Matrix Profile III: The Matrix Profile Allows Visualization of Salient Subsequences in Massive Time Series

Chin-Chia Michael Yeh, †Helga Van Herle, and Eamonn Keogh
University of California, Riverside, †University of Southern California
myeh003@ucr.edu, helga.vanherle@med.usc.edu, eamonn@cs.ucr.edu

***Abstract*— Multidimensional Scaling (MDS) is one of the most versatile tools used for exploratory data mining. It allows a first glimpse of possible structure in the data, which can inform the choice of analyses used. Its uses are multiple. It can give the user an idea as to the clusterability or linear separability of the data. It can help spot outliers, or can hint at the intrinsic dimensionality of the data. Moreover, it can sometimes reveal unexpected latent dimensions in the data. With all these uses, MDS is increasingly used in areas as diverse as marketing, medicine, genetics, music and linguistics. One of the strengths of MDS is that it is essentially agnostic to data type, as we can use any distance measure to create the distance matrix, which is the only required input to the MDS algorithm. In spite of this generality, we make the following claim. MDS is not (well) defined for an increasingly important data type, time series subsequences. In this work we explain why this is the case, and we propose a scalable solution. We demonstrate the utility of our ideas on several diverse real-world datasets. At the core of our approach is a novel Minimum Description Length (MDL) subsequence extraction algorithm. Beyond MDS visualization, this subsequence extraction subroutine may be a useful tool in its own right.**

Keywords*— *Time Series; Multidimensional Scaling; Visualization; Feature Extraction

I. Introduction

Multidimensional Scaling (MDS) refers to a family of related techniques used in exploratory data mining and information visualization. Given a set of N objects represented by their mutual distances in an $N \times N$ distance matrix. The MDS algorithm places each object in K-dimensional space such that the between-object distances are preserved as well as possible. Each object is then assigned coordinates in each of the K dimensions. K is often chosen to be 2, allowing an intuitive visualization of the data in a two-dimensional scatterplot as shown in Fig. 1.

MDS is one of the most versatile and commonly used tools in exploratory data mining. Its uses are myriad. For example, it can be utilized for music recommendation [1], under the assumption that if you like a particular artist, then you will probably like other artists that map to a nearby region of MDS space. It can also be used to quickly give a user an idea as to the clusterability of the data or its intrinsic dimensionality. One of the more interesting uses of MDS is finding *latent dimensions* in the data. For example, it has long been known that MDS of emotional artifacts (as expressed in tweets, facial expressions, song lyrics, political speeches etc.) often produce two latent dimensions: *Valence*, which refers to how positive or negative an event is, and *Arousal*, which reflects whether an event is exciting/agitating or calming/soothing.

MDS is useful for visualizing *time series data*, our data-type of interest. Fig. 1 shows individually *extracted* heartbeats from a time series taken from MIT-BIH Arrhythmia Database mapped to a 2D plane [2][3]. Note that while our goal is unsupervised MDS, in order to enhance the reader's appreciation of MDS for time series, here we use color to annotate the two classes labeled by original cardiologists [2].

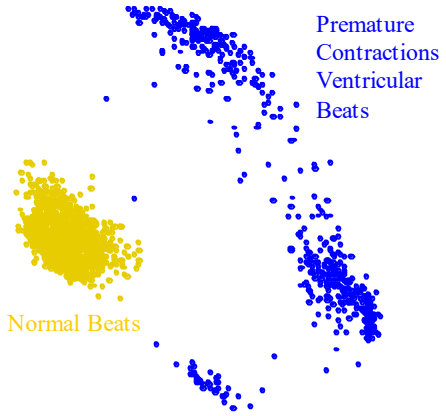

Fig. 1. A MDS plot generated from record 106 in MIT-BIH Arrhythmia Database [2][3]. Each point is a single heartbeat. The distance measure used is Euclidean distance. To enhance the reader's appreciation of MDS, here we use color to annotate the two classes labeled by the original cardiologists (best viewed in color).

This plot hints at the great utility of MDS. We showed the results to Dr. Van Herle, a noted cardiologist. She spent a few moments interacting with this plot (we have built a freely available tool that allows a user to click a point on the MDS plot and instantly see the corresponding time series in context [4]), and she discovered the following:

> The dense cloud of points at the left are indeed normal heartbeats. The remaining beats are ectopics (small changes in a heartbeat that is otherwise normal). The high variability of the remaining points can be explained in part because they are generated in different parts of the ventricle, so they are *polymorphic* Premature Ventricular Contractions (PVCs) with different morphologies depending on where in the ventricle they get initiated (Some people only have PVCs or ventricular arrhythmias that arise from one part of the ventricle all the time in which case they always look the same and are called *monomorphic*). However, the beats that form a cluster at the center top of the figure are mislabeled as PVCs. Instead they are *fusion beats*, something between a normal beat and a PVC. They are the result of a normal beat using a path other than the atrioventricular node to get to the ventricle.

This summary hints at the utility of MDS. At a minimum, instead of having to scroll through hours of ECG traces, a

cardiologist can glean an overview of the dataset, including the relative density of patterns, in just seconds.

However, this dataset had the individual beats carefully extracted by humans, *p*-wave to *p*-wave (aided by a carefully tuned beat-extraction algorithm) [2]. Suppose that this was not the case, could we still use MDS? One simple idea would be to randomly extract subsequences the length of a typical beat from the time series, without paying careful attention to the alignment. As we show in Fig. 2, the resulting MDS plot is reduced to a meaningless disk-like distribution.

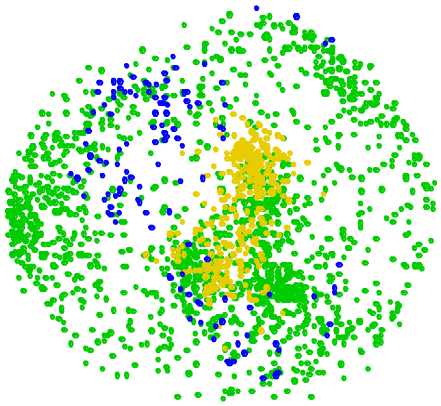

Fig. 2. A MDS generated from the same time series as in Fig. 1. Subsequences are randomly drawn from the time series. Yellow dots are subsequences contained normal beats, blue dots are subsequences contained ventricular premature beats, and green dots are subsequence contained neither.

At first blush this result is surprising. The data source is the same, and the lengths of time series subsequences are the same, so what caused the MDS plot to degenerate to what appears to be a random scatterplot? It is clearly not simply the *random sampling* of the ECG. Suppose we perform MDS on the famous Iris dataset [5]; then any large random sample will produce essentially the same MDS plot as the entire dataset[1].

One might imagine that instead of sampling the ECG we could just slide a window, the mean length of the heartbeats, across the time series and extract every subsequence, using them all as input into the MDS algorithm, however that produces results which are even worse [11].

In this work we will introduce an algorithm which can take in an arbitrarily long, unlabeled time series and produce intuitive and useful plots like the one shown in Fig. 1. A key observation that makes our algorithm possible is that we generally should not attempt to explain all the data in the plot, but rather only consider *salient* subsequences. We introduce a scalable and intuitive algorithm that exploits the Minimum Description Length (MDL) to extract such subsequences. We will demonstrate the effectiveness of our algorithm on cardiology, human activity, audio, seismic, and electrical power consumption time series.

The rest of this paper is organized as follows. In Section II we review related work and introduce the necessary definitions and background materials. In Section III we explain *why* time series subsequences cannot be simply plugged into a MDS framework, and we give our intuition as to how this problem may be solved, allowing us to formally introduce our algorithm in Section IV. We evaluate our algorithm with formal metrics and several diverse case studies in Section V, before offering conclusions in Section VI.

[1] This claim, and some similar claims made latter in this paper do not warrant the large amount of space to visually demonstrate them. However, in every case we have placed the relevant figures at [4].

II. Related Work and Background

A. Related Work

Since the early 1970's many researchers have shown that a two-dimensional MDS scatterplots of languages, dialects or accents often reveal latent dimensions that correspond to physical coordinates on Earth [6]. Note that in such cases the NSEW orientation can be flipped or rotated. Unsurprisingly, more recent explorations of genetic data also often reveal latent dimensions that are intrinsically spatial [7]. Note that spatial objects such as countries do not necessarily reveal latent dimensions that trivially encode their spatial distances. Instead, the latent dimensions depend on what "distances" are measured. For example, a classic (1970's) MDS demonstration using the perceived similarities of countries revealed latent dimensions that correspond to *Developed–Developing* and *Pro-western–Pro-communist* [8].

In spite of the large amount of research using MDS in such varied domains, we are not aware of any attempt to apply MDS on time series *subsequences*. Recall that Fig. 2 shows (and we will later explain *why*) that we cannot simply convert subsequences to whole time series.

In [9], the authors developed an MDS based visualization technique for multi-dimensional time series data. However, the objective is presenting relationship between "events", where an event is defined by the values of many different time series at that instant. In [10], the authors adopted self-organizing map (which can be seen as an alternative to MDS) to project subsequences of stock price time series to 2D plane. Because they only focused on stock price time series, they use a subsequence selection (or *interest point detection*) algorithm designed specifically for the domain and unlikely to generalize.

Note, the subsequence selection problem has applications beyond the visualization of time series subsequences, and variants of this problem have been considered before, usually in specialized domains. It is demonstrated in [11], clustering of time series subsequences is condemned to be meaningless if *all* subsequences are selected. This opens the chicken-and-egg paradox of only clustering subsequences that are clusterable. Various such methods for identifying *interesting* (or *salient*) subsequences are proposed over the years. Among them, [12] proposed a subsequence selection algorithm which identifies subsequences containing frequently appearing patterns under the MDL framework. The proposed method is both parameter free and shown effective in a diverse set of domains. As their algorithm is designed specifically for clustering, we cannot simply apply their subsequence selection algorithm in the subsequence visualization problem, nevertheless, this work is the starting point for our proposed algorithm.

More generally, the use of MDL in time series settings seems to be enjoying a significant increase in interest. For example, in a recent sequence of papers, Matsubara and collaborators have recently shown the general utility of MDL for a several time series problems including segmentation and anomaly-detection [13]. Keogh and colleagues have used MDL to tease out the *intrinsic* cardinality and dimensionality of time

series [14], to cluster from streams [12], and to predict when to stop labeling exemplars in a semi-supervised setting [15], and a team at Leiden University has applied MDL to a host of problems emanating from a single domain, telemetry from sensors on the Hollandse bridge [16].

B. Definitions and Notations

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

Definition 1: A *time series* T is an ordered list of numbers. $T = t_1, t_2, ..., t_n$ where n is the length of time series T.

We are interested in projecting *subsequences* into 2D space:

Definition 2: A *subsequence* $T_{i,m}$ of a time series T is a consecutive subset number sequence of length m which starts from position i. Formally, $T_{i,m} = t_i, t_{i+1},..., t_{i+m}$, $1 \leq i \leq n\text{-}m+1$.

We consider all possible subsequences (of a user chosen length) as candidates for projection into the MDS plot:

Definition 3: An all-subsequences set $\boldsymbol{A}$ of a time series T is an ordered set of all **possible** subsequences of T obtained by sliding a window of length m across T: $\boldsymbol{A} = \{T_{1,m}, T_{2,m}, ..., T_{n-m+1,m}\}$, where m is a user-defined subsequence length. We use $\boldsymbol{A}[i]$ to denote $T_{i,m}$.

As the space of all possible solutions for subsequence selection from an all-subsequence set is $O(2^{n-m+1})$, it is infeasible to evaluate all possible solutions in any reasonable sized time series. However, we can prune off a huge fraction of these possibilities by only considering subsequences that are similar to at least one other subsequence. We can efficiently search this reduced search space by exploiting a recently introduced data structure called the *matrix profile* [17]. For completeness we review its definition here.

Definition 4: A *matrix profile* P_A is a vector of the z-normalized Euclidean distances between each subsequence in an all-subsequences **set** $\boldsymbol{A}$ with their corresponding nearest neighbor within $\boldsymbol{A}$ (*trivial matches*, as explained in Fig. 7 must be excluded).

The i^{th} element in the matrix profile tells us the distance to the nearest neighbor of the subsequence of $T_{i,m}$. However, it does not tell us *where* that neighbor is located. This information is recorded in a companion data structure, the *matrix profile index*.

Definition 5: A *matrix profile index* I_A of an all-subsequences set $\boldsymbol{A}$ is a vector of integers where $I_A[i] = j$ if $\boldsymbol{A}[j]$ is $\boldsymbol{A}[i]$'s nearest neighbor.

As shown in Fig. 3 the matrix profile can be considered a "meta" time series which annotates a time series T. It has a host of interesting and exploitable properties, two of which we use in this work.

- The height of the matrix profile at location i reflects how far the subsequence of T beginning at i is from its nearest neighbor. As we will show in Section IV this suggests search order technique for our algorithm (lowest values first) and an admissible stopping strategy.
- As a practical matter [17] shows a host of techniques to allow the matrix profile to be efficiently computed. By exploiting these ideas we can handle very large datasets.

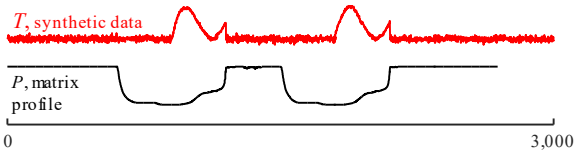

Fig. 3. A time series T, and its matrix profile P.

We will fully explain our algorithm in Section IV. At its heart is a search algorithm that must decide on which subsequences to place into the MDS matrix. As we will later show, this can be achieved with an MDL inspired cost function, however MDL requires *discrete* time series:

Definition 6: A discrete normalization function *DNorm* is a function that normalizes a real-valued $T_{i,k}$ into b-bit discrete value of range $[1, 2^b]$. The function is defined as:

$$DNorm(T_{i,m}) = round\left(\frac{T_{i,m} - min}{max - min} \times (2^b - 1)\right) + 1 \qquad (1)$$

where *min* and max are the minimum/maximum values in $\boldsymbol{S}$.

One might imagine that the dramatic reduction in precision by discretization will cause a loss of information. However, it has been empirically shown that this cardinality reduction has surprisingly little effect on both time series classification [18] and pairwise Euclidean distances [14]. To visually confirm this, in Fig. 4 we plot a heartbeat at various levels of precision.

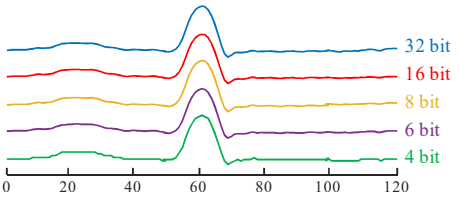

Fig. 4. A heartbeat represented at 32, 16, 8, 6 and 4-bit precision. It is difficult to notice the reduction in prediction until we vastly reduce the precision.

The MDL framework requires that we can compute the space needed to store a given (now) discrete time series:

Definition 7: The description length function *DL* is a function that calculates the number of bits required for storing a b-bit cardinality, length m time series T. The function is defined as:

$$DL(T) = m \times b \qquad (2)$$

For example, the number of bits required to store a 4-bit length 20 discrete time series is $4 \times 20 = 80$.

Because compression is at the heart of MDL, the MDL framework requires us to measure the space needed for lossless compression of the (now discrete) time series. In particular, we plan to use one time series as a *hypothesis*, and use it to encode one or more additional time series. A simple way to do this is to store the *positions* and *values* of only the differing elements.

Definition 8: A reduced description length function *RDL* is a function that calculates the number of bits required for storing a discretized b-bit length m time series T_C which is compressed relative to another b-bit length m time series T_H (T_C and T_H are referred as the *compressible* time series and *hypothesis* time series respectively). We denote this formally as:

$$RDL(T_C|T_H) = \gamma(T_C - T_H) \times (\log_2 m + b) \quad (3)$$

where $\gamma(\cdot)$ is a function that counts the number of non-zero element in the input vector (i.e., the number of mismatch element between T_C and T_H), $\log_2 m$ is the space required for storing the position of mismatch element, and b is the space required for storing the value of mismatch element.

Given the above definition, let us consider a toy example for our compression scheme. Given a 4-bit length 20 time series T_C:

T_C= 1 3 4 5 4 3 6 15 14 13 12 0 2 4 2 6 10 11 9 3

and a hypothesis 4-bit time series T_H with the same length:

T_H= 1 3 **3** 5 4 3 6 15 14 13 12 0 2 **3** 2 6 **4** **3** **1** **0**

Because there are 6 mismatches (highlighted) between T_C and T_H, the reduced description length of T_C after compression by T_H is:

$$RDL(T_C|T_H) = 6 \times (\log_2(20) + 4) = 49.93$$

In comparison to the uncompressed storage size (80 bit), our compression scheme achieves a compression ratio of 1.6. Note that in this toy example, only a single hypothesis time series is given. However, a *set* of hypotheses can be given when applying compression in the real system, thus we need additional space to store which hypothesis (among the given set of hypotheses) is used to compress the compressible. To address this issue, we generalize (3) into:

Definition 9: The general formulation for *RDL* which accounts for the additional information is:

$$RDL(T_C|T_H) = \gamma(T_C - T_H) \times (\log_2 m + b) + \log_2 h \quad (4)$$

where h is the total number of hypothesis in the given set of hypotheses $\boldsymbol{H}$. The term $\log_2 h$ is added to account for storing the position of hypothesis T_H in $\boldsymbol{H}$.

With the definitions listed above, we are ready to explain the cost function which assesses the total bit cost for storing all-possible subsequence set $\boldsymbol{A}$. As we mentioned before, our core algorithm is a search algorithm, and only a subset of $\boldsymbol{A}$ is explored at each search step. In other words, each subsequence in $\boldsymbol{A}$ is either explored and added to the compressible set $\boldsymbol{C}$, explored and added to the hypothesess set $\boldsymbol{H}$, or unexplored thus belongs to the *unexplored set* $\boldsymbol{U}$. The total bit cost function is formally defined as:

Definition 10: Given the compressible set $\boldsymbol{C}$, hypothesis set $\boldsymbol{H}$, and unexplored set $\boldsymbol{U}$, the total bit cost is computed as:

$$Bit(\boldsymbol{C}, \boldsymbol{H}, \boldsymbol{U}) = \sum_{T_C \in \boldsymbol{C}} \min_{T_H \in \boldsymbol{H}} RDL(T_C|T_H) + \sum_{T_H \in \boldsymbol{H}} DL(T_H) + \sum_{T_U \in \boldsymbol{U}} DL(T_U) \quad (5)$$

The total bit cost function simply adds the bit cost for storing each input set. Note, since only the compressible set $\boldsymbol{C}$ is compressed while both $\boldsymbol{H}$ and $\boldsymbol{U}$ is not compressed, we use (4) to compute the bit cost for $\boldsymbol{C}$ and we use (2) to compute the bit cost for $\boldsymbol{H}$ and $\boldsymbol{U}$.

Finally, we note that there are other possibilities to implement MDL for time series, including schemes that exploit Huffman coding, Shanon-Fano coding, run-length coding, delta encoding, etc. [13][14][16][18]. Moreover, one can consider the compression in the Fourier, Wavelet or piecewise polynomial space, etc. [14]. We have implemented and experimented with many of these variants, and they all produce very similar results. Thus in this work we use the simple formulation described above.

III. Observations on MDS for Time Series Subsequences

We begin by further enhancing the readers' intuition as to why MDS does not work for the time series subsequences (recall the contrast between Fig. 1 and Fig. 2). To do so we consider the discrete analogue of time series, *text*, and the discrete analogue of Euclidean distance, *hamming distance*.

Imagine we have a dataset consisting of just four words {`fat`, `cat`, `fog`, `dog`}. If we project the data into 2D space using MDS we obtain the plot shown in Fig. 5.

Fig. 5. Four words project into 2D space by MDS.

The plot is very intuitive. The rhyming pairs {`fat`, `cat`} and {`fog`, `dog`} are clustered to each side, with {`fat`, `fog`} slightly closer together than {`cat`, `dog`}, because {`fat`, `fog`} share the initial letter. This plot is a close analogue for what we saw in Fig. 1. However, this is not a perfect analogue to the task-at-hand, as we are interested in *subsequences*. Our task is more like the situation where we have a sentence such as:

```
a fat cat plays hide and seek in fog with dog
```

Even this example oversimplifies the problem, as the spacing provides exploitable clues, which are not generally available in real-valued time series. Thus let us consider:

```
afatcatplayshideandseekinfogwithdog
```

To perform MDS with this dataset, we can slide a window across the long string, extracting all subsequences of length three.

```
afa
 fat
  atc
   tca
    ...
```

This produces some spurious data, but we might imagine that MDS will take care of this in some way. Unfortunately, as we can see in Fig. 6 this is *not* the case.

The result of this experiment is somewhat disheartening. It is true that "`fat`" and "`cat`" did project to a similar location, but they are no closer than the spurious pairs {`atc`, `atp`} or {`nds`, `nfo`}. Even worse, "`dog`" and "`fog`" are no closer than

dozens of other pairings, including {eek, shi}, which do not even share a single letter. The situation is slightly better in 3D space, but that is of little comfort to us, as we want to make classic 2D MDS plots.

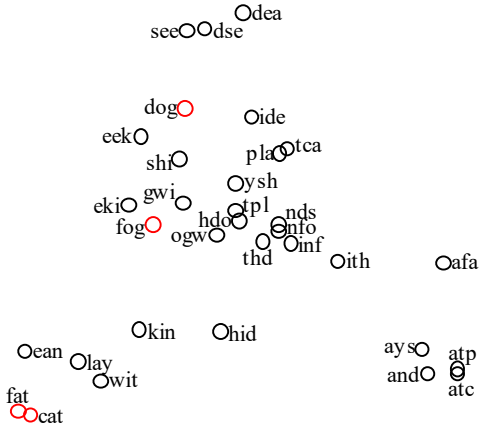

Fig. 6. The 2D MDS projection of all subwords of length 3 from afatcatplayshideandseekinfogwithdog.

Moreover, as poor that this result is, it actually understates the case. Under hamming distance sub-words such as "fat" have no correlation with the sub-words that are adjacent to it, in this case "afa" and "atp". In contrast, for *real-valued* time series, adjacent subsequences will have significant correlation (i.e. autocorrelation). Equivalently, adjacent subsequences will have a very small Euclidean distance. Such subsequences are widely known as *trivial matches*, and understood to have significant implications for several problems in time series data mining [19].

For concreteness, consider the subsequence highlighted in green beginning at location 4,000 in Fig. 7. This subsequence has a z-normalized Euclidean distance of (just under) 10.0 with both the purple and blue highlighted subsequences, and a Euclidean distance of less than 10.0 to *all* 698 subsequences in-between. In contrast, the red highlighted subsequence has only 27 such trivial matches within a distance of 10.0.

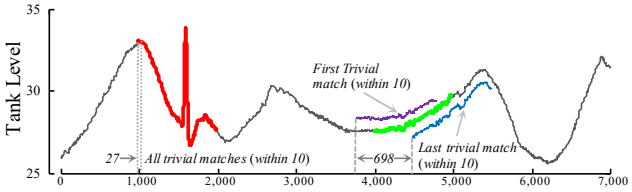

Fig. 7. A snippet of the LIC-002 from the delay cooking plant dataset.

One implication of this observation for our-task-at-hand (or any attempt at density estimation of subsequences) is that the green subsequence will appear to be near the center of a dense cloud, whereas the red subsequence will appear to live in a sparse region of the space. In general, the number of trivial matches a subsequence depends on the subsequence's *complexity* [20] with smooth/slowly evolving subsequences having many trivial matches, and noisy/fast evolving subsequences having very few.

One might imagine that we could solve this problem by somehow removing trivial matches, however no subsequence is itself *intrinsically* a trivial match; it is only a trivial match relative to other subsequences. This gives us a "chicken-and-egg" type subsequence selection problem that has been noted in the literature [11], but remains unsolved in the general case.

In summary, we claim that the MDS-for-*subsequences* problem is difficult, because even the much simpler MDS-for-*subwords* problem is hard, and the real-valued case has the additional issue of trivial matches and much greater dimensionality. Nevertheless, this exercise offers a suggestion for a solution to our problem. Our difficulties arise from trying to explain *all* the data, including subsequences that are redundant with other subsequences (*trivial matches*) and subsequences that are unique, perhaps because they correspond to the transitions between two pattern (i.e. the "atc" in "fatcat"). Both these issues can be solved by casting the problem in the MDL framework.

We have 35 characters in our string, each takes 5-bits to represent, giving us a total of 175 bits. To reduce this memory requirement, we could attempt dictionary compression. This is unpromising as there are no repeated subwords of length three or greater. However, we can consider one version of our rhyming pairs as being a model or *hypothesis* H_1 for the data and encode only the *difference* between the other occurrences. This would give us:

H_1 ={**X**:fog}
afatcatplayshideandseekin**X**___with$\mathbf{X}_d$__

In this case the hypothesis H_1 is only a good model for the subword "dog", as it is only here can we achieve compression after encoding the single differing letter[2]. If we continue the process, we would next find that...

H_1 ={**X**:fog}, H_2 ={**Y**:fat}
a**Y**___$\mathbf{Y}_c$__playshideandseekin**X**___with$\mathbf{X}_d$__

... also achieves some data compression.

After this we rapidly run out of options that can achieve compression. Note that the spurious pair {atc, atp} which mapped to adjacent points in Fig. 6 will not be extracted by this process, as they partly overlap (i.e. are *trivial matches*) with the two extracted hypotheses. Further note that if we hand over only the two hypotheses and the subsequences they encoded to MDS, then we are back to the successful plot shown in shown in Fig. 5.

This simple "hand-waving" (we limited the search space to our contrived patterns) example solves our toy example for the discrete case. In the next section we will provide a rigorous generalization to the real-valued time series case.

IV. MDS for Real-valued Time Series

We begin with a visual reiteration of the problem. As shown in Fig. 8, we are given just a time series T, and the user chosen length of patterns of interest m. From this input, we wish to extract all subsequences that are similar to at least one other (non-trivial) subsequence. We call this task the *subsequence selection* problem.

[2] Here we are ignoring the overhead costs of the dictionary/pointers etc. However, we have included the overhead in our MDL cost function as shown in (4).

Fig. 8. A time series that is an analogue of the string in Section III. We assign each latter a pattern and concatenate them to form the time series. The task at hand is to automatically extract the colored/bold subsequences, and (ideally) no other subsequences, with only the subsequences length m as an input.

In principle, **Definition 10** offers a solution. We can simply test all legal sets of subsequences, and choose the one that minimizes *the total bit cost*. However, this will be clearly intractable in general. Our solution to this relies upon the following two observations.

- As shown in Fig. 9, the Euclidean Distance is a very good proxy for the RDL measure, when the Euclidean Distance is small. Moreover, the Euclidean Distance is amiable to a host of optimizations that allow it to be computed many orders of magnitude faster.
- While finding the optimal subsequence set is intractable, in practice greedy search has been shown to produce high quality approximations for similar problems [18][14][13][22].

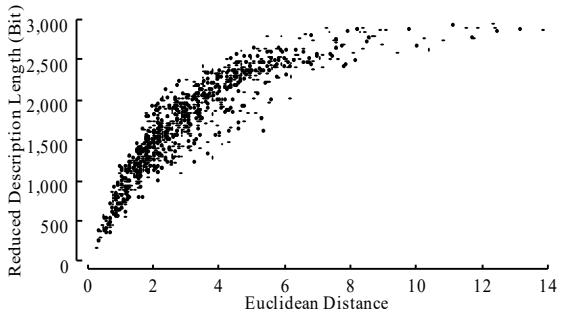

Fig. 9. A scatter plot created by taking random pairs A,B from the Strawberry dataset [21]. Each point represents ED(A,B) as the x-axis value, and RDL(A|B) as the y-axis value.

Thus, we use a Euclidean distance guided greedy search algorithm to solve the subsequence selection problem. In other words, the objective of our algorithm is to identify a compressible set $\boldsymbol{C}$ and a hypothesis set $\boldsymbol{H}$ within the all-subsequence set $\boldsymbol{A_T}$ of input time series T such that the total bit cost to store $\boldsymbol{A_T}$ is minimum.

As our solution is a greedy search algorithm, both $\boldsymbol{C}$ and $\boldsymbol{H}$ are initially empty. At each iteration, we identify the subsequence which provides the most benefit (in terms of lowering the total bit cost) if added as either a hypothesis or a compressible among a set of candidates, and add the subsequence to the corresponding set. The main algorithm is outlined in TABLE I, and its subroutines are outlined in TABLE II and TABLE III.

In the lines 1 and 2, the compressible set $\boldsymbol{C}$, hypothesis set $\boldsymbol{H}$, and unexplored set $\boldsymbol{U}$ are initialized. Both $\boldsymbol{C}$ and $\boldsymbol{H}$ are initially empty while $\boldsymbol{U}$ contains all-possible subsequences set $\boldsymbol{A_T}$. In line 3, the matrix profile *mp* and matrix profile index *mpi* are computed. The initial bit cost for the system is evaluated in line 4.

TABLE I. SUBSEQUENCE SELECTION ALGORITHM

Procedure *subsequenceSelection*(T, m, n_c, b)
Input: time series T, subsequence length m, number of candidate n_c, and number of bit b. **Output**: set of subsequences *Union*($\boldsymbol{C}$, $\boldsymbol{H}$)

```
1  C ← Ø, H ← Ø
2  U ← getAllPossibleSet(T, m)
3  [mp, mpi] ← STAMP(T, m)                          // [17]
4  bitCostOld ← Bit(C, H, U)                        // (5)
5  while True
6     cands ← pickCandidates(T, m, mp, b, n_c)      // TABLE II
7     [cand, type] ← pickBestCand(cands, H, mpi)    // TABLE III
8  remove cand (and its trivial matches) from mp
9     if type is hypothesis
10       H ← cand
11    else
12       C ← cand
13       bitCost ← Bit(C, H, U)                     // (5)
14       if bitCost > bitCostOld
15          break
16       end if
17       bitCostOld ← bitCost
18    end if
19    remove cand from U
20 end while
21 return Union(C, H)
```

Because both $\boldsymbol{C}$ and $\boldsymbol{H}$ are empty at this stage, the initial bit cost is simply the number of bits required for storing $\boldsymbol{A_T}$ uncompressed. At each iteration, *pickCandidates* (see TABLE II) is called in line 6 to select, extract, and discretize a set of candidates from $\boldsymbol{A_T}$. Next, in line 7, *pickBestCand* (see TABLE III) evaluate each candidate in the set of candidates and return the best candidate *cand* (in terms of reducing the overall bit cost of the system) and its *type* (i.e., compressible or hypothesis). Because we want to avoid extracting repeated subsequences in future iterations, *cand* (and its trivial matches) are removed from *mp* in line 8.

From line 9 to line 18, *cand* is added to either $\boldsymbol{C}$ or $\boldsymbol{H}$ based on its *type*. We only evaluate the terminal condition (line 13-17) when the *type* of *cand* is compressible because moving *cand* from $\boldsymbol{U}$ to $\boldsymbol{H}$ has little effect on the overall bit cost (bit cost of subsequences in $\boldsymbol{U}$ and $\boldsymbol{H}$ are both computed with (2) in (5)). Before moving to the next iteration, *cand* is removed from $\boldsymbol{U}$ in line 19 as *cand* is moved to either $\boldsymbol{C}$ or $\boldsymbol{H}$. Lastly, when the algorithm terminated, the union of $\boldsymbol{C}$ and $\boldsymbol{H}$ is returned for pairwise distance calculation and MDS algorithm.

TABLE II. IDENTIFY CANDIDATES

Procedure *pickCandidates*(T, m, mp, b, n_c)
Input: time series T, subsequence length m, matrix profile mp, number of bit b, and number of candidate n_c. **Output**: set of candidate *cands*

```
1  cands ← Ø
2  for i ← 1 to n_c
3     candIdx ← getMinIdx(mp)
4     cand ← T[candIdx:candIdx+m-1]
5     cand ← DNorm(cand, b)
6     cands ← cand
7     remove cand (and its trivial matches) from mp
8  end for
9  return cands
```

To select a set of candidates *cands* from all-possible subsequence set of T, we begin by select the lowest point in the matrix profile *mp* as shown in TABLE II line 3. This is heuristic choice based on our observation, illustrated in Fig. 9, that the Euclidean distance is an excellent proxy for RDL when the Euclidean distance is relatively small.

Next, the subsequence *cand* which corresponds to the lowest point is extracted, discretized, and added to *cands* in line 4-6. To avoid adding redundant subsequences (and trivial matches) to *cands*, we remove them from *mp* by replacing their corresponding *mp* values with `INF` in line 7 (similar to TABLE I line 8). After the loop is repeated n_c times, *cands* is populated with n_c discretized subsequences for returning in line 9.

TABLE III. IDENTIFY THE BEST AMONG CANDIDATES

Procedure *pickBestCand* (*T*, *cands*, ***H***, *mpi*)
Input: time series *T*, candidate *cands*, hypothesis set ***H***, and matrix profile index *mpi*.
Output: best candidate *bestCand*, and its type *bestCandType*

```
1   bestCand ← Ø, bestCandType ← Ø
2   bestBitsave ← -inf
3   for each cand in cands
4     // test as hypothesis
5     nn ← getNN(T, cand, mpi)
6     nn ← DNorm(nn, b)
7     bitsave ← DL(nn) – RDL(nn|cand)
8     if bitsave > bestBitsave
9       bestBitsave ← bitsave
10      bestCand ← cand
11      bestCandType ← hypothesis
12    end if
13    // test as compressible
14    for each hypo in H
15      bitsave ← DL(cand) – RDL(cand|hypo)
16      if bitsave > bestBitsave
17        bestBitsave ← bitsave
18        bestCand ← cand
19        bestCandType ← compressible
20      end if
21    end for
22  end for
23  return bestCand, bestCandType
```

Because we want to identify the best subsequence which benefits the system the most when added to either ***C*** or ***H*** within the set of candidate *cands*, we use the subroutine outlined in TABLE III to evaluate each candidate's ability to reduce the bit cost of the system, then return the best. At the beginning of the main loop (line 4-12), the subroutine tests a candidate *cand* as a hypothesis. We evaluate a subsequence's ability as hypothesis by computing the potential bit save provided by the subsequence. Specifically, we use *cand* to compress its nearest neighbor and calculate the before and after bit difference as shown in line 7. Then, we update the *best-so-far* when the bit difference is larger than the current best-so-far. Next, from line 13-21, the subroutine tests a candidate *cand* as a compressible. The test for compressible is straightforward, we just examine the bit difference when compress *cand* with each hypothesis in ***H***, and update the *best-so-far* based on the returned bit difference. Once the subroutine finished the candidate evaluation, the *best-so-far* and its type are returned in line 23.

A. Time and Space Complexity

Classic MDS is $O(N^3)$, where N is the number of extracted subsequences by algorithm shown in TABLE I and usually $N << n$. This would be untenable if we had to consider all the subsequences of a long time series, but recall that we only pass to MDS the relatively few subsequences extracted by the algorithm in TABLE I. *Given* the matrix profile (i.e., line 3 in TABLE I is precomputed), the time complexity is $O(N^2 n_c \max(m,\log n)+n)$, and the space complexity is $O((N+n_c)m+n)$.

This only leaves the time complexity of computing the *matrix profile*. This is naively $O(n^2 \log n)$ and our major time bottleneck. However, several factors that greatly mitigate this. As [17] shows, it is possible to produce a *high* quality approximation of the matrix profile two orders of magnitude faster than the *exact* computation. More significantly, when recording data for which you know *in advance* that you want to obtain the matrix profile of, you can compute it incrementally as the data arrives. In such cases, time complexity for the matrix profile computation is effectively zero, and we can produce MDS plots "on-demand", in just a few seconds.

To concretely ground these values, consider the case study in Human Activity Analysis (Section V.C), were we can compute everything (including exact matrix profile) an order of magnitude faster than real-time. Even for our largest dataset, the seismology case study (Section V.E), which has 1.7 million data points, we can still compute everything faster than real time.

V. EXPERIMENTAL EVALUATION

We have built a webpage [4] which contains all datasets and code used in this work, together with spreadsheets which contain the raw numbers. Given page limits, and the large amount of space required for a MDS plot, we have placed additional images and experiments at [4].

A. Effectiveness of Subsequence Extraction

We begin by examining the effectiveness of subsequence extraction, independent of our goal of MDS. To do this we consider all the datasets in the UCR Archive [21]. In our first experiment we do the following. For each dataset we place all the objects into a "bag" with an equal number of (z-normalized) random walk time series. We then task our extraction algorithm with extracting *just* the original exemplars. This is a challenging problem; a pair of random walks will often happen to be very similar to each other chance, and many of the original UCR datasets are very noisy.

Our algorithm can make two kinds of mistakes, not retrieving a true object (false negative) or by extracting a random walk (false positive). Thus, we report F-measure, which is commonly used in information retrieval problems to summarize precision and recall. Note, in practice, maximizing precision when the number of retrieved item is above certain threshold would be a better performance measurement for our task (visualization with MDS). We still choose to report F-measure in this section to showcase that our subsequence selection algorithm is capable of obtaining reasonable F-measure for other tasks (e.g., clustering, semi-supervised learning, rule finding). In the interests of space we defer the full annotated table of results to [4], and in Fig. 10 show a visual summary of the results on four randomly chosen datasets.

Again, recall that this problem is a good proxy for our task-at-hand, but actually harder, thus we do not need or expect to achieve perfect F-measures.

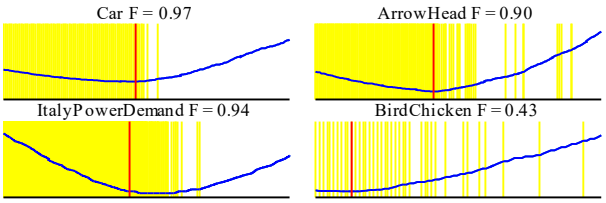

Fig. 10. A visual summary of the effectiveness of our algorithm in *the whole sequence* setting. The blue curve shows the total bit cost at each iteration. The red line is the threshold chosen using the total bit cost curve. The yellow stripes indicate iterations where true positives are retrieved. The F-measure computed using the threshold for each dataset is displayed in the title.

A perfect algorithm would create plots with a solid "wall" of yellow to the left of the red vertical line, and no yellow stripes to the right of it.

These results are very promising, but the setup is arguably too contrived, as all the data are already *whole* sequences, not *subsequences*. As shown in Fig. 11, to make the problem more closely model the task-at-hand, we randomly concatenated the true data objects with random walks.

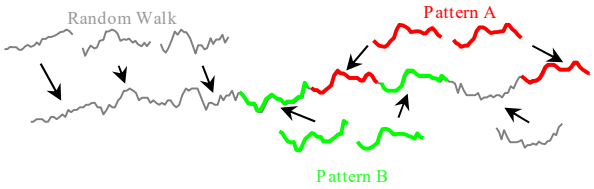

Fig. 11. A toy example for sequences concatenation. In this example, we randomly concatenated 4 random walk sequence and 4 pattern sequences from ItalyPowerDemand dataset.

This variant means we have to slightly modify our scoring function, as we may extract what is essentially a true object, but is shifted slightly and thus contains a few elements of random walk. We will denote any extracted subsequence that is at least 80% a true object is a true positive. As Fig. 12 shows, this more realistic setting only slightly decreases performance.

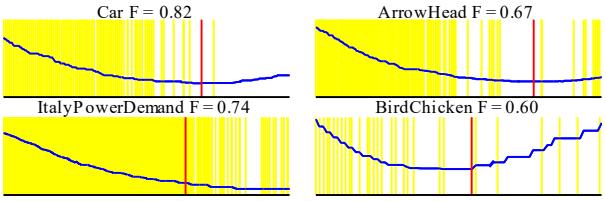

Fig. 12. A visual summery of the effectiveness of our algorithm in the *subsequence* setting. See the caption of Fig. 10 for key to these plots.

Note, the F-measure of BirdChicken increased slightly when moving to the harder case. This seems to be due to the fact that shift invariant distance measurement is required for identifying correct sequence for this dataset. As the solution space for subsequence selection problem also contains the shifted version of each subsequence (i.e., trivial matches), shift invariant is enforced by allowing matches between shifted version of each subsequence. Overall, more than half of the dataset we tested have F-measures greater than 0.5.

To further demonstrate the utility of our ideas, in the following sections we consider some case studies in domains for which we have access to some external data or expertise that allow a post-hoc evaluation of the results. As shown in Section III, the key to produce a successful MDS plot is to reduce the amount of irrelevant objects (i.e., maximizing precision). We only report precision for the following sections.

B. A Case Study in Heart Beat Analysis

Here we revisit the issue of ECG time series used in Section I. Instead of using the ground truth segmentation, we applied our subsequence selection algorithm to unsegmented data. The resulting MDS plot is shown in Fig. 13. Note that the four subsequences labeled as false positives are probably *true* positives that the original cardiologists neglected to annotate [2], however we retained the original labels, and the precision is 0.99.

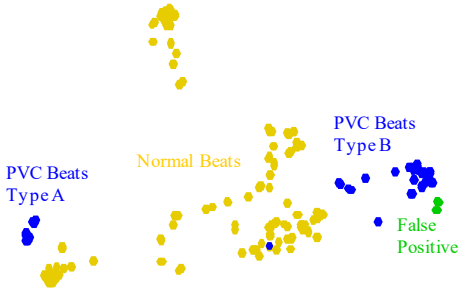

Fig. 13. The MDS plot of the heart beat dataset. Note that the PVC beats are polymorphic, forming two very distinct clusters, whereas normal beats form a single cluster. Aside from the two false positives, there is also one PVC beat is misplaced within the cluster of normal beat. We have colored the dots with the *original* data annotations [2].

We showed the results to Dr. Van Herle, a noted cardiologist. She agreed with the original PVC vs. Normal classifications, but noted the MDS plot had revealed an unexpected finer level of detail. To lightly paraphrase:

> While **A** and **B** are both PVCs, their morphology (which is related to where in the ventricle they initiate) are different. It appears that type **B** is a right bundle branch pattern, coming from right side of the heart, and Type **A** is more likely to be the of the fusion of a normal beat and an aberrant beat. Moreover, there is also evidence of a retrograde *P*-wave in type **B**.

This observation hints that our subsequence selection algorithm preserved the utility of MDS. The MDS plot provides a meaningful overview plot of the dataset rather than meaningless plot as shown in Fig. 2.

C. A Case Study in Human Activity Analysis

We examined the activity dataset of [22]. This dataset consists of a 13.3 minute 10-fps video sequence of an actor performing various activities. From this data, the original authors extracted 721 channels of optical flow time series, and each time series' length is 8,000. We consider only one randomly chosen time series, which as shown is Fig. 14 is suggestive of structure in places, but very noisy.

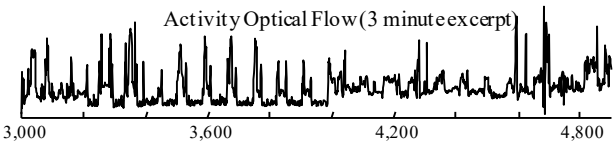

Fig. 14. A three-minute snippet of the human activity dataset.

Note that unlike motion-capture or accelerometer data, optical flow does not have an obvious intuitive physical

meaning; it is simply the case that if two actions are *visually* very similar, some subset of their time series representations should also be similar [22].

Given the full dataset we use our algorithm to produce an MDS plot of 12-second subsequences, Fig. 15 shows the result, annotated by a post-hoc examination of the full video. The precision for this dataset is 1.0. Gratifyingly, the results do not show any false positives, but there are false negatives in the sense that there are activities which were not extracted. However, it is the nature of optical flow features that they are localized in space, thus it is not surprising that we do not find behaviors that manifest in other regions. However, by choosing other optical flow time series we *do* find such features, we relegate such examples to [4].

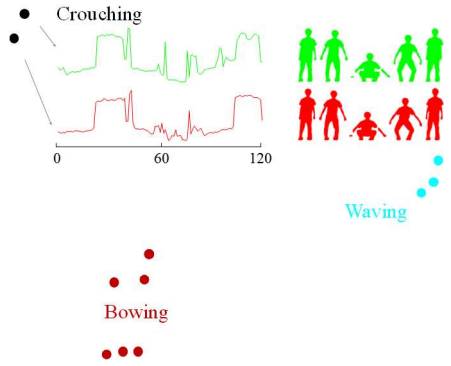

Fig. 15. The MDS plot for the activity dataset for 12-second queries. In the top-left, two data points are annotated by showing the extracted subsequences and film strips of the binarized video data.

D. A Cast Study in Audio

We examined a short audio sequence consisting of the (sung with piano accompaniment) nursery rhyme, *London Bridge is Falling Down* [23]. We converted this 203 second-long audio file into an MFCC representation and considered only the 2nd coefficient. We use a subsequence length of 200 (two seconds). Fig. 16 shows the resulting MDS plot.

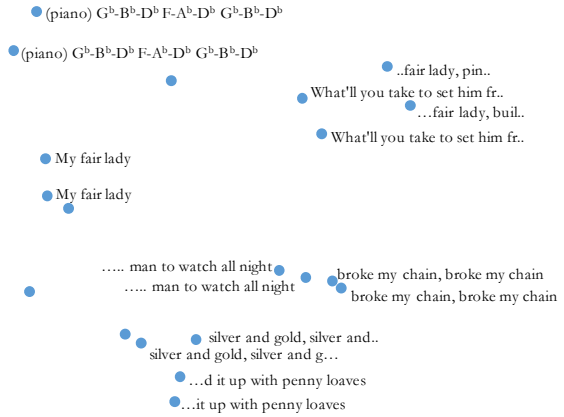

Fig. 16. An MDS plot of a time series extracted from the 2nd MFCC of a performance of *London Bridge is Falling Down*. Selected subsequences are annotated by examining the original data.

The reader familiar with the song will note that the results are very good, but not perfect. There are a handful of rhyming phrases missed. However, recall that we only considered one of the twelve MFCC coefficients.

E. A Case Study in Seismology

We applied our algorithm to 24 hours of seismic data from Mammoth Lakes, a geologically active region that lies on the edge of the Long Valley Caldera in Northern California. The data was recorded on February 17th, 2016, and the sampling rate is 20 Hz. It may not be obvious to the reader that such data could contain repeated patterns. Recall that earthquakes are waves, and as such are subject to both reflection and refraction as they travel from source to seismograph. As this path may consist of many different substrates with greatly differing acoustic properties, repeated earthquakes from the same location typically have an acoustic signature that is highly conserved. We tested our algorithm with a query length of 2.5 seconds, and the resulting MDS plot is shown in Fig. 17.

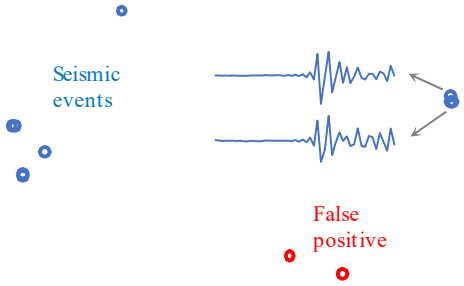

Fig. 17. An MDS plot from the seismic data recorded at a station near Mammoth Lakes on February 17th, 2016. True positives are labeled as blue and false positives are labeled as red. We slightly jittered the data to allow visualization of points that perfectly coincide.

The extracted subsequences form four clusters in the MDS plot. By referring to the associated annotation of the seismic data, we can assess whether a selected subsequence contains true seismic events (i.e., earthquakes). Out of the eleven selected subsequences, nine of them *definitely* contain seismic events (the precision is 0.82), while the two false positive subsequences form a cluster by themselves. It may be slightly disappointing that we have discovered a false positive, but recall that we searched a dataset of length 1,728,000, thus there are approximately 1.5 trillion pairs of subsequences that might happen to be similar just by chance.

F. A Case Study in Power Comsumption

To further illustrate the effectiveness of our method, we tested our method on a power consumption dataset from [24]. We consider the fridge freezer telemetry from the entire year of 2014, examining 1,8 million data points. The subsequences length was set to 1,325 (i.e. about eight hours). Fig. 16 shows the resulting MDS plot.

Note that subsequences from summer are distributed widely along x-axis while the ones from fall spread widely along y-axis. In contrast, subsequences from spring and winter form dense clusters near the center with spring being slightly more spread. When we examined the subsequences, we found that the periods of the repeated pattern (i.e., a rectified square wave) are longer in summer and autumn compared to spring and winter. The off-center subsequences of spring are from the end of spring, and they are visually similar to subsequences from summer.

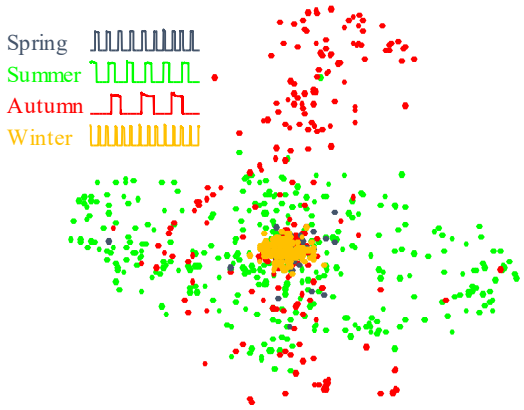

Fig. 18. The MDS plot for the power consumption time series of a fridge freezer with 8.5 hours queries during the year of 2014. The subsequences are colored based on the season[3]. Examples of extracted time series for each season are plotted at the upper right corner.

While the relationship between the subsequence patterns and the season is striking, it is important to note that this is the result of a "quick-and-dirty" mapping to standard calendar division. We did not consider the actual weather at the location of interest, nor did we have access to any confounding factors (i.e. school breaks or religious holidays) that might further explain some of features of the plot.

VI. Discussion and Conclusions

Our case studies offer strong support for our claim that "*we generally should not attempt to explain all the data, but rather only consider salient subsequences*". For example, in the seismology and power consumption datasets we only explain 0.0127% and 3.5% of the data respectively.

The power consumption dataset also hints at a useful technique that we wish to further explore. Once we project plots into 2D space, it may be useful to color them by some other variable that was recorded in parallel. This idea is called *brushing* in the visualization community [25]. For power consumption we brushed on *calendar* information, however the technique could be more general than that.

We only considered Euclidean Distance as the underlying distance measure in our work. It is known that Dynamic Time Warping (DTW) can outperform Euclidean Distance for some data mining tasks [26], however several independent research efforts have suggested that performance gap between the two measures quickly diminishes for pair-wise problems (joins, motif discovery, clustering) as we consider larger and larger datasets [26].

Our work has one possible limitation in that we are assuming that *salient* subsequences are subsequences that are approximately repeated, at least once. This idea, that *conserved* patterns are most likely to be of interest is a fundamental assumption in bioinformatics, and has growing acceptance in time series applications [12][13][16][19]. Nevertheless, it may be interesting to augment our work to allow definitions of saliency based on other criteria. One idea would be to augment our current plots by also projecting the most *interesting*, most *novel* [26] or most *surprising* patterns under some suitable definition. However, such definitions will almost certainly need to be domain dependent.

[3] Spring is from March 2014 to May 2014, summer is from June 2014 to August 2014, fall is from September 2014 to November 2014, and winter is from December 2014 to February 2015.

Acknowledgment

We thank Dr. Helga Van Herle for her medical expertise, Dr. Sang-Hee Lee for her help in music annotation and Dr. Gareth Funning for providing and helping to annotate the seismology data.

References

[1] J. C. Platt, "Fast embedding of sparse similarity graphs," In *NIPS* 2003
[2] A. L. Goldberger et al., "PhysioBank, PhysioToolkit, and PhysioNet: components of a new research resource for complex physiologic signals," *Circulation* 101(23):e215-e220, 2000.
[3] G. B. Moody and R. G. Mark, "The impact of the MIT-BIH arrhythmia database," *IEEE Eng in Med and Biol* 20(3):45-50, 2001.
[4] Supporting page: http://www.cs.ucr.edu/~eamonn/MatrixProfile.html.
[5] R. A. Fisher, "The use of multiple measurements in taxonomic problems," *Annals of Eugenics*, 1936.
[6] P. Black, "Multidimensional scaling applied to linguistic relationships," In *Cahiers de l'Institut de Linguistique Louvain*, vol. 3, pp. 13-92, 1973.
[7] L. B. Scheinfeldt, S. Soi and S. A. Tishkoff, "Working toward a synthesis of archaeological, linguistic, and genetic data for inferring African population history," *Proc Natl Acad Sci USA,* 8931–38, 2010.
[8] J. Kruskal and M. Wish, *Multidimensional Scaling*, Sage Publications, Beverly Hills, 1978.
[9] D. Jäckle, F. Fischer, T. Schreck and D. A. Keim, "Temporal MDS plots for analysis of multivariate data," *IEEE Transactions on Visualization and Computer Graphics*. vol. 22, no. 1. pp. 141-150. Jan. 31 2016.
[10] F. Wanner, et al."Integrated visual analysis of patterns in time series and text data - Workflow and application to financial data analysis," *Information Visualization*, 2015.
[11] J. Lin, E. Keogh and W. Truppel, "Clustering of streaming time series is meaningless," in *Proceedings of the 8th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery*, 2003.
[12] T. Rakthanmanon, E. Keogh, S. Lonardi, and S. Evans. "Time series epenthesis: clustering time series streams requires ignoring some data" *IEEE ICDM* 2011.
[13] Y. Matsubara, Y. Sakurai and C. Faloutsos, "AutoPlait: automatic mining of co-evolving time sequences," *SIGMOD*. 2014.
[14] B. Hu, T. Rakthanmanon, Y. Hao, S. Evans, S. Lonardi and E. Keogh, "Discovering the intrinsic cardinality and dimensionality of time series using MDL," *2011 IEEE 11th International Conference on Data Mining*
[15] N. Begum, B. Hu, T. Rakthanmanon and E. Keogh, "A Minimum Description Length Technique for Semi-Supervised Time Series Classification," *IRI 2013*: 171-192.
[16] U. Vespier, A. Knobbe, S. Nijssen and J. Vanschoren, "MDL-based Analysis of Time Series at Multiple Time-Scales." *ECML-PKDD*. 2012.
[17] 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," *ICDM* 2016.
[18] M. Shokoohi-Yekta, Y. Chen, B. Campana, B. Hu, J. Zakaria and E. Keogh, "Discovery of meaningful rules in time series," *ACM SIGKDD* 2015.
[19] B. Y. Chiu, E. J. Keogh and S. Lonardi, "Probabilistic discovery of time series motifs," *ACM SIGKDD* 2003: 493-498.
[20] G. E. A. P. A. Batista, X. Wang and E. J. Keogh, "A complexity-invariant distance measure for time series," *SDM* 2011: 699-710.
[21] Y. Chen, E. Keogh, B. Hu, N. Begum, A. Bagnall, A. Mueen and G. Batista, *The UCR Time Series Classification Archive*. 2015.
[22] A. Veeraraghavan, R. Chellappa and A. Roy-Chowdhury, "The function space of an activity," *Computer Vision and Pattern Recognition*. 2006.
[23] London Bridge (poem) www.youtube.com/watch?v=9VIGQCbAgL0
[24] D. Murray et. al., "A data management platform for personalised real-time energy feedback," *In Proce of the 8th International Conference on Energy Efficiency in Domestic Appliances and Lighting*, 2015.
[25] M. A. Hearst, "User interfaces and visualization," *Modern Information Retrieval*.
[26] H. Ding et al., "Querying and mining of time series data: experimental comparison of representations and distance measures," *PVLDB* 2008
[27] E. J. Keogh, S. Lonardi and B. Y. Chiu, "Finding surprising patterns in a time series database in linear time and space," *KDD*, 2002.