Title

Introducing Mplots: scaling time series recurrence plots to massive datasets

Permalink

https://escholarship.org/uc/item/38z3g689

Journal

Journal of Big Data, 11(1)

ISSN

2196-1115

Authors

Shahcheraghi, Maryam

Mercer, Ryan

Rodrigues, João Manuel de Almeida

et al.

Publication Date

2024

DOI

10.1186/s40537-024-00954-1

Peer reviewed

Journal of Big Data

RESEARCH Open Access

Introducing Mplots: scaling time series recurrence plots to massive datasets

Maryam Shahcheraghi[1*], Ryan Mercer[1], João Manuel de Almeida Rodrigues[2], Audrey Der[1], Hugo Filipe Silveira Gamboa[2], Zachary Zimmerman[1], Kerry Mauck[1] and Eamonn Keogh[1]

*Correspondence:
sshah073@ucr.edu

[1] University of California, Riverside, USA
[2] Libphys-UNL NOVA Faculty of Science and Technology, Lisbon, Portugal

Abstract

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 work, 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 will demonstrate the utility of our ideas for data exploration, segmentation, and classification in domains as diverse as astronomy, bioinformatics, entomology, and wildlife monitoring.

Keywords: Time series, Anomalies, Similarity matrix

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 [40, 43]. 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. [7, 22]. 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

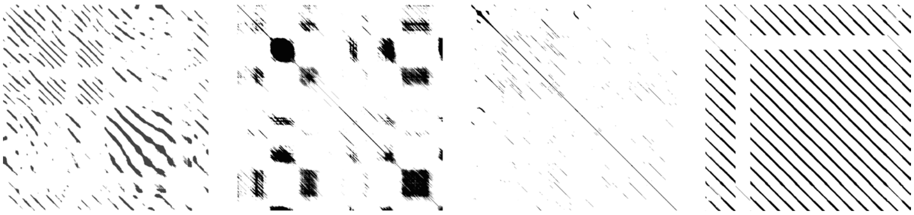

Fig. 1 Examples of Mplots hint at the diversity and utility of this data structure. Like many plots in the paper, these figures suffer somewhat from the size of reproduction. We encourage the interested reader to visit [31] which has larger figures and videos. Here we show binarized Mplots, but more generally Mplots allow a spectrum of colors to indicate degree of similarity

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 [40, 43, 45].

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 [27], economics [34], music [10], physiology [36, 37], neuroscience [19], earth sciences [3], medicine [1, 42] and engineering [22]. As noted in a founding paper on the topic, "information obtained from recurrence plots is often surprising, and not easily obtainable by other methods" [7].

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 [7, 20, 22, 27].
- 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 Sect. "Searching Massive Mplots" 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 > 1000, 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 rest of this paper is organized as follows. In Sect. "Definitions and notation" we review the definitions that are required to comprehend the techniques explained in this paper. After that we clarify the main differences between Mplots and true recurrence plots in Sect. "Related work", along with a discussion of related work. Our proposed techniques are described in Sect. "Algorithms that scale up Mplots". To help develop the readers skill in the interpretation of the Mplot patterns, we present annotated examples in Sect. "Interpreting Mplots". Section "Interpreting Mplots: reverse engineered" explains how you can "reverse engineer" Mplot interpretations to automate the discovery of hypothesized structure in the time series. We present a detailed empirical analysis on diverse datasets in Sect. "Experimental evaluation". Finally, in Sect. "Conclusions" we offer conclusions and directions for future work.

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 $\mathbf{T}$ 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 [40]. Note the subsequence length here takes on a similar role to the embedding dimension in discrete dot plots [20, 22]. 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 $\mathbf{T}_{\mathrm{B}}^{(j,m)}$.

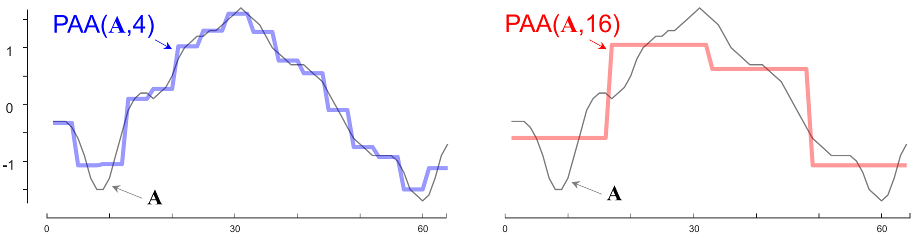

Fig. 2 A time series A with n = 64 downsampled with PAA. left) Downsampled 1 in 4, right) Downsampled 1 in 16. While the 1 in 16 plot has lost significant detail, the 1 in 4 downsampling does preserve the basic shape of the time series

The distance can be computed very efficiently using the MASS algorithm [25]. 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 [17]. 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.

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 $\mathbf{A} = \mathbf{B}$, this definition is logically equivalent to a self-join of **A**. The popular Matrix Profile is simply the vector of length $|\mathbf{A}|$ that contains the minimum (non-diagonal) value in each column [40, 43, 45]. 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 $\mathbf{A} \neq \mathbf{B}$, 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.

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. [13]. 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 1000. Moreover, these vectors need to be normalized before being compared (surprisingly, normalization generally takes longer than the distance computation [28]). 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 [8] and the references therein. However, Mplots are particularly direct and intuitive. Moreover, unlike say Viz-trees [18], 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.

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.

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($m \times n^2$) [22, 27]. As we show in Sect. "Experimental evaluation", 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 distance calculation [28]. In this section we

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

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  $\mu, \sigma \leftarrow$ computeMeanStd($T_A$, $T_B$, $m$)          //see (Rakthanmanon et al. 2013)
3  for $diag$ = 1 to Mplot_column_count
4    for $row$ = 1 to Mplot_row_count – $diag$ + 1
5      $col \leftarrow row + diag - 1$
6      $QT_{row,col} \leftarrow$ ComputeDotProduct($T_A^{(col,m)}$, $T_B^{(row,m)}$)    //see (Zhu et al. 2016)
7      d ← CalculateDistanceProfile($T_A^{(col,m)}$, $T_B^{(row,m)}$)     //see (1)
8      Mplot($row$, $row + diag$ - 1) = d
9  return Mplot
```

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 [43, 45]. Let us consider the formula for calculating a cell of distance profile ($d^{(i,j)}$).

$$\boldsymbol{d}^{(i,j)} = \sqrt{2\boldsymbol{m}\left(1 - \frac{\boldsymbol{QT}^{(i,j)} - \boldsymbol{m}\boldsymbol{\mu}^{i}\boldsymbol{\mu}^{j}}{\boldsymbol{m}\boldsymbol{\sigma}^{i}\boldsymbol{\sigma}^{j}}\right)} \tag{1}$$

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 $\mathrm{T}^{(i,m)}$, 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.

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 [28, 43] and [45] 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 × 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 [44]. For example, in SPLAT($\mathbf{A}$,$\mathbf{B}$,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 $|\mathbf{A}| = 10{,}000$, and found it took S seconds. We can then predict that building a Mplot for size time series $\mathbf{T}$, of length $n\prime$, will take $SplatTimePredict(\mathbf{T}, n\prime) = S * (n\prime/|\mathbf{A}|)^2$.

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 $\mathbf{T}$ to create $\mathbf{T}_{\mathrm{PAA}} = \mathrm{PAA}(\mathbf{T},p)$, where $p = getPaaFactor = n\prime/|\mathbf{A}|$, 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 [43, 45], 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 Sect. "Removing the memory bottleneck".

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 8 K (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 [33].

Assume that the entire area of an 8 K 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

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

user_patience = $t$
estimated_time ← SplatTimePredict($T_A$, $T_B$, $m$)
if estimated_time > user_patience
    $p$ ← getPaaFactor($T_A$)
    $T'_A$ ← paa($T_A$, $p$)
    $T'_B$ ← paa($T_B$, $p$)
    $m'$ ← floor($m/p$)
    Mplot ← SPLAT($T'_A$, $T'_B$, $m'$)        // see Table 1
else
    Mplot ← SPLAT($T_A$, $T_B$, $m$)        // see Table 1
return Mplot
```

length 7680 and 4320 in well under one second on a standard desktop (by way of contrast, if $m = 512$, existing brute-force algorithms take about 840 s). 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 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($T_A$(seg$_A$), $T_B$(seg$_B$), $m$)        //see Table 2
```

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 ($T_A{\prime}$, $T_B{\prime}$) and the reduced subsequence length ($m{\prime}$) 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.

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 7680 and 4320, which reduces the memory footprint by a factor of ~ 31,000.

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

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 $\mathrm{T}_{\mathrm{feature}}$, Subsequence length $m$, Patch size $p$, Overlap size $ov$,
Output: The patch with top-1 $\mathrm{T}_{\mathrm{feature}}$ best_patch

1  best_patch = Null
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 ← $\mathrm{T}_{\mathrm{feature}}$(best_patch, curr_patch)
7  return best_patch
```

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

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 2000 by 2000 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, $\mathrm{T}_{\mathrm{feature}}$, exists. In Table 4 we show how we can use the *PiecewiseSPLAT* algorithm to find the patch that contains the top-1 $\mathrm{T}_{\mathrm{feature}}$.

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. $\mathrm{T}_{\mathrm{feature}}$. 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 $\mathrm{T}_{\mathrm{feature}}$.

Thus far we have glossed over the nature of $\mathrm{T}_{\mathrm{feature}}$. 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 [20], 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 [27], 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. [22]. 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 Sect. "Experimental evaluation" 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.

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" [41]. 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..`binge watching`. If there is repeated structure longer than three, such as ...`notwithstanding her demandingnesses`..., 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 subsubsequences 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 s. 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

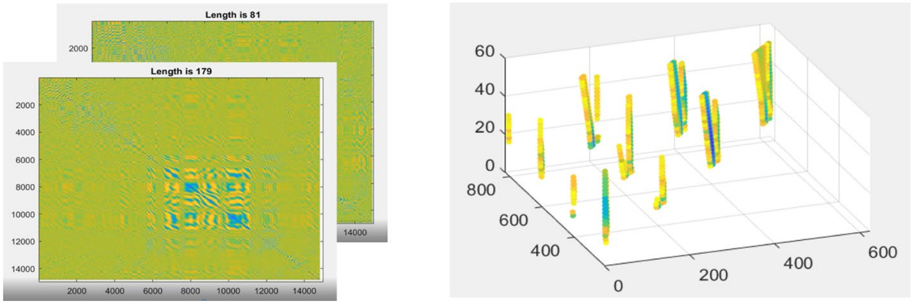

Fig. 3 Left) Screen grabs from a Mplot video. Right) A 3D Mplot shows how motifs change as a function of the subsequence length

current authors is the behavior of sap feeding insects [4, 5, 11, 38], 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.

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 [31] 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. Figure 4 shows an example of a multifocal Mplot on some insect electrical penetration graph (EPG) telemetry.

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.

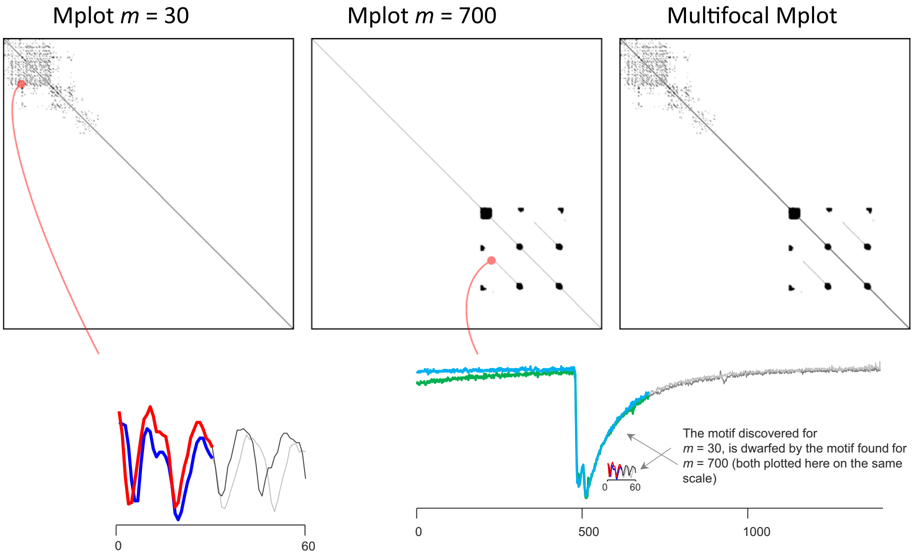

Fig. 4 Left) A Mplot with $m = 30$ discovers conserved periodic behavior corresponding to xylem ingestion [11] but fails to discover conserved behavior at longer time frames. Center) A Mplot with $m = 700$ discovers conserved periodic behavior corresponding to intercellular passage but cannot represent the shorter xylem ingestion behavior. Right) A multifocal Mplot can simultaneously represent conserved behavior at both scales

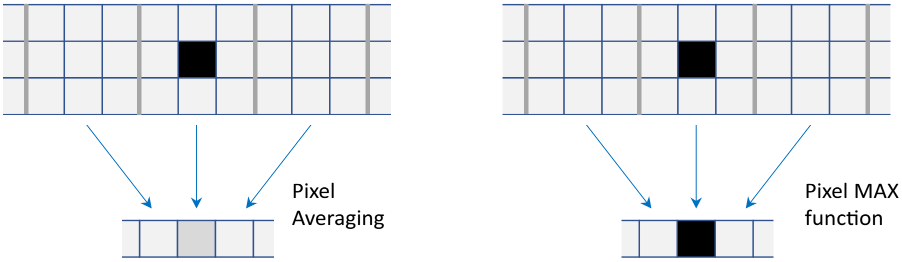

Fig. 5 Left) A naïve averaging of pixels can "blur" out features when downscaling. Right) In contrast, while a MAX aggregation may create some small amount of spatial uncertainty, it preserves the strength ("color") of the discovered motif

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.

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
```

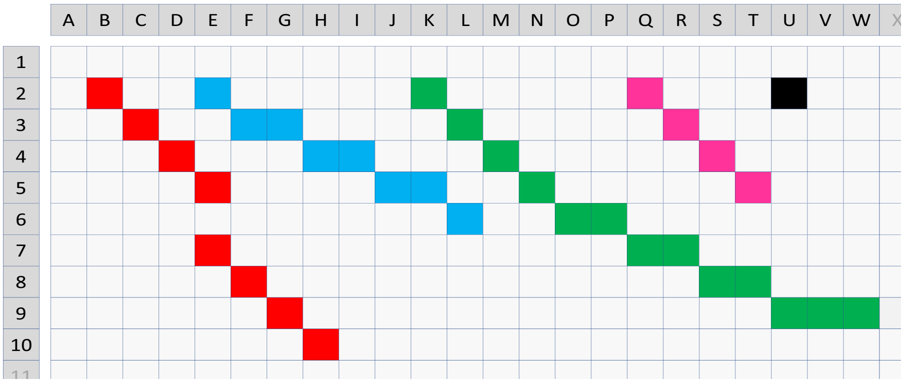

Fig. 6 Some examples of patterns we may see on a Mplot. Here we assume $m=4$ was used to create this plot

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 8 K 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.

Interpreting Mplots

There are many useful guides to interpreting recurrence plots/dot plots available [22]. We will not duplicate those efforts here. However, as we noted in Sect. "Related work", 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 $k \pm 1$, i and $k \pm 2$, 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.

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 [28]. This is an important benefit, as finding motifs with invariance to warping (i.e. Dynamic Time Warping [28]) 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.

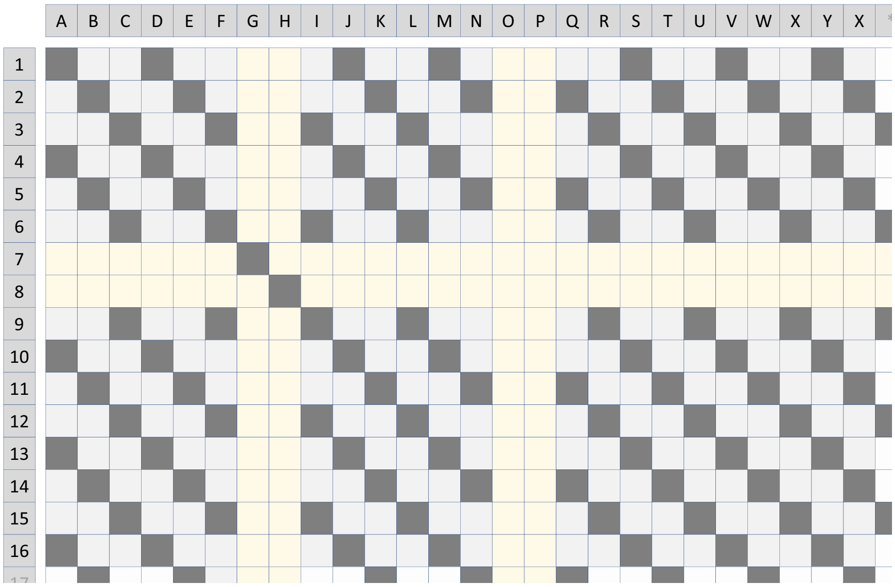

Fig. 7 A hypothetical Mplot. Note that there are two "crosses" formed by the sparse rows {7,8} and the sparse columns {G,H} and {O,P}

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 [40, 43]). 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.

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.

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

best_patches = [];
candr = [];                // Mplot rows with less similarity to others
candc = [];                // Mplot columns with less similarity to others
for $row = 1: T\_Length - p; row + p - ov$
    for $col = row : T\_Length - p ; col + p - ov$
        $T_A, T_B = T(col{:}col+p-1), T(row{:}row+p-1)$
        Mplot = SPLAT($T_A, T_B, m$)        // see Table 1
        candr.append(rows in Mplot | sum(rows) < mean(rows)) //rows with less black pixels
        candc.append(cols in Mplot | sum(cols) < mean(cols))) //columns with less black pixels
for patch in intersection (candr, candc)
    hlines = HoughTransform(patch)
    if length(hlines) > 0
        best_patches.append(patch)
return sorted(best_patches, black_pixel_count, 'descending')
```

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.

In line 1, we define an empty list to store the possible best patches. Lines 2 and 3 introduce 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 [6]. 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 Sect. "Pooling SPLAT" 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 [16], Time Series Shapelets [40], Time Series Novelets [24], etc. More exciting is the possibility of that this framework will be used to discover structures that did not occur to the current authors.

Experimental evaluation

To ensure that our experiments are reproducible, we have built a website [31] that contains all the data/code used in this work. All experiments were conducted on an Intel® Core i7-9700CPU at 2.80 GHz 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 [31] 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.

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.

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?

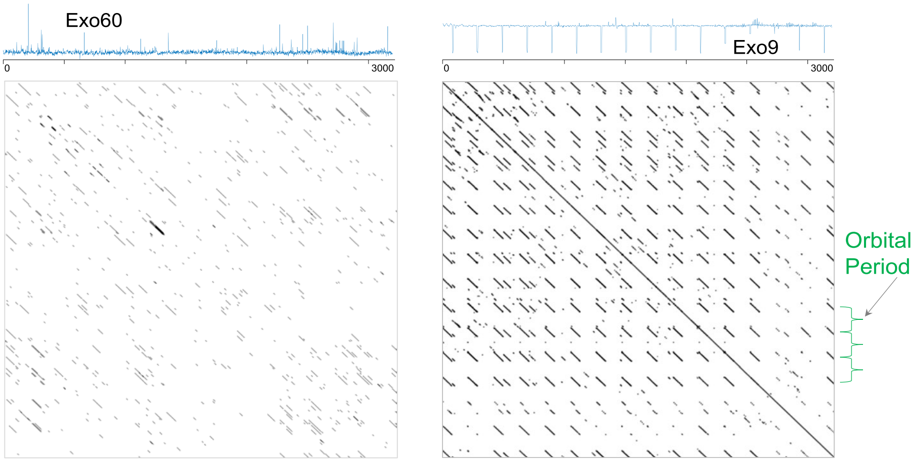

Fig. 8 Top.left) A star-light curve from a star believed not to have an exoplanet. Top.right) A star light curve from a star known to have an exoplanet. Bottom.left) The Mplot of the planetless star is relatively featureless. Bottom.right) The Mplot of Exo9 reveals not only the existence of an exoplanet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

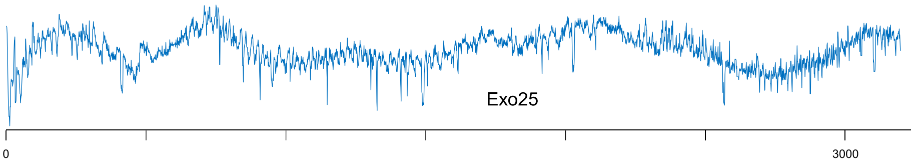

Fig. 9 The star light curve for Exo25. Does it suggest the existence of an Exoplanet?

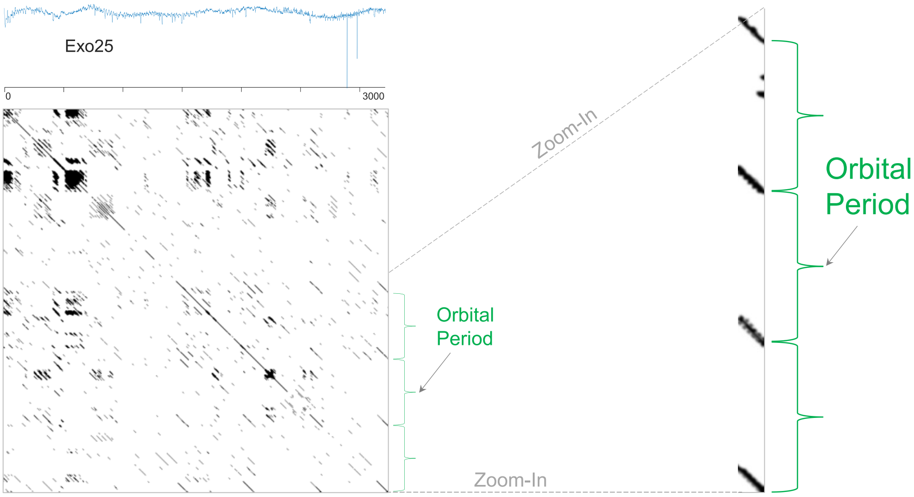

Fig. 10 The star-light curve for Exo25 with its Mplot ($m = 100$). While there is noise reflecting the original data's noise, there is the unmistakable signature of an exoplanet with an orbital period about three times longer than Exo9 (Cf. Fig. 8.bottom.right)

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

A visual inspection offers strong evidence for the existent of an exoplanet. As the callout 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 [31] 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.

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 [40, 43]. 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 [23] 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

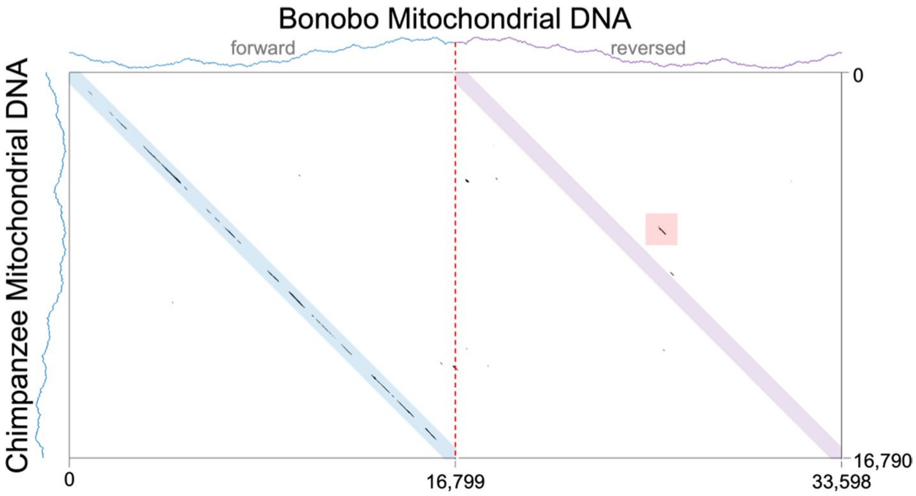

Fig. 11 A contrast-Mplot revealing mitochondrial DNA subsequences are shared between Bonobos and Chimps, but absent from Humans. The region highlighted in red indicates a reversed and offset Bonobo subsequence relative to the Chimp sequence

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$_{\mathrm{AB}}$ between T_A and T_B, then compute two Matrix Profiles MP$_{\mathrm{AC}}$ and MP$_{\mathrm{BC}}$. We habituate through the following indexed definition:

$$\mathbf{ContrastMplot}^{(\mathrm{i,j})} = \min\left(MP^{i}_{AC}, MP^{j}_{BC}\right) - Mplot^{(i,j)}_{AB}$$

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 [14], so we will investigate whether there exist DNA subsequences shared between them, but which is absent from humans.

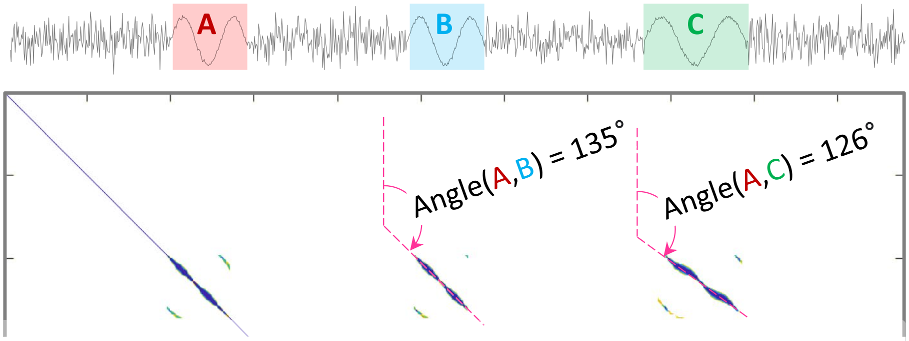

Fig. 12 Top) A toy time series with three sine-wave patterns embedded. Note that instance C is about 37% longer than the other two instances A and B. Bottom) The corresponding Mplot shows that the difference in lengths manifests as a difference in angle

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 [29] 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 [14]. 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.

Finding rescaled motifs using PiecewiseSPLAT

As we noted in Sect. "Removing the human visual attention bottleneck" 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 [43]. If the difference in scale is relatively small, say < 8 to 20%, then there are a handful of techniques to address such cases [39]. However, here we are interested in motifs that may dramatically differ in scale, say up to 300%.

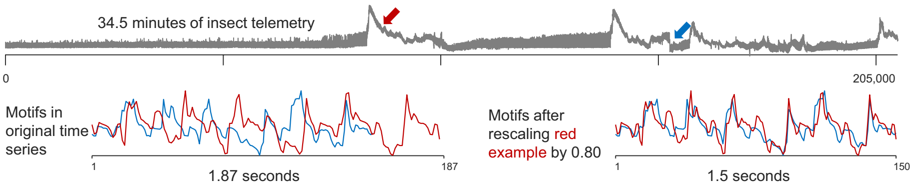

Fig. 13 Top) Telemetry from an insect pest feeding on a plant. bottom) A multi-scale motif discovered in the data can only be seen as conserved after one instance is rescaled by a factor of 1.25

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

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 $L\text{-}m \times \sqrt{2}$, 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:

$$\text{ScalingFactor}(\text{A}, \text{C}) = \frac{1}{\tan\left(\text{Angle}(\text{A}, \text{C}) - 90^{\circ}\right)}$$

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 transform [6]. 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.

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 [40].

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 1400 known species [35]. 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 [29] 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 [29].

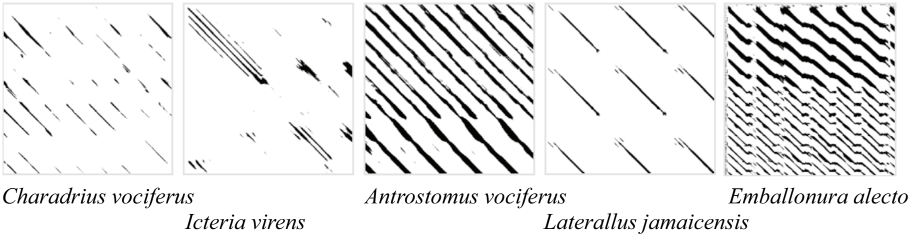

Fig. 14 Five randomly chosen six-second snippets of animals that both fly and produce sound at night. The four leftmost examples are all birds. The rightmost example is a bat, which is unique here in having "stripes" that are not perfectly parallel to the diagonal

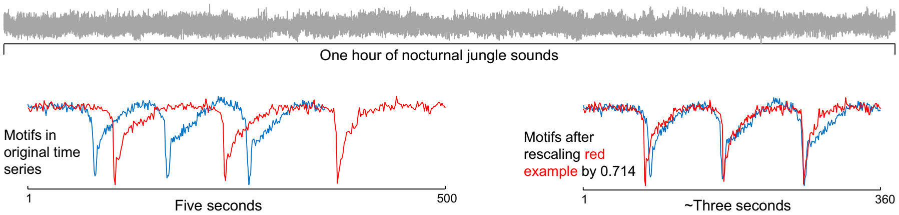

Fig. 15 Top) A one-hour dataset containing bird sounds, and a total of 20 seconds of bat sound. Bottom) If we use PiecewiseSPLAT to search for motifs that have at least 1.35 rescaling, the top-1 motif is a bat vocalization

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 Sect. "Interpreting Mplots". Consider Fig. 14.right.

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 $\pm 9.5°$ to the diagonal, indicating a rescaling factor of 1.40. As shown in Fig. 15.

The top-1 motif was indeed a bat vocalization. This experiment took 81 min, 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.

Searching massive Mplots

Recall that in Sect. "Interpreting Mplots: reverse engineered" 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

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

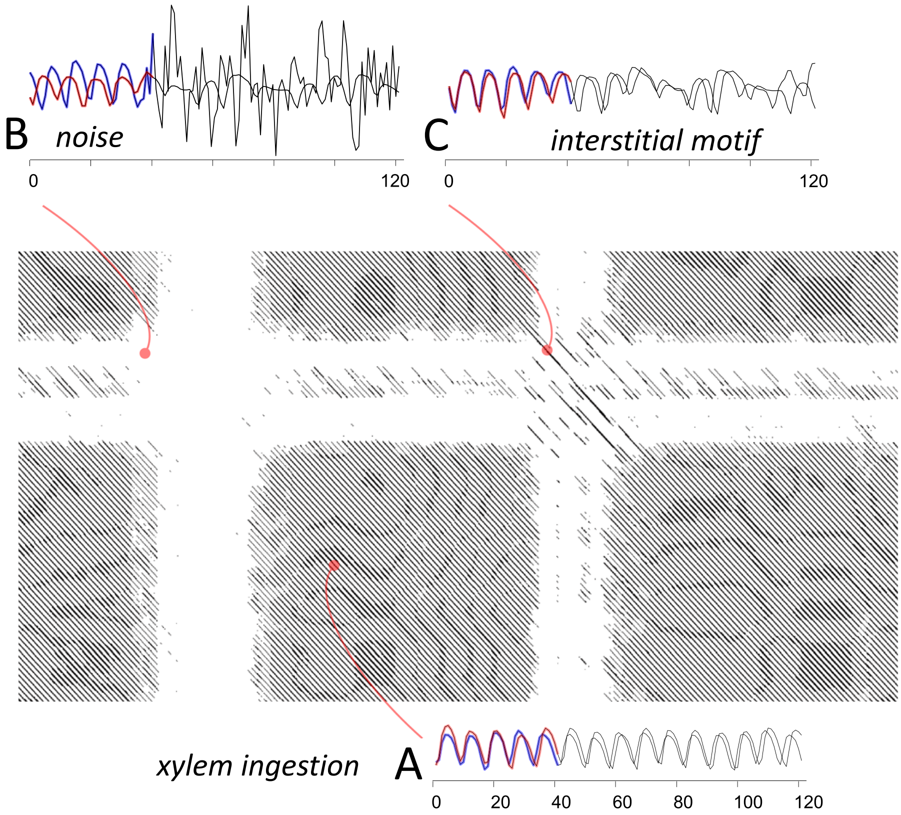

Fig. 16 An Mplot with the three corresponding pairs of time series extracted from an Asian citrus psyllid (*Diaphorina citri*). The value of *m* was forty (the length of the colored prefix in the call-out plots), and we show the following eighty datapoints for context. **A** A typical bout of xylem ingestion 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

would look like on a Mplot, then build a 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 [4, 38]. 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. 16A 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 40 min [11]. 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

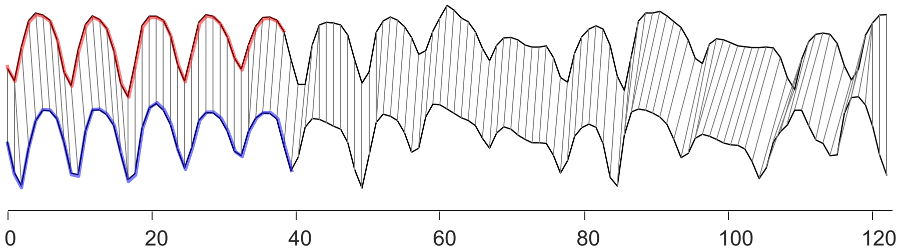

Fig. 17 A larger reproduction of the interstitial motif shown in Fig. 16C

in a Faraday cage in an attempt to mitigate electronic noise interference [26]). 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 h long recording.

Figure 16 allows us to illustrate the three possibilities that make up our dataset. Figure 16A 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 [4, 38]. Such patterns make up more than 99% of the Mplot. Figure 16B shows a white cross with an empty intersection. This corresponds to a noisy region in the time series. Figure 16C 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.

We illustrate the similarity of the two time series by showing the Dynamic Time Warping alignment between them [28]. 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 [5]. 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 [4] 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 h) for the telltale white crosses. Each patch was of size 10,000 by 10,000 and took about 5.3 s to process. As there are 10,000 patches, the entire process took about 8.5 h. 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.

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

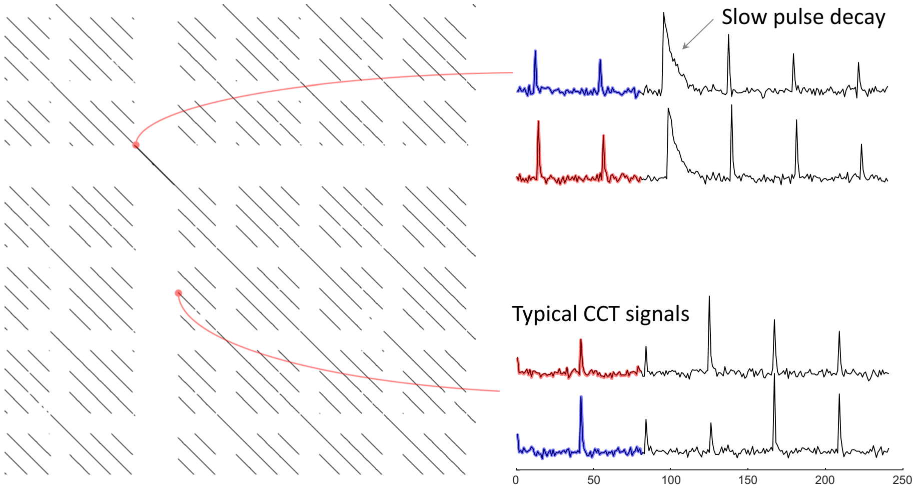

Fig. 18 Left) A zoom-in of an AB-Mplot created with CCT telemetry from two mice. Right) The value of *m* was eighty (the length of the colored prefix in the call-out plots), and we show the following 160 datapoints for context

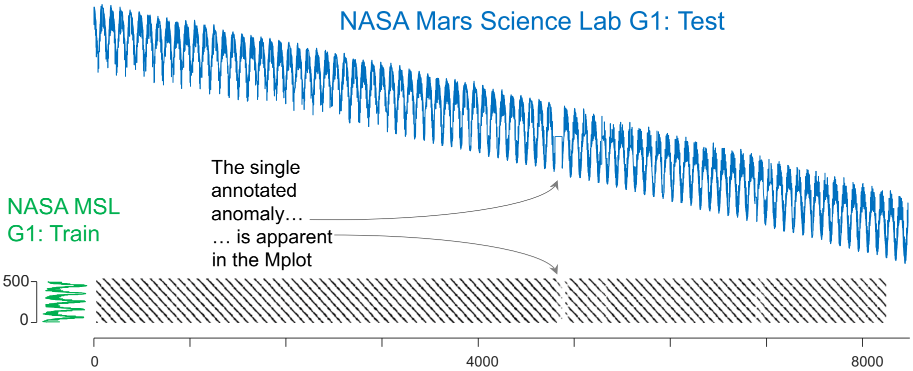

Fig. 19 The G1 trace from NASA MSL [15] is used to create an AB Mplot (The training data is longer that shown here. However, it is highly redundant, so we only used the first 500 datapoints.) with $m = 80$. The only anomaly annotated in the official NASA record is at 4797 to 4871, and it shows up in the Mplot as an obvious break in the continuity of the diagonal lines (recall the broken "red streak" example in Fig. 6)

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) [21]. 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 [21] to understand that, as illustrated in Fig. 18 top.right, this is a rare motif of slow pulse decay.

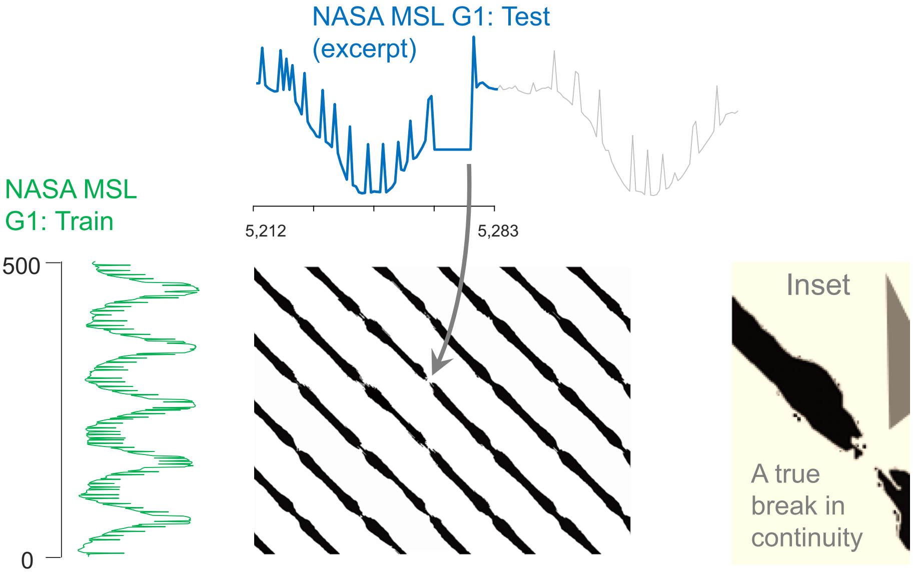

Fig. 20 A zoom-in of Fig. 19. At about location 5271 there is an apparent anomaly, similar to, but much shorter than the officially recorded anomaly (Not shown, there is a similar, but smaller anomaly at 6879 to 6894). The zoomed-in inset shows that the anomaly causes a break in the continuity of the diagonal lines of the Mplot

Using Mplot as anomaly detectors

Most of our examples thus far have concentrated on the discovery of conserved structure (motifs), and on the self-join use of Mplots. In this example we show that Mplots can also be useful for discovering violations of conservation, which are (in most contexts) called time series anomalie*s*. Moreover, here we will consider an AB-Join, not a self-join. That is to say we consider a matrix is $\mathbf{DP}_{\mathrm{AB}}^{(j,m)}$, where $\mathbf{A} \neq \mathbf{B}$, (cf. Definition 4). Note that, as shown in Fig. 19, such Mplots are not generally square.

In [15], Hundman and his colleagues introduced a dataset that has since become widely studied and has been cited more than one thousand times. The Mars Science Laboratory (MSL) rover dataset is a set of telemetry anomalies corresponding to actual spacecraft issues involving various subsystems and channel types. Beyond the excitement of the domain, the dataset is attractive because it has unusually good provenance. The labels come from "expert-labeled data derived from Incident Surprise, Anomaly (ISA) reports", and we are reassured that "All telemetry channels discussed in an individual ISA were reviewed to ensure that the anomaly was evident in the associated telemetry data, and specific anomalous time ranges were manually labeled for each channel.". Moreover, the data also comes with positive only training data.

In Fig. 19 we show MSL:G1 and its corresponding Mplot.

There is one anomaly labeled in this dataset by NASA's ISA report beginning at location 4797. A casual glance at the Mplot can clearly locate the anomaly. It does not seem to be a particularly hard problem, and perhaps the dataset is ill-suited to the strong claims made by those using this dataset to compare rival algorithms.

However, when using the Mplot to investigate this dataset we noticed two other subtle breaks in the diagonal stripes. In Fig. 20 we show a zoom-in of one of the relevant regions and its corresponding Mplot.

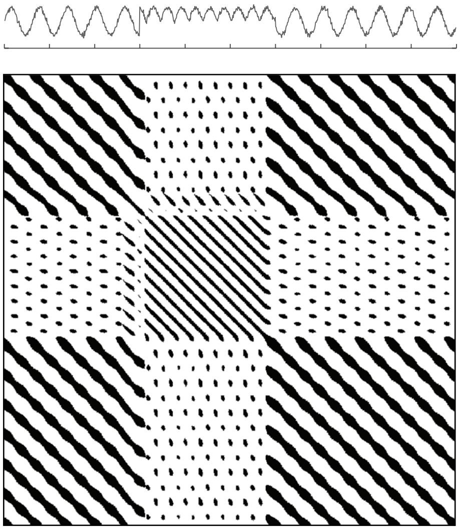

Fig. 21 Regime changes produce block-like Mplots

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

We wrote to the original authors and asked them to examine our findings. They confirmed that these two examples are true positives, missed by the original annotators.

As an aside, it is interesting to note that while at least one hundred papers have explicitly experimented on this dataset, to the best of our knowledge, none of them have reported noting these unknowledge true positives. However, most of these papers report results (typically F1) using four significant digits. If we correctly labeled the data based on the revised acknowledgement of ground truth, this would change at least two of those digits.

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. Figure 21 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 [9], 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.

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

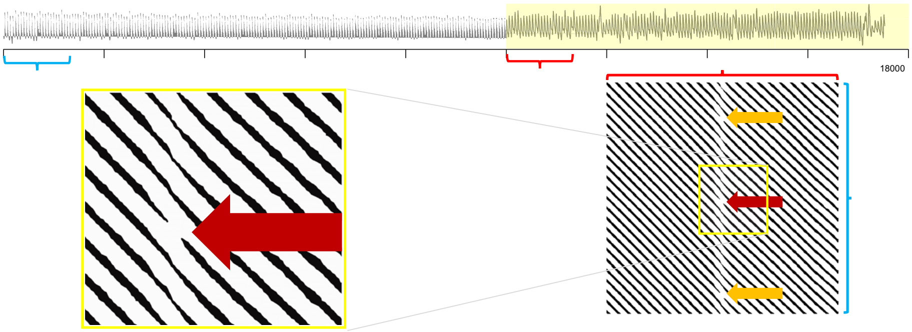

Fig. 22 Top) The PulsusParadoxus$_{SP02}$ segmentation problem is very subtle. Bottom.left) A zoom-in of the Mplot close to the regime change revels a break in the diagonal streak. Bottom.right) A zoom-out indicate that these breaks happen once in every eight beats

thirty-two diverse datasets. We use the evaluation metric suggested by the creators of the datasets [12]. Table 7 summarizes the results.

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 [12] 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. 22 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 [12], 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.

However, there is an interesting clue as shown in Fig. 22 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.

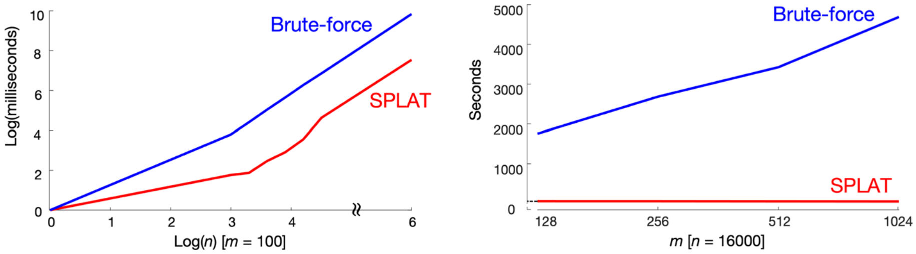

Fig. 23 SPLAT execution time vs. brute-force algorithm—note that both the left figure's axis are in log scale

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

Speed and scalability

In Fig. 23left 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. 23right we hold the length of the time series to a fixed 16,000, and test the effect of increasingly large values of m.

The reader will observe that we can compute a million length time series in about 9.5 h 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 s. We refer the reader to visit [31] for more results and the GPU code.

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" [30]. They also consider a dataset of size 1 M, finding it took 68.94 s 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 [30] 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.

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.

Acknowledgements
We thank all the donors of datasets, and the domain experts that provided context and interpretation of the patterns discovered.

Author contributions
M.S: Writing, Implementation, Experiment Design. R.M. Bioinformatics Experiment Design. J.R. Segmentation Experiment Design. A.D. Creation of Accompanying Videos. Z.Z Porting Algorithms to GPU. K.M: Entomology Experiment Design. E.K: Funding, Editing.

Funding
We gratefully acknowledge funding from Accenture, Mitsubishi Labs and NSF Award 2103976.

Data availability
To ensure that our experiments are reproducible, we have built a website [31] that contains all the data/code used in this work. Shahcheraghi [32].

Declarations

Competing interests
The authors of this paper declare that they have no conflict of interest.

Received: 23 November 2023 Accepted: 2 July 2024
Published online: 20 July 2024

References

1. Afonso LCS, Rosa GH, Pereira CR, et al. A recurrence plot-based approach for Parkinson's disease identification. Future Gen Comput Syst. 2019;94:282–92. https://doi.org/10.1016/j.future.2018.11.054.
2. Alaee S, Mercer R, Kamgar K, Keogh E. Time series motifs discovery under DTW allows more robust discovery of conserved structure. Data Min Knowl Discov. 2021;35:1–48. https://doi.org/10.1007/s10618-021-00740-0.
3. Almeida-Ñauñay AF, Benito RM, Quemada M, et al. Recurrence plots for quantifying the vegetation indices dynamics in a semi-arid grassland. Geoderma. 2022;406: 115488. https://doi.org/10.1016/j.geoderma.2021.115488.
4. Bonani JP, Fereres A, Garzo E, et al. Characterization of electrical penetration graphs of the Asian citrus psyllid, Diaphorina citri, in sweet orange seedlings. Entomol Exp Appl. 2009;134:35–49. https://doi.org/10.1111/j.1570-7458.2009.00937.x.
5. Chesnais Q, Mauck KE. Choice of tethering material influences the magnitude and significance of treatment effects in whitefly electrical penetration graph recordings. J Insect Behav. 2018;31:656–71. https://doi.org/10.1007/s10905-018-9705-x.
6. Duda RO, Hart PE. Use of the Hough transformation to detect lines and curves in pictures. Commun ACM. 1972;15:11–5. https://doi.org/10.1145/361237.361242.
7. Eckmann J-P, Kamphorst SO, Ruelle D. Recurrence plots of dynamical systems. Europhys Lett (EPL). 1987;4:973–7. https://doi.org/10.1209/0295-5075/4/9/004.
8. Fang Y, Xu H, Jiang J. A survey of time series data visualization research. IOP Conf Ser Mater Sci Eng. 2020;782:22013. https://doi.org/10.1088/1757-899x/782/2/022013.
9. Foote J, Cooper M. Media segmentation using self-similarity decomposition. Proceedings of SPIE—The International Society for Optical Engineering. 2022; 5021. https://doi.org/10.1117/12.476302.
10. Fukino M, Hirata Y, Aihara K. Coarse-graining time series data: recurrence plot of recurrence plots and its application for music. Chaos Interdiscip J Nonlinear Sci. 2016;26:23116. https://doi.org/10.1063/1.4941371.
11. George J, Kanissery R, Ammar E-D, 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. 2020. https://doi.org/10.3390/insects11010048.
12. Gharghabi S, Ding Y, Yeh C-CM, et al. Matrix profile VIII: domain agnostic online semantic segmentation at superhuman performance levels. 2017; pp 117–126.
13. Gibbs AJ, Gibbs AJ, McIntyre GA. The diagram, a method for comparing sequences: its use with amino acid and nucleotide sequences. Eur J Biochem. 1970;16:1–11. https://doi.org/10.1111/j.1432-1033.1970.tb01046.x.
14. Green RE, Malaspinas A-S, Krause J, et al. A complete neandertal mitochondrial genome sequence determined by high-throughput sequencing. Cell. 2008;134:416–26.
15. Hundman K et al. Detecting spacecraft anomalies using LSTMs and nonparametric dynamic thresholding. In: Proceedings 24th ACM SIGKDD Intl. Conf. Knowledge Discovery and Data Mining, 2018; pp. 387–395.

16. Imamura M, Nakamura T, Keogh E. 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. Association for Computing Machinery, New York, NY, USA; 2020, pp 1114–1122.
17. Keogh E, Chakrabarti K, Pazzani M, Mehrotra S. Dimensionality reduction for fast similarity search in large time series databases. Knowl Inf Syst. 2001;3:263–86. https://doi.org/10.1007/PL00011669.
18. Lin J, Keogh E, Lonardi S, et al. VizTree: a tool for visually mining and monitoring massive time series databases. In: Proceedings of international conference on very large data bases. 2004; pp 1269–1272.
19. Lopes MA, Zhang J, Krzemiński D, et al. Recurrence quantification analysis of dynamic brain networks. Eur J Neurosci. 2021;53:1040–59. https://doi.org/10.1111/ejn.14960.
20. Malige F, Djokić D, Patris J, et al. Use of recurrence plots for identification and extraction of patterns in humpback whale song recordings. Bioacoustics. 2020. https://doi.org/10.1080/09524622.2020.1845240.
21. Marimon X, Traserra S, Jiménez M, et al. Detection of abnormal cardiac response patterns in cardiac tissue using deep learning. Mathematics. 2022. https://doi.org/10.3390/math10152786.
22. Marwan N, Carmen Romano M, Thiel M, Kurths J. Recurrence plots for the analysis of complex systems. Phys Rep. 2007;438:237–329. https://doi.org/10.1016/j.physrep.2006.11.001.
23. Mercer R, Alaee S, Abdoli A, et al. 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.
24. Mercer R, Keogh E. 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.
25. Mueen A, Zhu Y, Yeh M, et al. The fastest similarity search algorithm for time series subsequences under Euclidean distance. In: http://www.cs.unm.edu/~mueen/FastestSimilaritySearch.html. 2017.
26. Nalam V, Louis J, Patel M, Shah J. 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. 2018. https://doi.org/10.21769/BioProtoc.2950.
27. Phillipson RA. Complex long-term variability of X-ray binaries and active galaxies revealed by novel methods. In: American Astronomical Society Meeting Abstracts #236. 2020;p 122.02.
28. Rakthanmanon T, Campana B, Mueen A, 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). 2013.
29. Ratcliffe JM, Elemans CPH, Jakobsen L, Surlykke A. How the bat got its buzz. Biol Lett. 2013. https://doi.org/10.1098/rsbl.2012.1031.
30. Rawald T, Sips M, Marwan N. PyRQA—Conducting recurrence quantification analysis on very long time series efficiently. Comput Geosci. 2017;104:101–8. https://doi.org/10.1016/j.cageo.2016.11.016.
31. Shahcheraghi M. mplot. 2022. https://sites.google.com/view/mplot/. Accessed 6 Mar 2023.
32. Shahcheraghi M, Mercer R, Rodrigues JDA, et al. Matrix profile XXVI: Mplots: scaling time series similarity matrices to massive data. In: 2022 IEEE International Conference on Data Mining (ICDM). Los Alamitos, CA, USA: IEEE Computer Society; 2022. p. 1179–84.
33. Shneiderman B. 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.
34. Soloviev VN, Serdiuk O, Semerikov SO, Kiv AE. Recurrence plot-based analysis of financial-economic crashes. In: M3E2-MLPEED. 2020.
35. Tabak M, Murray K, Lombardi J, Bay K. Automated classification of bat echolocation call recordings with artificial intelligence. 2021.
36. Takakura I, Hoshi R, Santos M, et al. Recurrence plots: a new tool for quantification of cardiac autonomic nervous system recovery after transplant. Braz J Cardiovasc Surg. 2017. https://doi.org/10.21470/1678-9741-2016-0035.
37. Webber CL, Zbilut JP. Dynamical assessment of physiological systems and states using recurrence plot strategies. J Appl Physiol. 1994;76:965–73. https://doi.org/10.1152/jappl.1994.76.2.965.
38. Willett D, George J, Willett N, et al. Machine learning for characterization of insect vector feeding. PLoS Comput Biol. 2016;12: e1005158. https://doi.org/10.1371/journal.pcbi.1005158.
39. Yankov D, Keogh E, Medina J, et al. Detecting time series motifs under uniform scaling. In: Proceedings of the ACM SIGKDD international conference on knowledge discovery and data mining. 2007; pp 844–853.
40. Yeh C-CM, Zhu Y, Ulanova L, et al. Matrix profile I: all pairs similarity joins for time series: a unifying view that includes motifs, discords and shapelets. 2016; pp 1317–1322.
41. Zhang Y, Hou Y, OuYang K, Zhou S. Multi-scale signed recurrence plot based time series classification using inception architectural networks. Pattern Recogn. 2022;123: 108385. https://doi.org/10.1016/j.patcog.2021.108385.
42. Zhu X-C, Zhao D-H, Zhang Y-H, et al. Multi-scale recurrence quantification measurements for voice disorder detection. Appl Sci. 2022. https://doi.org/10.3390/app12189196.
43. Zhu Y, Zimmerman Z, Senobari NS, 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.
44. Zilberstein S. Optimizing decision quality with contract algorithms. In: Proceedings of the 14th International Joint Conference on Artificial Intelligence—Volume 2. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA; 1995. pp 1576–1582.
45. Zimmerman Z, Kamgar K, Senobari NS, 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. Association for Computing Machinery, New York, NY, USA; 2019. pp 74–86.

Publisher's Note