Matrix Profile XXVI: Mplots: Scaling Time Series Similarity Matrices to Massive Data

Maryam Shahcheraghi[1] Ryan Mercer[1] João Manuel de Almeida Rodrigues[2] Audrey Der [1]
Hugo Filipe Silveira Gamboa[2] Zachary Zimmerman[1] Eamonn Keogh[1]
[1]University of California, Riverside [2]Libphys-UNL NOVA Faculty of Science and Technology
{sshah073, rmerc002, ader003, zzimm001}@ucr.edu jmad.rodrigues@gmail.com hgamboa@fct.unl.pt eamonn@cs.ucr.edu

***Abstract*—Time series similarity matrices (informally, recurrence 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 we can largely remove 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 attention by either humans or algorithms. We will demonstrate the utility of our ideas for data exploration, segmentation, and classification in diverse domains.**

Keywords—Time Series; Anomalies; Similarity Matrix

I. INTRODUCTION

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 called recurrence plots, dot plots, self-similarity matrices, similarity plots, time series similarity matrices etc. [2][11]. For concreteness we will call the variant of interest here *time series similarity matrix plots* or just *Mplots*. Mplots have many uses in data mining. They can be used for visual exploration of data, or various features can be extracted from them, and then fed into other algorithms. For example, the *Matrix Profile* is an increasingly popular time series analytical tool that is directly extracted from a Mplot, by recording the smallest (off-the-diagonal) value in each column [20][21][23].

A simple Mplot makes comparisons within time series **A**, as shown in Figure 1, and we can also use this representation to compare and contrast *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 are used in astronomy [14], economics, music, physiology, neuroscience, earth sciences, astrophysics and engineering [11]. As noted in a founding paper on the topic, "*information obtained from recurrence plots is often surprising, and not easily obtainable by other methods*" [2].

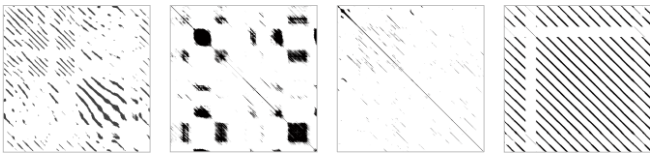

Figure 1: Examples of Mplots hint at the diversity and utility of this data structure. Like many plots in the paper, these figures suffer somewhat from the size of reproduction. We encourage the interested reader to visit [25] which has larger figures and videos. Further note that here we show binarized Mplots, but more generally Mplots allow a spectrum of colors to indicate degree of similarity.

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

- **CPU**: Classic Mplots require processing that is quadratic in the length of the time series, and linear in the length of the subsequences. This seems to have limited their use to time series with a length of about 20,000 [2][10][11][14].
- **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.

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

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

II. Motivation and 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. [6]. Moreover, many of the uses of recurrence plots are also uses of Mplots. However, it is worth explicitly pointing out 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 be over 1,000. Moreover, these vectors need to be normalized (surprisingly, normalization generally takes longer than the distance computation). 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 [3] and the refences therein. However, Mplots are particularly direct and intuitive. Moreover, unlike say Viztrees [9], 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, two days with similar spending patterns on flowers and restaurants (Here we assume the USA Mother's Day).

III. Defniitions 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 shapes of small regions called *subsequences*.

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

We can measure the distance between any two time series *subsequences* of equal length using a distance measure. In this work, we use the ubiquitous z-normalized Euclidean distance [20]. Note the subsequence length here takes on a similar role to the *embedding dimension* in discrete dot plots [10][11]. 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.

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 $\mathbf{T_A}$ and a query subsequence $\mathbf{T}_{\mathrm{B}}^{(j,m)}$.

The distance can be computed very efficiently using the MASS algorithm [13]. 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 [8]. If we wish to downsample a time series by a factor of *d*, we indicate this by PAA(**A**,*d*). As Figure 2 shows, on many datasets it is possible to significantly downsample the data, while retaining the essential features.

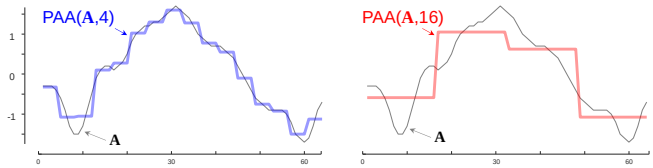

Figure 2: A time series **A** with *n* = 64 downsampled with PAA. *left*) Downsampled 1 in 4, *right*) Downsampled 1 in 16. While the 1 in 16 plot has lost significant detail, the 1 in 4 downsampling does preserve the basic shape of the time series.

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

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

When **A**= **B**, this definition is logically equivalent to a self-join of **A**. The popular Matrix Profile is simply the vector of length |**A**| that contains the minimum (non-diagonal) value in each column [20][21][23]. The state-of-the-art Matrix Profile algorithms (SCRIMP, SCAMP) can compute this incrementally, without ever having to have the entire matrix in main memory at one time. When **A**≠ **B**, this definition is logically equivalent to an **AB**-join. Such joins are frequently used in recurrence plots to visualize the differences between two DNA sequences, but surprisingly, to the best of our knowledge, there are very few uses of real-valued **AB**-join time series.

IV. SCALING 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.

A. 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 $O(m \times n^2)$ [11][14]. As we show in Section VI, 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. In this section we show that we can completely remove the dependence on m and make Mplot's a true $O(n^2)$ algorithm with tiny constant factor.

The idea of making the time complexity of a Mplot independent of the m value is similar to the Matrix Profile algorithms proposed in [21][24]. Let us consider the formula for calculating a cell of distance profile ($d^{(i,j)}$).

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

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

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

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

Table 1: The SPLAT Algorithm to compute Mplots

```
Function:  SPLAT(T_A, T_B, m)
Input:     T_A:  Reference time series
           T_B:  Query time series
           m:    Subsequence length
Output:    Mplot: Distance matrix
1  Mplot = nan(length(T_A)-m+1, length(T_B)-m+1)
2  μ,σ ← computeMeanStd(T_A, T_B, m)                    //see [15]
3  for diag = 1 to Mplot_column_count
4      for row = 1 to Mplot_row_count - diag + 1
5        col ← row + diag - 1
6        QT_row,col ← ComputeDotProduct(T_A^(col,m), T_B^(row,m))   //see [21]
7        d ← CalculateDistanceProfile(T_A^(col,m), T_B^(row,m))     //see (1)
8        Mplot(row, row + diag - 1) = d
9  return Mplot
```

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

By taking advantage of the techniques in [15], [21] and [23] in addition to the mean and standard deviation, the dot product can also be calculated in O(1). So, a time complexity of $O(n^2)$ is achieved which is the minimum required to compute all the values in a Mplot with $n \times n$ cells.

The SPLAT algorithm can efficiently compute large Mplots, but we may task it with a long time series that would take longer to compute than the user's patience allows. To address this issue, we can create a *contract algorithm* version of SPLAT, parameterized by the maximum amount of time the user is willing to wait [22]. For example, in SPLAT(**A**,**B**,*m*,4), the user is requesting the best approximation that can be computed in four seconds or less.

To achieve this user-requested time limit, we will approximate the time series with PAA (Recall Figure 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 **T**, of length n', will take *SplatTimePredict*(**T**, n') = $S * (n'/|\mathbf{A}|)^2$.

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

Although the minimum possible time complexity has now been achieved with these ideas, we will run into issues with memory usage for a long time series. A Mplot needs all its cell values in memory, unlike say the state-of-the-art Matrix Profile algorithms [21][23], which only require keeping the minimum of each column of Mplot. Thus, memory becomes the next bottleneck. Our proposed solution to this issue is described in Section IV-B.

B. Removing the Memory Bottleneck

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

The reader may wonder why we should use the time and memory resources required to compute large matrices, when none of the available screens are able to display them at native resolution. Note that the highest resolution in a commercially available system is currently 8K (7680×4320 pixels). In fact, there is a reason to compute a Mplot at a resolution greater than can be (currently) displayed. We propose to create *multi-resolution* approach, which allows a user to initially see an approximation of a massive Mplot, and interactively zoom-in on any areas that catch her eye as requiring a more detailed inspection. When the zoomed-in patch is requested, one of two things happens:

- If the zoomed-in patch was precomputed at the required resolution, we can simply fetch it from memory.

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

Note that this style of user interaction echoes the widely known visual information seeking mantra given by Ben Shneiderman: *Overview first, zoom and filter, then details-on-demand* [17].

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

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

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

In Table 2 we formalize these ideas, beginning with the main MultiResSPLAT algorithm.

Table 2: The MultiResSPLAT Algorithm

```
Function:  MultiResSPLAT(T_A, T_B, m)
Input:     T_A:   Reference time series
           T_B:   Query time series
           m:     Subsequence length
Output:    Mplot: Distance matrix
1  user_paitience = t
2  estimated_time ← SplatTimePredict(T_A, T_B, m)
3  if estimated_time > user_patience
4      p ← getPaaFactor(T_A)
5      T'_A ← paa(T_A, p)
6      T'_B ← paa(T_B, p)
7      m' ← floor(m/p)
8      Mplot ← SPLAT(T'_A, T'_B, m')          // see Table 1
9  else
10     Mplot ← SPLAT(T_A, T_B, m)             // see Table 1
11 return Mplot
```

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

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

Table 3: The MultiResSPLAT Zoom Algorithm

```
Function:  MultiResSPLATZoom(T_A, T_B, m, plt)
Input:     T_A:   Reference time series
           T_B:   Query time series
           m:     Subsequence length
           plt:   existing Mplot
Output:    Mplot: Distance matrix
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
```

The algorithm starts by obtaining the user-requested patch from an existing Mplot (*plt*) in line 1. This request normally comes from a classic rectangular selection tool. As the user selects a rectangle region on *plt*, the four corners of the selected area are returned as *lx*, *rx*, *uy*, *dy*, which are left/right x and up/down y values, respectively. Then in lines 2 and 3, the coordinates are mapped to the exact locations in both input time series T_A (seg_A) and T_B (seg_B). Finally, the MultiResSPLAT is called on the new subsets of the input time series and the new zoomed-in Mplot is returned with line 4.

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

C. Removing the Human Visual Attention Bottleneck

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

show the user the full-scale piecewise patches consecutively. However, that simply shifts the bottleneck to human visual attention, which is an even more precious resource.

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

Table 4: The PiecewiseSPLAT Algorithm

```
Function:  PieceWiseSPLAT(T_A, T_B, m, p, ov)
Input:     T_A:       Reference time series
           T_B:       Query time series
           T_feature  This is the feature we wish to find
           m:         Subsequence length
           p:         Patch size
           ov:        Overlap size
Output:    best_patch: The patch with top-1 T_feature
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 ← T_feature(best_patch, curr_patch)
7  return best_patch
```

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

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

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

- **Bioacoustics**: Malige et. al use Mplots [10], to analyze humpback whale communications, and explicitly define features on the matrix such as *song* and *theme*.
- **Astronomy**: Phillipson et. al. use Mplot to investigate stochastic light curves of Active Galactic Nuclei [14], 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), and etc. [11]. Note that not all proposed features can be computed piecewise using the algorithm in Table 4 (some features require random access to all parts of the matrix), but the vast majority can.

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

D. Parameter-Free Mplots: 3DMplots and Mplot Movies

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 two such examples, and we suspect that the community may discover many more.

Recall that Mplots have a single parameter, the subsequence length *m*. Because SPLAT is so fast, this allows two techniques to get rid of even this parameter, and thus make Mplots parameter-free.

- We can produce Mplot movies, by creating a Mplot for all possible values of *m* and writing each 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.

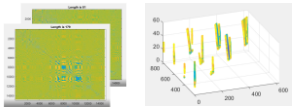

Figure 3: *left*) Screen grabs from a Mplot video. *right*) A 3D Mplot shows how motifs change as a function of the subsequence length.

While these 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 [25] to see a gallery of them used in various domains.

E. 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 Figure 4.*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.

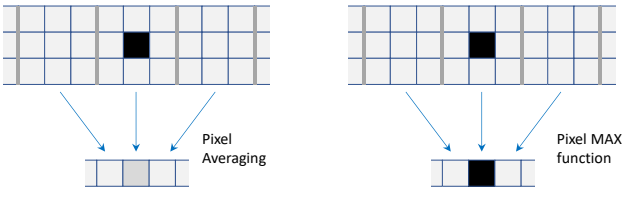

Figure 4: *left*) A naïve averaging of pixels can "blur" out features when downscaling. *right*) In contrast, while a MAX aggregation may create some small amount of spatial uncertainty, it preserves the strength ("color") of the discovered motif.

To mitigate this issue, we propose to take explicit control of how Mplot images are resized. Instead of simply averaging the pixels, we allow arbitrary aggregation functions. For example, to help highlight motifs we can use a MAX function as shown in Figure 4.*right*, and to preserve discords (anomlaie/differnces) 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:     T_A:   Reference time series
           T_B:   Query time series
           m:     Subsequence length
           s:     Mplot output size
           f:     Aggregate function
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 - diag + 1
6        d ← compute_Mplot_value@(row,col) //see Table 1
7        row', col' ← row/ n_rowslice, col/ n_colslice
8        pooled_Mplot(row' , col') ← f(d,pooled_Mplot(row', col'))
9  return pooled_Mplot
```

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

V. Interpreting Mplots

There are many useful guides to interpreting recurrence plots/dot plots available [11]. We will not duplicate those efforts here. However, as we noted in Section II, there are several differences between *true* recurrence plots and Mplots, and some of those differences effect the interpretation of plots. In Figure 5 we show some examples of patterns that are unique to Mplots. When discussing the time series that created these patterns, we use the familiar expository trick of using *text* as a proxy for time series, and *hamming distance* as a proxy for Euclidean distance.

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

In principle Mplots are similar, and a motif that was exactly m datapoints long could produce a single dot (U2). However, even if the natural motifs in the time series are exactly m datapoints long, the use of parameter m would tend not to produce a single point, but a smeared line. The reason is that if two subsequences beginning at locations i and k, are a close match, then we will still have a reasonably close match for i and $k\pm1$, i and $k\pm2$, etc. This is not true for the discrete strings of dot plots. Therefore, if we see a diagonal streak on a Mplot built with parameter m, whose length in the x-axis is d, we should interpret this as the existence of a motif of length a little greater than d. Thus, the pattern seen at Q2 suggests the existence of a motif of length five or six, not just four. This suggests a strategy for setting the value of m. We should set it to be *less than* the length of the motifs we want or expect to find.

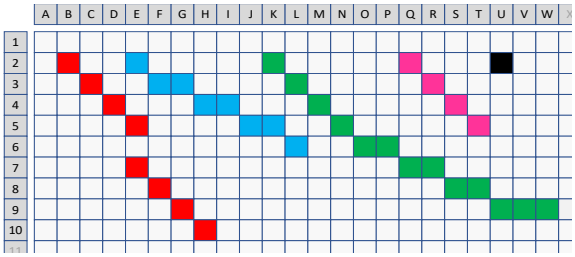

Figure 5: Some examples of patterns we may see on a Mplot. Here we assume $m = 4$ was used to create this plot

One of the patterns that are unique to Mplot is the curved line shown in beginning at K2. This suggest 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".

The pattern 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 pattern 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.

VI. Experimental Evaluation

To ensure that our experiments are reproducible, we have built a website [25] that contains all the data/code used in this work. All experiments were conducted on an Intel® Core i7-9700CPU at 2.80GHz with 16 GB of main memory, unless otherwise stated. As noted above, the format of this publication does not lend itself well to Mplots. We encourage the reader to visit [25] 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.

A. *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 Figure 6.*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.

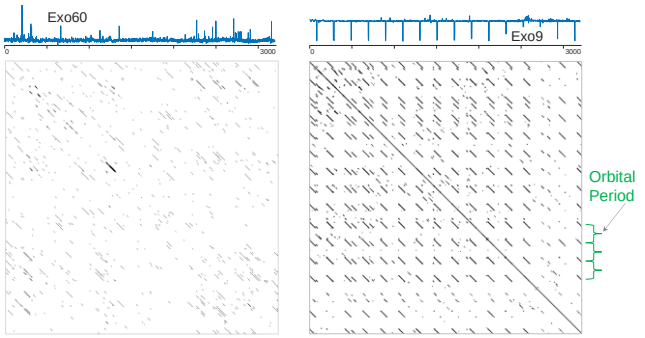

Figure 6: *top.left*) A star-light curve from a star believed not to have an exoplanet. *top.right*) A star light curve from a star *known* to have an exoplanet. *bottom.left*) The Mplot of the planetless star is relatively featureless. *bottom.right*) The Mplot of Exo9 reveals not only the existence of an exoplanet but tells us its orbital period.

As Figure 6 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 Figure 7, does it show an Exoplanet?

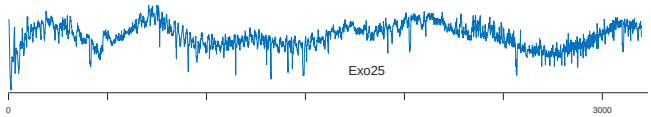

Figure 7: The star light curve for Exo25. Does it suggest the existence of an Exoplanet?

To answer this question, we built the Mplot in Figure 8.

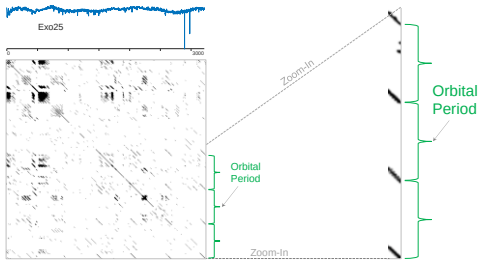

Figure 8: The star-light curve for Exo25 with its Mplot (m=100). While there is noise reflecting the original data's noise, there is the unmistakable signature of an exoplanet with an orbital period about three times longer than Exo9 (Cf. Figure 6.*bottom.right*).

In [25] we have a gallery of additional exoplanets discovered with this technique.

B. *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).

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 [20][21]. 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 [12] is a recent tool for discovering *contrasting patterns* across time series, that is, behaviors that are repeated within one time series but are absent from another. Since the Contrast Profile is defined "lego-like", by combining several Matrix Profiles, this suggests that its definition could be retroactively generalized to Mplots.

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

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

We adapt this to create the semantic definition we desire:

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

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

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

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

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

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

```
T1 = 0,   for i = 1 to length(DNAstring)
        if DNAstringi = A, then Ti+1 = Ti + 1
        if DNAstringi = C, then Ti+1 = Ti - 1
        if DNAstringi = G, then Ti+1 = Ti - 2
        if DNAstringi = T, then Ti+1 = Ti + 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 [7], so we will investigate whether there exist DNA subsequences shared between them, but which is absent from humans.

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

In the ContrastMplot shown in Figure 9, 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 [24] 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 [7]. 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.

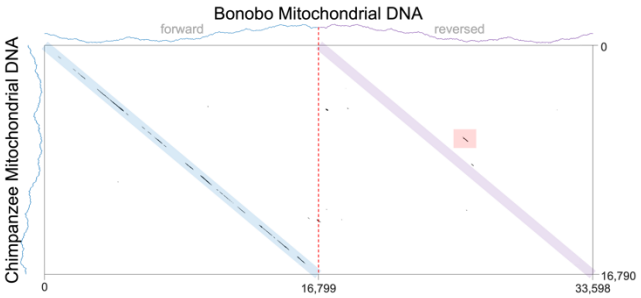

Figure 9: A Contrast-Mplot revealing mitochondrial DNA subsequences are shared between Bonobos and Chimps, but absent from Humans. The region highlighted in red indicates a reversed and offset Bonobo subsequence relative to the Chimp sequence.

C. *Finding Rescaled Motifs using PiecewiseSPLAT*

As we noted in Section IV.*C* 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 [21]. If the difference in scale is relatively small, say <8 to 20%, then there are a handful of techniques to address such cases [19]. However, here we are interested in motifs that may dramatically differ in scale, say up to 300%.

To discover such rescaled motifs we can search Mplots with PiecewiseSPLAT. Figure 10 illustrates the main insight.

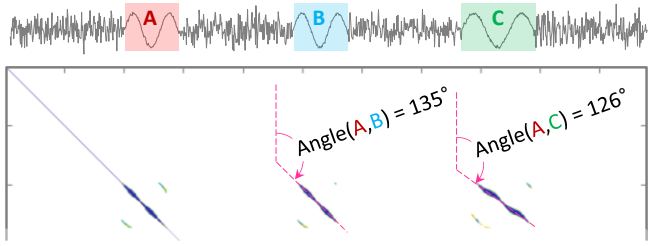

Figure 10: *top*) A toy time series with three sine-wave patterns embedded. Note that instance **C** is about 37% longer than the other two instances **A** and **B**. *bottom*) The corresponding Mplot shows that the difference in *lengths* manifests as a difference in *angle*.

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

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

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

Thus, we can reduce the rescaled motif discovery problem to the task of finding lines in an image, and that problem is easily solved by the classic Hough transform [1]. 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 Figure 11.

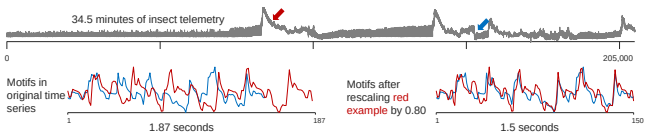

Figure 11: *top*) Telemetry from an insect pest feeding on a plant. *bottom*) A multi-scale motif discovered in the data can only be seen as conserved after one instance is rescaled by a factor of 1.25.

Here we see a motif discovered in telemetry from an insect. Because the two instances of this motif differ in length by a factor of 1.25, classic methods cannot find them [20].

D. Hunting for Chiroptera with PiecewiseSPLAT

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

For example, suppose a biodiversity survey needs to examine audio recorded at night to look for examples of bats. Existing bat classifiers have only been tested on a handful of the 1,400 known species [18]. We would like to have a *general* method to capture any species of bat. The problem is compound by the fact that many birds also sing at night.

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 a constant but changes, as [16] notes "*Over the course of an attack, bats increase call production rate*". This suggests an exploitable idea, we might expect that these changes in call rate would produce Mplot structures *not* parallel to the diagonal, as discussed in Section V. Consider Figure 12.*right*.

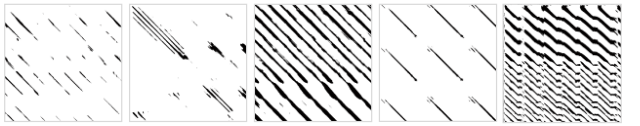

Figure 12: Five randomly chosen six-second snippets of animals that both fly and produce sound at night. The four leftmost examples are all birds. The rightmost example is a bat, which is unique here in having "stripes" that are *not* parallel to the diagonal.

These Mplot snippets are diverse, but note that the bird examples all have structure that is parallel to the diagonal. In contrast, the bat call is unique in that it has lines that are at an angle to the diagonal, telling us that the bat produced the motif twice, at two different speeds. Birds are only using sound to communicate[1], bats are using sound for a completely different purpose, and occasionally producing this unique feature.

To test our hypothesis, we embedded a twenty-second snippet of bat hunting audio into a one-hour audio file containing diverse bird songs. We searched for lines that had an angle of at least ±9.5° to the diagonal, indicating a rescaling factor of 1.40. As shown in Figure 13

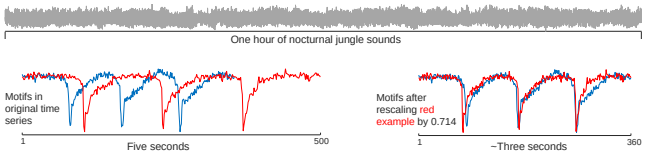

Figure 13: *top*) A one-hour dataset containing bird sounds, and a total of 20 seconds of bat sound. *bottom*) If we use PiecewiseSPLAT to search for motifs that have at least 1.35 rescaling, the top-1 motif is a bat vocalization.

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

E. 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 14 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 [4], 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.

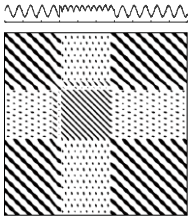

Figure 14: Regime changes produce *block-like* Mplots.

The result is a 1D function called the novelty function. The change point events are represented by local maxima (peaks) in the novelty function, which are then discovered with a peak finding algorithm. To test the utility of this algorithm we compared to three state-of-the-art semantic segmentation algorithms on a benchmark of 32 diverse datasets. We use the evaluation metric suggested by the creators of the datasets [5]. Table 6 summarizes the results.

Table 6: A comparison of Mplot with three SOTA algorithms

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

In interpreting these results note the following:

- Our algorithm is better than AutoPlait, about the same as HOG-1D, and worse, but not dramatically so, than FLOSS.
- We could have done better by tuning our algorithm, but to avoid *overtuning* we set *m* to be the same value as used by the authors of [5] for FLOSS. Thus, these results should be seen as a lower bound for SPLAT's performance.

SPLAT segmentation has an 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 Figure 15.*top*. As noted in [5], this problem cannot be solved by visual inspection. The ground truth 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.

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

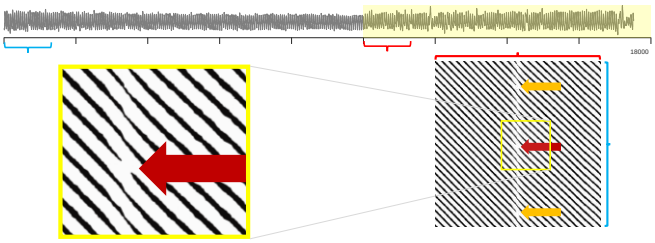

Figure 15: *top*) The PulsusParadoxus$_{SP02}$ segmentation problem is very subtle. *bottom.left*) A zoom-in of the Mplot close to the regime change revels a break in the diagonal streak. *bottom.right*) A zoom-out indicate that these breaks happen once in every eight beats.

However, there is an interesting clue as shown in Figure 15.*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 SPO2. 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. Mason, this is exactly what we are seeing here.

F. Speed and Scalability

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

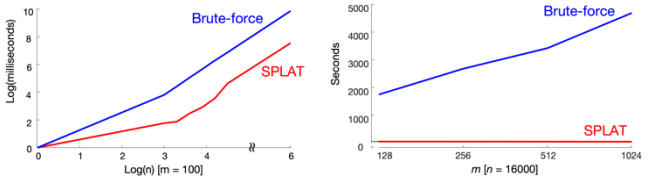

Figure 16: SPLAT execution time vs. brute-force algorithm – Note that both the left figure's axis are in log scale.

The reader will observe that we can compute a million length time series in about 9.5 hours using PiecewiseSPLAT. This is extremely fast given that the brute-force algorithm takes 5.4 years. The reader will appreciate we can further accelerate this by leveraging the hardware. To test this, we ported SPLAT to GPUs, and used a 1× Nvidia Tesla P100 GPU, finding that we can process a time series of length one million in just 6.2 seconds. We refer the reader to visit [25] for more results and the code.

VII. Conclusions

We introduced SPLAT, an algorithm that allows us to construct Mplots that are orders of magnitude larger than those that can be currently computed. We have shown that such Mplots can be used for tasks in domains as diverse as astronomy, medicine and biodiversity monitoring. We have made all code and data freely available to allow the community to confirm our results and build upon our ideas.

References

[1] R. O. Duda and P. E. Hart, "Use of the Hough Transformation to Detect Lines and Curves" in 1972, Commun. ACM, vol. 15, no. 1, pp. 11–15.

[2] J. Eckmann, S. Kamphorst, and D. Ruelle, "Recurrence Plots of Dynamical Systems," in 1987, Europhysics Letters (EPL), vol. 4, no. 9, pp. 973-977.

[3] Y. Fang, H. Xu, J. Jiang, "A Survey of Time Series Data Visualization Research," in 2020, IOP Conference Series: Materials Science and Engineering, vol. 782, no. 2, p. 22013.

[4] J. Foote and M. Cooper, "Media Segmentation using Self-Similarity Decomposition," in 2002, Proceedings of SPIE, vol. 5021.

[5] S. Gharghabi, et al, "Matrix Profile VIII: Domain Agnostic Online Semantic Segmentation at Superhuman Performance Levels," in 2017, IEEE International Conference on Data Mining (ICDM), pp. 117–126.

[6] A. J. Gibbs, A. J. Gibbs, and G. A. McIntyre, "The diagram, a method for comparing sequences: its use with amino acid and nucleotide sequences," in 1970, Eur J Biochem, vol. v. 16, no. 1, pp. 1-11.

[7] R. E. Green et al., "A Complete Neandertal Mitochondrial Genome Sequence Determined by High-Throughput Sequencing," in 2008, Cell, vol. 134, pp. 416-426.

[8] E. Keogh, K. Chakrabarti, M. Pazzani, and S. Mehrotra, "Dimensionality Reduction for Fast Similarity Search in Large Time Series Data Bases," in 2001, KAIS vol. 3, no. 3, pp. 263-286.

[9] J. Lin, E. Keogh, S. Lonardi, J. Lankford, and D. Nystrom, "VizTree: A Tool for Visually Mining and Monitoring Massive Time Series Data Bases," in 2004, Proceedings of VLDB, pp. 1269-1272.

[10] F. Malige, D. Djokić, J. Patris, R. Sousa-Lima, and H. Glotin, "Use of recurrence plots for identification and extraction of patterns in humpback whale song recordings," in 2020, Bioacoustics.

[11] N. Marwan, et al, "Recurrence Plots for the Analysis of Complex Systems," in 2007, Physics Reports, vol. 438, no. 5, pp. 237-329.

[12] R. Mercer, S. Alaee, A. Abdoli, S. Singh, A. Murillo, and E. Keogh, "Matrix Profile XXIII: Contrast Profile: A Novel Time Series Primitive that Allows Real World Classification," in 2021 (ICDM), pp. 1240-45.

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

[14] R. A. Phillipson, "Complex Long-Term Variability of X-ray Binaries and Active Galaxies Revealed by Novel Methods," American Astronomical Society Meeting #236, June. 2020, vol. 236, p. 122.02.

[15] T. Rakthanmanon, et. al., "Addresing Big Data Time Series: Mining Thrillions of Time Series Subsequences Under Dynamic Time Warping," in 2013, TKDD, vol. 7.

[16] J. M. Ratcliffe, C. P. H. Elemans, L. Jakobsen, and A. Surlykke, "How the bat got its buzz," in 2013, Biology Letters, vol. 9.

[17] B. Shneiderman, "The Eyes Have it: A Task by Data Type Taxonomy for Information Visualizations," Proc 1996 IEEE Symposium on Visual Languages, pp. 336-343.

[18] M. Tabak, K. Murray, J. Lombardi, and K. Bay, "Automated classification of bat echolocation call recordings with artificial intelligence," in 2021.

[19] D. Yankov, E. Keogh, J. Medina, B. Chiu, and V. Zordan, "Detecting time series motifs under uniform scaling," in 2007, Proceedings of the ACM SIGKDD, pp. 844-853.

[20] C.-C.M. Yeh, et. al., "Matrix Profile I: All Pairs Similarity Joins for Time Series: A Unifying View That Includes Motifs, Discords and Shapelets," in 2016, IEEE ICDM, pp. 1317-1322.

[21] Y. Zhu, et. al., "Matrix Profile II: Exploiting a Novel Algorithm and GPUs to Break the One Hundred Million Barrier for Time Series Motifs and Joins," in 2016, IEEE 16th ICDM, pp. 739-748.

[22] S. Zilberstein, "Optimizing Decision Quality with Contract Algorithms," in 1995, Proceedings of the 14th International Joint Conference on Artificial Intelligence, vol. 2, pp. 1576-1582.

[23] Z. Zimmerman et al, "Matrix Profile XIV: Scaling Time Series Motif Discovery with GPUs to Break a Quintillion Pairwise Comparisons a Day and Beyond," in 2019, in Proc' of the ACM SoCC.

[24] "BLAST: Basic Local Alignment Search Tool." https://blast.ncbi.nlm.nih.gov/Blast.cgi (accessed May 25, 2022).

[25] "Mplots" https://sites.google.com/ucr.edu/mplots/