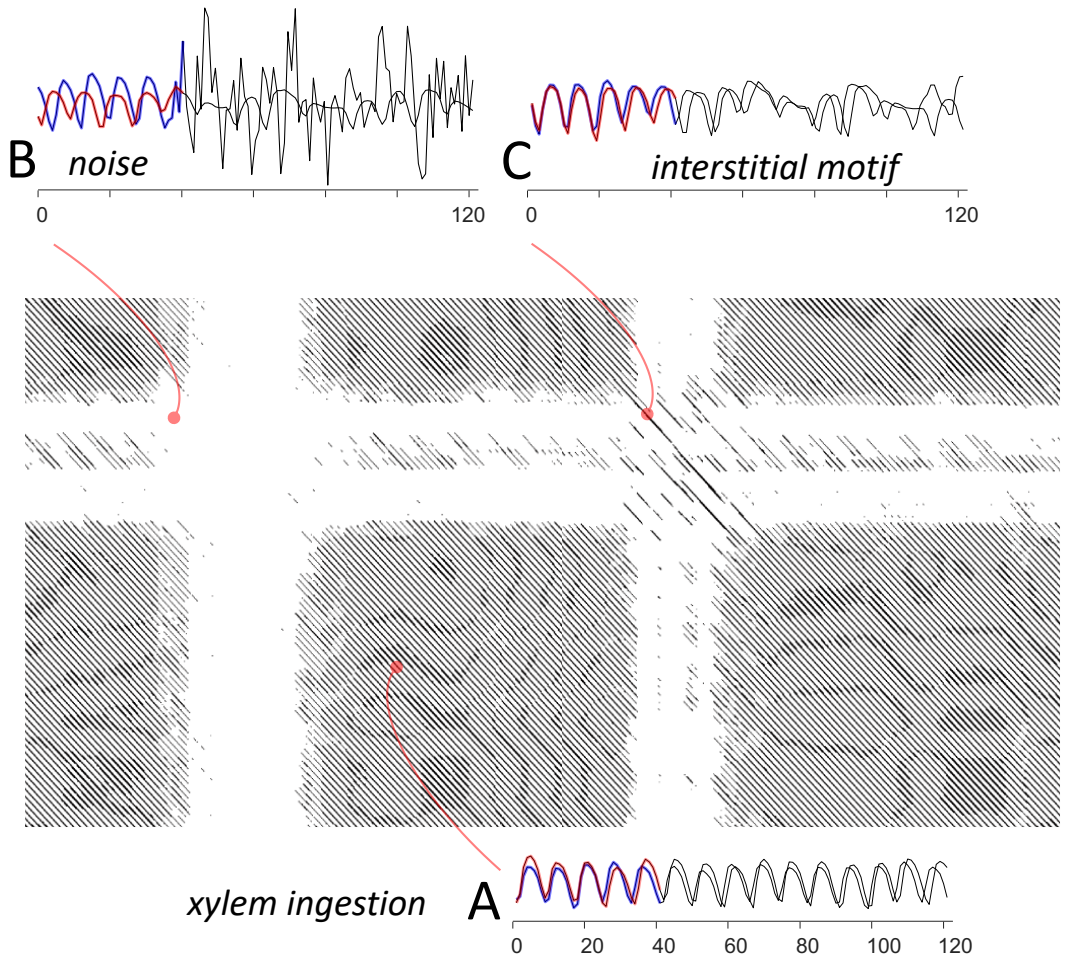

Fig.16 An Mplot with the three corresponding pairs of time series extracted from an Asian citrus psyllid (*Diaphorina citri*). The value of *m* was forty (the length of the colored prefix in the call-out plots), and we show the following eighty datapoints for context. *A*) A typical bout of xylem ingestion shows metronome-like regularity. *B*) The white cross with an empty intersection corresponds to a section of noise (cf. Fig.7). *C*) The white cross with diagonal strip in its intersection corresponds to a rare motif, that occurred between two bouts of *xylem ingestion*.

Fig.16 allows us to illustrate the three possibilities that make up our dataset. Fig.16.*A* shows a dense run of parallel lines, corresponding to the typical *xylem ingestion* waveform (in the literature, this is often called the G phase or G waveform [5][48]. Such patterns make up more than 99% of the Mplot. Fig.16.*B* shows a white cross with an empty intersection. This

corresponds to a noisy region in the time series. Fig.16.*C* shows a white cross with diagonal lines in intersection. This corresponds to what we have dubbed an *interstitial* motif. In Fig.17 we show this motif at a larger scale, to allow the reader to appreciate how well conserved it is.

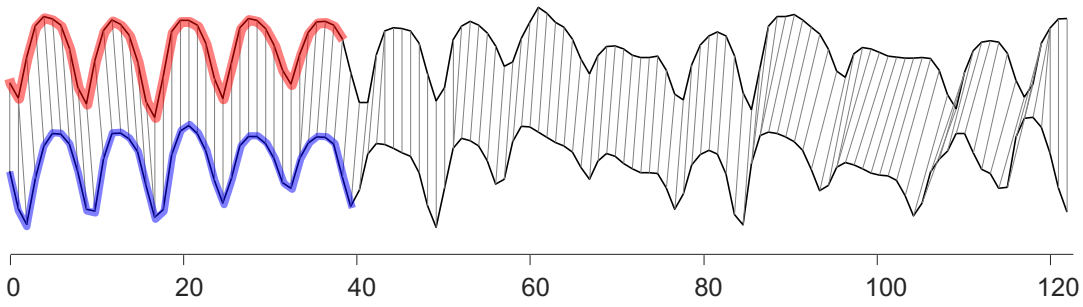

Fig.17 A larger reproduction of the interstitial motif shown in Fig.16.*C*.

We illustrate the similarity of the two time series by showing the Dynamic Time Warping alignment between them [37]. The is only a small amount of warping but is enough such that these two 120-datapoint long subsequences are *not* similar under the classic Euclidean distance. In a sense, we can see the Mplot as revealing a "piecewise" Euclidean distance similarity by showing a diagonal (but slightly wavy) line.

One of the current authors is entomologist who is an expert on EPG data [7]. Although not involved in the collection of this dataset, she believes the interstitial motif shows the insect is transitioning between C phase (navigation through the mesophyll tissue) and the G phase. In [5] they observed that waveform G was always followed by a return to waveform C. This would also explain why it is somewhat regular but not 100% consistent, as C phase has some variability depending on the nature of the tissues the stylet (the insect's needle-like mouthpart) is traveling through.

We use piecewise Mplot to search 1,000,000 datapoints (2.7 hours) for the telltale white crosses. Each patch was of size 10,000 by 10,000 and took about 5.3 seconds to process. As there are 10,000 patches, the entire process took about 8.5 hours. To give the reader an appreciation as to how large a Mplot this is, if we printed out the entire Mplot at the scale shown in Fig.16[3], it would comfortably cover a soccer field.

Finally, we want to demonstrate that the "white cross" heuristic can be a general technique for finding rare motifs in the presence of common motifs, so we will consider a completely different data domain. Here we address the problem of examining telemetry from Contraction in Cardiac Tissue (CCT), which are mechanical contractile signals at the tissue level (the signals are related to, but distinct from the more familiar ECGs [27]. As shown in Fig.18.*bottom.right*, most of such data looks like noise with periodic spikes. This generally produces the classic pattern of diagonal stripes in a Mplot. However, as shown in Fig.18.*left*, when comparing two traces with an AB-Mplot, we occasionally see a white cross with a diagonal strip in the intersection. Here we can use the annotations provided by the creators of the dataset [27] to understand that, as illustrated in Fig.18.*top.right*, this is a rare motif of *slow pulse decay*.

[3] 100 datapoints is about one centimeter, given the scale shown in Fig.16 and this journals format.

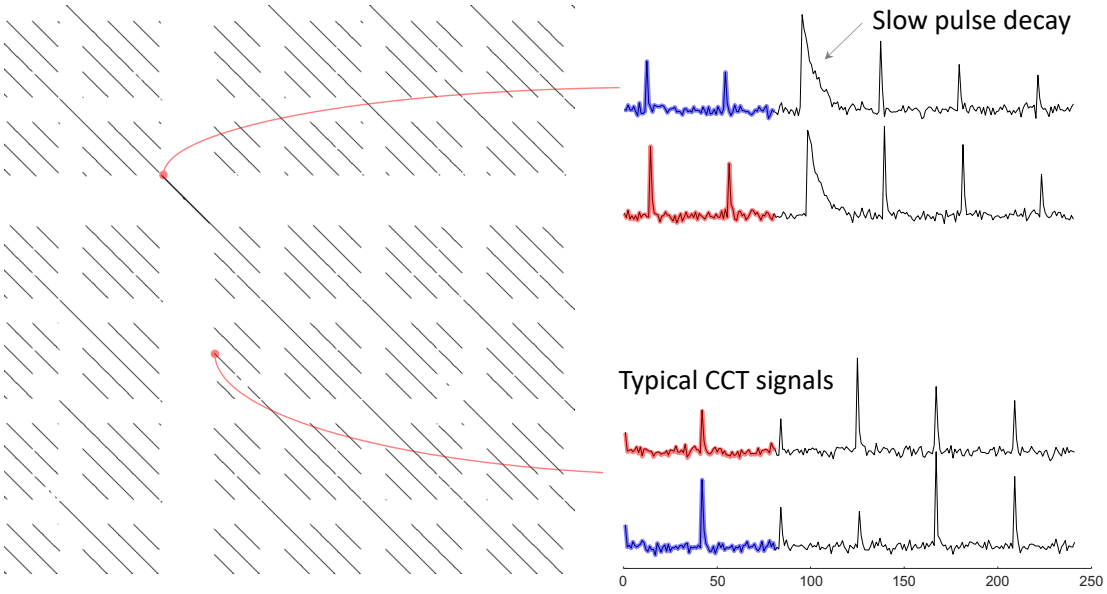

Fig.18 *left*) A zoom-in of an AB-Mplot created with CCT telemetry from two mice. *right*) The value of *m* was eighty (the length of the colored prefix in the call-out plots), and we show the following 160 datapoints for context.

3.3.6 Mplot Based Segmentation

Many researchers have independently noted that if the time series being examined in a Mplot comprises of multiple *regimes*, the Mplot will reflect that fact with a "block-like" structure. Fig.19 illustrates this with a toy example. This suggests that we could formalize this observation to produce a Mplot semantic segmentation algorithm. To search for segmentation points we slightly adapt the method defined in [14], that is used in audio signal information retrieval. This process involves searching for transitions between block structures using the correlation of a checkerboard kernel with the diagonal of the matrix.

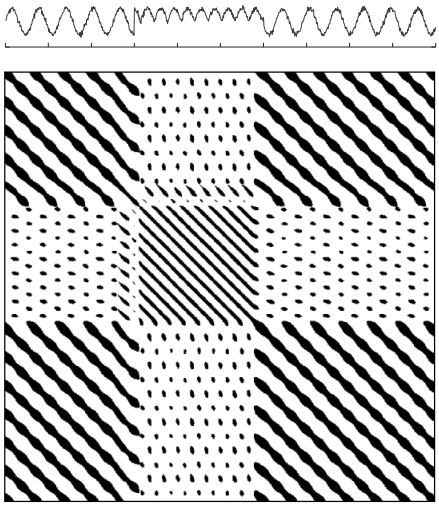

Fig.19 Regime changes produce block-like Mplots.

The result is a 1D function called the novelty function. The change point events are represented by local maxima (peaks) in the novelty function, which are then discovered with a peak finding algorithm. To test the utility of this algorithm we compared to three state-of-the-art semantic segmentation algorithms on a benchmark of thirty-two diverse datasets. We use the evaluation metric suggested by the creators of the datasets [18]. Table 7 summarizes the results.

Table 7. A comparison of Mplot with three SOTA algorithms.

	FLOSS	**AutoPlait**	**HOG-1D**
win |lose|draw over Mplot	20 | 7 | 4	8 | 22 | 2	17 | 13 | 2

In interpreting these results note the following:

- Our algorithm is better than AutoPlait, about the same as HOG-1D, and worse, but not dramatically so, than FLOSS.

- We could have done better by tuning our algorithm, but to avoid *overtuning* we set *m* to be the same value as used by the authors of [18] for FLOSS. Thus, these results should be seen as a lower bound for SPLAT's performance.

SPLAT segmentation has a significant advantage over the other methods, it can give insight into the *cause* of the regime change. For example, consider the PulsusParadoxus$_{SP02}$ problem shown in Fig.20.*top*. Note that SPO_2, also known as oxygen saturation, is a measure of the amount of oxygen-carrying hemoglobin in the blood relative to the amount of hemoglobin not carrying oxygen.

As noted in [18], this problem cannot be solved by visual inspection. The ground truth is known by access to out-of-band data. Nevertheless, both SPLAT and FLOSS correctly segment it. But what *caused* the change? If we saw non-linear structure in the blocks off the diagonal, we could attribute the regime change to a change of heart *rate*, but this is not the case here.

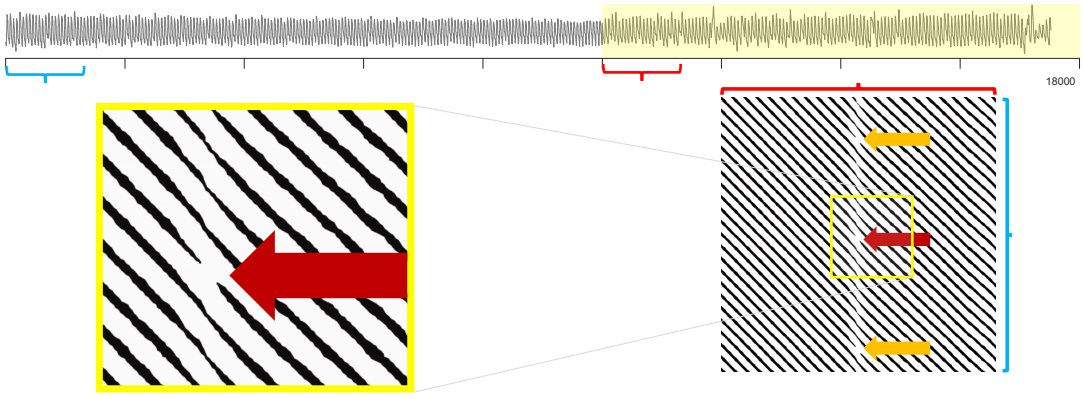

Fig.20 *top*) The PulsusParadoxus$_{SP02}$ segmentation problem is very subtle. *bottom.left*) A zoom-in of the Mplot close to the regime change revels a break in the diagonal streak. *bottom.right*) A zoom-out indicate that these breaks happen once in every eight beats.

However, there is an interesting clue as shown in Fig.20.*bottom*. There is a slight reduction in the degree of conservation of heartbeats, that happens about once every eight

beats. The reader will appreciate that the ratio of typical respiration rate to heartbeat rate is about eight-to-one.

Normally we should not expect respiration to effect SPO_2. However, if the pericardium, a sac-like structure surrounding the heart, is damaged during surgery, it can fill with fluid and then deep breaths can cause pressure on the heart (this is called Cardiac tamponade) and reduce its efficiency in producing oxygenated blood. According to Dr. Greg Mason (Clinical Professor of Medicine, David Geffen School of Medicine at UCLA) this is exactly what we are seeing here.

3.3.7 Speed and Scalability

In Fig.21.*left* we evaluate the time needed for SPLAT for increasingly long time series (n) when the subsequence length (m) is set to 100. Then, in Fig.21.*right* we hold the length of the time series to a fixed 16,000, and test the effect of increasingly large values of m.

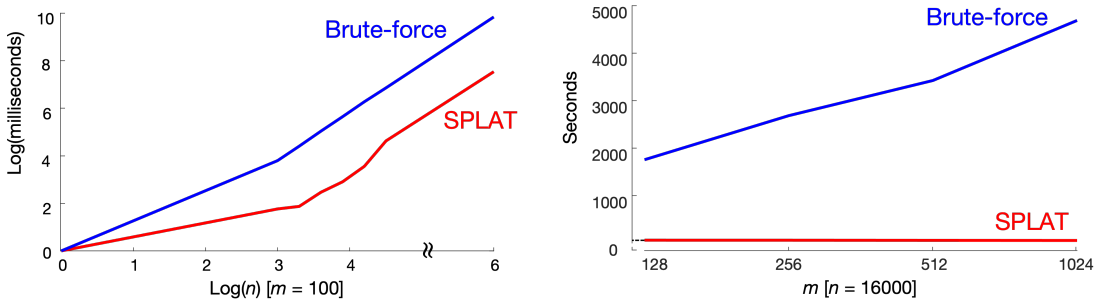

Fig.21 SPLAT execution time vs. brute-force algorithm – Note that both the left figure's axis are in log scale.

The reader will observe that we can compute a million length time series in about 9.5 hours using PiecewiseSPLAT. This is extremely fast given that the brute-force algorithm would take 5.4 years.

We can further accelerate our algorithm by leveraging the hardware. To test this, we ported SPLAT to GPUs. As the results in Table 8 show we can process a time series of length one million in just 6.3 seconds. We refer the reader to visit [41] for more results and the GPU code.

Table 8. Pooled Mplot Timing Results (in seconds) on 1×Nvidia GPU P100.

Time series length	Mplot 100 × 100	Mplot 1k × 1k	Mplot 4k × 4k	Mplot 8k × 8k
128k	0.20	0.21	0.26	0.46
256k	0.49	0.47	0.55	0.73
512k	1.58	1.57	1.64	1.84
1M	6.01	5.99	6.05	6.27

In a just published paper the authors introduce PyRQA, "*a software package that efficiently conducts recurrence quantification analysis… leveraging the computing capabilities of a variety of parallel hardware architectures*" [39].They also consider a dataset of size 1M, finding it took 68.94 seconds to process. This is an order of magnitude slower than the time we required. Moreover, our results in Table 8 used a *single* Nvidia P100 GPU, whereas [39] use *four*, much faster NVIDIA GeForce GTX 690 GPUs. The two software packages are not identical in features, nevertheless, this comparison does hint at the efficiency of our proposed algorithms.

3.4 Conclusions

We introduced SPLAT, an algorithm that allows us to construct Mplots that are orders of magnitude larger than those that are typically computed. We have shown that such Mplots can be used for tasks in domains as diverse as astronomy, medicine, entomology, and biodiversity monitoring. Our proposed algorithms are so scalable that for the first time, space and time complexity are no longer bottlenecks, but human attention is. Therefore,

we further show that our ideas can support patchwise search of massive Mplots, to find a handful of patches that are worth bringing to the attention of a user.

We have made all code and data freely available to allow the community to confirm our results and build upon our ideas.

4 MOTIF-ONLY MATRIX PROFILE: ORDERS OF MAGNITUDE FASTER

While SPLAT has significantly advanced our ability to generate and utilize Mplots across various domains, the exploration of large-scale data often requires not just visualization but also the efficient detection of patterns within the data. To further enhance our ability to manage and interpret vast datasets, we now shift our focus to the critical task of motif detection in time series data. This transition from large-scale visualization to rapid pattern recognition underscores the next step in our journey.

In this chapter, we introduce a novel, multi-resolution algorithm that accelerates motif detection by orders of magnitude. Additionally, we establish the first-ever lower bound for the matrix profile, providing a new theoretical foundation for this important task. This advancement complements our previous work by addressing both the visualization and pattern recognition needs in large datasets.

4.1 Geometric Intuitions

Before introducing our definitions and notation in the next section, we will take a moment to visually review the ideas behind lower bounding, the triangular inequality and the combination of these two techniques. While the triangular inequality is a commonly used tool for indexing etc., our work is unusual in that it exploits triangular inequality twice, hence this review may help sharpen the readers intuition for the contributions in Section 4.3.

In Fig.22.*left*, we show two points, R_1 and B_1 that are 9.0 units apart in Euclidean space.

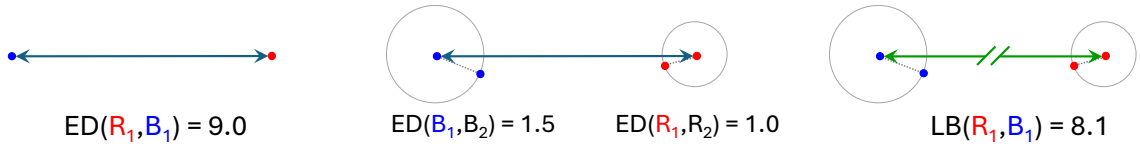

Fig.22 left) We know points R_1 and B_1 are 9.0 units apart. center) We further know that B_2 is 1.5 units from B_1, and that R_2 is 1.0 units from R_1. right) Here we assume we do not know the true distance between R_1 and B_1 but we know a lower bound for it, 8.1.

Further assume that we know B_2 is 1.5 units from B_1 and that R_2 is 1.0 units from R_1, as is illustrated in Fig.22.*center*. From this we can derive a new fact, a lower bound of the distance between B_2 and R_2 is 6.5, which we computed as a Lower Bound with two applications of Triangular Inequality:

$$LBTI(B_2,R_2) = ED(R_1,B_1) - [ED(R_1,R_2) + ED(B_1,B_2)] = 6.5$$

Note that while the LBTI here is positive, if the "circles" shown in Fig.22.*center* were relatively large, then this function could be negative, which we snap to zero.

We can generalize this idea to the case where we do not know the *exact* distance $ED(R_1,B_1)$ but, as shown in Fig.22.*right*, we only have some lower bound for it, i.e. $LB(R_1,B_1) \leq ED(R_1,B_1)$. We can still compute a (now *weaker*) lower bound between B_2 and R_2 by changing a term in LBTI.

$$LBTI(B_2,R_2) = LB(R_1,B_1) - [ED(R_1,R_2) + ED(B_1,B_2)] = 5.6$$

To make it clear that how we can exploit this information imagine that we know "for free" the information in Table 9.

Table 9: Incomplete information about pairwise distances between objects in R & B

	R_1	R2	B_1	B2
R_1		1.0	?	?
R2			?	?
B_1				1.5
B2	$LB(R_1,B_1) = 8.1$			

Further imagine we are tasked finding the *closest pair* between any R point and any B point. Naively, we could compute the four missing values in Table 9 and select the minimum. But instead, we leverage the intuitions presented in Fig.22 to find the answer, while doing less work.

Suppose that we randomly choose one of the four missing values to compute, say $ED(R_2,B_2)$ and discover it is 6.5, establishing a *best-so-far*. In this case we can see that:

- $\{R_1,B_1\}$ cannot be a closer pair, because we have:

$$ED(R_1,B_1) \geq LB(R_1,B_1) \text{ which is } 8.1$$

Since 8.1 is greater than our *best-so-far*. $\{R_1,B_1\}$ is pruned.

- $\{R_1,B_2\}$ cannot be a closer pair, because we have:

$$ED(R_1,B_2) \geq LB(R_1,B_1) - ED(B_1,B_2), \text{ which is } 8.1 - 1.5$$

Since 6.6 is greater than our *best-so-far*. $\{R_1,B_2\}$ is pruned.

- $\{R_2,B_1\}$ cannot be a closer pair, because we have:

$$ED(R_2,B_1) \geq LB(R_1,B_1) - ED(R_1,R_2), \text{ which is } 8.1 - 1.0$$

Since 7.1 is greater than our *best-so-far*. $\{R_2,B_1\}$ is pruned.

Thus, in this case, we only had to compute one new distance calculation, not four.

There is an obvious generalization of the above ideas. Suppose as shown in Fig. 23 we have *many* points in the R and B sets.

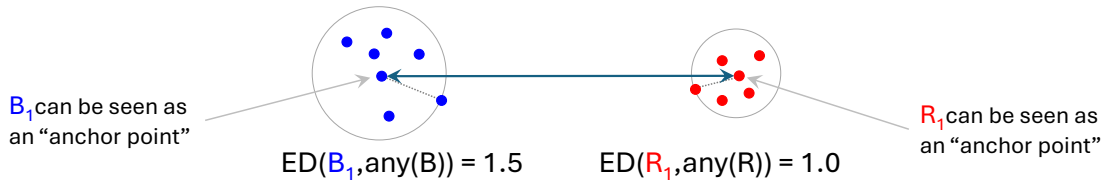

Fig. 23 We can generalize the triangular inequality pruning to include a cohort of points, simply by recording the distance between an anchor point and the most distant member of the cohort.

We could record each of these point's distance from its "anchor point" R_1 or B_1. However, this would require significant memory overhead. It sufficient to record just the largest distance and use that as a bound for all its cohort.

4.2 Definitions and Background

We begin by outlining the definitions and notations used in this work. We start with the data type of our interest, which are *time series*.

Definition 4: A *time series* $\mathbf{T} = t_1, t_2, \ldots, t_n$ is a sequence of real-valued numbers.

We are not interested in the global properties of a time series but rather shapes of small regions called *subsequences*.

Definition 5: A *subsequence* $\mathbf{T}^{(i,m)}$ is a contiguous subset of values from T starting at index i with length m.

The Z-normalized nearest neighbor distance for all subsequences of length *m* in T can be stored in a meta time series, the *matrix profile* [50].

Definition 3: A *matrix profile MP* of time series T is the vector of the z-normalized Euclidean distance between all subsequence $T^{(i,m)}$ and their nearest neighbor $T^{(j,m)}$ in T.

When finding the nearest neighbor to each subsequence, we enforce an *exclusion zone* of *m*/2 before and after location *i* to avoid considering the trivial matches [8]. The lowest values in a Matrix Profile (there will always be a tied pair) correspond to the Top-1 motifs, that is to say, the pair of subsequences that have the lowest mutual Euclidean distance.

4.3 Lower Bounding the Matrix Profile

As the examples in the previous section hinted at, we plan to create a *lower bound* for the MP. This lower bound will be the same length as the true MP, but faster to compute. As shown in Fig. 24 our starting point is to use the Piecewise Aggregate Approximation (PAA) [22] to downsample T.

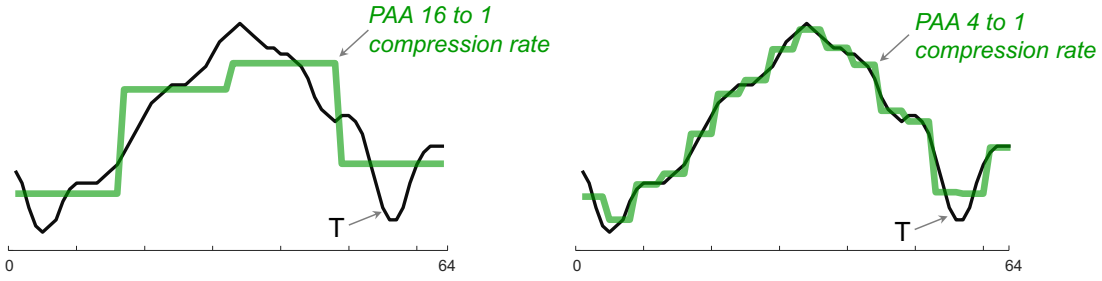

Fig. 24 left) A course 16-to-1 approximation of a time series ***T***. right) A fine 4-to-1 approximation of the same time series.

The PAA can be defined as [22]:

Definition 4: 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. More specifically, for each window *i*, the appropriate value is calculated by the following equation:

$$\overline{t_i} = \frac{k}{n} \sum_{j=\frac{n}{k}(i-1)+1}^{\frac{n}{k}i} t_j$$

Note that the examples in Fig. 24 and all subsequent examples assume that the length of the time series is a power-of-two. This is not a requirement, the PAA is defined when m and/or n are arbitrary integers, this just simplifies exposition.

Assume we apply PAA to T using a downsample rate (dsr) producing an output T′ with length n/dsr. If we compute the MP on T′ then the computation is dsr^2 times faster than computing the MP on T. This "proxy" MP has two issues. First it is too short by a factor of dsr. This is easy to fix by upsampling, simply repeating each value dsr times. The second issue is that the proxy MP is "weak". The true MP records the distances between subsequences of length m, but the proxy MP considers distances between subsequences of length just m/dsr, which in general have a lower value. To correct this we multiply T′ by $\sqrt{dsr}$ [22]. We call the result the *Approximate Matrix Profile*.

Definition 6: The approximate *Matrix Profile* AMP_{dsr} is the MP vector computed on T′. The computed MP is then upsampled by the same factor, by repeating every value for dsr times, and multiplied by $\sqrt{dsr}$.

Note that a special case of the AMP is $AMP_{1\text{-to-}1}$ which is simply the original Matrix Profile. In Fig. 25 we show how this notation harkens back to the examples in the previous section. Note that each t_i ($i = \beta$d+1), is an *Anchor* point, and the following dsr-1 values are called *cohort* points.

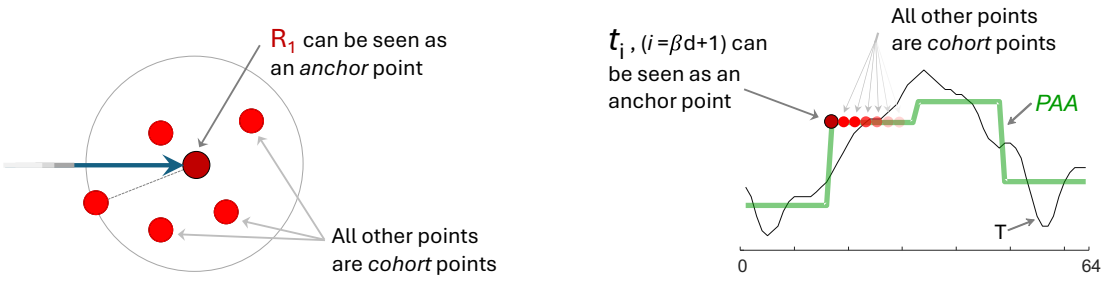

Fig. 25 left) (cf. Fig. 23) The geometric example considered in the previous section has a perfect analog with the AMP (right).

Why did we make this connection? As shown in Fig. 26, if we plot AMP and MP together it *appears* that the AMP *lowers bounds* the MP, however this is not the case!

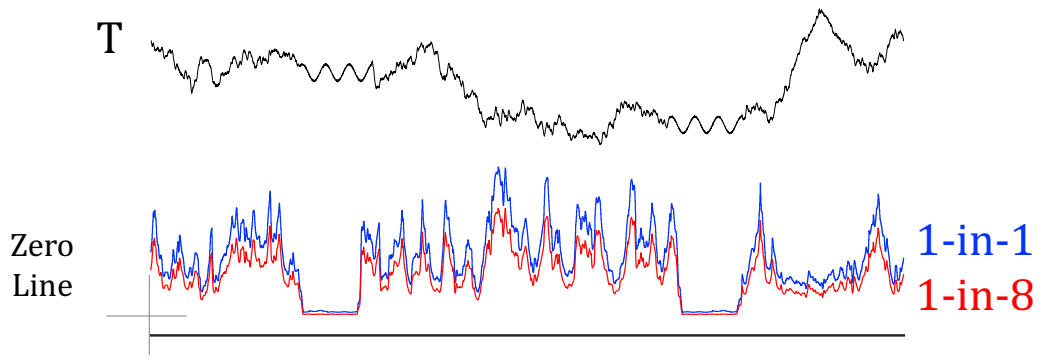

Fig. 26 The MP and the $AMP_{8\text{-to-}1}$ for time series ***T***. Visually the AMP appears to be a lower bound for the MP, but this is not the case.

The AMP is only guaranteed to be a lower bound for MP only at locations that are aligned to anchor points, and only then it is lower bounding to other subsequences that *also* happen to align to anchor points.

To define a true lower bound for all values in MP, including all the cohort subsequences, will take additional work. We begin by introducing the *K-Triangular Inequality Profile*.

Definition 7: A *K-Triangular Inequality Profile KTIP*, is defined as $d^{(i,x)}$ for x∈ (i, j), where t_i and t_j are two consecutive anchor points and t_x is t_i's furthest cohort point.

The individual values in the KTIP can be seen analogous to the circle shown in Fig. 25.*left*. They can be seen as defining a region of variability or uncertainty of subsequence shape around the anchor point. To make a true lower bound, we must "compensate" for this uncertainty by subtracting it from the true lower bounding information that we do know. In the example in Section 4.1, we did this with

$$LBTI(B_2,R_2) = LB(R_1,B_1) - [ED(R_1,R_2) + ED(B_1,B_2)]$$

In the notation below, we show a perfect analogue of this, a final lower bound that comprised of a lower bound, minus the sum of two triangular inequality-based compensation.

Based on the intuition explained in Section 4.1, we propose using KTIP to define *Lower Bound Matrix Profile*.

Definition 8: A *Lower Bound Matrix Profile lbMP$_{dsr}$*, is a vector of distance values, lb_{dsr} = $[lb_1, \ldots, lb_{n-m+1}]$. Where $lb_i = \max[amp^i_{dsr} - (ktip^i + ktip^j)]$ for $j \neq i$.

lbMP$_{dsr}$ is a parameterizable lower bound for the MP. Note that it has an interesting special case; when dsr is 1 we have lbMP$_{1\text{-to-}1}$ = MP. Thus, the MP is a special case of lbMP$_{dsr}$.

More generally, as dsr is set to larger values, the time needed to compute it decreases, but the tightness also decreases. To see this, we need a formal metric. We define TLB, the *Tightness of the Lower Bound* as:

$$TLB = 1 - [\ ED(MP, lbMP_{dsr}) / ED(MP, ZeroLine)\ ]$$

As illustrated in Fig. 27.*right*, TLB ranges from zero to one, with zero being a useless lower bound, and becoming more effective as it approaches one.

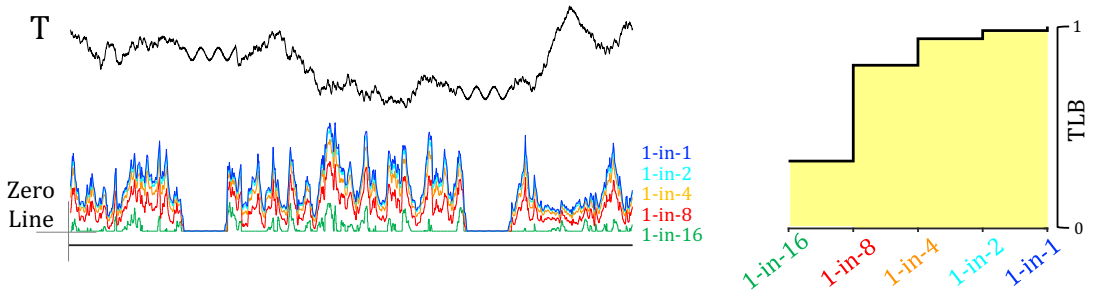

Fig. 27 left) Five lbMPs of time series ***T*** for various levels of downsampling. Note that the special case of lbMP$_{1\text{-in-}1}$ is just the normal Matrix Profile. right) The Pareto frontier of the time needed to compute each lbMP$_{dsr}$ vs. its tightness.

How can we use a lower bound? All Matrix Profile algorithms initialize a *best-so-far* variable (*bsf*) to infinity, and then incrementally reduce it until it is the true distance of the top-1 motif pair. This suggest a simple pruning rule; any region of the time series that corresponds to a section of the lbMP that is greater than the current *bsf* can be admissibly pruned.

As shown in Fig. 27, we have a *parametrizable* lower bound. This suggests the need for careful consideration of the trade-off involved. We could invoke a fast computation, but only obtain a weak lower bound. Alternatively, we could spend additional computational resources to obtain a tighter lower bound. This will almost certainly allow us to prune more, but will this more aggressive pruning pay for itself? This is a very difficult thing to

optimize, as the minimum useful tightness of a lower bound depends on the current value of *bsf* variable. However, the current value of *bsf* variable will change as the algorithm runs. Moreover, the current *bsf* variable itself depends on two things:

- The final true distance of the top-1 motif pair. Clearly that is a lower bound for the best *bsf* variable we can see.
- How fast the algorithm finds at least a good (i.e. *low*) *bsf* variable. This in turn depends on the algorithm's search strategy, and the data itself.

To make this concrete, consider the two timeseries shown in Fig. 28. In Fig. 28.*left*, a $\text{lbMP}_{\text{8-to-1}}$ is sufficient to prune almost all the data. A $\text{lbMP}_{\text{2-to-1}}$ used here would take sixteen times longer to compute but is no more effective at pruning. In contrast, in Fig. 28.*right*, a $\text{lbMP}_{\text{8-to-1}}$ can prune almost nothing and is simply a waste of computation.

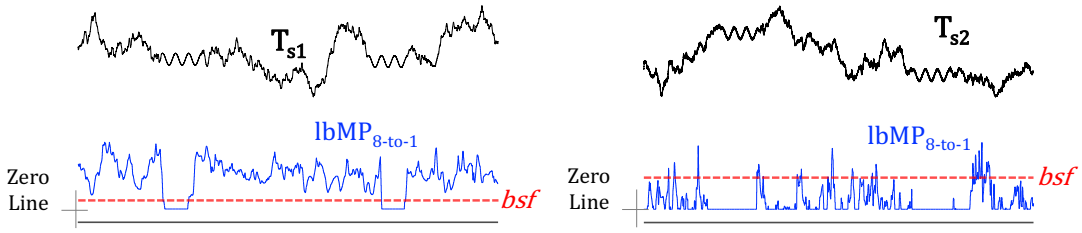

Fig. 28 In these examples, the pruning algorithm has a bsf that is ~8% greater than the true final motif distance. left) A 8-to-1 aMP can prune almost all the data. right) A 8-to-1 aMP cannot prune any data.

In the next section we introduce MOMP, an algorithm that solves this issue. MOMP begins with a coarse lbMP_{dsr}, and then iteratively passes any surviving (i.e. unpruned) data to finer lbMP_{dsr} steps. In the limit, the finest lbMP_{dsr} is the $\text{lbMP}_{\text{1-in-1}}$, which is just the classic Matrix Profile.

This strategy has two notable effects, it removes the "guesswork" of choosing the right down-sampling level. In addition, this strategy tends to quickly reduce the *bsf* variable. This in turn has two positive effects. It maximizes the pruning effectiveness, hence speeding up the algorithm, and it makes the algorithm strongly *anytime*.

4.4 MOMP

In this section we introduce the MOMP algorithm. For simplicity of presentation, we consider only the case of Top-1 motif, self-join, and *m* is a power-of-two. However, all the generalizations of these assumptions are trivial and have already been implemented [31].

4.4.1 Introducing MOMP

We begin by giving the intuition behind MOMP. The core idea is to attempt to prune as much of the time series as possible with the coarsest (and therefore *cheapest*) lower bound. Any time series that survives that pruning is then considered at sampling rate that is twice as fine. The fact that the data is twice as fine means that the Matrix Profile computation would be four times as slow *if* all the data was unpruned. However, if any amount of data was pruned, this finer computation will have been accelerated. We iteratively continue this *prune-then-upsample* step until the unsampled data is at the original (i.e. 1-in-1) sample rate. At this point the Matrix Profile algorithm searches the remaining data and returns the true best motif pair.

This algorithm is formalized in Table 10.

Table 10: The MOMP algorithm

Function:	MOMP(T, m)
Input:	T : Input time series m: Subsequence length
Output:	momp_out: Distance matrix
1	T_0 = T
2	dsr = m/32 # Set initial coarse down sample rate
3	bsf = inf
4	full_ktip = computeKTIP(T_0, m, dsr) #Table 11
5	while true
6	ip = full_ktip(:, log2(dsr))
7	lbMP, local_bsf = computeLBMP(T, m, dsr, ip) #Table 12
8	bsf = refineBSFloc(T_0, m, local_bsf, bsf) #Table 13
9	prnT = prune(T_0, m, lbMP, bsf) #Table 14
10	T, dsr = prnT, dsr/2
11	if dsr == 1
12	mp, motifloc = SCAMP(T, m) #Or *any* other MP algorithm
13	return min(mp), prnT.indices(motifloc)

The algorithm starts by taking in the input time series, T and the user's choice of subsequence length m. In line 1, the input time series is assigned to T_0 as the original input. Then in line 2, the initial downsampling rate (dsr) is set. In line 3 the *bsf* value is initialized to infinity. Line 4 computes the full KTIP matrix explained in Table 11 on T_0. Each column of which is later used at the appropriate downsampling level.

The *prune-then-upsample* loop starts at Line 5. Line 6 selects the corresponding KTIP array, then uses it in line 7 to compute the lbMP. The lower bound algorithm outlined in Table 12 returns the $lbMP_{dsr}$ along with a *bsf* value. We call this the local *bsf*. This local *bsf* only represent motifs that start at an anchor point (recall Fig. 25.*right*). However, a tighter motif pair might be possible if we also consider the cohort points around the anchor point. Thus line 8 executes a quick local tuning on T_0 to refine the local *bsf* value and return the (generally smaller) current *bsf*.

In Line 9 the algorithm admissibly prunes any section of the time series that has a corresponding region of the lbMP that is greater than the *bsf*, and in line 10 the pruned time series is promoted to the next iteration. Moreover, the downsampling rate is divided by two, so that the next iteration is working with data that is twice as finely sampled. Line 11 checks to see when downsampling rate reaches one, at that point the exact MP computation is done on whatever regions of the time series has survived pruning up to that point. Line 12 computes the final MOMP results and line 13 returns the output.

Note that there are no parameters for MOMP. We do need to pick a starting downsample rate. As shown in line 2 of Table 10, we use (m/32)-to-1, but replacing the 32 with 128, 64 or 16 makes no measurable difference. The number of times the loop beginning at line 5 iterates is $\log_2(m/32)$.

This outline explained MOMP. We can now consider it subroutines in more detail, by further examining the details of KTIP computation (Table 11), lbMP computation (Table 12), refinement (Table 13) and pruning (Table 14) in more detail.

In line 2, the KTIP matrix is initialized as all NaN values. This matrix is the length of the MP array, and its column count is set to the number of MOMP steps. A temp variable is defined in line 3 for temporary storage of minimum values.

Then lines 4 to 13 compute the minimum distances, and in line 12 and 13 the required KTIP values are stored. Finally, line 14 returns the full KTIP matrix. The KTIP is then used in computing the lbMP using the algorithm in Table 12.

Table 11: K-Triangular Inequality Profile Algorithm

Function:	computeKTIP(T, m, dd)
Input:	T: Input time series m: Subsequence length dd_0: Initial downsampling rate
Output:	ktip*:* lower bound Matrix Profile
1	n = len(T)
2	ktip = nan(n-m+1, log2(dd_0))
3	temp = nan(n-m+1, 1)
4	for diag = 1:dd_0
5	for rr = 1:n-m-diag+2
6	cc = row + diag – 1
7	dist ← ED(T(rr:rr+m-1), T(cc:cc+m-1))
8	if dist < temp(rr)
9	temp(rr) = dist
10	if dist < temp(cc)
11	temp(cc) = dist
12	if ispow2(diag)
13	ktip(:, log2(diag)) = temp
14	return ktip

Table 12: Lower Bound Matrix Profile Algorithm

Function:	computeLBMP(T, m, dd, ip)
Input:	m: Subsequence length dd: Downsampling rate ip: ktip for dd step
Output:	lbMP*:* lower bound Matrix Profile
1	lbMP = nan(size(amp))
2	dT = PAA(T, dd)
3	amp = SCAMP(dT, m/dd)
4	for i = 1:len(amp)
5	lbMP(i) = max jϵ[1: len(amp)], j ≠ i (amp(i)-ip(i)-ip(j))
6	lbMP ← upsample(lbmp, dd)
7	return lbMP, min(lbMP)

In line 1, the lbMP array is initialized as NaN values. Then in line 2, the input time series is downsampled by 1 in dd using PAA (recall Definition 4). Line 3 computes the MP on the downsampled time series. This allows lines 4 and 5 to compute the lower bound for every value in *amp*. Line 6 upsamples the computed lbMP by factor dd to maintain the original length. Finally line 7 returns the lbMP and its minimum value which is an approximate *bsf*. Recall that, as shown in Fig.22.*right*, this *bsf* is limited to representing motifs that start at an anchor point. In Table 13, we use a refinement function to adjust the approximate *bsf* locally, by considering the cohort points to the right of the anchor points that currently have the *bsf* value.

Table 13: Best-so-far Local Refinement

Function:	refineBSFloc(T, m, local_bsf, bsf)
Input:	T: Input time series m: Subsequence length local_bsf: bsf found by lbMP bsf: current bsf value
Output:	bsf: updated bsf value
1	i,j ← local_bsf.loc
2	segA = T(i-dd+1: i+m+dd-1)
3	segB = T(j-dd+1: j+m+dd-1)
4	[mp, minloc] = SCAMP([segA, segB], m)
5	If min(mp) < bsf
6	bsf ← min(mp)
7	bsf.loc ← T.indices(minloc)
8	return bsf

In line 1, the location of the local motifs found by lbMP is assigned to *i* and *j*. Then in lines 2 and 3, two small segments of time series are made, covering the approximate motif locations and the *dd* cohort points to the right. Then in line 4, a classic matrix profile is computed on this tiny subset of T. Lines 5 to 7 updates the *bsf* value if a better motif is found, and Line 8 returns the *bsf*. Then we use the approximate and refined *bsf* values to prune T, using the algorithm in Table 14 and continue to the next iteration.

In line 1, the pruned time series is initialized as an empty array. The target subsequences are found in line 2 by applying the best-so-far threshold to lbMP values. Then in lines 3 and 4, the target subsequences are concatenated. The overlaps and alignment of

concatenation are handled in the original code, but this is glossed over for brevity. Line 5 returns the pruned time series.

Table 14: Pruning Algorithm

Function:	prune(T, m, lbMP, bsf)
Input:	T: Input time series m: Subsequence length bsf: refined bsf value
Output:	prnT: pruned time series
1	prnT = []
2	tgts ← locate(lbMP ≤ bsf)
3	for t in tgts
4	prnT.concatenate(T(t:t+m-1))
5	return prnT

4.4.2 MOMPs Cost Model

The cost for MOMP is almost completely dictated on the final prune rate at the time we finish computing on the 2-to-1 down-sampled data, and are about to pass the non-pruned data at full resolution to the classic MP algorithm in line 13 of Table 10. Thus, the speed achieved over classic MP is basically: $1/(1-prune\ rate\ at\ DSR_{2-to-1})^2$.

However, this model ignores the overhead cost of lines 1 to 12 of Table 10. Here there is some variability. In the best case, the coarsest down-sampling might prune almost everything, and this overhead would be almost immeasurably small. However, let us consider the worst case, in which nothing is pruned. The overhead would consist of fruitless computation of the SCAMP algorithm at down-sampled data with dsr of 2-to-1, 4-to-1, 8-to-1 etc. These would have a cost of $1/2^2$, $1/4^2$, $1/8^2$ etc., which sums to an overhead of ~33%.

There is also a small amount of overhead for down-sampling/up-sampling, computing the KTIP, refining the *bsf* etc. Empirically, the total overhead is typically less than a factor of two. Thus, as shown in Fig. 29, the acceleration achieved by MOMP over MP is well modeled as: Speedup = $(1/(1-prune\ rate\ at\ DSR_{2-to-1})^2)/2$

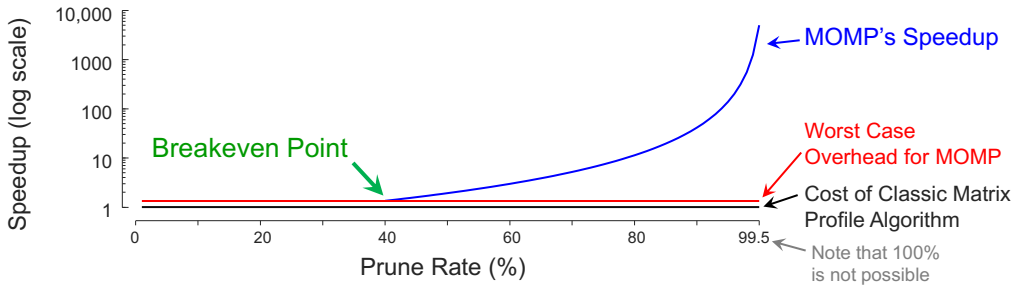

Fig. 29 The cost model for MOMP

As show in Fig. 29 this implies that MOMP can "breakeven", that is to say, cover the cost of the overhead, once the prune rate reaches about 40%. As the prune rate gets higher than that, the speed up rapidly makes the overhead inconsequential. For example, if we can prune 90% of the data, the speed-up is about forty-one times faster, and if we prune 99% of the data, the speed-up is about 1,250 times faster. Remarkably, as we will show in Section 4.5, we do see such prune rates and speedups in real-world datasets.

4.4.3 Observations about MOMP

Here we discuss some of the properties of MOMP.

- MOMP is independent of the base Matrix Profile algorithm used. It can use STAMP, STOMP, SCAMP, SCRIMP, SCRIMP+, or their GPU multi-threaded versions or their reduced precision versions [36] etc.
- Some of the original MP algorithms, notably SCRIMP++ are strongly *anytime* algorithms. As we show in Fig. 34, MOMP not only inherits this property, but it can convert the currently batch-only MP algorithms such as STOMP or SCAMP to *become* strongly anytime algorithms.
- MOMP is independent of the lower bound used. In principle MOMP could be used with any new lower bound invented in the future. There is a simple test to see if any newly proposed lower bound could help accelerate MOMP, it needs to be above the Pareto frontier shown in Fig. 27. For the simpler problem of *time series similarity search*, there is a rich history of alternative lower bounds [22], we suspect this will be a fruitful area for future research.

- Since the introduction of the Matrix Profile in 2016, the community has produced many extensions including Motif-Joins, Consensus Motifs, Chains, K-motifs, Novelets, multi-dimensional motifs etc. MOMP can be used to accelerate all these primitives. For brevity we confine our demonstration to Motif-Joins in Section 4.4.4. Note that MOMP does not accelerate *discord* discovery, but there is already DAMP, an ultra-fast *upper* bound method for this task.
- MOMP is more *sampling-rate invariant* than classic MP algorithms. Imagine Alice and Bob both record the exact same hour of EOG data using a Edanusa X12 recording device [12], but Alice at the lowest setting of 50 Hz, and Bob at the highest setting of 400 Hz. As shown in Fig. 32, in this case they both find the same top motif of length twenty seconds, but while Alice's search takes 109 seconds, Bob's search takes 117 minutes. This is about sixty-four times longer, as we expect from the $O(n^2)$ time complexity of SCAMP. However, MOMP is only 5.1 times slower when using the finer resolution data[4]. This is because the performance of MOMP is more influenced by the *intrinsic* dimensionality of the data, and not the actual sampling rate. This is a very useful property, as many datasets are arguably greatly oversampled.
- The worst-case *time* overhead for MOMP as shown in Fig. 29 is remarkable. Most algorithms based on lower bounding in the dimensionality reduced space have good *best* cases, but in the worst case they can be orders of magnitude slower than a simple brute force search. For example, the recently introduced Attimo[5] has a good best case for motif discovery [6], but as we show in [31], in the worst case it is hundreds of times slower than brute-force.
- The worst-case *space* overhead for MOMP is also remarkable, it is just $O(n)$. Once again, many time series algorithms have *best* cases, but untenable worst cases. For example, the space complexity for QuickMotif is $O(m + survivors)$ [23]. In the best case the number of *survivors* might be small, but in the worst case there can be $O(n^2)$ *survivors*. Previous

[4] MOMP's subquadratic scaling is due to its improving pruning ability in the finer data. For the low-sample-rate data MOMP prunes 91.4% of the data, but for the high-sampling-rate data it prunes 96.9% of the data. Full details in [31]

[5] Attimo is not an *exact* time series motif discovery algorithm. It finds the best motif with a user specified probability [6].

work noted that "*the memory footprint for Quick-Motif tends to be very large*" [51], and that was when considering datasets that are less than one-hundred the size of the datasets we consider in this work.

4.4.4 Limitations of MOMP

There are two bad cases for MOMP. Interestingly, they are essentially two opposite cases:

- ***Nothing* is a Motif**: If the data is just random noise, then the BSF variable will never be significantly smaller than any part of LB (see line 2 of Table 14) and no pruning will take place. Of course, in such a situation, there are no semantically meaningful motifs, because the closest pair of subsequences will only be slightly closer than any random pair of subsequences [4].
- ***Everything* is a Motif**: Suppose the data consists of perfect sine wave with a little noise. Here there are visually satisfying motifs, but any random subsequence with its nearest neighbor will almost be as similar. Because of this, all lower bounds will be close to zero everywhere and no pruning is possible.

Obviously, we do not normally expect to work with such pathological datasets, however real datasets may approach either case. There are two other scenarios where MOMP does not offer significant improvements. First, if n is small, SCAMP is so fast that there is little room for improvement. The second case is if m is small. Given that MOMP is doing a multi-resolution search, if m is very short, there simply are not multiple resolutions to search over.

4.5 Experimental Evaluation

To ensure that our experiments are reproducible, [31] which contains all data/code for the results, in addition to many experiments that are omitted here for brevity.

Unless otherwise stated, all experiments were run on a Dell XPS 8920, with Intel Core i7-7700 CPU @ 3.6GHz and 64GB RAM. While there is great interest in using HPC for time series motif discovery [36][54][56], here we intentionally use a dated and underwhelming machine.

Note that we mostly consider motif lengths that are a power-of-two. This is *not* a limitation of our work; the PAA representation is defined for arbitrary lengths [22]. However, later research may wish to build alternative lower bounds based on wavelets or DFT, and both those representations have their best cases for *n* and/or *m* equals a power of two [22].

We use a highly optimized MATLAB version of batch-only SCAMP as both the base algorithm for MOMP and our rival strawman [56]. Because both approaches use the same base algorithm on the same hardware, any speed differences can be solely attributed to our algorithm.

There is almost no other algorithm we can compare to. The recent Attimo [6] and HIME [16] are *approximate* algorithms, but we are interested in *exact* search. Quick-Motif [23] is an exact algorithm, and we do compare to it.

There are dozens of papers that expand on the Matrix Profile. However most, such as *Motiflets* [40], use a standard algorithm such as SCAMP, but offer different ways to extract the motifs once the Matrix Profile has been computed.

4.6 Establishing Two Important Facts

We begin by establishing two facts that we will use for the rest of our empirical evaluations.

- We can perfectly predict how long SCAMP will take, given only length of the input time series (see Appendix B).
- The cost model introduced in Section 4.4.2 is reasonable.

To see this, we can conduct an experiment to measure the speed up obtained by MOMP in the face of increasing noise. We created a random walk of length 2^{20}, and using SCAMP we measured how long it takes to find the Top-1 motifs of length 2^{11}. As shown in Fig. 30.*right*, it takes SCAMP about 4,976 seconds (about 1.38 hours).

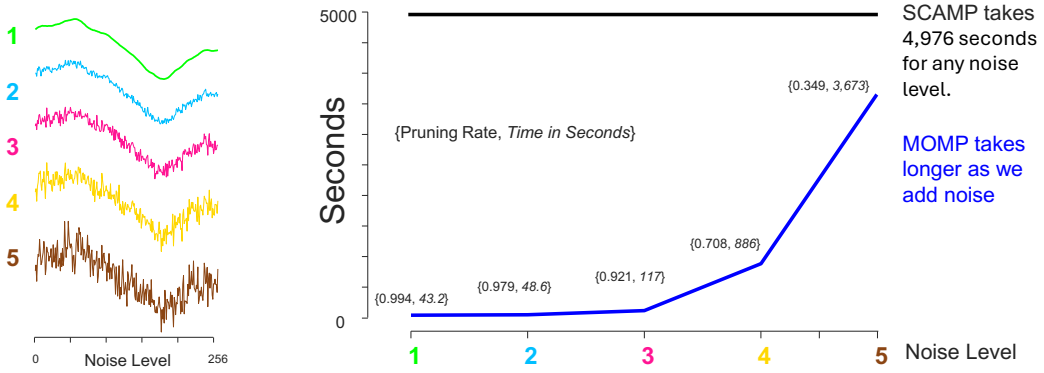

Fig. 30 left) The first 256 datapoints of the increasingly noisy datasets considered. right) The time for SCAMP is independent of noise level. However, while MOMP is initially two orders of magnitude faster, it slows down in the face of increasing noise.

We can see that MOMP is impressively fast, about 115 times faster than a direct use of SCAMP. However, as we add noise to the dataset, MOMP begins to slow down. This is to be expected. In the limit, a very noisy dataset puts us in the "*nothing is a motif*" situation explained in Section 4.4.4.

Now let us consider the raison d'etre for this experiment.

- Here it took 4,976 seconds for SCAMP to process $n = 2^{20}$ datapoints. How long would it take to process say $n = 654{,}321$? Using the formula in Appendix B we predict it would take 1,937 seconds. Empirically measuring it, it takes 1,939 seconds, less than 1% error.
- Consider the speed up obtained for the noisiest experiment. Our cost model predicts we should take 84.8% of the time for SCAMP. We actually took 73.8% of the time. This suggests our cost model is, if anything, a little conservative.

These observations tell us that we can report either the wall-clock time or just the cost model's predicted speedup. This is useful for two reasons. For some of the experiments we wish to do, SCAMP would take years or longer, we clearly cannot wait that long. Secondly, the speed up numbers are independent of the MP algorithm, the quality and optimization of the algorithm, and the hardware used. We use the most appropriate of the two measures on a case-by-case basis below, all measures are archived at [31].

4.7 MOMP's Speed Up on Diverse Datasets

In Table 15 we evaluate MOMP on a collection of diverse datasets from science, industry and medicine. The times for MOMP, and the times for SCAMP that are less than 12 hours are measured wall-clock times. Longer times are extrapolated as discussed in Section 4.6.

Table 15: SCAMP vs. MOMP on various datasets.

Name	Length *n*	Time SCAMP	Time MOMP	SpeedUp
EOG400 Hz (Fig. 32)	1,439,997	2.03 hours	0.03 hours	60.7
SWaT Attack 7	449,919	12.0 minutes	2.3 minutes	5.0
$\text{Human}_{\text{Y-chromosome}}$ [6][16]	26,415,043	28.6 days	0.11 days	259.3
EEG P_2O_2	2,472,001	6.1 hours	1.75 hours	3.3
SleepElectromyography	5,983,000	35.5 hours	0.85 hours	41.1
EOG (during sleep study)	8,490,000	71.05 hours	4.82 hours	14.7
Respiration (Challenge)	1,799,997	3.19 hours	0.15 hours	20.4
Insect $\text{EPG}_{\textit{Kryder trifoliata}}$	7,583,000	2.36 days	0.64 days	3.6
Insect $\text{EPG}_{\textit{Poncirus trifoliata}}$	7,583,000	2.36 days	0.033 days	231.5
Chicken Behavior	8,595,817	3.02 days	0.107 days	28.1
Kittiwake (Flying wild bird)	1,288,330	1.63 hours	0.11 hours	13.6
HAR Ambient Sensor	1,875,227	3.27 hours	0.098 hours	35.4
Electroencephalography	6,375,000	1.67 days	0.01 days	144.3
WaterDemand	2,100,777	4.35 hours	0.131 hours	32.8
Micro PMU	62,208,000	158.9 days	0.19 days	800.3
Stator Winding	1,330,816	1.74 hours	0.033 hours	52.4
Household electrical demand	5,153,051	1.09 days	0.0087 days	125.6
Wind Turbine	5,231,008	1.12 days	0.068 days	16.4
NASA SolarWind	8,066,432	2.67 days	0.012 days	220.0
ECoG (Finger Flexion)	23,999,997	23.65 days	0.126 days	188.2
SpanishEnergy	25,232,401	26.15 days	0.51 days	51.3
EEG CinC (Fig. 33)	1,593,750	2.50 hours	0.013 hours	181.2
WearablesWetLab (Fig. 35)	4,031,969	16.0 hours	0.097 hours	165.4
Physical Activity in Youth NIH	4,741,248	22.2 hours	0.17 hours	123.8
OSA Sleep Apnea	8,340,000	68.3 hours	2.2 hours	30.9
Random Walk	67,108,864	164.64 days	0.064 days	2565.8

While *negative* speedups are logically possible (see Fig. 29), none are observed. The speedups range from 3.3 (EEG P_2O_2) to 800.3 (Micro PMU) with a mean of 117.5. It is difficult to see any obvious predictors of speedup (beyond the factors discussed in Section 4.4.4). The two insect EPG datasets are not visually distinct yet have dramatically different speedups. It is only with a *post-hoc* analysis we see that Insect EPG$_{\textit{Poncirus trifoliata}}$ happened to have a better conversed motif, allowing for more aggressive pruning.

We do not have space to provide detailed provenance for each dataset, but this information is archived at [31]. However, in most cases the motifs make intuitive sense the domain experts we asked to review the findings. For chicken behavior dataset the motif reflects *dustbathing*. For small values of *m*, *pecking* is the most conspicuous and frequent motif in chickens. However, here we take advantage of the scalability of MOMP to look for much longer motifs in 24-hour period. Healthy chickens normally only dustbathe every two days [34], so this discovery of two bouts in a single day offers evidence of possible infestation by parasitic mites.

For the Kittiwake dataset, the motif reflects the bird (*Rissa tridactyla*) transitioning from gliding behavior to flapping flight. For EOG data, the motif represents corneal reflex blinks. For WearablesWetLab, the act of *pouring*.

In Fig. 31 we show representative examples of the motifs found during these experiments.

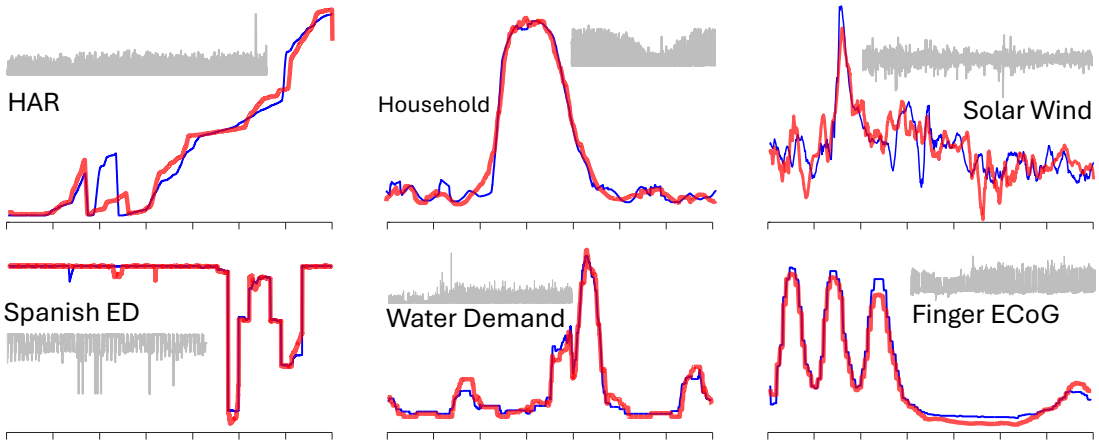

Fig. 31 Six representative motifs found during the experiments described in Table 15. Inset in gray are the entire original datasets.

Note that we deliberately included one tiny dataset, SWaT-Attack-7 in Table 15, as it is the only dataset Quick-Motif could process without running out of memory [23]. When we use the original m = 16,384, Quick-Motif *still* ran out of memory[6]. But by reducing m to 4,096 we got it to work. Quick-Motif was about seventeen times slower than SCAMP. Quick-Motif has three parameters w/l, λ and ε, it is possible that better settings of these parameters could close the time gap, but its memory footprint issue seems insurmountable.

4.8 Searching The Entire Human Genome

In Table 15 we converted the Human Y chromosome (H_Y) to a time series using the algorithm in Appendix A andsearched it for motifs. This dataset's length is 26,415,043 bp (base pairs), and it was used as a capstone experiment in two papers that could only searched it *approximately* [6][16]. Note that it is by far the smallest of the 23 human chromosomes.

To demonstrate the scalability of MOMP we consider an experiment that would be otherwise untenable. We began by converting the entire human genome (2,727,047,427 bp) into 23 time series representing all the human chromosomes. We searched H_1 for the best motif of length 65,536. We now can ask the following question. *Which of the remaining 22 chromosomes have a motif as well conserved as H_1*?

To answer this question, we simply placed MOMP in a loop that ran 22 times. The only difference to normal MOMP is that instead of initializing the *bsf* in line 3 of Table 10, we initialize a global *bsf* outside of MOMP, and pass that value into MOMP. We can the exploit the anytime algorithm property of MOMP by stopping the search of a chromosome, if we find a qualifying small motif distance.

The largest chromosome is H_2 with a n = 238,357,386. To find the exact motif in just H_2, we must compare or prune 25,734,779,520,915,612 candidate subsequence pairs of length 65,536. If each comparison took one microsecond, this would require 815 years. SCAMP cleverly reduces the factor in m to just O(1), and thus takes only 5.07 years. Here MOMP did a full search, taking 1.08 hours, about 41,000 times faster.

In five of the twenty-two Chromosomes *anytime* MOMP's early abandoning did work. For example, on H_9 (n = 121,526,601) anytime MOMP found a qualifying motif in 12.4 minutes, whereas SCAMP takes 1.32 years.

Note that the H_1 motif does not have zero distance, as zero-distance motifs can be found in linear time. Consider these excerpts we mapped into the original DNA string:

Motif-1 (103,286,375 bp omitted) ..TGGCACAATGTCAC..

|||||| |||||||

Motif-2 (103,426,107 bp omitted) ..TGGCAC-ATGTCAC..

We can see that there are small local micromutations (either the insertion of **A** into Motif-1 *or* the deletion of **A** from Motif-2) that happened after the transposition event. By counting these micromutations it is possible to approximately predict when the transposition event occurred.

4.9 AB Join MOMP

The genomes of the Human and Chimp are nearly identical in structure, except for five large-scale inversions (and one chromosome fusion, which we ignore here). Let us use a *join* to find one such large-scale inversion. Here we follow the notation of the original MP paper [50], and generalize the MP to consider $J_{\mathbf{AB}}$, which is simply a motif consisting of the subsequence in **B**, that is closest to *some* subsequence in **A**. We denote this generalization $J_{\mathbf{AB}}$-MOMP.

A DNA inversion appears as a time series subsequence reversed in time. So, if the i^{th} chromosome does not have an inversion, we should expect that the Euclidean distance of the Top-1 join-motif between the Human's i^{th} chromosome (Hereafter H_i) and C_i is much smaller than the Top-1 join-motif distance between the H_i and Backwards (C_i), as the reversal will not reveal hidden structure.

If, however, the chromosome does have an inversion, we should expect the Euclidean distance of the Top-1 join-motif between the H_i and Backwards (C_i) to be about the same as the top-1 join-motif distance between original chromosomes. This is because our *digital* reversal undoes a *biological* inversion that took place sometime after the human chimp split six million years ago.

We first consider H_{21} and C_{21} which have lengths of size 35,186,393 and 32,724,802 respectively. Here the reversed search found a motif pair with a distance of 164.49, which is dramatically (and visually, see Fig. 32) further apart than the normal motif pairs distance of just 27.33.

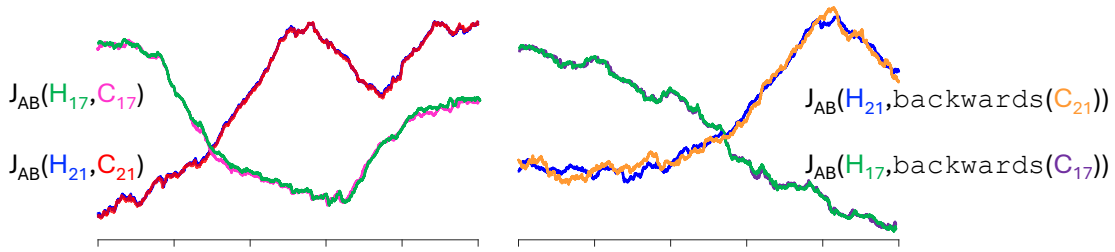

Fig. 32 Top-1 motifs pairs from the four AB-join experiments. The pair corresponding to the reversal of C_{21} is visibly less conserved than the others, looking no better conserved that random chance, suggesting that this chromosome is not the source of an inversion.

We then considered H_{17} and C_{17} which have lengths of size 79,910,446 and 81,665,723 respectively, finding that the reversed search found a motif pair with a distance 54.45, which is actually slightly less than the normal motif pairs distance of 76.87, strongly (and correctly, see [44]) indicating that this pair of chromosomes *has* a large-scale inversion.

The average pruning rate for all four joins was 0.934 which means that our experiments ran about 139 times faster than J_{AB}-SCAMP. Note that the pruning rates are not particularly high here because for most of the joins, we are in the "*everything is a motif*" situation as described in Section 4.8. Furthermore, note that the anytime convergence is so fast here,

that as a practical matter we could have answered the original question in a few minutes on a typical laptop.

This is not an original biological finding, and it could be done better/faster in the original DNA representation. Our example simply shows that MOMP allows accelerated time series similarity joins between massive datasets. To the best of our knowledge, these joins are, by two orders of magnitude, the largest time series joins ever attempted [50].

4.10 MOMP as an AnyTime Algorithm

Consider the 8.8 hour-long EEG CinC example in Table 15. As shown in Fig. 33.*top* the discovered motif is unusually well conserved and perfectly periodic, something we do not expect to see in real medical data. A few seconds of introspection by a medical technician would have them realize that the motif does not reflect a biological signal. After about 6.69 hours, the electrode patch that holds the sensor to the patient's skin became loose. When this happens the sensor does not "flatline", but instead emits a calibration signal, this is what this motif represents.

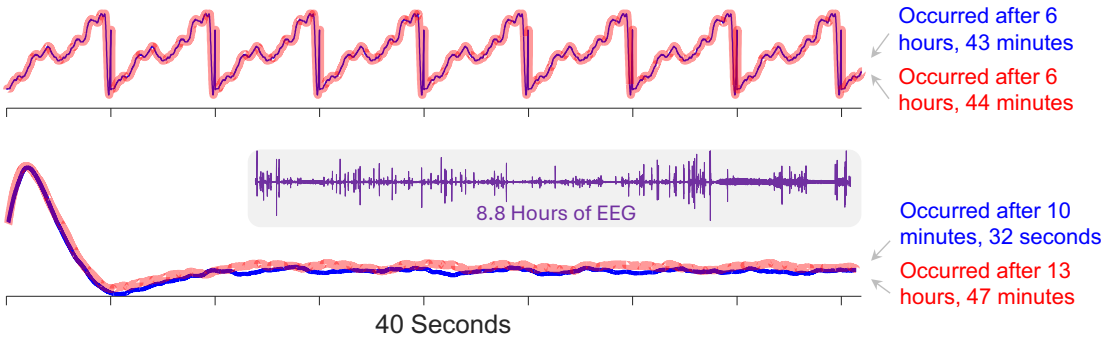

Fig. 33 top) The Top-1 motif discovered in the EEG CinC dataset is just a calibration signal. It looks too well conserved to be natural, given how noisy the data is (gray inset). After removing the calibration signal, the Top-1 motif is a biological signal (bottom).

When a user sees this, she can delete the calibration signals and rerun motif discovery. However, consider the three following scenarios. If we had used SCAMP to find this motif, this cycle would have taken 2.50 hours. Using MOMP as a batch algorithm, it would have only taken about 79 seconds. However, using MOMP as any anytime algorithm and visually inspecting the current *bsf* motif, we would have seen this calibration motif after about eight seconds.

This observation motivates the use of an anytime framework for motif discovery. Although the full algorithm may take minutes or hours to finish, often a visual inspection in the first few seconds will allow the user to stop because:

- It is clear that the motif that will be returned is pathological, as in the EEG case above, or
- The current best-so-far motif is already interesting enough to warrant investigation that there is no point in continuing, or at least the current run can be run in the background while the user examines the current motif.

A useful property of MOMP is that it is an *anytime* algorithm, even if the underlying MP algorithm is not. To see this, let us conduct an experiment. We created a random walk of length 2^{20} and using SCAMP (a *batch-only* algorithm) we measured how long it takes to find the Top-1 motif $m = 2^{12}$. Fig. 34 shows that this takes 34.7 minutes.

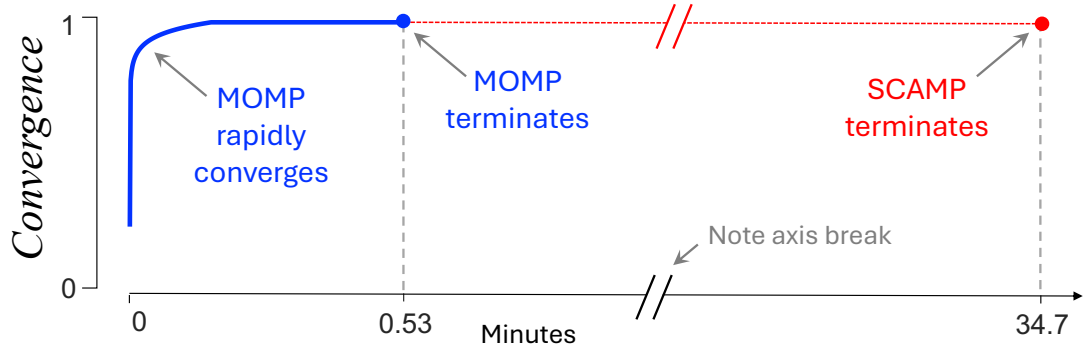

Fig. 34 The time needed for SCAMP and MOMP to find the Top-1 Motif. For MOMP, we also plot its anytime convergence rate.

We repeated the experiment with MOMP, this time also measuring how fast it converges. This *convergence* is the normalized difference between the final motif distance and the *bsf* distance recorded each time through the loop in line 8 of Table 2. We defined a score to evaluate the convergence rate. The score is:

$$Convergence = 1 - \frac{currentBSF - finalBSF}{initialBSF}$$

Here we define *initialBSF* as the average distance between two random subsequences (empirically determined). Because MOMP is data dependent, we averaged over one

hundred runs. As Fig. 34 shows MOMP is an ideal anytime algorithm, quickly converging on high-quality motifs.

4.11 The Utility of Motif Discovery

While the utility of motif discovery as a subroutine in downstream analysis is well established, for completeness we will show a concrete example. Suppose we wish to annotate a long video. If we have a companion time series, it may be easier to find motifs in the time series, and visually check if the corresponding video snippets show semantically similar behaviors. As shown in Fig. 35, this idea does bear fruit.

Fig. 35 left) We searched for motifs in time series recorded in parallel to a 40.5 minute video showing benchwork in a wet lab. right) The motifs do correspond to a repeated behavior, pouring.

Here MOMP took 5.8 minutes, much faster than real time. Our cost model predicts SCAMP would take 16.0 hours. We took advantage of the relatively small data size to run SCAMP to competition, finding it took 15.8 hours.

Similarly, for a doctor to annotate a sleep study, they must examine about eight hours of sleep telemetry. Working with Dr. <blinded> we performed a simple experiment to see if motif discovery could reduce this labeling burden. The task was to annotate a dataset for an Obstructive Sleep Apnea (OSA) study. OSA is characterized by "*repetitive episodes of intermittent hypoxemia*" [9], and of course "repetitive" strongly suggests "motifs". As shown in Fig. 36 we searched a seven-hour respiration dataset for motifs of length 80 seconds (the length was suggested by Dr. <blinded>).

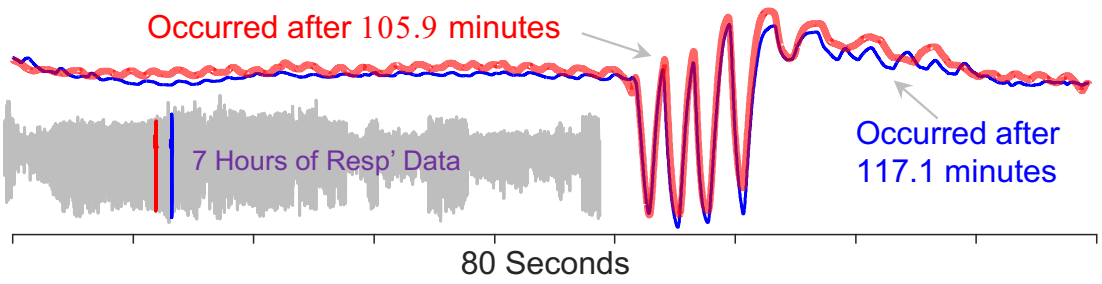

Fig. 36 This motif corresponds to "cycles of intermittent hypoxemia with cyclical desaturation-reoxygenation". inset) the full seven-hour respiration dataset.

Having found a motif of interest, we can simply search for other occurrences to help label the dataset.

Here, MOMP took 2.21 hours, which is faster than real time. Our cost model predicts SCAMP would take 68.5 hours. We again took advantage of the small data size to run SCAMP to completion, finding it actually took 68.3 hours.

4.12 Conclusions

We introduced lbMP, the first lower bound to the Matrix Profile, and we further introduced MOMP, a parameter-free algorithm that exploits lbMP to accelerate exact motif discovery. We have shown that MOMP accelerates motif discovery by up to two orders of magnitude. There are many opportunities for future work. In the past motif discovery was always compute-bound. The use of MOMP is bringing us close to the point were dealing with secondary memory will become an interesting research challenge. In addition, precedent in time series similarity search, suggests that once a lower-bounding framework for an important problem is introduced, the community is very creative in inventing increasingly tighter lower bounds [22].

5 FUTURE WORK

The research presented in this thesis opens several exciting avenues for future work. Below, we outline potential directions for extending the contributions made by SPLAT and MOMP, as well as addressing new challenges and opportunities that have emerged from our findings.

1. **Generalization of SPLAT:**
    - **Introduction of New Patterns:** While SPLAT has demonstrated its utility across diverse domains, there is significant potential to generalize its applicability further. This can be achieved by introducing new patterns that cater to specific use cases in various fields such as climatology, genomics, and social network analysis. Developing a library of such patterns will enhance SPLAT's versatility and impact.
    - **Exact Computation on GPUs:** Currently, SPLAT leverages GPU computing for near-approximate results. Future research should focus on adapting SPLAT to utilize GPU acceleration while maintaining exact computation. This would involve optimizing algorithms for parallel processing, reducing approximation errors, and ensuring computational integrity.
    - **Integration with Real-Time Data Streams:** Extending SPLAT to handle real-time data streams can greatly enhance its applicability, particularly in domains like finance, environmental monitoring, and online behavior analysis. This would require the development of robust algorithms that can process and analyze data in real-time without compromising accuracy.
    - **Collaborative Filtering and Recommendation Systems:** Leveraging SPLAT for collaborative filtering can improve recommendation systems by identifying patterns and similarities in user behavior, preferences, and interactions. This application can enhance the performance of e-commerce platforms, social networks, and content delivery services.
2. **Enhancement of MOMP and Matrix Profile Algorithms:**
    - **Memory Management and Optimization:** As MOMP significantly accelerates motif discovery, the next challenge lies in managing memory efficiently. Research should be directed towards developing innovative memory management techniques and optimizing data structures to handle large datasets without compromising performance. Exploring hybrid

storage solutions that balance between RAM and secondary memory could prove beneficial.

- **Tighter Lower Bounds:** Building on the introduction of lbMP, future work should aim to develop tighter lower bounds for Matrix Profile calculations. Drawing inspiration from the community's creativity in time series similarity search, researchers can devise new algorithms that further enhance the efficiency and accuracy of motif discovery. These improvements could unlock new applications and deeper insights from time series data.
- **Scalability to Ultra-High Dimensional Data:** Extending MOMP to handle ultra-high dimensional datasets will broaden its application scope, particularly in fields like bioinformatics, neuroimaging, and sensor networks. This involves addressing the curse of dimensionality through advanced dimensionality reduction techniques and parallel processing.
- **Adaptive and Incremental Learning:** Developing adaptive and incremental versions of MOMP that can learn from new data without retraining from scratch will enhance its efficiency and applicability. This can be particularly useful in dynamic environments where data evolves over time, such as stock market analysis and adaptive control systems.

3. **Human-Centric Enhancements:**
   - **Patchwise Search Optimization:** Given that human attention has become the primary bottleneck in analyzing massive Mplots, enhancing patchwise search methodologies is crucial. Future work should focus on refining these search techniques to ensure that the most relevant and insightful patches are identified quickly. This can involve integrating machine learning models to predict user interests and streamline the search process.
   - **User Interface and Experience:** Developing intuitive user interfaces that facilitate interaction with large Mplots and enable efficient patchwise searches is essential. Incorporating visual analytics and interactive

exploration tools will empower users to extract meaningful insights with minimal effort.

- **Personalized Visualization and Reporting:** Customizing visualization and reporting tools to align with user preferences and expertise levels can significantly improve user engagement and decision-making. This includes developing dashboards that highlight key findings and trends relevant to specific users or applications.

4. **Exploration of Secondary Memory Challenges:**
    - **Addressing Secondary Memory Bottlenecks:** As MOMP brings motif discovery close to the limits of compute-bound constraints, dealing with secondary memory becomes a pertinent research challenge. Future work should explore advanced storage solutions, such as distributed file systems, cloud-based storage, and memory-mapped file techniques, to handle large-scale data efficiently.
    - **Efficient Data Retrieval and Indexing:** Improving data retrieval and indexing mechanisms to minimize latency and maximize throughput when accessing large datasets stored in secondary memory is critical. This could involve developing new indexing algorithms and optimizing existing ones to ensure rapid data access and retrieval.
    - **Fault Tolerance and Data Integrity:** Ensuring fault tolerance and data integrity in secondary memory storage solutions is vital, particularly for applications requiring high reliability and accuracy. Future research should focus on developing robust error detection and correction mechanisms, as well as backup and recovery strategies to safeguard data integrity.

By pursuing these future directions, the research community can build upon the foundational work presented in this thesis, further advancing the fields of time series analysis, motif discovery, and scalable algorithm design. The continued evolution of SPLAT and MOMP will undoubtedly contribute to solving increasingly complex problems across a wide range of scientific and industrial domains.

REFERENCES

[1] L. C. S. Afonso *et al.*, "A recurrence plot-based approach for Parkinson's disease identification," *Future Generation Computer Systems*, vol. 94, pp. 282–292, 2019, doi: https://doi.org/10.1016/j.future.2018.11.054.

[2] S. Alaee, R. Mercer, K. Kamgar, and E. Keogh, "Time series motifs discovery under DTW allows more robust discovery of conserved structure," *Data Min Knowl Discov*, vol. 35, pp. 1–48, Mar. 2021, doi: 10.1007/s10618-021-00740-0.

[3] A. F. Almeida-Ñauñay, R. M. Benito, M. Quemada, J. C. Losada, and A. M. Tarquis, "Recurrence plots for quantifying the vegetation indices dynamics in a semi-arid grassland," *Geoderma*, vol. 406, p. 115488, 2022, doi: https://doi.org/10.1016/j.geoderma.2021.115488.

[4] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft, "When Is 'Nearest Neighbor' Meaningful?," *ICDT 1999*, LNCS, vol. 1540.

[5] J. P. Bonani, A. Fereres, E. Garzo, M. Miranda, B. Appezzato-da-Glória, and J. Lopes, "Characterization of electrical penetration graphs of the Asian citrus psyllid, Diaphorina citri, in sweet orange seedlings," *Entomol Exp Appl*, vol. 134, pp. 35–49, Mar. 2009, doi: 10.1111/j.1570-7458.2009.00937.x.

[6] M. Ceccarello and J. Gamper, "Fast and Scalable Mining of Time Series Motifs with Probabilistic Guarantees," *Proceedings of the VLDB Endowment*, vol. 15, pp. 3841–3853, May 2023.

[7] Q. Chesnais and K. E. Mauck, "Choice of Tethering Material Influences the Magnitude and Significance of Treatment Effects in Whitefly Electrical Penetration Graph Recordings," *J Insect Behav*, vol. 31, no. 6, pp. 656–671, 2018, doi: 10.1007/s10905-018-9705-x.

[8] B. Chiu, E. Keogh, and S. Lonardi, "Probabilistic discovery of time series motifs," in Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, 2003, pp. 493-498, doi: 10.1145/956750.956808.

[9] N. Dewan, F. Nieto, V. Somers, “Intermittent hypoxemia and OSA: implications for comorbidities,” *Chest*, vol. 147, no. 1, pp. 266-274, Jan. 2015.

[10] R. O. Duda and P. E. Hart, “Use of the Hough Transformation to Detect Lines and Curves in Pictures,” *Commun. ACM*, vol. 15, no. 1, pp. 11–15, Jan. 1972, doi: 10.1145/361237.361242.

[11] J.-P. Eckmann, S. O. Kamphorst, and D. Ruelle, “Recurrence Plots of Dynamical Systems,” *Europhysics Letters (EPL)*, vol. 4, no. 9, pp. 973–977, Nov. 1987, doi: 10.1209/0295-5075/4/9/004.

[12] Edanusa-X12 Tech Sheet, *Edan USA*, Apr. 29, 2024. [Online]. Available: https://edanusa.com/wp-content/uploads/sites/D12/2020/11/x8-x12-PM-spec-sheet.pdf

[13] Y. Fang, H. Xu, and J. Jiang, “A Survey of Time Series Data Visualization Research,” *IOP Conf Ser Mater Sci Eng*, vol. 782, no. 2, p. 22013, Mar. 2020, doi: 10.1088/1757-899x/782/2/022013.

[14] J. Foote and M. Cooper, “Media Segmentation using Self-Similarity Decomposition,” *Proceedings of SPIE - The International Society for Optical Engineering*, vol. 5021, Jun. 2002, doi: 10.1117/12.476302.

[15] M. Fukino, Y. Hirata, and K. Aihara, “Coarse-graining time series data: Recurrence plot of recurrence plots and its application for music,” *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 26, no. 2, p. 23116, 2016, doi: 10.1063/1.4941371.

[16] Y. Gao and J. Lin, “HIME: discovering variable-length motifs in large-scale time series,” *Knowledge and Information Systems*, vol. 61, May 2019.

[17] J. George *et al.*, “Feeding Behavior of Asian Citrus Psyllid [Diaphorina citri (Hemiptera: Liviidae)] Nymphs and Adults on Common Weeds Occurring in Cultivated Citrus Described Using Electrical Penetration Graph Recordings,” *Insects*, vol. 11, no. 1, 2020, doi: 10.3390/insects11010048.

[18] S. Gharghabi, Y. Ding, C.-C. M. Yeh, K. Kamgar, L. Ulanova, and E. Keogh, “Matrix Profile VIII: Domain Agnostic Online Semantic Segmentation at Superhuman Performance Levels,” Jun. 2017, pp. 117–126. doi: 10.1109/ICDM.2017.21.

[19] A. J. Gibbs, A. J. Gibbs, and G. A. McIntyre, “The diagram, a method for comparing sequences: its use with amino acid and nucleotide sequences,” *Eur J Biochem*, vol. v. 16, no. 1, pp. 1-11–1970 v.16 no.1, 1970, doi: 10.1111/j.1432-1033.1970.tb01046.x.

[20] R. E. Green *et al.*, “A Complete Neandertal Mitochondrial Genome Sequence Determined by High-Throughput Sequencing,” *Cell*, vol. 134, pp. 416–426, 2008.

[21] M. Imamura, T. Nakamura, and E. Keogh, “Matrix Profile XXI: A Geometric Approach to Time Series Chains Improves Robustness,” in *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020, pp. 1114–1122. doi: 10.1145/3394486.3403164.

[22] E. Keogh, K. Chakrabarti, M. Pazzani, and S. Mehrotra, “Dimensionality Reduction for Fast Similarity Search in Large Time Series Databases,” *Knowl Inf Syst*, vol. 3, no. 3, pp. 263–286, 2001, doi: 10.1007/PL00011669.

[23] Y. Li et al., “Quick-motif: An efficient and scalable framework for exact motif discovery,” in *Proceedings of the IEEE International Conference on Data Engineering (ICDE)*, 2015, pp. 579–590.

[24] J. Lin, E. Keogh, S. Lonardi, J. Lankford, and D. Nystrom, “VizTree: a Tool for Visually Mining and Monitoring Massive Time Series Databases.,” in *Proceedings of International Conference on Very Large Data Bases*, Mar. 2004, pp. 1269–1272. doi: 10.1016/B978-012088469-8/50124-8.

[25] M. A. Lopes *et al.*, “Recurrence quantification analysis of dynamic brain networks,” *European Journal of Neuroscience*, vol. 53, no. 4, pp. 1040–1059, 2021, doi: https://doi.org/10.1111/ejn.14960.

[26] F. Malige, D. Djokić, J. Patris, R. Sousa-Lima, and H. Glotin, “Use of recurrence plots for identification and extraction of patterns in humpback whale song recordings,” *Bioacoustics*, Mar. 2020, doi: 10.1080/09524622.2020.1845240.

[27] X. Marimon, S. Traserra, M. Jiménez, A. Ospina, and R. Benítez, “Detection of Abnormal Cardiac Response Patterns in Cardiac Tissue Using Deep Learning,” *Mathematics*, vol. 10, no. 15, 2022, doi: 10.3390/math10152786.

[28] N. Marwan, M. Carmen Romano, M. Thiel, and J. Kurths, “Recurrence plots for the analysis of complex systems,” *Phys Rep*, vol. 438, no. 5, pp. 237–329, 2007, doi: https://doi.org/10.1016/j.physrep.2006.11.001.

[29] R. Mercer, S. Alaee, A. Abdoli, S. Singh, A. Murillo, and E. Keogh, “Matrix Profile XXIII: Contrast Profile: A Novel Time Series Primitive that Allows Real World Classification,” in *2021 IEEE International Conference on Data Mining (ICDM)*, 2021, pp. 1240–1245. doi: 10.1109/ICDM51629.2021.00151.

[30] R. Mercer and E. Keogh, “Matrix Profile XXV: Introducing Novelets: A Primitive that Allows Online Detection of Emerging Behaviors in Time Series,” in *2022 IEEE International Conference on Data Mining (ICDM)*, 2022, pp. 338–347. doi: 10.1109/ICDM54844.2022.00044.

[31] MOMP, “MOMP,” Jun. 01, 2024. [Online]. Available: https://sites.google.com/view/momp2024

[32] A. Mueen *et al.*, “The Fastest Similarity Search Algorithm for Time Series Subsequences under Euclidean Distance,” *http://www.cs.unm.edu/~mueen/FastestSimilaritySearch.html*, Aug. 2017.

[33] V. Nalam, J. Louis, M. Patel, and J. Shah, “Arabidopsis-Green Peach Aphid Interaction: Rearing the Insect, No-choice and Fecundity Assays, and Electrical Penetration Graph Technique to Study Insect Feeding Behavior,” *Bio Protoc*, vol. 8, Feb. 2018, doi: 10.21769/BioProtoc.2950.

[34] A. Olsson and L. Keeling, “Why in earth? Dustbathing behaviour in jungle and domestic fowl reviewed from an Animal Welfare perspective,” *Applied Animal Behaviour Science*, vol. 93, pp. 259–282, May 2005.

[35] R. ~A. Phillipson, “Complex Long-Term Variability of X-ray Binaries and Active Galaxies Revealed by Novel Methods,” in *American Astronomical Society Meeting Abstracts #236*, Jun. 2020, vol. 236, p. 122.02.

[36] A. Raoofy, R. Karlstetter, M. Schreiber, C. Trinitis, and M. Schulz, “Overcoming Weak Scaling Challenges in Tree-Based Nearest Neighbor Time Series Mining,” in

Proceedings of the 2023 IEEE International Conference on Big Data (Big Data), 2023, pp. 317–338.

[37] T. Rakthanmanon *et al.*, "Addressing Big Data Time Series: Mining Trillions of Time Series Subsequences Under Dynamic Time Warping," in *ACM Transactions on Knowledge Discovery from Data (TKDD)*, Mar. 2013, vol. 7. doi: 10.1145/2500489.

[38] J. M. Ratcliffe, C. P. H. Elemans, L. Jakobsen, and A. Surlykke, "How the bat got its buzz," *Biol Lett*, vol. 9, 2013.

[39] T. Rawald, M. Sips, and N. Marwan, "PyRQA—Conducting recurrence quantification analysis on very long time series efficiently," *Comput Geosci*, vol. 104, pp. 101–108, 2017, doi: https://doi.org/10.1016/j.cageo.2016.11.016.

[40] P. Schäfer and U. Leser, "Motiflets: Simple and Accurate Detection of Motifs in Time Series," *Proceedings of the VLDB Endowment*, vol. 16, pp. 725–737, May 2023, doi: 10.14778/3574245.3574257.

[41] M. Shahcheraghi, "mplot," *https://sites.google.com/view/mplot/*, 2022. https://sites.google.com/view/mplot/ (accessed Mar. 06, 2023).

[42] B. Shneiderman, "The eyes have it: a task by data type taxonomy for information visualizations," in *Proceedings 1996 IEEE Symposium on Visual Languages*, 1996, pp. 336–343. doi: 10.1109/VL.1996.545307.

[43] V. N. Soloviev, O. Serdiuk, S. O. Semerikov, and A. E. Kiv, "Recurrence plot-based analysis of financial-economic crashes," 2020.

[44] M. Suntsova and A. Buzdin, "Differences between human and chimpanzee genomes," *BMC Genomics*, vol. 21, Suppl. 7, 535, 2020.

[45] M. Tabak, K. Murray, J. Lombardi, and K. Bay, "Automated classification of bat echolocation call recordings with artificial intelligence." Jun. 2021. doi: 10.1101/2021.06.23.449619.

[46] I. Takakura *et al.*, "Recurrence Plots: a New Tool for Quantification of Cardiac Autonomic Nervous System Recovery after Transplant," *Braz J Cardiovasc Surg*, vol. 32, Mar. 2017, doi: 10.21470/1678-9741-2016-0035.

[47] C. L. Webber and J. P. Zbilut, “Dynamical assessment of physiological systems and states using recurrence plot strategies,” *J Appl Physiol*, vol. 76, no. 2, pp. 965–973, 1994, doi: 10.1152/jappl.1994.76.2.965.

[48] D. Willett, J. George, N. Willett, L. Stelinski, and S. Lapointe, “Machine Learning for Characterization of Insect Vector Feeding,” *PLoS Comput Biol*, vol. 12, p. e1005158, Mar. 2016, doi: 10.1371/journal.pcbi.1005158.

[49] D. Yankov, E. Keogh, J. Medina, B. Chiu, and V. Zordan, “Detecting time series motifs under uniform scaling,” in *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Apr. 2007, pp. 844–853. doi: 10.1145/1281192.1281282.

[50] C.-C. M. Yeh *et al.*, “Matrix Profile I: All Pairs Similarity Joins for Time Series: A Unifying View That Includes Motifs, Discords and Shapelets,” Mar. 2016, pp. 1317–1322. doi: 10.1109/ICDM.2016.0179.

[51] C.-C. M. Yeh et al., “Time series joins, motifs, discords and shapelets: a unifying view that exploits the matrix profile,” *Data Mining and Knowledge Discovery*, vol. 32, May 2018, doi: 10.1007/s10618-017-0519-9.

[52] Y. Zhang, Y. Hou, K. OuYang, and S. Zhou, “Multi-scale signed recurrence plot based time series classification using inception architectural networks,” *Pattern Recognit*, vol. 123, p. 108385, 2022, doi: https://doi.org/10.1016/j.patcog.2021.108385.

[53] X.-C. Zhu, D.-H. Zhao, Y.-H. Zhang, X.-J. Zhang, and Z. Tao, “Multi-Scale Recurrence Quantification Measurements for Voice Disorder Detection,” *Applied Sciences*, vol. 12, no. 18, 2022, doi: 10.3390/app12189196.

[54] 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,” in *2016 IEEE 16th International Conference on Data Mining (ICDM)*, 2016, pp. 739–748. doi: 10.1109/ICDM.2016.0085.

[55] S. Zilberstein, “Optimizing Decision Quality with Contract Algorithms,” in *Proceedings of the 14th International Joint Conference on Artificial Intelligence - Volume 2*, 1995, pp. 1576–1582.

[56] Z. Zimmerman *et al.*, “Matrix Profile XIV: Scaling Time Series Motif Discovery with GPUs to Break a Quintillion Pairwise Comparisons a Day and Beyond,” in *Proceedings of the ACM Symposium on Cloud Computing*, 2019, pp. 74–86. doi: 10.1145/3357223.3362721.