DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Visualization of Semantically Meaningful Representations of Large Multi-dimensional Time Series on Supercomputers

Gabrielle Poerwawinata

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Visualization of Semantically Meaningful Representations of Large Multi-dimensional Time Series on Supercomputers

Visualisierung von Mustern aus großen Multi-dimensionalen Zeitreihen auf Supercomputern

Supervisor:	Prof. Dr. rer. nat. Martin Schulz
Advisors:	Amir Raoofy, M.Sc., Alessio Netti, M.Sc.
Author:	Gabrielle Poerwawinata
Chair:	Computer Architecture and Parallel Systems
Submission:	15th of July 2020

I confirm that this master's thesis in informatics is my own work and I have documented all sources and material used.

Garching, 15th of July 2020 Gabrielle Poerwawinata

Acknowledgments

I want to thank Amir Raoofy for his dedication, patience and excellent guidance in the duration of this thesis. Further, I believe this thesis would have been incomplete without support from Allessio Netti with his hands-on technical advice, R. Karlstetter for his insight on gas turbine sensors, and E. Mayer and T. Odaker with Unreal Engine tutorial.

I would also like to express my gratitude to Prof. Martin Schulz for giving me the opportunity to work in the CAPS chair. Working on this topic has been quite a challenge but also an excellent oportunity.

Many thanks to M. Maiterth and D. Tafani for their encouragement in pursuing a thesis work in this chair. Finally, I want to thank my parents, siblings and friends(Vikas, Setiadi, Auliya, Mustika, Stefan, Aqid, Pinar and many more) for their continous support.

Abstract

Visualizing high-dimensional data is a cornerstone in scientific data mining. The meaningful representation of knowledge discovery in time series helps the expert to understand their data behavior. In high-dimensional data, visualization often becomes more prominent as it can help the user in identifying the patterns, classifications, and relationships within contributing variables.

In time series analysis, Euclidean distance based time-series similarity plays a big role. A novel method based on Euclidean distance so-called "matrix profile", has demonstrated use in pattern analysis, clustering, rule discovery. Its capability to be computed in parallel has distinguished its performance compare to other techniques. It enables us to work with many sensors and very long time-series data.

Due to the irregularities of the sensors data, we use an unsupervised approach to infers the feature with no specific application domain. Therefore, the developed model should be able to be applied for any kind of time series data. Furthermore, additional study is included to infer the background of the data source. In many data investigations, the background study helps in developing a hierarchical systematic which guides the user to narrow down the exploration space for a certain aspect of analysis.

After information extraction, visualization as the terminal of this study: research on insightful ways to show time series similarity, segmentation, and data clustering has to be performed. As there are many ways to visualize the data in design, information delivery, and engagement with the users, we limit the research work on the first two parts. In a small section, we also review the opportunities to create a more interactive way of visualization.

Keywords: data mining, high dimensional, time series similarity, matrix profile, data visualization, HPC performance counters, gas turbine.

Contents

1. Introduction

This chapter summarizes the current state of time series mining, high dimensional visualization and the thesis objectives itself.

1.1. Fundamental of Time Series Mining

Measurements or value reading over time lead to a collection of organized data called time series. Major time series mining-related tasks include query by content, anomaly detection, motif discovery, prediction, clustering, classification, segmentation, and anomaly detection [1]. Nowadays, time series analysis covers real-life problems in various fields of research. The increasing use of time series data has led to a great deal of interest and development in extracting meaningful information using various data mining tools. Extracting useful insight from time-series mining, however, includes numerous facets of complexity. Particular difficulties arise from time-series dimensionality and similarity measures method selection.

With the rapid growth of sensors reading, time series mining algorithms need to adapt increasingly massive datasets. For instance, in the case of the Internet of Things (IoT), an increasing number of powerful sensors and context information has made it important to perform such time series analysis more efficiently by considering all available computational resources. Some attempts to compute the high dimensionality of time series in parallelizing mode has been done in [2], [3].

Basics of time series analysis include similarity finding. Two main approaches are known, Euclidean Distance and Dynamic Time Warping. Based on the computed distance, further steps are needed in recognizing the recurring pattern and partitioning time series into regimes. Another interesting aspect of time series mining is to exploit their essential motifs. Time series motifs itself are pairs of subsequences. Understanding the highly recurring motifs in time series is beneficial in building predictions or clustering.

1.2. High Dimensional Data Visualization

The term data visualization expresses the idea to represent information in a graphical form (instead of using a table). In high-dimensional data, information needs to be transformed into datapoint which consists of n-dimensional (row-)vectors of numerals. A set of points, or datasets are arranged in a matrix in which the number of columns is equal to the number of dimensions. Each row displays one entry of an n-dimensional point [5]. Later, data transformation corresponds to dimensionality reduction, regression, subspace clustering, feature extraction, data sampling, and abstraction. The results in mining the datasets should be valid and concise before going to be mapped and transformed visually.

Visual mapping focuses on visual encodings based on axes (e.g scatterplots, parallel co-ordinate plots), pixels, together with animation and perception. View transformation focuses on-screen space and rendering for various visual structures as well as screen space measures for reducing clutter or artifacts and highlighting important features [6]. After the validation of the visual mapping, the process can be continued further using more advanced technologies and involving more perspectives, such as using AR, VR, and human interaction [7].

1.3. Thesis Objectives

The motivation of this thesis is to prove that we can obtain information insight and to create a visualization design based on the matrix profile output analysis. We want to see whether the output works accurately towards large multidimensional time series. The main task is focused on producing essential time series subsequences (motifs) and region segmentation, regardless of the data domain knowledge. Due to no domain aspect, we evaluate our analysis within two data sources which are HPC's performance counters and gas turbines. To retrieve important characteristics mentioned above, we might need to develop a new approach in utilizing matrix profile output for our time series mining and visualization target. Results of these tasks at the end, need to be evaluated to proof their correctness by comparing with other technique or fundamental theory.

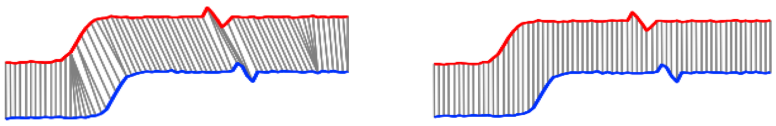
2. Background

This chapter describes the background studies of time series mining, the curse in high dimensionality, matrix profile and visualization. We also briefly explain the background of using matrix profile in time series mining.

2.1. Key Concepts in Time Series Mining

As previously mentioned, the fundamental task of time series mining is to find similarity and recurring pattern. This is often continued with advanced analysis like anomaly detection, motif discovery, prediction, clustering, classification and segmentation [1].

Time-series similarity measurement The similarity measure in time series is the most essential ingredient of time series clustering and classification systems. This calculation needs to be done efficiently for multidimensional time series data. The two common choice of this measure are DTW and ED distance. Eucliden distance is faster but DTW is more accurate [ED vs DTW Figure 2.1].



(a) Dynamic Time Wrapping (DTW)

(b) Euclidean Distance (ED)

Figure 2.1.: Basic methods for time series similarity. In terms of accuracy DTW provides better results, but slower computation compare to ED [8].

One key aspect in time series mining is to work with lighter representation of raw data using dimensionality reduction. After establishing a true distance measure for the raw data (e.g with Euclidean distance), the distance between a pair of time series in lower bound space must lower or equal to the true distance in normal space. There are several representatives of

time series similarity: lock-step (Euclidean Distance), feature-based (Fourrier coefficients), model-based and elastic measures [9]. Fourier coefficients [Fourier Equation 2.2] uses Discrete Fourier Transform, data representation instead of raw time series, enable us to perform half summation. Hence the computation can be accelerated [9]. This becomes the fundamental in matrix profile algorithm that we will visit later. A detailed comprehension of raw time series data to *Fourier series* conversion, can be learned more from [10].

$$d_{L_n}(\mathbf{x}, \mathbf{y}) = \left(\sum_{i=1}^{M} (x_i - y_i)^2 \right)^{\frac{1}{2}} \qquad (2.1) \qquad d_{FC}(\mathbf{x}, \mathbf{y}) = \left(\sum_{i=1}^{\theta} (\hat{x}_i - \hat{y}_i)^2 \right)^{\frac{1}{2}} \qquad (2.2)$$

Clustering Clustering on set of data is trying to group data according to some similarity based on distance metric. The goal of clustering in time series is given a time series database and similarity measures, find the set of clusters that maximizes inter-cluster distance and minimizes intra-cluster variance. Several approaches for time series clustering are clipped data representations, Auto-Regressive and K-Means. In K-Means, the determination of the optimal k clusters can be attained using the elbow method. The idea of elbow method is to run clustering of all possible k and measures the sum of squared errors (SSE) [11].

Classification The classification task seeks unlabeled time series T, known as TSC (Time Series Classification) problem, and assign it to one class from a set of predefined class. Several approaches such as using Singular Value Decomposition, however, the computation cost is high. Other common classifiers: Nearest Neighbor, Support Vector Machines, Decision Forest; machine learning techniques: neural networks or Bayesian classification. Deep Neural Network model can also be used for TSC problem [12].

Segmentation There are two definitions of "segmentation" in the literature: an approximation of a signal with Piecewise Linear Representation (PLR) [PLR illustration Figure 2.2] or partition of a dataset into several regions. The first definition aims at creating an accurate approximation of time series by pertaining its meaning while reducing dimensionality. Formally, given a time series T, construct a model of reduced dimensionality $\bar{T}$ such that $\bar{T}$ closely approximates T with a threshold of ε_r. The after results could be similar to moving average or median in time series pre-processing. PLR has been used to DTW, similarity search, clustering, and classification algorithm. When trying to find the approximation line, linear interpolation or linear regression is used [13]. While the second definition of segmentation which is more into "semantic segmentation", can be recognized as a type of clustering with the additional constraint that the elements in each cluster are contiguous in time [14]. On the

other hand, it can also be identified as a classification task if the ground truth classes as the time series background in contiguous time are known.

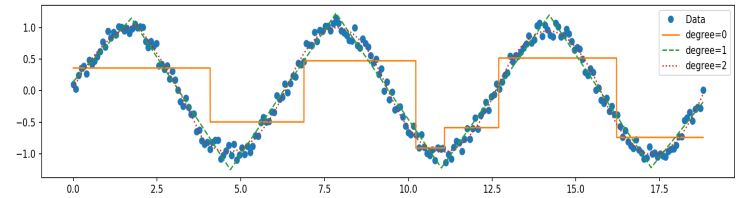


Figure 2.2.: PLR fits in a noisy *sine* waves using *pwlf* library [15] to a segmented constant(degree=0), piecewise linear(degree=1) and a piecewise quadratic model(degree=2).

Motif Discovery Motif discovery aims to find every subsequence that appears recurrently in a longer time series. Recurrent pattern exploration is beneficial in exploring meaningful clusters. The interest of this topic has been triggered by the observation that subsequences clustering produced meaningless result [16] if clustering involves arbitrary subsequences taken from its time series native. Later, the notion of motifs can be applied to many different tasks, such as modeling normal versus anomaly behavior implies finding recurrent two distinctive motifs in a series. Moreover, time-series classification can be speed-ups by constructing motifs for each class.

Anomaly Detection The common approach to detect anomalies is to create first a normal behavior of time series and characterize subsequence that diverge far [1]. In other words, anomaly detection tries to find outlier data points relative to some standard signal. As bottom line principle, statistical approach begins by detecting the signal outlier using low pass filter (Z-score), Chebyshev theorem to some current machine learning techniques such as regression using LSTMs, RNNs, DNNs, density-based techniques (k-NN and isolation forest) can be used to detect time series anomaly [17].

2.2. The Curse of High Dimensionality

One limitation concerning the data learning algorithm is the curse of high dimensionality, where an exponential increase in data size is unavoidable. As shown in [18], the volume of a unit-radius sphere with respect to the dimension of the space also show that if the number of dimension increases until a certain point, the volume of radius sphere will go nearly to zero means that there is no value of adding multivariate features [high-dimensional phenomena Figure 2.3].

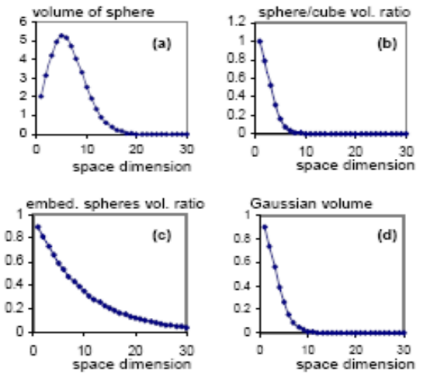


Figure 2.3.: Four phenomena in high-dimensional spaces [18].

In multi-dimensional Gaussian distribution, the probability of randomly picked data from the center in valuable 1^{st} dimension would be 90%. As the dimension increases, this probability will become lower [18]. Therefore, in terms of clusters visualization, the preliminary steps of dimensionality reduction are needed. Dimensionality reduction enables exploratory data analyses by reducing the complexity of the datasets and still preserving reasonable distance between cases or subjects.

Dimensionality in time series has several meanings. It can mean the number of metrics involved (multivariate time series) or the value of a single time series. Some popular dimensionality reduction algorithms for multivariate time series such as PCA and t-SNE. Because of the expensive computation of PCA, in the time series clustering task, where data has temporal relation, some researchers are considering multi-dimensional scaling (MDS) to create low-dimensional representation [19] [20]. Where the dimensionality reduction of time series value can be solved by PLR, SAX or bits compression with MDL (will be discussed in a later chapter).

2.3. Matrix Profile for Time Series Similarity

Pattern finding in time series starts from similarity measurement between pairs of time series or between itself (self-join). The latter can be defined as a similarity join problem, where given individual data object, one can retrieve the nearest neighbor for every object [21]. Initial work of similarity join [22] uses conversion to the lower-dimensional representation of time series using SAX. The lower dimensionality was determined by certain regions, e.g *a, b, c* in [SAX Figure 2.4]. From there, the similarity is found by computing the distance between the sequence of lower-dimensional representation.

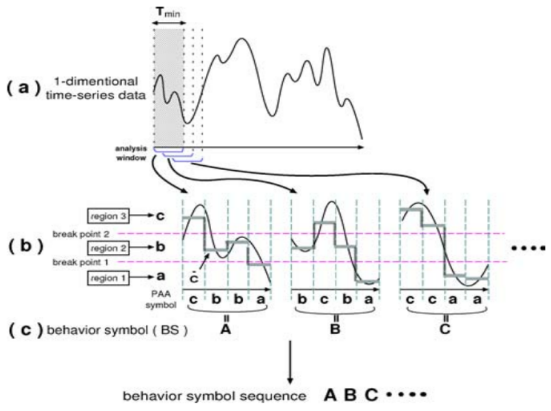


Figure 2.4.: Time series transformation to SAX sequences [22].

On the other hand, Matrix Profile computes the full distance matrix of all subsequences in one-time series with all subsequences in another time series, without converting to lower bits of time series. From the matrix profile, we can retrieve the 1-NN of subsequences in time series [21] [matrix profile Figure 2.5]. The companion vector comes with *matrix profile distance* is *matrix profile indices* where it tells the location of nearest neighbor [matrix profile indices Figure 2.6]. By having a reference of the most similar neighbor in time series, this will help us to look for patterns and motifs in time series.

Matrix profile algorithm returns two vectors:

- *Matrix profile distance*, the Euclidean distance of the subsequence $\boldsymbol{T}_{i,i+m}$ to its nearest neighbor elsewhere in $\boldsymbol{T}$.

- *Matrix profile index*, is the location of the i^{th}'s nearest neighbor in $\boldsymbol{T}$.

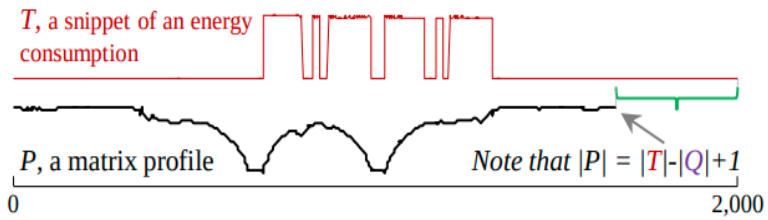


Figure 2.5.: A time series T and its self-join matrix profile P. Self-join means that a single time series compare with itself [21].

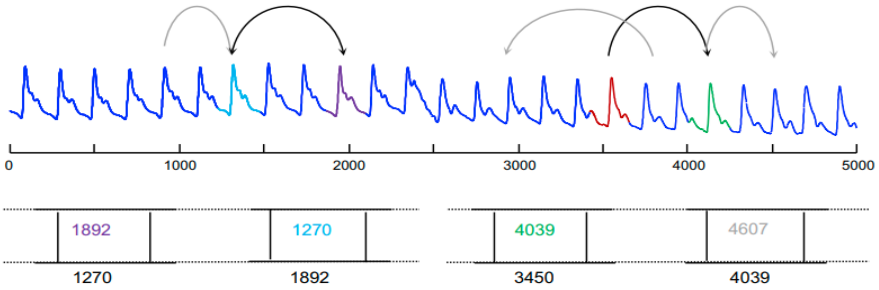


Figure 2.6.: An illustration of Matrix Profile indices, where it indicates the nearest neighbor of an mp index. It can be symmetric, but not always true, e.g 1270 to 1892 is symmetric however 3450 to 4039 is not [14].

2.4. Multidimensional Pattern Recognition

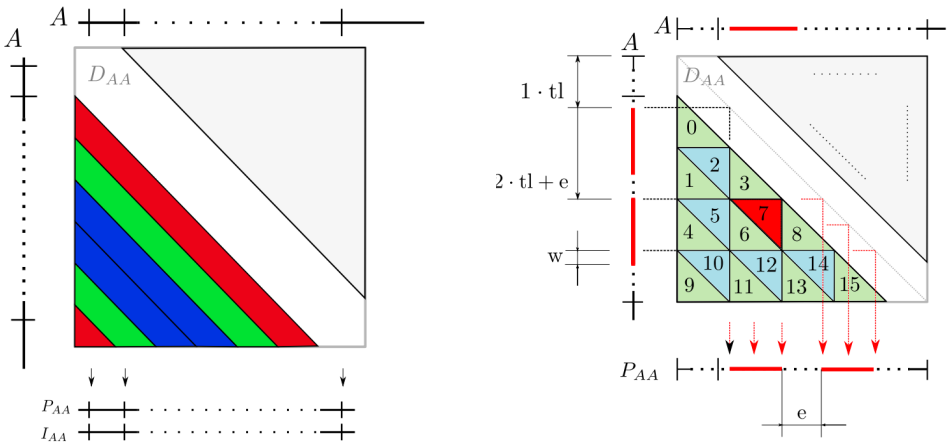
Multivariate or multidimensional implies the exploration of multiple signals in parallel. In many multidimensional pattern exploration research, they need to transform the data into reduced representation [23]. In 2005 Tanaka et.al [22], many dimension time-series were transformed into 1-D using PCA. However, this approach will not able to capture activities from each sensor with small timing and length difference. This also comes with the fact that not all sensors are synchronously working together. A group of sensors might be idle during the first 5 seconds, then they work together with other groups according to the activation, then some sensors might accelerate, some might go back to the idle phase. This complexity in multidimensional time-series analysis needs to be solved by capturing the signals from each sensor separately.

Dimensionality reduction has to be done to capture meaningful features, in which dif-

ferent pattern behaviors of signals from each sensor still can be captured. When only a single-dimensional to the analysis, fewer patterns will be detected because repeated signals in continuous time are treated as a single pattern. Moreover, one can expect, a hidden abnormal behavior that occurs from a single sensor can be detected while the rest of the sensors are working properly.

2.5. Utilization of HPC Resources

Utilization of HPC systems for matrix profile computation has been done in some research works [2],[3] [matrix profile cells partition in parallel Figure 2.7]. Each tile represents a segment of time series pairs while in mSTAMP computation. This will allow the computation to scale to a very large input size and distribute the computation into different machine [25]. The increasing amount of data has made it important to extend the algorithm using multicore programming, parallelization, and GPU/HPC acceleration [3], [26].



(a) Trivial parallelization among 3 processes

(b) Tiling scheme of distributed parallelization among 16 processes

Figure 2.7.: Partitioning illustration of work among processes for self-join computation where A is the time series computed against itself. $P_{A,A}$ is the matrix profile distance and $I_{A,A}$ is the matrix profile index. Distributed parallelization is the more refined work of [3].

3. Related Works

This chapter covers the theoretical foundations: matrix profile syntax conventions, motif discovery and segmentation, sub-sequence clustering using matrix profile, visualization approach, and hierarchical data analysis.

3.1. Definitions

This section introduces common terms used in time series mining. All definitions are taken from [27].

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

Definition 2 A subsequence $T_{i,m}$ of a $\boldsymbol{T}$ is a continuous subset of the values from T of length m starting from position i. It can be formulated into $T_{i,m} = [t_1, t_2, ..., t_{i+m-1}]$.

Definition 3 A time series motif is the most similar subsequence pair of a time series $\boldsymbol{T}$.

Definition 4 A *distance profile* $\boldsymbol{MPdistance} \in \mathbb{R}^{n-m+1}$ of a time series $\boldsymbol{T}$ and a subsequence $T_{i,m}$ is a vector that stores $dist(T_{i,m}, T_{j,m}) \forall j \in [1, 2, ..., n - m + 1]$.

Definition 5 A *matrix profile* $\boldsymbol{MP} \in \mathbb{R}^{-+\mathbb{1}}$ of a time series $\boldsymbol{T}$ is a meta time series that stores the z-normalized Euclidean distance between each subsequence and its nearest neighbor, where n is the length of $\boldsymbol{T}$, and m is the given subsequence length.

Definition 6 A *multidimensional time series* $\boldsymbol{T} \in \mathbb{R}^{dxn}$ is a set of time series $T^(i) \in \mathbb{R}^n$: $T = [T^{(1)}, T^{(2)}, ..., T^{(d)}]^T$ where d is the dimensionality of $\boldsymbol{T}$ and n is the length of $\boldsymbol{T}$.

Definition 7 A *multidimensional subsequence* $T_{i,m} \in \mathbb{R}^{dxm}$ of a multidimensional time series $\boldsymbol{T}$ is a set of univariant subsequences from $\boldsymbol{T}$ of length m starting from position i. Formally, $T_{i,m} = [T_{i,m}^{(1)}, T_{i,m}^{(2)}, ..., T_{i,m}^{(d)}]^T$.

Definition 8 The ***k*-dimensional distance function** or $dist^{(k)}$ computes the distance between two multidimensional subsequences by using only the best $\boldsymbol{k}$ out of d dimensions.

Definition 9 A ***k*-dimensional matrix profile** $\boldsymbol{P} \in \mathbb{R}^{-+\mathbb{1}}$ of a multidimensional time series $\boldsymbol{T}$ is a meta time series that stores the z-normalized Euclidean distance between each subsequence and its nearest neighbor, where $\boldsymbol{n}$ is the length of $\boldsymbol{T}$, d is the dimensionality of $\boldsymbol{T}$, $\boldsymbol{k}$ is the given number of dimension, and $\boldsymbol{m}$ is the given subsequence length.

Definition 10 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.

Definition 11 1NN-join function: given two all-subsequences sets $\boldsymbol{A}$ and $\boldsymbol{B}$ and two subsequences $\boldsymbol{A}[i]$ and $\boldsymbol{B}[j]$, a 1NN-join function $\Theta_{1nn}(\boldsymbol{A}[i], \boldsymbol{B}[j])$ is a Boolean function which returns "true" only if $\boldsymbol{B}[j]$ is the nearest neighbor of $\boldsymbol{A}[i]$ in the set $\boldsymbol{B}$.

3.2. Motif Discovery and Segmentation

Motifs can correspond to complex behaviors that capture common sequences of state transitions. This works by comparing structure similarity in subsequences. Hence, motif discovery target is to collect the most similar pairs in time series T. Here, we will review mSTAMP and basic ideas of MASS (Mueen's Algorithm for Similarity Search) time series algorithm for similarity search and motif discovery.

3.2.1. mSTAMP Algorithm

mSTAMP is one of the naive algorithms to compute matrix profile for multidimensional time series. It computes k combinations out of d dimensions in combinatorial search space. The complexity of the algorithm is $O(dlogdn^2)$ time and $O(dn)$ space. Because of these low complexities in time and space, it has enabled mSTAMP to perform efficiently [28]. When looking for dimensionality reduction, constrained or unconstrained search of included dimensions can be applied. Constrained search looks for a predetermined set of dimensions from the user. When the user has little knowledge about the reasonable dimensions to choose, it is

necessary to support unconstrained search. The difference is when choosing k number of dimensions where $1 \leq k \leq d$, and typically $k \ll d$; where k is done by the algorithm as the natural dimensionality of a repeated structure in the data.

Unconstrained search in mSTAMP works by dimensionality reduction using Minimum Description Length (MDL) principle [28]. In the first option, given d dimensional data, matrix profile also returns d dimensional matrix profile. Then from each of the dimension in matrix profile, we take the distance profile value and mapped it into elbow plot, where the breaking point in elbow plot casts the best k dimensionality[elbow plot Figure 3.1].

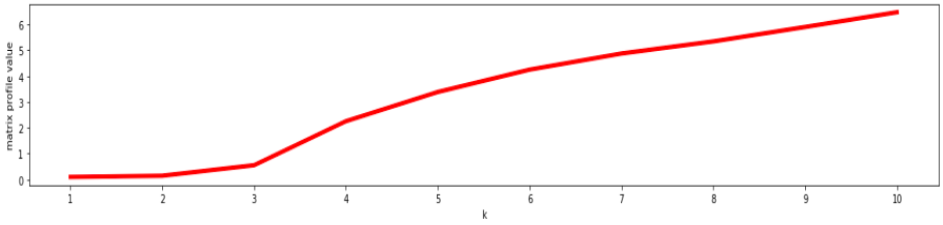


Figure 3.1.: Elbow curve for 10 dimensional time series. The turning point in elbow plot shows the best k dimensionality, which in this case is 3.

The most essential reason of using matrix profile based method because of its scalability and time performance compare to other methods in similarity pairs finding. STAMP is the only algorithm that does full joins on time series subsequences. Referencing to the computation of 2^{18} data points, the comparison to the other method: TSFR (MapReduce), HSDJ (MapReduce), Optimized Nested Loop Table 3.1:

$\mathbf{TSFR_{DAA}}$	$\mathbf{HDSJ_{I-SAX}}$	ONL	STAMP
51.7 hours	19.6 min	28.1 hours	1.17 hours

Table 3.1.: Time series similarity techniques comparison to STAMP [21].

3.2.2. Minimum Description Length (MDL)

MDL shows that the best description of the data is given by the model which compresses it the best.

$$DNorm(T_{i,m}) = round\left(\frac{T_{i,m} - min}{max - min} \times (2^b - 1)\right) + 1 \tag{3.1}$$

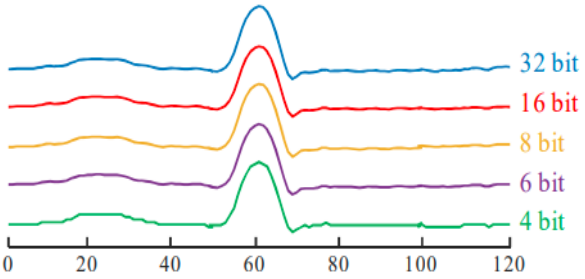


Figure 3.2.: Time series subsequence represented at different bit precision [19].

For any time series T, to determine how many bits it takes to represent, we can approximate the *Kolmogorov* complexity by using *Huffman coding*. The compressed file size is an upper bound to the description length (DL) of the time series. Therefore we can approximate the total number of bits required by looking at the encoded file size from the *Huffman coding*.

$$DL(T) = |HuffmanCoding(T)| \tag{3.2}$$

The idea to use this concept is by comparing a real value time series data with its hypothesis (reduction). This lower bound shape of the real value time series can be approximated with DFT, APCA, SAX, PLR [19].

3.3. Semantic Segmentation

Semantic segmentation divides time series into several regimes. Assume we have a system S, which can be divide into some regimes (e.g An exercise routine contains of *warm-up*, *stretching*, *resistance training* and *cool-down* [29]). Segmentation on a data sequence has been successfully developed by using Hidden Markov Models (HMMs). Another approach, FLUSS algorithm from Garghabi, et.al [14] that also extends mSTAMP algorithm. While HMM exploits statistical changes to mark the segment cuts, FLUSS uses counting of the crossing arcs number. The crossing arcs are basically the MP*indices* returned from mSTAMP. The fewer arcs crossing across a region, the more likely that time series region becomes the cuts (segment border) between a region and its neighbor.

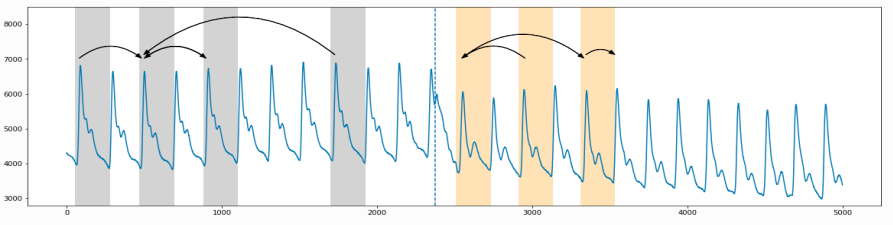


Figure 3.3.: The arcs crossing across their own regime (colored in grey and yellow). The dashed vertical line is the hint for the breaking point of regimes change.

The task of FLUSS is to produce companion time series Arc Curve (AC), which annotates the regimes changes likelihood in time series. The intuition is as follows: within a single regime we can expect that most subsequence will have a nearest neighbor close to them, which if we imagine that one subsequence pointing to another subsequence and vice versa Figure 3.3, we can think that these subsequence are in the same regimes. Therefore, the previous lemma of fewer arcs crossing could be a hint of regimes change.

FLUSS takes a time series $\boldsymbol{T}$ and pre-defined subsequence length as inputs. FLUSS outputs AC vector of length n, where each index i contains the number of how many nearest neighbor arcs from the MP*index* spatially cross over i. Regimes change detection can be spotted by looking at this AC vector, where Arc Curve has a low value at the location of the regimes change[AC transformed into CAC Figure 3.4 in detecting regimes changes].

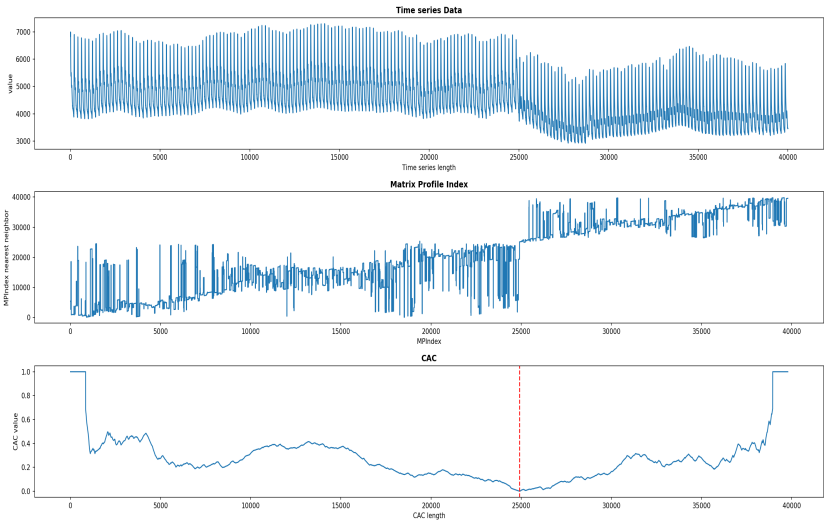


Figure 3.4.: The CAC (Corrected Arc Curves, where AC has been corrected with inverted parabola) minimizes in index where regimes changes [30].

An ideal AC shape resembles an inverted parabola with its height $\frac{1}{2n}$. Further, regimes can be extracted from the CAC. This regimes extraction works like *k-means* clustering, where it requires k number of cluster as user input. Their Regime Extracting Algorithm (REA) searches for k lowest *"valley"* points in the CAC. FLUSS sets up an exclusion zone to prevent the second segment cut is chosen from the consecutive point. The exclusion zone is important to pick the next regimes with some distance from the current segment cut.

3.4. Subsequences Clustering

Clustering on time series can be recognized into two variants, whole clustering on complete selected sequences in or subsequences clustering. Time series subsequences cannot be simply plugged into a MDS framework. Given a single time series, sometimes in the form of streaming time series, individual time series subsequences are extracted with a sliding window. Clustering is then performed on the extracted time series sub-sequences [16]. The other way of clustering time series is by means similar to conventional clustering of discrete objects. It clusters similar time series into the same cluster.

3.4.1. Multidimensional Scaling (MDS)

MDS is one of the related techniques used for dimensionality reduction before visualization. Given k dimensional objects, MDS can preserve the distance between objects as well as possible. MDS based visualization has been done in subsequence with different events and range. Keogh et.al [16] demonstrates a clustering of time series subsequences is bad where *all* subsequences are involved. In identifying *important* subsequences, MDL framework can be used as subsequence proposing algorithm using *discrete* time series normalization Equation 3.1. The normalization tries to find out the most optimum bit to compress the time series subsequence without reducing the precision subsection 3.2.2. In summary, MDS in time series has been used for the subsequence selection problem by testing all subsequences and pick one that minimizes the total bit cost.

3.4.2. Top k-motifs Algorithm

Top motifs are subsequences or some region in a time series that has appeared many times. One motif represents a single subsequence that recurrence many times. Hence, in k top motifs, there are several different k subsequences which represent k different motifs. As

Euclidean distance in matrix profile is an excellent proxy for picking the location of motifs candidate, semantically meaningful sub-sequence in time series will also have local extrema in matrix profile[motif and discord Figure 3.6-part(a)]. Discovered motifs is beneficial in recognizing frequent patterns. In Figure 3.5 shows a use case of multivariate time series pattern visualization in the data center.

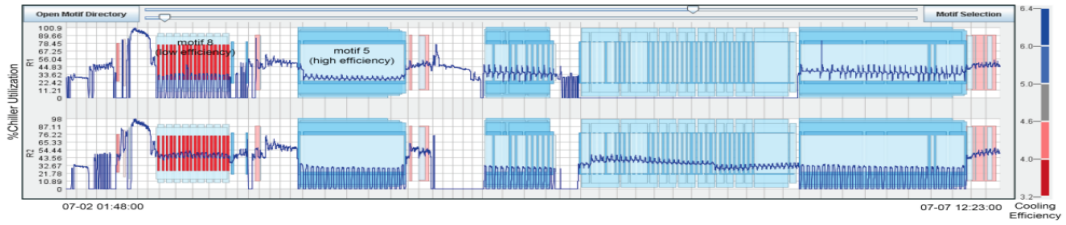
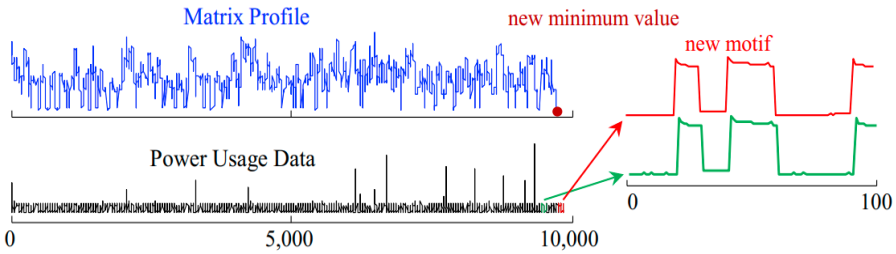


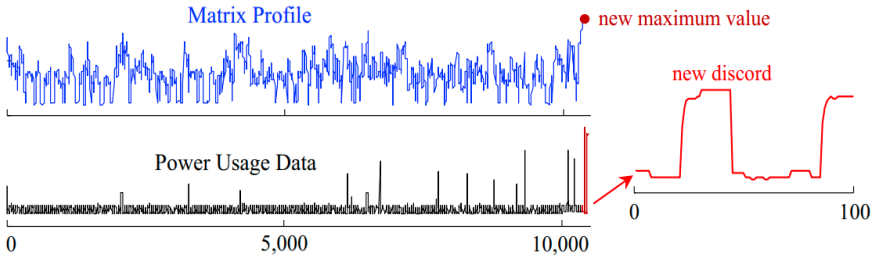
Figure 3.5.: Motifs visualization in describing high and low efficiency in data center [31].

3.4.3. Top k-discord Algorithm

Discords in time series data represent a subsequence candidate which has the farthest euclidean distance with the rest of the data which might be a candidate of an anomaly in the dataset. However, not all unusual patterns are important. Finding top-*K* discords on many occasions is more important than only discovering the most or all unusual patterns. Facilitated by matrix profile, top-*K* discords discovery can be investigated by taking the highest point in matrix profile distance[motif and discord Figure 3.6-part(b)].



(a) The minimum value of the matrix profile corresponds to a pair of time series motifs in the power usage data. The best time series motifs detected in red and green color.



(b) The maximum value of the matrix profile corresponds to discord or anomaly in raw time series data.

Figure 3.6.: An example of motifs and discords discovery on power usage data. Source [21]

3.5. Time Series Pattern Visualization

Temporal data is subdivided into temporal and structural domains. Visualization of the temporal domain focuses on the historical change of time evolution, while the structural domain focuses on the correlation of values in the data. Frequently, structural domain analysis will ignore the aspect of the time evolution in time series data [24]. Most often visualization of temporal data is in shapes of plotline or stacked graph. The complexity of visualizing data increases where data are too large, not only in terms of data point but also in the number of features observed. Ordinary visualization of such data can lead to overcrowded and cluttered displays [24].

The most important criteria from a visualization point of view are the following: when dealing with large volumes of data, additional analytical methods have to be included to derive a higher level of abstraction of the data[32]. Visualization techniques that can exploits pattern similarity become important. Time series are commonly represented as line charts, but a considerable amount of work has examined alternatives visual encodings, such as horizon graphs [33] and heatmap or colorfields [34].

A clustered color with the light and saturation color can be useful to display the pattern in time series. Some of the visualization using hierarchical analysis in the start before they map the data following to its saturation color properties. Visualization based on color saturation can be represented in CircleView Figure 3.7 or Line graph (colorfields). The color follows the intensity of the value, for instance, the more important feature has a more intense color. In many cases of scientific visualization, the transition of color intensity can be created by creating a sequence of linear space or using complex numbers. Another common approach for the time series visualization is by using a stream graph (*ThemeRiver*) Figure 3.9.

In visualization, it is important to fit every feature to visualization space. To this end, hierarchical structures to find data granularities, such as by analyzing the data correlation, information gain and statistical methods to get the concentration of interesting part of the data based on user objectives. Improvisation on data granularity can be improved by employing tree structure like SpaceTree, HyperSlices, Hierarchical Parallel Coordinates and Multiresolution approach Figure 3.7.

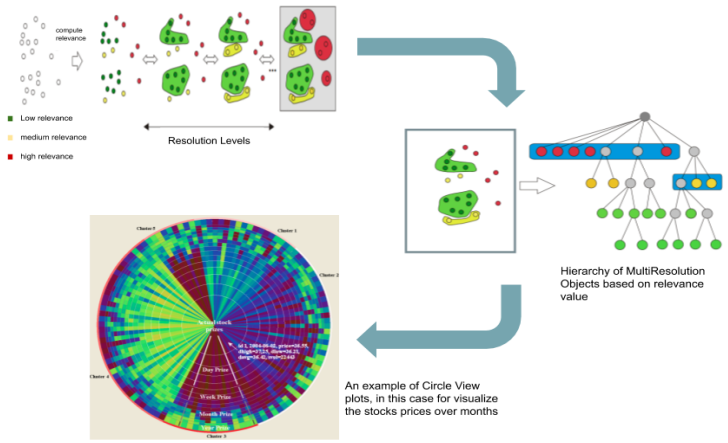


Figure 3.7.: Multiresolution compute hierarchy of views that present the data at different levels of detail. It shows more relevant objects at higher detail. The more relevant the data the more contrast the color intensity. Source: [32]

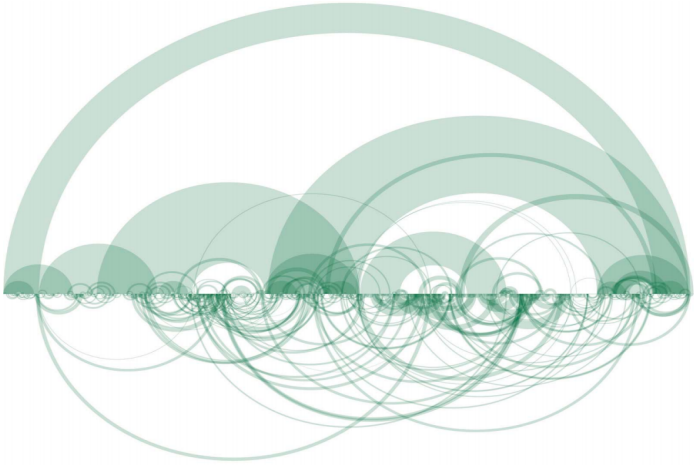
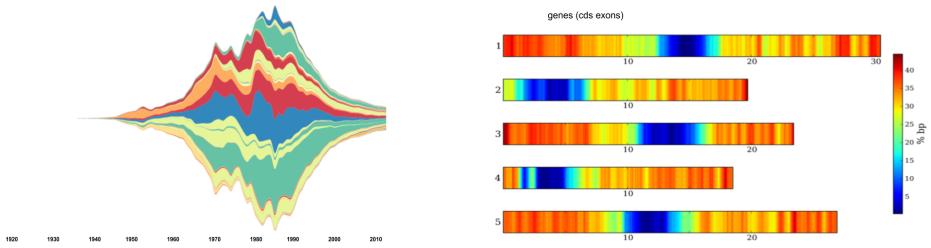


Figure 3.8.: Arc diagram visualization of *Für Elise* music structure [33]



(a) Stream graph of time series data over years.

(b) A-thal genes. Source:Helmholtz Munich

Figure 3.9.: Each color in stream graph corresponds to flow of a single feature while color in heat map correlates with value intensity to the color maps (*blue-red*).

Time series data can be processed later into something more meaningful such subsequences clustering. One of the visual representations of clustering is a scatter plot. An example of such data visualization is Embedding Projector Figure 3.10 by Tensorflow [34]. In this project, user can interact with the visualized points for instance with mouse clicking where it will show more details about a point and its cluster. It also provides other features such as doing dimensionality reduction over iteration until the points are converged.

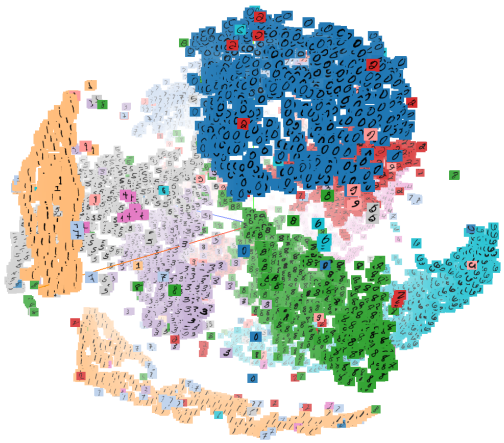


Figure 3.10.: Embedding Projector by Tensorflow on Mnist data [35]

3.5.1. Interactive Data Visualization

There are starting point tools for data visualization with projection on a 2D screen such as Matlab, Matplotlib, OpenGL, or a web-based visualization using WebGL. A study [36] started the development of data visualization in VR 3D using *Unity 3D*^{TM}. They combined web browser interaction as an interface to an immersive 3D space that facilitates the exploration of data in VR in a "game-like" environment. One approach is to have "scatter plot" look like using point cloud based element in the game engine environment [37].

Figure 3.11.: Illustration of point cloud prototype in Unreal Engine (plugin by LRZ).

4. Experimental Approach

This chapter summarizes the thesis roadmap and background study in data source and preparation.

4.1. Thesis workflow

This work roadmap consists of several steps from data production, preparation, data mining, and relevant visualization tools [thesis work plan Figure 4.1].

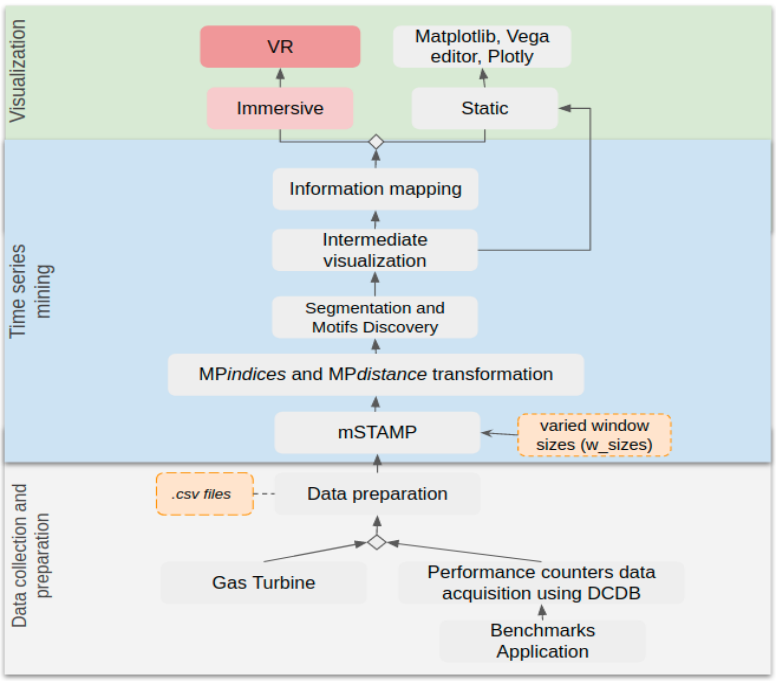


Figure 4.1.: Thesis work plan.

The works started by producing and collecting data from different sources. In collecting data from performance counters, we need to set up a script that will run different application

benchmarks. Hence, background knowledge on how to acquire the intended dataset is needed. On the other hand, for the gas turbine data, because the data was already batched, we can continue the step directly into data preparation.

Our time series mining uses the Matrix Profile approach. Data preparation before mSTAMP includes grouping several sensors together and data filtering. In mSTAMP, the window sizes are identified based on the length of interesting subsequences in the dataset. The output from mSTAMP (MP*indices* and MP*distance*) can then be further analyzed to produce useful insight, which will be the main focus of this thesis. After developing the analysis model, we will evaluate these results to assess the accuracy of the analysis. Further, some visualization approaches will be explored to present the results in a meaningful manner. In this work, we will only focus on static visualization and further possibilities to reach an immersive visualization.

4.2. HPC's Performance Counters Case Study

To generate the sensor data in the HPC environment, events trigger to some parts in the system can be produced by running a common benchmarks application. In this work, we deployed benchmarks reading using *Coral2* [38].

4.2.1. *Coral2* Benchmarks

Application benchmarks in this thesis work include Kripke, AMG, Nekbone, and LAMMPS from *Coral2* [38]. Nekbone targets scalable science while the other focuses in throughput measurement. Nekbone solves a Poisson equation where each process having the same amount of computational work. Kripke has a wavefront algorithm which stresses more the memory throughput on a structured grid algorithm. AMG tested parallel performance on unstructured multigrid methods which consume main memory bandwidth. LAMMPS, is used to simulate materials modeling and is intended to measure compute, memory bandwidth, and network latency performance.

Data are obtained using DCDB [39] framework, in which it has defined the level of the performance counters that can be retrieved. It would be beneficial if the application runs on a certain MPI configuration [40] and be able to acquire the data only from related sensors. For instance, we can define how many cores the application will run using `I_MPI_PIN_DOMAIN`. We can also investigate the related metrics via MPI (Message Passing Interface) debug options.

In summary, DCDB can perform information up to the system core level. These data will be grouped belongs to their criteria in subsection 4.2.2.

4.2.2. Top-Down Approach for HPC's Performance Counters

Yasin et.al [41] designed a top-down analysis approach to identifies critical bottlenecks in CPUs. The counters from DCDB will be conformed using this approach. There is a limitation where some counters have no pair match with DCDB specification. Therefore, we will approximate a matching pairs between available DCDB sensors and required PMU counters in [41], is described in [top-down for DCDB Figure 4.2] and [DCDB's sensors group Table A.2].

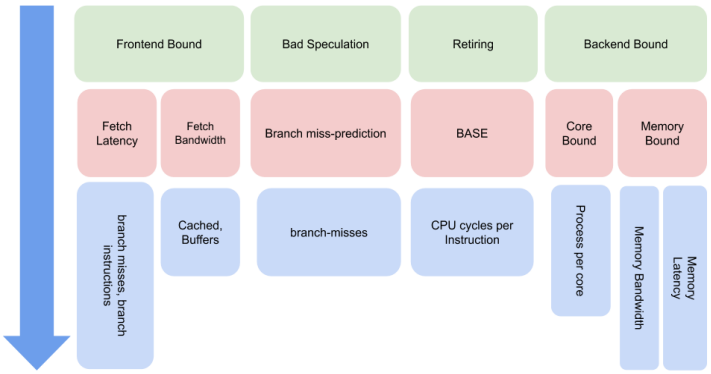


Figure 4.2.: The top-down categorization for HPC's performance counters.

4.3. Gas Turbine Case Study

Gas turbine produces a huge amount of data[46]. As we stated earlier, we will perform our analysis on different data source. The difference between this use case and the performance counters lies on no ground-truth labels available in the analysis (unconstrained search subsection 3.2.1). Meanwhile, in HPC's performance counters, we can label the time duration where the application runs which turn into our ground truth in the analysis.

We grouped gas turbine sensors into speed (TNH_RPM), power, temperature, pressure, valve position, and gas detector [gas turbine sensors group section A.4]. Power sensor will be kept in each group to see the impact of its down-time and up-time towards reading

of the other sensors in a group. Then sensors with multiple reading and similar prefixes with others in the same group are averaged together. We also apply data sampling and filtering to the sensors based on its noise level. Based on the behavior in the dataset, we perform two different analyses: when the machine has gone through multiple idle phases or steady run in a certain time duration. When conducting an experiment for a stable run, the data regions of idle time is removed by removing any value below the mean value of the speed.

For statistics review, this dataset is available in second precision. The total size of a single sensor is around 32M data points. After a 20 seconds sampling rate is applied, it has reduced to 1.7M data points. For current analysis, only some parts in the colored boxes are taken[gas turbine data sample Figure 4.3]. The data size in both regions, idling and steady, are 60000 and 35000 data points respectively.

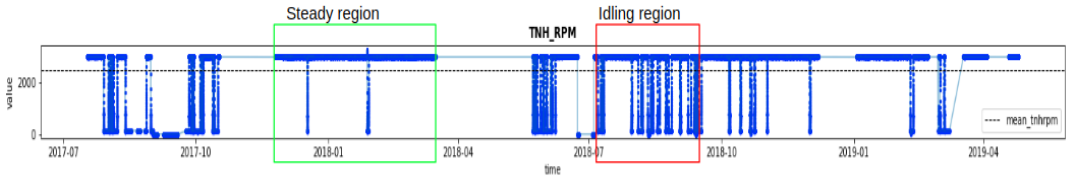


Figure 4.3.: Data investigation on sample focuses in regions across red (time series with idle phases) and light blue (stable regions with no idle phase) colors.

4.3.1. Filtering

When time-series data has high noise, it could create more dissimilarity instead of similarity between time series pairs in matrix profile computation. In this case, filtering helps to smooth the data to a certain threshold t. Although the purpose of filtering is to reduce the noise, the subsequences shape from original data has to be in recognition for matrix profile computation. Therefore, when filtering is needed, we try first with different filter window sizes to see how far information loss is accepted. We filter gas turbine data before mSTAMP, while we skip this part for *Coral2* because the subsequence shapes are already clear.

A widely used class of filters is moving windows. The wider the window size q the more noise will be removed, however, it can cause more data points loss. Moving average by window size also removes some points by q in the first part of data. Therefore, the q parameter is decided by how fine expected granularity of the output and the length of raw time series data $q \ll n$. Another approach of data filtering is by applying a weighted exponential filter (EMA or EWMA). This filtering option does not eliminate the first q part of the data. Although, the

outlier suppression is weaker than moving mean [42], for our analysis we chose this filtering method to avoid any data points removal.

4.4. Outlier Removal

The intention to use outlier removal in our analysis is to remove the high frequency noise in MP*indices* after being transformed with MDL. This intuition occurs where a segment in dataset should have similar pointers distribution to other region in dataset Figure 3.3. Turns out, applying low-pass filter (using moving average and standard deviation) to the MP*indices* works better compared to without pre-liminary transformation on MP*indices*. Outlying points can be set by dynamic threshold t of the sigma-rule in interval $[\mu - t * \sigma, \mu + t * \sigma]$ and $|z_i| > t$.

$$z_i = \frac{x_i - \mu}{\sigma} \tag{4.1}$$

4.5. Method Evaluation

Basic precision and recall will be used to measure the performance of the analysis. They consist of TP (True Positives), FP (False Positives), FN (False Negatives), TN (True Negatives) [43].

$$Precision = \frac{TP}{TP + FP} \quad (4.2) \qquad Recall = \frac{TP}{TP + FN} \quad (4.3) \qquad F1score = \frac{2 * P * R}{P + R} \quad (4.4)$$

These performance metrics are used to measured all detected label in our segmentation and motifs discovery analysis. Precision is the fraction of all detected segment or motif labels, whereas recall is the fraction of all real segments or motifs that are successfully detected.

		Actual	
		+	-
Predicted	+	TP	FN
	-	FP	TN

Table 4.1.: Definition of TP, FP, FN and TN in comparing the actual and predicted label.

5. Matrix Profile, Motif Discovery and Segmentation

As previously mentioned time series mining in this thesis study is based on the matrix profile output from mSTAMP, where the flow of the algorithm can be described:

Algorithm 1: mSTAMP Algorithm [21]

input : T_A, T_B and window size with length m

output: $MPdistance_{A,B}$, $MPindex_{A,B}$, $MPdims_{A,B}$

1 $n_b \leftarrow$ Length(n_b)
2 $MP_{A,B} \leftarrow$infs, $MPIndex_{A,B} \leftarrow$zeros, $idxes \leftarrow$1:$n_b - m + 1$
3 **foreach** *idx in idxes* **do**
4 QT $\leftarrow$ SlidingDotProducts_FFT(Query Q, Time series T)
5 μ_Q, σ_Q, M_T, $\Sigma_T \leftarrow$ ComputeMeanStd(Q,T)
6 D $\leftarrow$ CalculateDistanceProfile(Q, T, QT, μ_Q, σ_Q, M_T, Σ_T)
7 $MP_{A,B}$, $MPIndex_{A,B} \leftarrow$ ElementWiseMin($MP_{A,B}$, $MPIndex_{A,B}$, D, idx)
8 **return** $MP_{A,B}$, $MPIndex_{A,B}$

When calculating the distance profile algorithm 1-line 6, the algorithm uses Equation 5.1 which applies Pearsons' correlation $\rho_{A,B}$, instead of using z-normalized euclidean distance Equation 2.1.

$$\delta_{A,B} = \sqrt{2m\left(1 - \frac{Q_{A,B} - m\mu_A\mu_B}{m\sigma_A\sigma_B}\right)} = \sqrt{2m(1 - \rho_{A,B}))} \tag{5.1}$$

The main hyperparameter tuning involves in the algorithm is the window size. Window sizes are currently chosen by visual observation on the raw time-series dataset, wherever some parts are more likely leads to recognition of data pattern. The chosen window with size m slides across T and by this definition the subsequence $T_{i,m}$ will be compared with the rest of data T. When doing the similarity of multidimensional time series, every time series subsequence captured in a window size of m will be compared to subsequences in all dimensions. The output at the end of the algorithm is the value when 1NN-join is true,

indicated by local minima in matrix profile distance.

5.1. Matrix Profile with Different Window Sizes

The decision of window sizes in mSTAMP algorithm needed will depend on the length of the pattern that we would like to observe. For illustration we describe matrix profile analysis on a dummy time series data which consists of multiple *sine, tooth saw, and square* signal forms Figure 5.1. We would like to see how our matrix profile output looks like given these signals on multidimensional time series Figure 5.2.

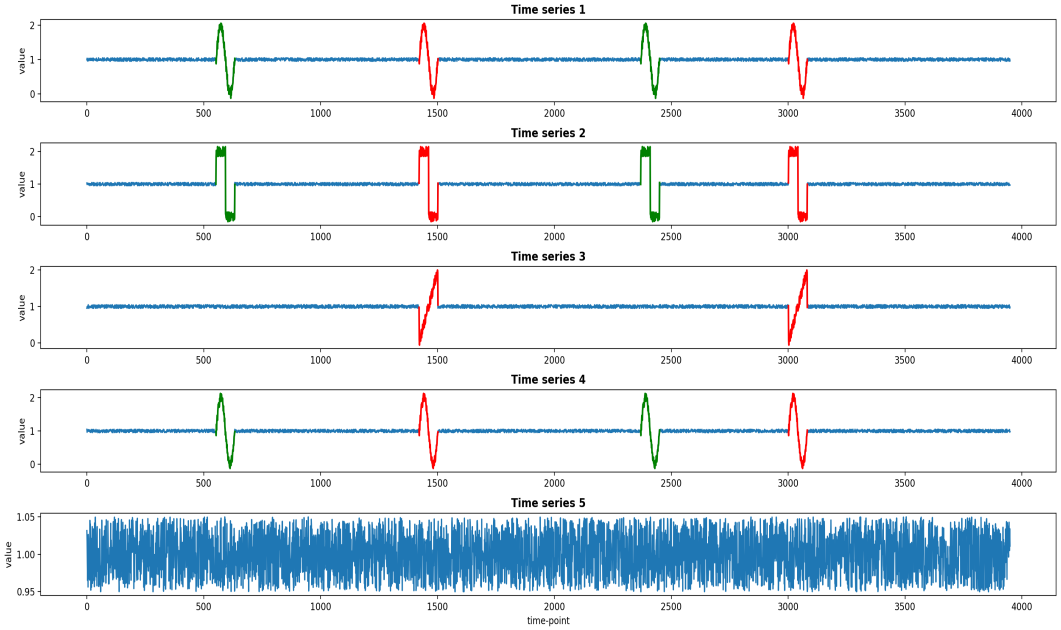


Figure 5.1.: Repeated different signals *(sine, toothsaw, square)* and random. The length for each signal pattern is 80. The coloring red and green are to show the area of time series data that we want to identify in latter task.

We can see, when m is set close to the pattern length of the signals (m = 80), *MPdistance* indicates a local minima which covers the signal pattern. Unique subsequences $T_{i,m}$ with $m \leq 80$ might be detected from the 5^{th} dimension, however, mSTAMP does not recognize any impact on the analysis from including this dimension. It is demonstrated by the 5^{th} dimension of matrix profile value, local minima moving far from 0. In the intuition of mSTAMP algorithm, any unique pattern in original T data can be investigated by tracing the regions of local minima on matrix profile.

Inspection on the signal pattern in Figure 5.1 colored in **green**, it is showing signal on 1^{st}, 2^{nd}, and 4^{th} dimensions on the time-series T data. If we look into the matrix profile distance for these three dimensions, we can see the local minima in matrix profile of the first 3 dimensions are close to zero, thereafter the local minima moving away from 0. This knowledge leads to the conclusion that 3 out of 5 dimensions are important in recognizing the **green** pattern. Now, if we investigate the **red** pattern, the signals present on 1^{st}, 2^{nd}, 3^{rd} and 4^{th} dimensions. The MP*distance* are still close to 0 until the 4th dimensions. Therefore four dimensions are important in recognizing the **red** pattern. This is basically the intuition of elbow curve in selecting the k dimensionality.

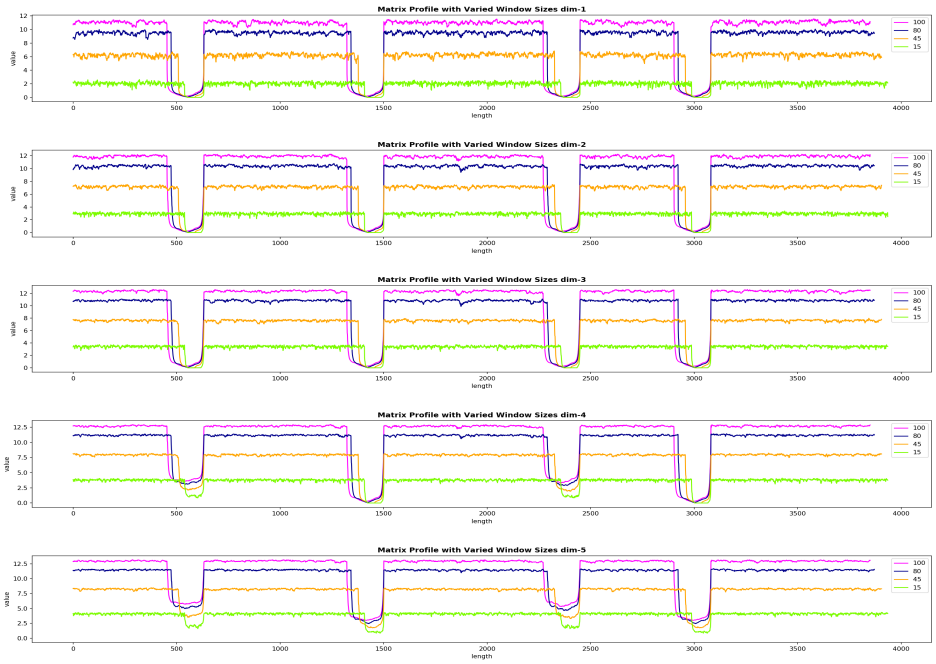


Figure 5.2.: Multidimensional matrix profile distances (MP*distance*)given multidimensional time series in Figure 5.1 with varied window sizes. As matrix profile returns n-m+1, where n is the length of a single T, m is the window size, thus it will not cover *m-1* data points in time-series T tail into the analysis.

5.1.1. MDL for dimensionality reduction

We can now put our multidimensional MP*distance* into our core analysis, which consists of finding its MDL and transforming the multidimensional MP*index* based on their MDL. As MDL is usually used to find the best compression of time series data, here we use MDL to

find the best k dimension algorithm 2-Part I. MDL can be found by compressing it to the most optimized bit by Huffman coding, however, we take another approach by calculating the derivative of multidimensional MP*distance* value (elbow point) for each data point. This comes from the idea in [28], that k-dimensional MP*distance* can be produced by solving $\min_x \|\boldsymbol{MPdistance}\|_0 \, \forall j \in [1, 2, ..., n - m + 1]$, where k is the vector of included dimensions. In terms of k expression, because of the programming nature, k will start from 0 while it means 1^{st} dimension instead of 0^{th}. Hence, when talking about the best k dimension, we refer to *k+1* dimension Figure 5.3.

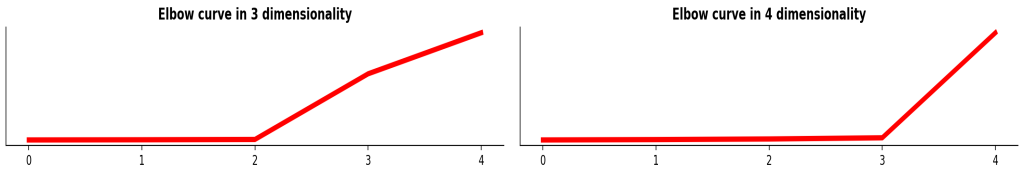


Figure 5.3.: Elbow curves from Figure 5.4 of detected orange and green colored patch respectively. The orange colored patch involves at best 3 dimensions, while the other one involves at best 4 dimensions.

For every data point, we can then pick the value of data point i in MP*index-k*, where the whole output later is known as our filtered MP*index*. As a reminder, MP*index* hints to another area in T where it informs the location of their nearest neighbor. The output of our filtered MP*index* usually still has noises in this indexing system. The perception where a cluster of indexes in a certain subsequence $T_{i,m}$ should in all point to the somewhere another region in the same cluster. In other words, MP*index* of a subsequence $T_{i,m[a]}$ should point altogether to the most similar subsequence $T_{i,m[b]}$. Aggregation of MP*index* by reducing the variation around a certain range of the index is important to have a finer visualization Figure 5.4.

Algorithm 2: MP*index* Filtering towards its MDL properties

input : MP*distance*, MP*index*, *f_window*, *sigma*
output: *adapted_mpi*

```
// Part I: MDL computation for each data point in T
mdl_dim ← []
for i ← 0 to length of MPindex do
    for k ← 0 to size of MPdims do
        // derivative of elbow curve
        elbow_dim ← []
        der ← MPindex_{k+1,i} - MPindex_{k,i}
        elbow_dim.append(der)
    k ← min_{elbow_dim}
    // k is the best dimensionality
    mdl_dim.append(k)
// Part II: MPindex filtering based on its MDL for each data points in T
adapted_mpi ← []
min_dim ← min_{mdl_dim}
max_dim ← max_{mdl_dim}
// min_dim, max_dim can be replaced by x_1 and x_2, where min_{mdl_dim} ≤ x_1 ≤ x_2 ≤ max_{mdl_dim}
for i ← 0 to length of mdl_dim do
    if mdl_dim[i] ≥ max_dim and mdl_dim[i] ≤ min_dim then
        // the dimension is within dimensionality range
        adapted_mpi.append(mdl_dim[i])
    else adapted_mpi.append(0);
// apply low pass filter and linear interpolate, deviation ≈ 1σ to 2σ
adapted_mpi ← lowpass_filter(adapted_mpi, f_window, sigma)
return adapted_mpi
```

The filtering of MP*index* starts by canceling the pointers that are perceived as noise. For instance, in this simple 5-dimensional case, dimensionality in the regions where the signal is not meaningful can be ignored or k set to zero. This step will reduce the complexity of MPindex filtering in the next step. However, this step is mostly skipped because the signal patterns in the dataset are not distinctive. Subsequently, pointers in MP*index* still need to be aggregated by removing the sudden indexing shift.

The outlier removal technique is the approach to complete the task. Filtering by low-pass filter using parameter *f_window* (line 19) and later where the indicated outliers with deviation more than 1 to 2 σ are removed and replaced by linear interpolation of its closest neighbors. The deviation should be around these values because we want to smooth out the noise in MP*index* without eliminating the meaningful pointers shifting in some regions. After getting through all steps, the MP*index* is now more representative and we called it Adapted MP*index* for future reference Figure 5.4. To note down, every figure that represents pattern recognition in the next section will have their experiment setup reference, as a change in one of these values can affect the final result and also for easier investigation, see section A.2.

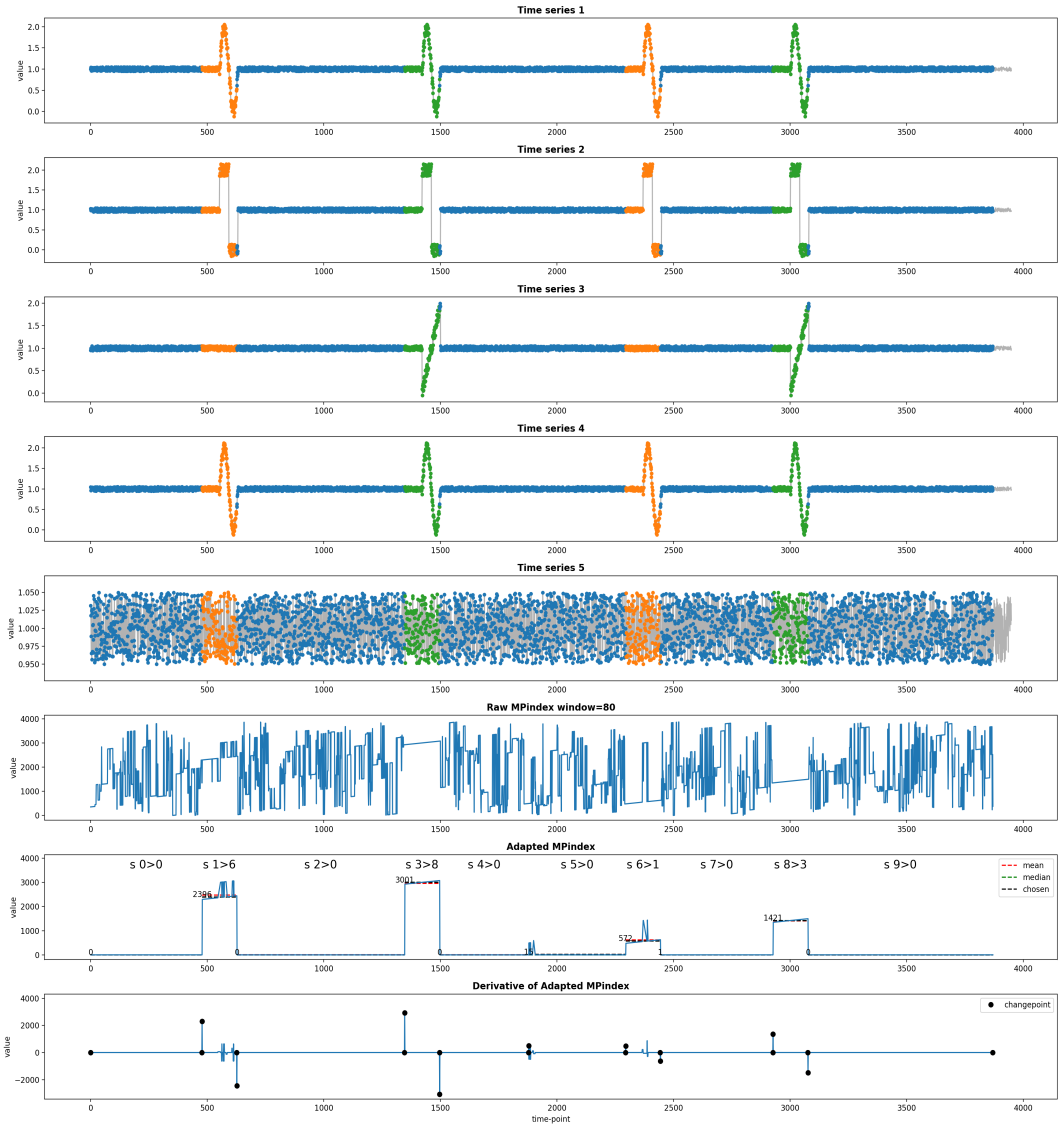


Figure 5.4.: Application of adapted MP*index* to the multidimensional T. The transformation of smoothing out the pointer noise from MP*index* to adapted MP*index* and goes until recognizing meaningful time series segment. [setup: sampling 1s, EMA=0, mp_window=80, f_window=43, sigma=1, cp_gap=140, cp_peak_t=400, mpindex_var_t=1000].

In the next section, we will describe the paths in obtaining the segment length, motif location, and the coloring scheme based on adapted MP*index* regardless of its multidimensional aspect.

5.1.2. Local Extremum in Matrix Profile Distance

Investigation on MP*distance* becomes fundamental in discovering motifs, segmentation, and clustering tasks. First, meaningful regions in the matrix profile need to be defined. This is done by selecting the peak/valley points in matrix profile which can be similar to searching for local minima or maxima in signal processing. In order to obtain these values in a matrix profile, one approach is to use the combination of gaussian kernel algorithm 3, scaling, and rolling the matrix profile values. The other option is by using *find_peaks* library in Scipy which works by comparing neighboring values in a signal or *find_peaks_cwt* with its wavelet transformation.

Algorithm 3: Local minima using Gaussian Filter

input : 1D array
output: *local minima*

1 gaussian_shape ← scipy.general_gaussian(p←1, standard_deviation, gaussian_width)
2 convolved ← FourrierConvolve(gaussian_shape, *distance*)
3 convolved ← $\mu_{MPdistance} / \mu_{convolved} \times$ convolved
4 convolved ← roll(convolved, window_size)
5 peaks ← relativeExtrema(convolved, elementWiseLessComparison)
6 `// select to only lowest peaks under median value of MP`*`distance`*
7 **return** *peaks*

The algorithm mentioned above needs parameter tuning such as the shape of the Gaussian with adjusting the p value. Thereafter, the data is filtered to the gaussian window that we already have using FFT convolve (line 4-6). The local extremum can be obtained when the Gaussian fits the data. The median value of the data is needed to set the threshold for picking the valley peaks with a value smaller than the median (line 9-10).

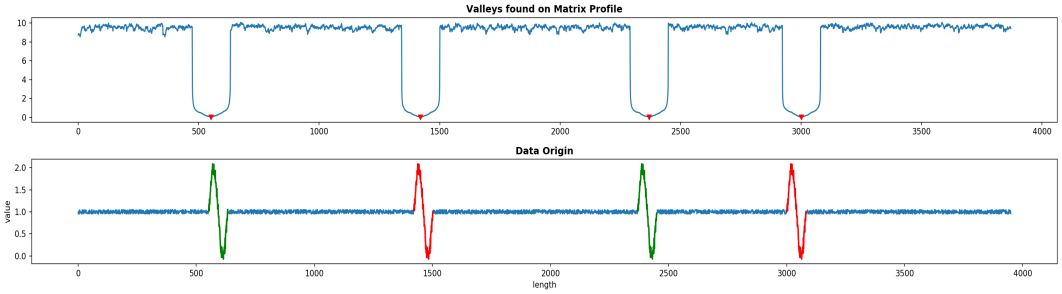


Figure 5.5.: Local extremum (marked with red points) found on a matrix profile with relative to its original data. Finding the peak valleys can determine the occurrence of patterns in time series.

In some cases, the technique mentioned above still could not find the peaks accurately, the other approaches are also being used interchangeably as long as the local extremum can be retrieved. After several experiments, most of the time, *find_peaks* performed the best. The results difference are shown in Figure A.8 and Figure A.9.

5.2. Matrix Profile-based Motifs Discovery

Motifs in time series represent a particular trend that happens which can be recurring many times in localized or different regions in data. Because multidimensional time-series data are involved in motif discovery tasks, our problem is described as multidimensional motif discovery. Retrieved peaks from the previous step are the best indication of motifs location, however, this alone can not be used to return the top motif. The length m of each subsequence motif needs to be known.

Ideally, when the pointers of the MP*index* are shifting far from the current average pointers value, it indicates another new regime such as described in section 3.3. While we previously took the abstraction of the MP*index* (adapted MP*index*), this leads to a new way of deriving the subsequence length required for each motif. A sudden jump of pointers in adapted MP*index* will show peak when the forward differential Equation 5.3 is applied. Meanwhile, the slope in constant increment or decrement in adapted MP*index* pointers will stay insignificant.

$$MPindex'_{fd}(x) = MPindex(x+1) - MPindex(x) \tag{5.2}$$

where

$$MPindex = adapted\_mpi \tag{5.3}$$

algorithm 4 describes the steps in obtaining the segment lengths m of $T_{i,m}$ is needed to recognize the subsequence motifs. Two parameters that are involved in this step are the minimum distance between changepoints *cp_gap* to marked down the segment region and the minimum height of the spike *cp_peak_t* occurred from taking the first derivative of MP*index*. It does not necessarily take the highest value of the spike that occurred is the valid changepoints or cuts between regions.

As shown in Figure 5.4 changepoints even occurred with a height between 140 and 200. In the same figure, other changepoints are detected around time-point $\approx$ 1800. We can

assume the *cp_gap* ≈ 140 will safely capture the important signals with length m and able in ignoring the rest. Hence a harmonize fit of *cp_gap* and *cp_peak_t* is necessary. Because the adjustment of these two values is currently taken by visual inspection with looking back into the original data, this also becomes part of the future works whether it is possible to create an optimization to tackle this problem.

After segments in the dataset are found respective to their changepoints, only some of them are important and could lead to meaningful motifs discovery. Motif in the matrix profile has a minimum distance to its nearest neighbor, while it is true by selecting the minimum values (peaks) in *MPdistance*. This also has been mentioned in subsection 3.4.2, where the idea behind top-k motifs is by looking for a couple of points (P_i, P_j) where the walk of $i \rightarrow j$ is the index range of motif window, where $j - i \leq m$. Therefore, the segment locations where there is one or more MP*distance* local minima lie in between are the indication of the top motifs.

Algorithm 4: Changepoints detection

input : adapted_mpi, cp_gap, cp_peak_t
output: changepoints

```
// Given adapted MPIndex and window size for matrix profile computation,
find the changepoints for data segmentation
// Part I
for i ← 0 to length of adapted_mpi do
    // derivative of Adapted MPIndex
    der_mpi ← adapted_mpi_{i+1} - adapted_mpi_i
// Part II: segment the time series based on der_mpi; find the
changepoint, a number of list will be created following to the number of
segments
list_seq, current_seq, changepoints ← new List
// window size in matrix profile can be used as approximation of tolerated
length between spikes in der_mpi
cp_gap ≈ window_length
for i ← 0 to length of adapted_mpi do
    if |der_mpi[i]| ≥ cp_peak_t then
        if length of current_seq ≤ cp_gap then
            list_seq.append(current_seq)
            current_seq ← empty list
            current_seq.append(i)
        else current_seq.append(i);
    else current_seq.append(i);
    list_seq.append(current_seq)
for i ← 0 to length of list_seq do
    // head and tail of the list_seq are the initial index and last index
    of a segment respectively
    changepoints.append(list_seq_head, list_seq_tail)
return changepoints
```

[Multidimensional motif algorithm 5] explains the process to find motifs by locating the peaks of minimum MP*distance*. The matrix profile algorithm that is used to compute the distance similarity of k multidimensional time-series will also return k dimensional MP*distance*. There-

fore, the steps in Part I- algorithm 2 need to be applied in finding the best k dimensionality for each MP*distance* data points. Afterwards, algorithm 3 can be enforced on the final 1-d MP*distance*.

Algorithm 5: Multidimensional Motif Discovery

input : changepoints, MP*distance*
output: motif/s location

```
foreach d ∈ MPdistance do
    peaks ← find_peaks(d) // local extrema in matrix profile distance
    peaks_list.append(peaks)
motifs_location, colors ← new Array
// color_1 for motif, color_2 for
motifs_color ← color_1, color_2
ismotif ← False
for i ← 0 to length of changepoints do
    // changepoints_{i,0}:  initial index of a segment
    initial_idx ← changepoints_{i,0}
    // changepoints_{i,1}:  last index of a segment
    last_idx ← changepoints_{i,1}
    for j ← 0 to length of peaks_list do
        // there is local minima in matrix profile between changepoints,
        which could indicate a motif
        if peaks_list_j ≥ initial_idx and peaks_list_j ≤ last_idx then
            motifs_location.append(initial_idx, last_idx)
            color.append(color_1)
            ismotif ← True
            break
        else color.append(color_2)
        ;
    if ismotif==False then
        color.append(color_2)
return color, motifs_location
```

Now as we already have the changepoints between segments and location of meaningful motifs, the effort continues by coloring the region of motifs and the rest with different colors.

In line. 16, the subsequences $T_{i,m}$ that represent a motif are colored differently with the rest of the non-motif subsequences (line 22). For instance, in case of gas turbine dataset, we want to detect the number of down-time right before the machine turns to idle.

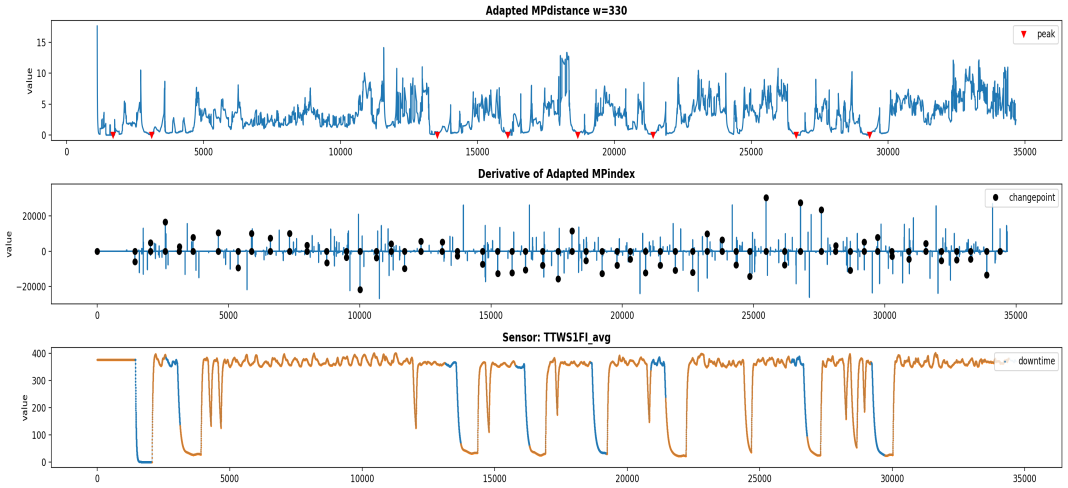


Figure 5.6.: Local minima (marked with red dots) found on the MP*distance* hints the location of interesting motif (marked in blue) which is the down-time before going to idle. The pair of *changepoint* helps in determining the motif window size.

We took a gas turbine data sample for almost a week's duration. The behavior of the gas turbine system shown in Figure 5.6 indicated there are eight times (marked in blue) that the machine went off to idle phase during this time sample, see Figure A.6 for full reference. This experiment was conducted on the power and temperature sensors group. On the other hand, the recognition is not always 100% correct which is shown in Figure A.7 using power and valve sensors group, see precision and recall in Table 5.1. There we can see a slight misclassification (in red patches), where the algorithm recognizes the transition from idle phase to up-time instead down-time.

Dataset	TP	FP	FN	P	R	F1score
Figure A.6 Power and temperature group	8	0	0	1	1	1
Figure A.7 Power and valve position group	7	1	1	$\frac{7}{8}$	$\frac{7}{8}$	0.875

Table 5.1.: Performance metric on down-time to idle phase motif recognition in gas turbine data. TP, FP, FN is the number of correct downtime recognition, the number of where the down-time motif is not recognizable and the number of the non-downtime motif is detected as down-time respectively.

We also try motif discovery for stable run data [gas turbine stable run Figure A.4], where the motif target is a short downtime (the value suddenly drops for a very short time window,e.g. sensor TTWS1F1_avg with time-point close to 50K). The setup of window size for mSTAMP is taken based to the length of a single drop motif. After getting the first derivative of the *adapted_mpi* and local minima in MP*distance*, in these variables we can see that there are no distinctive match between gap in the derivative and local minima location in MP*distance*. Therefore, the intended motif detection could not be achieved. In the same dataset, we can see motif candidates with larger window size as shown in Figure A.5. The analysis returns 2 local minima in MP*distance* which signifies a region with two sensors having a big transition. These length of the motif subsequence are defined by the gap length between changepoints.

5.3. Matrix Profile-based Segmentation

First, we would like to review a work that also develops a time series segmentation using MP*index* section 3.3. The construction of time series segmentation using matrix profile has relied on the number of crossing arcs across regions. The term "arcs" here means indexing pointers in MP*index* returned by mSTAMP. By counting the frequency on how many arcs crossing a certain region, they derived the segment cut based on the lowest number of crossing arcs. Apart from it can generate an exact number of labels, after experimenting CAC with a variety of datasets, we can conclude that the technique will produce segment cuts if there is a motif uniqueness between regions Figure 3.4, where the shift between one region in MPindex is visible. This prevents arcs or indexing pointers from crossing to another region far away with a similar motif. Detection on similar motifs within a region will enforce the MP*index* pointers to point a group of indices within its region and ignore other regions with similarity candidates.

Algorithm 6: CAC construction for Segmentation [14]

input : MP*index*, L: subsequence length
output: Filtered MP*index*

```
n ← new Array[length of MPIndex]
AC = CAC = nnmark ← new Array of zeros[length of L]
nnmark(i,j) ← NN(i) // count once the crossing arc if symmetrical
  neighboring, i → j and j → i
numArcs ← 0
for i ← 1 to L do
    numArcs ← numArcs + nnmark[i]
    AC[i] ← numArcs
IAC ← parabolic curve of length n and height 1/2 n
CAC ← min(AC/IAC, 1)
return CAC
```

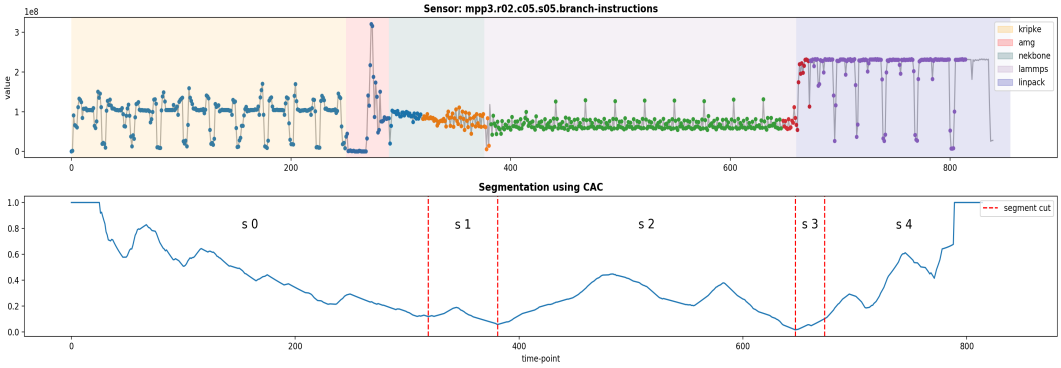


Figure 5.7.: *Coral2* (fetch latency group) dataset segmentation using CAC. The background patch colors are the ground truth while the predicted segment are colored in blue, orange, green, red and violet via scatter plot. [setup: sampling 1s, EMA=0, mp_window=26, filter=no, L=26].

Based on the fundamental work of CAC, we proposed a technique to approach time series segmentation using *changepoints* and *adapted_mpi* that we obtained from previous steps. After computing *adapted_mpi* derivative, we would pick a fairly good *cp_gap* and *cp_peak_t* threshold values for algorithm 4 input. It is recommended to take *cp_gap* with a value less or equal to the smallest segment in the dataset (if the ground truths are known). For instance, in *Coral2* benchmark dataset Figure 5.8, the smallest segment is from AMG application which is approximately 50 time-point. Hence, in this experiment, *cp_gap* is set to 50. The *mp_window*

for mSTAMP input, is chosen by considering the subsequence length m of a single unique motif algorithm 7.

The central distribution of MP*index* over a segment is taken as an approximated location to another segment with a similar motif pattern(line 8-9 in algorithm 7). Mean or median value is chosen depends on the MP*index* distribution in that segment(line 14-22). Ideally, MP*index* pointers in a segment should not abruptly change. The threshold value *mpindex_var_t* determines an allowed pointer variance range within a segment. When pointer variance passes the threshold, it is better to consider the median over mean. At the end of the algorithm, we can retrieve the potential matching segment in the time series dataset (line 18,22,24). To describe this better, we put the segment pair on top of the **Adapted MPindex** plot in each figure, e.g s 0>0.

The difference of our segmentation algorithm in contrast with CAC is the number of the intended label will not be guaranteed as these come naturally with the selection of *cp_gap* and *cp_peak_t*. Experiment shown in Figure 5.8 is able to recognize 4 different labels, however experiment with the same configurations in other DCDB sensors group Figure A.1 can retrieved 5 different segments (see number of segments in **Adapted MP*index*** plot). Meanwhile, the number of segments in CAC can be pre-defined before the algorithm.

Algorithm 7: Segmentation mapping

input : changepoints, mpindex_var_t
output: mapping of segments

```
segment_pairs ← [{}]
for i ← 0 to length of changepoints do
    // find the average or median of pointers for each segments in
    adapted_mpi
    // changepoints[i][0]:  initial index of a segment
    initial_idx ← changepoints[i][0]
    // changepoints[i][1]:  last index of a segment
    last_idx ← changepoints[i][1]
    average ← mean_{adapted_mpi[initial_idx:last_idx]}
    median ← median_{adapted_mpi[initial_idx:last_idx]}
    // find the max and minimum of pointers for each segment
    mpi_min ← min_{adapted_mpi[initial_idx:last_idx]}
    mpi_max ← max_{adapted_mpi[initial_idx:last_idx]}

    if mpi_max-mpi_min ≤ mpindex_var_t then
        // use average of a segment, to find segment mapping
        for j ← 0 to length of changepoints do
            if initial_idx ≤ average and last_idx ≥ average then
                destination[i] ← j
    else // use median of a segment, to find segment mapping
    for j ← 0 to length of changepoints do
        if initial_idx ≤ median and last_idx ≥ median then
            segment_pairs[i] ← j
    ;
return segment_pairs
```

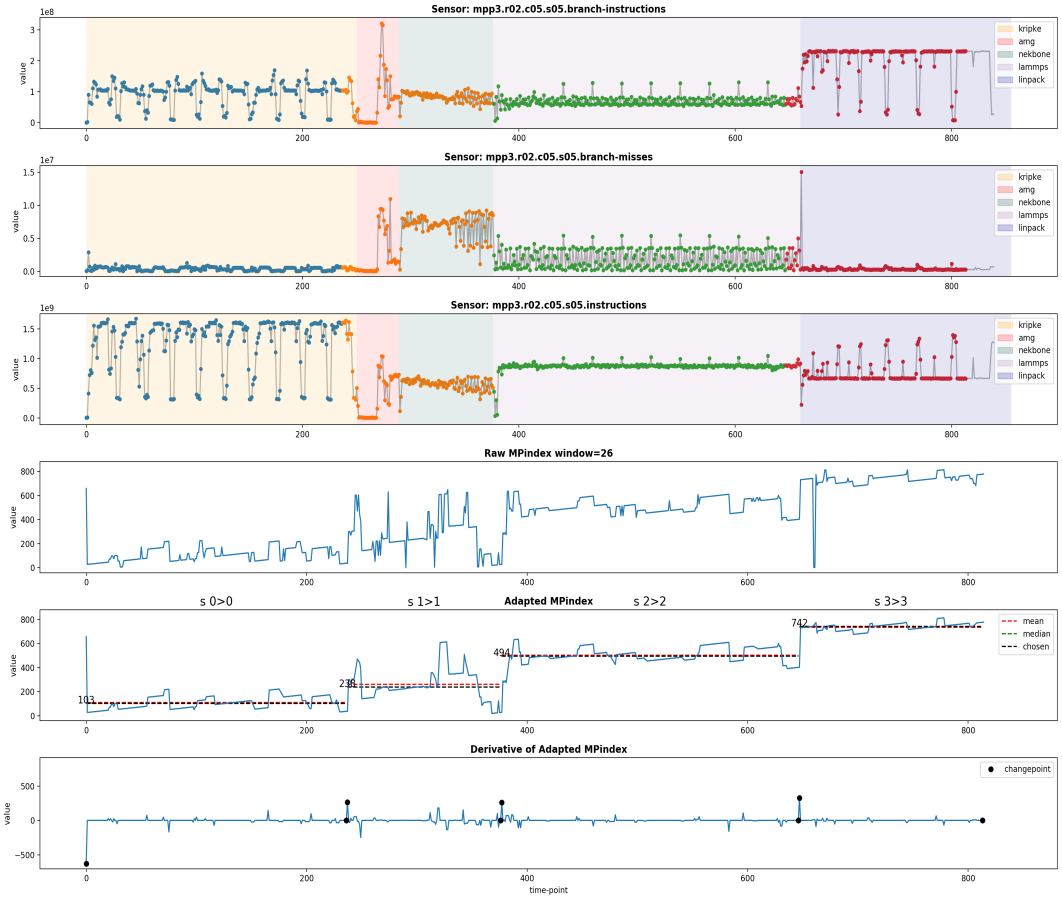


Figure 5.8.: *Coral2* (fetch latency group) segmentation [setup: sampling 1s, EMA=0, mp_window=26, f_window=11, sigma=1, cp_gap=50, cp_peak_t=190, mpindex_var_t=400].

The border between AMG and Nekbone in Figure 5.8 is not segmented properly. Further inspection on the region of AMG and Nekbone was done by trying with different experiment setups with a focus in tuning *mp_window* parameter. The rest parameters setup will follow the intuition of returned MP*index* from mSTAMP. After trying these time region alone in the whole process, our algorithm can distinguish these two regions over two different segments Figure 5.9. Hence, we took this *mp_window* into account back to the full dataset.

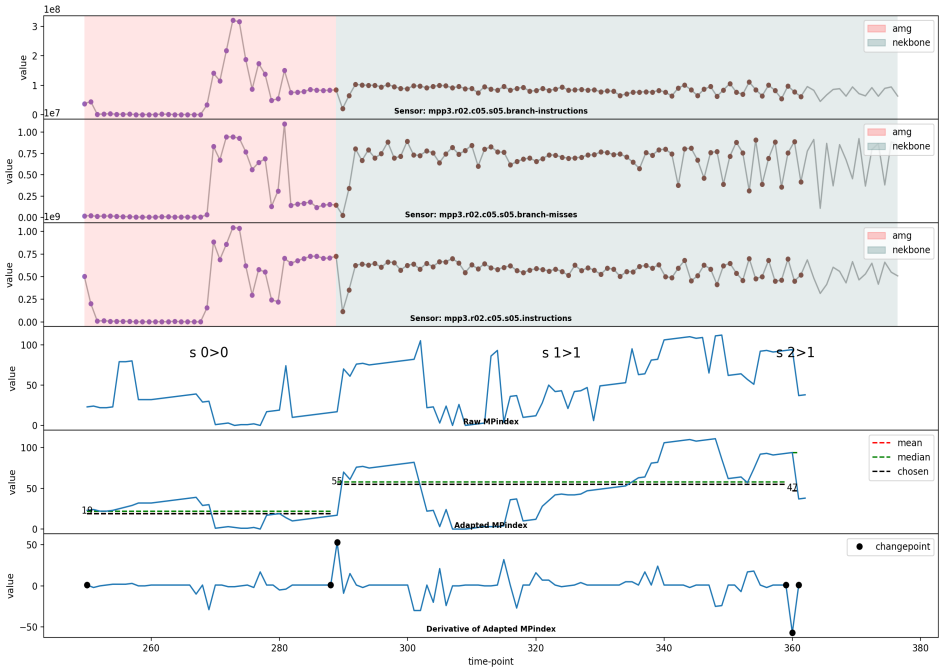


Figure 5.9.: *Coral2* (fetch latency group) partial segmentation from Figure 5.8 [setup: sampling 1s, EMA=0, mp_window=15, f_window=7, sigma=1, cp_gap=30, cp_peak_t=40].

Modification on *mp_window* from 26 to 15 has changed the segmentation Figure 5.10. It recognizes 5 segments instead of 4, which is identical to the number of ground truth labels. Although there are 5 different segments, our coloring scheme will follow the symmetrical rule because index 207 (pointer in s 1) is covered within the Kripke region, hence the coloring scheme in algorithm 8 will draw predicted Kripke and AMG with the same color, blue. This has resulted in the dataset having 4 different colors instead of 5.

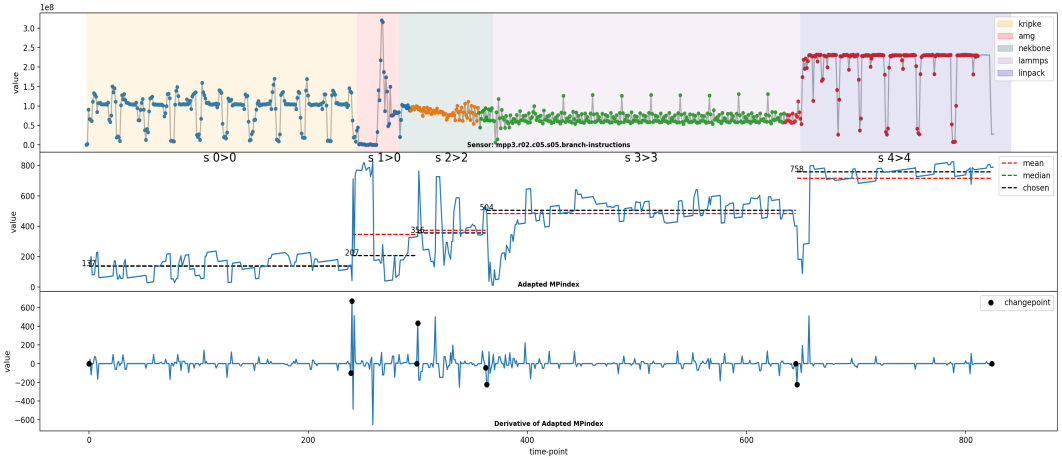


Figure 5.10.: *Coral2* (fetch latency group) segmentation in Figure 5.8 with different setup from Figure 5.8 [setup: sampling 1s, EMA=0, mp_window=15, f_window=7, sigma=1, cp_gap=50, cp_peak_t=200, mpindex_var_t=400].

Dataset	TP	FP	FN	P	R	F1score
Figure 5.7 FL segmentation with CAC, w=26	733	39	54	0.94	0.931	0.94
Figure 5.8 FL segmentation own analysis, w=26	725	85	30	0.89	0.96	0.92
Figure 5.10 FL segmentation own analysis, w=15	766	0	50	1	0.938	0.968

Table 5.2.: Performance metric from each experiment. FL: Fetch Latency dataset

The performance metric is measured by counting the total data point of our segment prediction. True positives mark every segment that can represent one ground-truth label. When there is a predicted segment that overreaching the ground truth labels boundary, it is considered as false negatives. False positives happen when a different label is recognized as the same label as the other, for instance, Nekbone is recognized altogether in s1 with AMG Figure 5.8.

6. Visualization

6.1. Literature Review

The final step in this thesis work is to represent our findings in better or more compelling visualization. Apart from that, an exploration in visualization has aided the writer in developing the method in the previous chapter. Over the years, there are many approaches to high-dimensional visualization research pathways. Categorization of recent advances research from major visualization venues (e.g Visweek, EuroVis, PacificVis, TVCG), visualization pipeline can be summarized in Table 6.1:

Data Transformation			
Dimension Reduction	Subspace Clustering	Regression Analysis	Topological Data Analysis
linear projection, non-linear projection, Control points projection, distance metric, precision measures	dimension space exploration, subset of dimension, Non-Axis-Parallel Subspace	Optimization design steering, structural summaries	Morse-smale complex reeb graph contour tree topological creatures

Visual Mapping					
Axis Based	Glyphs	PixelOriented	Hierarchy Based	Animation	Evaluation
Scatterplot Matrix, Parallel Coordinate, Radial Layout, Hybrid Construction	Per-Element Glyphs Multi-Object Glyphs	Jigsaw Map, Pixel Bar Charts, Circle Segment, Value and Relation Display	Dimension, Hierarchy, Topology-based	GGobi, TripAdvisor	Scatterplot Guideline, PCPs Effectiveness

Views			
Illustrative Rendering	Continuous Visual Representation	Accurate Color Blending	Image Space Metrics
Illustrative PCP, 3D Scatter Plot, Magic Lens	Continuous Scatterplot, Continuous Parallel Coordinate, Splatterplots	Hue-Preserving, Blending	Clutter Reduction, Pargnostics Pixnostics

Table 6.1.: Research categorization based on different stages of the visualization pipeline, with each, reflects common approaches. Source [44]. Note: cells in violet denotes the approaches used in this study.

6.2. Approach

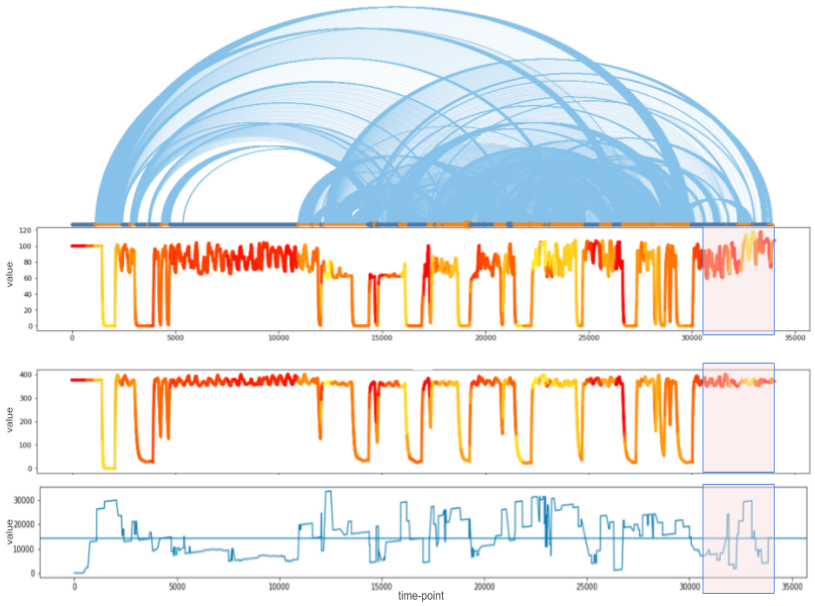
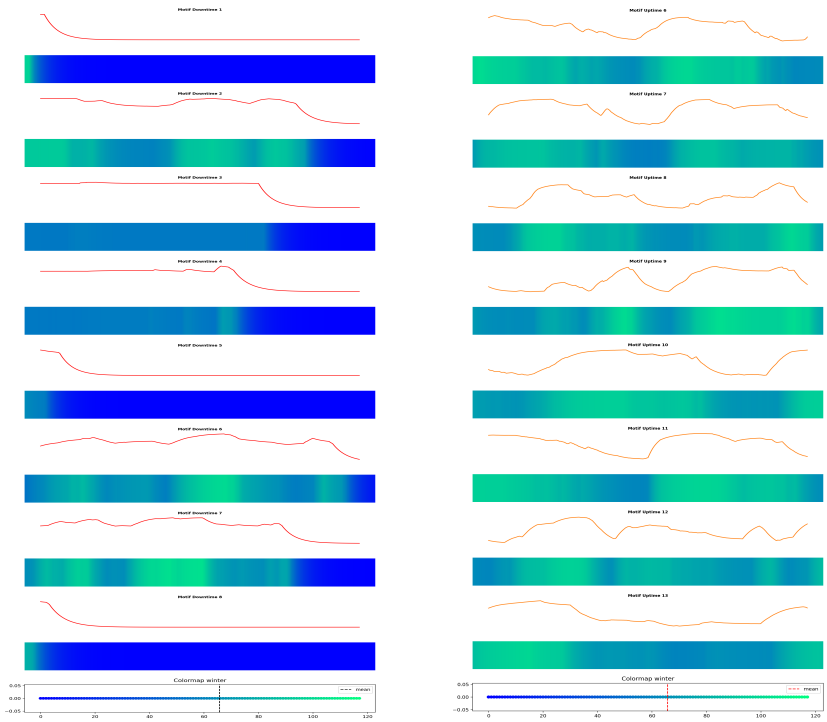


Figure 6.1.: Arc diagrams visualization over gas turbine dataset with many idling phases. The arc diagrams and coloring were drawn based on *adapted_mpi* without taking its first order of derivatives.

Our first attempt was to see whether applying *adapted_mpi* alone and pairing it with *Matplotlib* colormaps, will produce a good visual recognition on idling phases. From the figure, enforcing colormaps directly to *adapted_mpi* could not produce good pattern recognition. Later, another approach using arc diagrams was expected to improve the recognition of the pattern. This technique requires further MP*index* pointers and region filtering, such as only draw arcs if the value exceeds a certain threshold, by limit the arcs crossing distance. This is because irregular and abrupt changes of pointers in *adapted_mpi* will end up in overcrowded and chaotic arcs. This suggestion also persuaded by the fact that pointers in *adapted_mpi* do not always have symmetry relationship- algorithm 8. In [33] study showed arc diagrams visualization, where the mapped or connected subsequences are symmetrical.

In the same figure, we can also see red patches which illustrate that putting a threshold value (horizontal blue line) is still not enough to filter down the arcs in ignoring the non-idling region, while it worked well to the region around 5000 to 10000. Such a case occurs because indices between 5K and 10K are pointing to itself, while some indices in red patch (up-time region) are referencing to the region between 15K and 30K (idle regions). Therefore, further

steps in recognizing the exact segmentation are necessary.



(a) Down-time motifs.

(b) Up-time motifs.

(c) Mean of DWATT sensor.

Figure 6.2.: Extracted motif pairs with colorfields visualization for down-time and up-time classes- Figure 5.6. The colormap midpoint is shifted according to the mean of gas turbine DWATT sensor.

In reviewing motif visualization, we picked again the gas turbine dataset with contains the idling phase, as it has two distinct motifs, either idle or up-time motif. Our approach is to use color fields. As mentioned in section 7.1, for some cases humans perceive time series motifs better in *Colorfields* compare to *LineGraph*. After retrieving idle motifs in Figure 5.6, visualization in Colorfields starts by defining the colormap range. First of all, in this dataset, the value range between the up-time and idle phases can be separated properly by averaging the dataset Figure 6.2-section(c).

Later, this average value becomes the new midpoint in colormaps. Shifting the midpoint of colormaps will maintain the color range from Tosca to light-green for up-time (above mean=65) and dark-blue to Tosca-green for down-time (below mean=65). If the colormaps are not shifted following to the mean of the dataset, every low slope in up-time will be represented in dark blue as well. We can see in Figure 6.2-section(a), that every time the down-time transition occurs, color fields evolves to dark-blue, while in up-time motifs, color fields remain green Figure 6.2-section(b).

Algorithm 8: Time Series Segmentation Coloring

input : destination

output: colors grouping for each segment

```
// bruteforce in filling the color for each segment
colors ← Array[length of destination)]
for i ← 0 to length of destination do
    if colors[i] == "" then
        c ← next(matplotlib_color_cycle)
        colors[i] ← c
        colors[destination[i]] ← c
    // traceback to all destination keys
    for j ← 0 to length of destination do
        if destination[i] == j then
            colors[destination[j]] ← c
// final check for all coloring slots
for i ← 0 to length of destination do
    if colors[i] == "" then
        colors[i] ← colors[destination[i]]
return colors
```

In coloring the segments which were produced by Coral2 dataset experiments Figure 5.8, Figure 5.10, Figure A.1, there are at least 3 different scenarios such as described in Figure 3.3, e.g there are 2 segments **A** and **B**:

- Segment **A** points to index within itself.
- Segment **A** points to index within segment **B**, however pointer in **B** does not reference

back to segment **A** (asymmetry relationship).

- Segment **A** points to index within segment **B** and **B** also points back to **A** (symmetry relationship).

Our current approach is by filling all available segment slots with a different color at the start. While filling the color of every segment, it also finds out whether the current segment is pointing to another segment that already has a color (line. 9- algorithm 8). From here, the algorithm will match up the current segment coloring with its neighbor. This also defines that some segments belong to the same group. A brute-force approach of coloring segments with the same or different colors might create false recognition although the segmentation method works well. In our previous example, the size of ground truth labels is relatively small. We will keep the intention to see the performance of this coloring scheme in more massive ground truth labels and repeated labels over the dataset in our future works.

Following the idea, topological spaces in graph problem can be a candidate for better-connected segments tracing in order to give color them the same or differently [45] Figure 6.3. For instance, based on the connection path, segment 8^{th} and 6^{th} should be colored the same, although there is no direct pointer from segment 8^{th} to 6^{th}. Later, based on the group of colors, it can lead to the development of segments clustering.

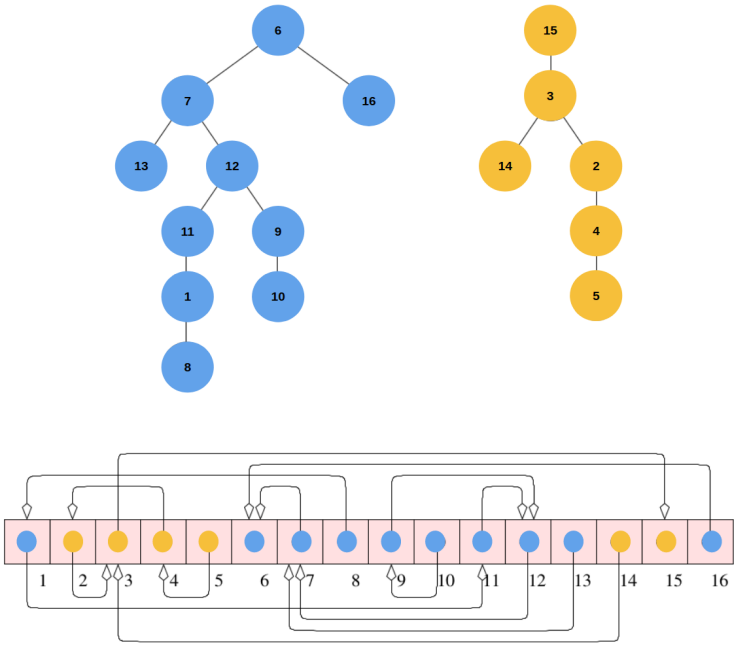


Figure 6.3.: Illustration of topological approach in time series segmentation coloring. The top two trees represents two disjoint sets in different color group. Bottom: linear array which stores the trees in ordering of the nodes. Image was recreated from Edelsbrunner, et.al [45].

The result from our segmentation coloring method can be proven using this theory. For instance in one of our experiment Figure 5.4, the sequence of the segmentation pairs are:

{s_0→s_0} {s_1→s_6} {s_2→s_0} {s_3→s_8} {s_4→s_0} {s_5→s_0} {s_6→s_1} {s_7→s_0} {s_8→s_3} {s_9→s_0}

By following the source and destination of every segment pair, we obtained three disjoint sets Figure 6.4 of color group. This validates the correctness of our coloring scheme- algorithm 8.

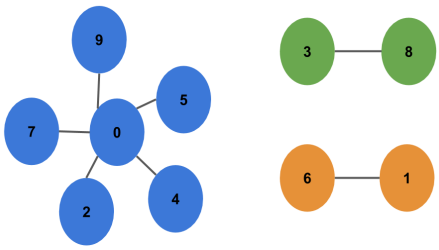


Figure 6.4.: Topological approach validates the correctness of our coloring scheme in Figure 5.4. It returns 3 disjoint sets in which the connected nodes in each group matched with the our segmentation coloring method's output.

The last topic in our visualization method is visualization using point cloud element, which basically can be prototyped using 3D scatter plot to retrieve its coordinates candidate. Each point represents a subsequence of time series with length m. The subsequence length should be close to the mp_window size when producing the segmentation. Hence, for instance, if a segment has a length of 100, divided by *mp_window* of 5, it will produce a 20 points cloud. In relation to multi-dimensionality, each point will represent n-subsequences from n-dimensionality. As a reminder, different segments either it close or crossing far which are represented in the same color should be clustered together in one center. Therefore, we need to have a map of a segment number and their color, then group the segments with similar colors together.

For visualization in 2D or 3D space, our current approach in picking the central coordinate of each cluster is by giving random values. Each point in the same group initially will be given the same coordinate values then the distribution around was adapted by adding Gaussian noise to each coordinate in the point. This method will give us clustered points as the initial coordinate values are equal and were shifted nearby by the noise. This visualization method enables us to explore the density of a segment compare to another. However, this work will be limited at this point, as we need to develop a better way to generate the cluster center coordinates, distribution of points in a cluster, and distances between clusters.

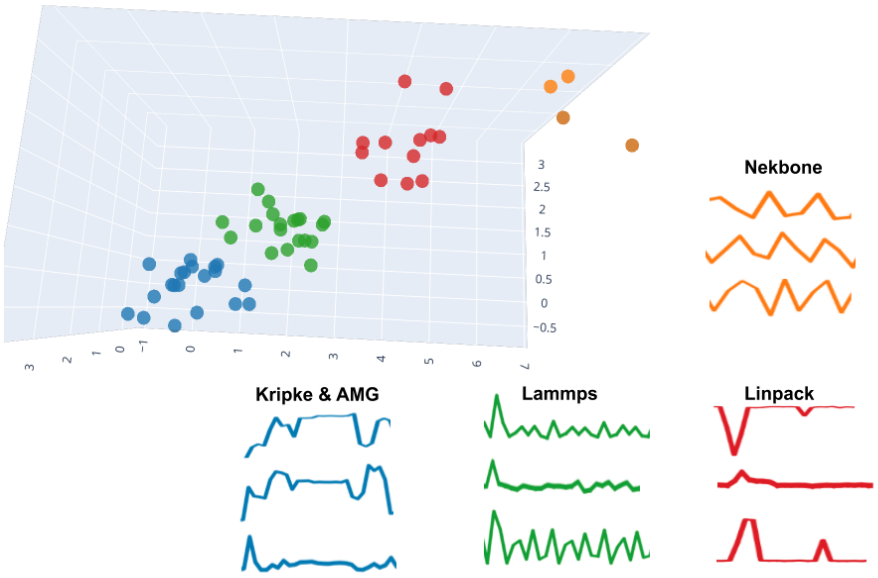


Figure 6.5.: Motifs density visualization of *Coral2* experiment- Figure 5.10 with scatter plot. Each point represents three subsequences which were deducted from 3 different sensors (mpp3.r02.c05.s05.branch-instructions; branch-misses; instructions). From the visualization, Nekbone looked to have spent less time in the benchmark run represented by less density scatter plot, compare to the other 3 applications (Kripke&AMG, Lammps, Linpack).

7. Evaluation and Conclusion

Matrix profile is a new method for motif discovery and meaningful segmentation with the advantage of scalability for very large datasets. For instance, in a non-parallelized version of mSTAMP, it took approximately 2.5 days of computation for processing around 400000 data points per sensor such as the experiment in Figure A.6. It also works well in high-dimensional as it can select naturally the best k dimensionality. However, we might state as well as involving more than one dimension in the analysis can be useful in extracting patterns. For a real-life example, a constant repeated arms movement might be considered as a single pattern. If we put another dimension into the analysis, e.g legs movement, with different speeds (walking or running), the analysis will detect two distinctive patterns instead of one, such as pattern 1 (arms movement while walking) and pattern 2 (arms movement while running).

7.1. User studies on time series similarity visualization

The value of the final visualization structure must be highly considered from a user perspective. Therefore, here we give one research example on user studies in visualization impression. Research in [46] conducted user studies with different backgrounds towards visualization in time series similarity. The time series visualization encoding techniques ranging from line charts(LC ) , horizon graphs(HG) and color fields(CF ). The task of the participants was to select the most similar time series in different visualization to a query(time series). The experiment started by collecting around 100-time series which is the nearest neighbor with the query. Measurement of participants' confidence was based on the type of the answers following to similarity algorithm (Euclidean Distance ED or Dynamic Time Wrapping DTW) and the shape of visualization(LC, HG, CF), time performance and their subjective assessment of the visualization structures. Subjective assessments (agreement) were scored by the probability of the Brennan coefficient, which assumes all q categories are selected by chance with the same probability $\frac{1}{q}$.

In terms of visualization encodings preference, participants' are more likely to complete the task faster with CF(7.5sec) where HG(9.1sec) and LC(13.9sec) are on the average. However, in subjective assessments, agreement value is generally low for HG and CF which implies higher subjectivity of participants' choices. Corresponding to the similarity algorithm, HG is showing a better tendency towards DTW, as ED answers were chosen with no consistency among participants. In conclusion for this experiment, participants preferred results returned by DTW. Color variations communicate high-level patterns, while shape and position reveal details. Participants may have focused on the high-level patterns in color while considering shape and position as secondary factors. Their results also suggest that CF is less appropriate for domains that require invariance to temporal wrapping (e.g DTW).

7.2. Limitations and Future Works

In our developed analysis model, we need to setup hyperparameters and preparation differently for each dataset. First of all, we need to have a best initial guess for matrix profile window based on the targeted analysis. Afterwards, some datasets might need to have data filtering before getting into the main process. Then, based on the output we might need to review the pre-processing step that we applied earlier on the original data for gaining a better decision. The parameters tuning also needed in every part of the algorithm especially in adapting the MP*index* in a way, after being processed it can still maintain essential cuts and information. As different parameters setup with a single numerical different can affect the final result, we can also apply probability measurement for each parameter setup. We might take an HMM (Hidden Markov Model) approach for counting the probability of segment pairing.

For future work in visualization, in the earlier time, we wanted to reach immersive data visualization starting with the application of point cloud using the game engine framework. This part of work is a very interesting thing to be explored later as it is not only involved more technological aspects and flexibility for data visualization itself.

7.3. Conclusion

We presented an evaluation of matrix profile in extracting pattern when handling multidimensional time series. We include 3 phases of visualization starting from data collection and formatting, then continued with data mining and transformation and in the final, we chose

one or two feasible visualization approaches. We also developed a further implementation of matrix profile to find its best k dimensionality and using this element to adapt the original MP*index* (matrix profile indices) and MP*distance* (matrix profile distance). The adapted profile index and distance are useful in obtaining the insight of various window sizes of trends or motifs. From time series motifs, we can extend the work into time-series segmentation.

The methods were experimented at first with simple multidimensional datasets then continues to be applied in real data. When generating performance counters data, the author can simulate a synthetic data generation in a controlled environment. Moreover, the ground truth of which applications were running in a specific time range is useful for validating the performance of the developed model. On the other hand, analyzing gas turbine data resembles investigation on a black box for the author.

The idea of defining segments using adapted_mpi at the end is more or less similar to SAX sequence Figure 2.4, however not to create a sequence fpr the raw time series data but for the MP*index*. The main difference is in SAX, lower dimensionality means an approximation of the time series data to their linear space. On the other hand, matrix profile does not need a reduced representation of the raw time data, while enabling us to explore data in high dimensional features. Thus, terms of "dimensionality reduction" in our work is to find out the best number of features or metrics that need to be included in the analysis.

The results of this work answered the research objectives- section 1.3. After many trials and errors, we can conclude that our model extension using matrix profile can be applied to reach our visualization targets. This finding also might contribute to the time series mining knowledge in the future.

A. Appendices

A.1. Results

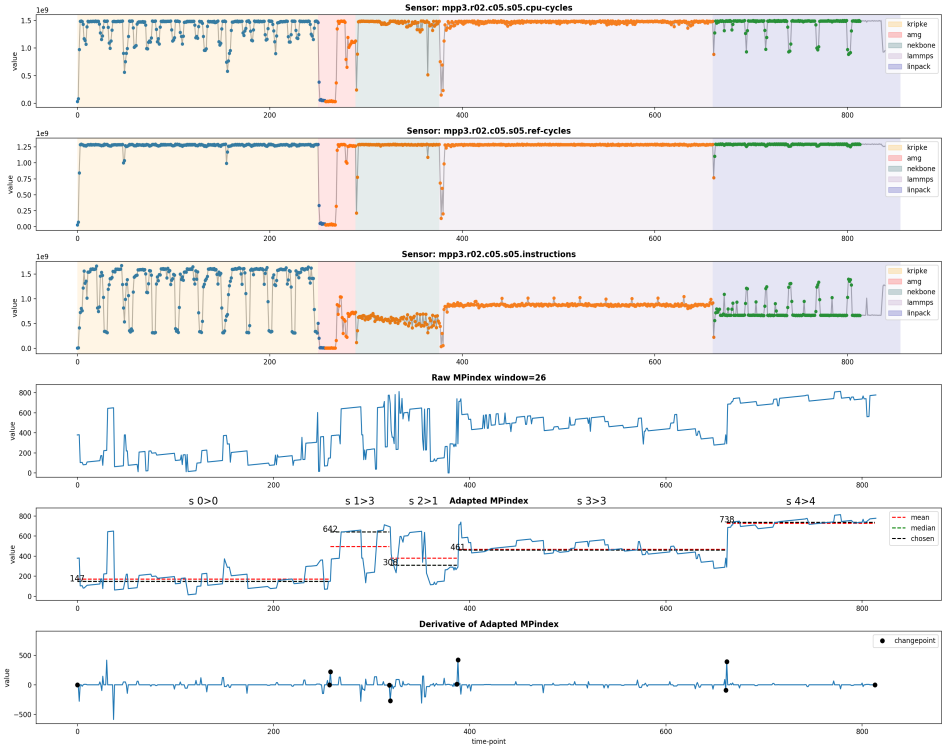


Figure A.1.: *Coral2* (base group) segmentation. It resulted in 4 segmentation regions with 3 different colors. [setup: sampling 1s, EMA=0, mp_window=26, f_window=11, sigma=1, cp_gap=50, cp_peak_t=190, mpindex_var_t=400]

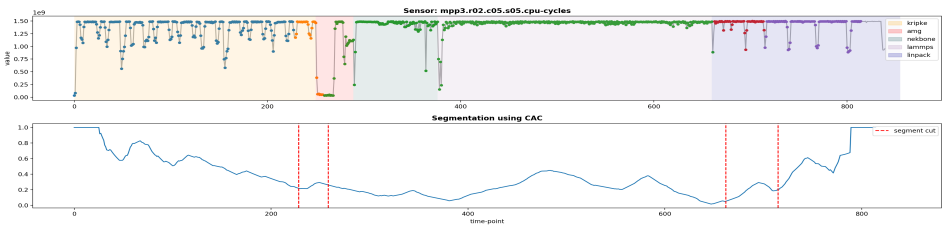


Figure A.2.: *Coral2* (base group) segmentation. [setup: sampling 1s, EMA=0, mp_window=26, filter=no, L=26]

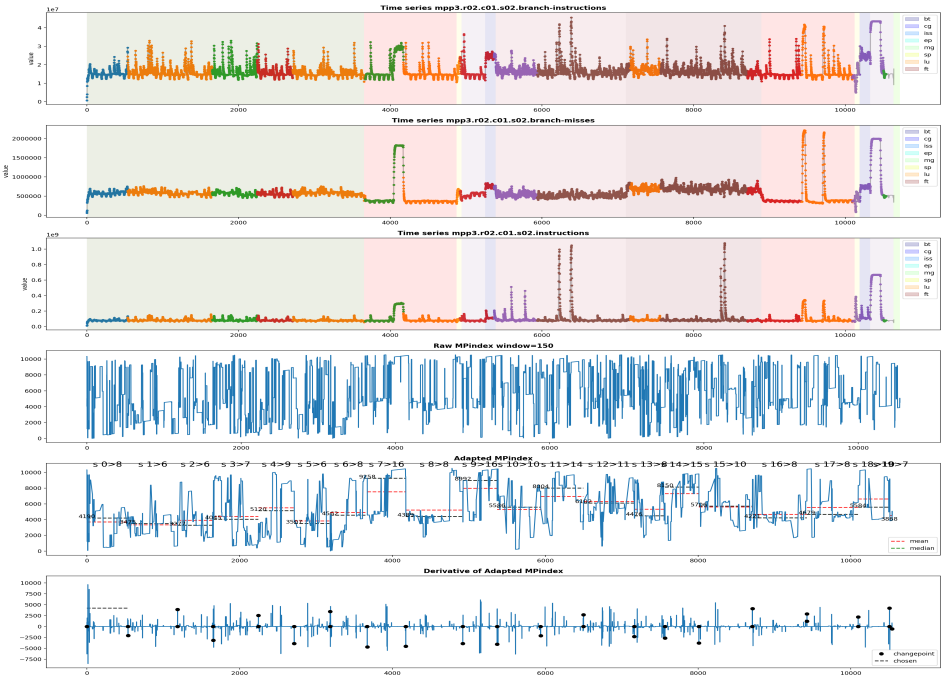


Figure A.3.: *NASA benchmark* (fetch latency) segmentation. An example of bad subsequence shape makes it difficult to be analyzed. [setup: sampling 1s, EMA=0, mp_window=100, f_window=101, sigma=1, cp_gap=50, cp_peak_t=190, mpindex_var_t=400]

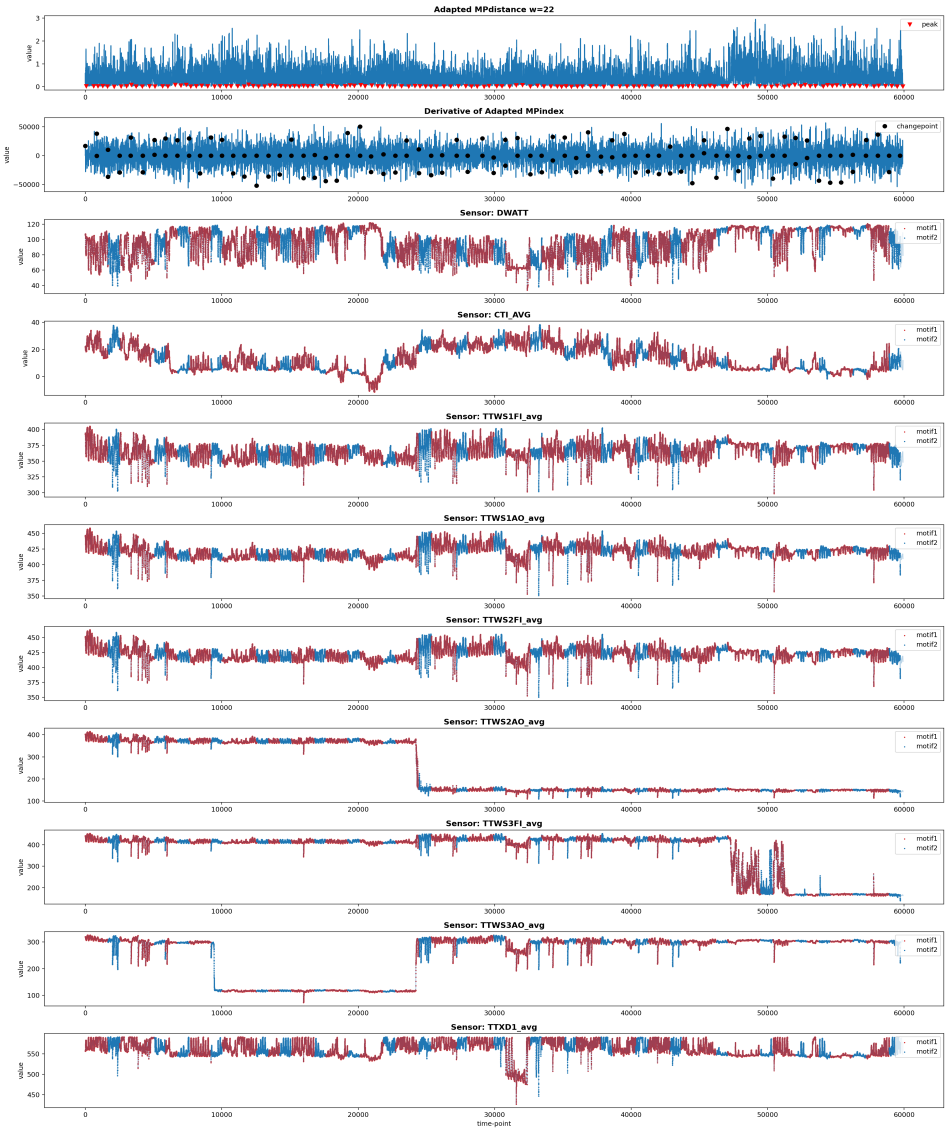


Figure A.4.: Motif Discovery in Gas Turbine (power and temperature) sensors steady run, period:2017-12 to 2018-03. [setup: sampling 20s, EMA=20, mp_window=22, f_window=11, sigma=1, cp_gap=800, cp_peak_t=26000]

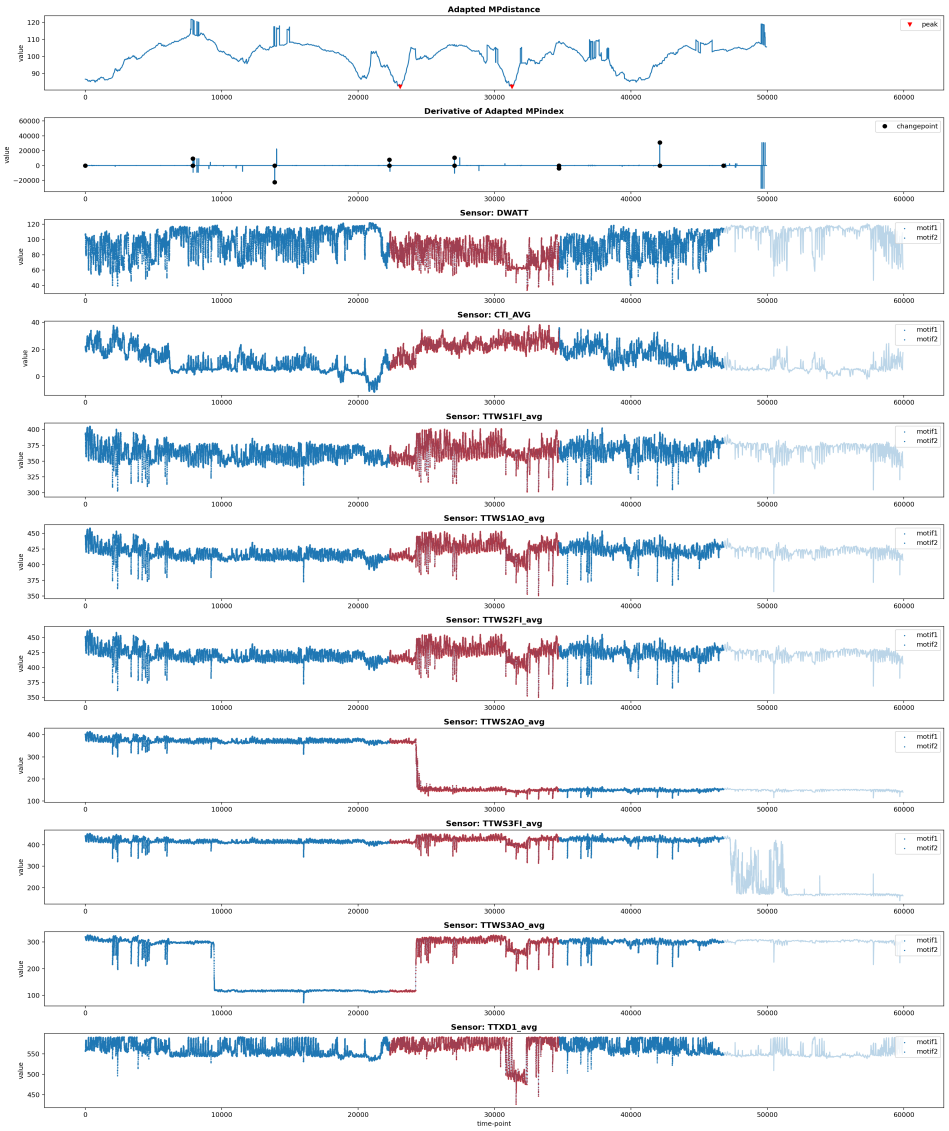


Figure A.5.: Motif Discovery in Gas Turbine (power and temperature) sensors steady run, period:2017-12 to 2018-03. [setup: sampling 20s, EMA=20, mp_window=10000, f_window=5001, sigma=1, cp_gap=4000, cp_peak_t=2000]

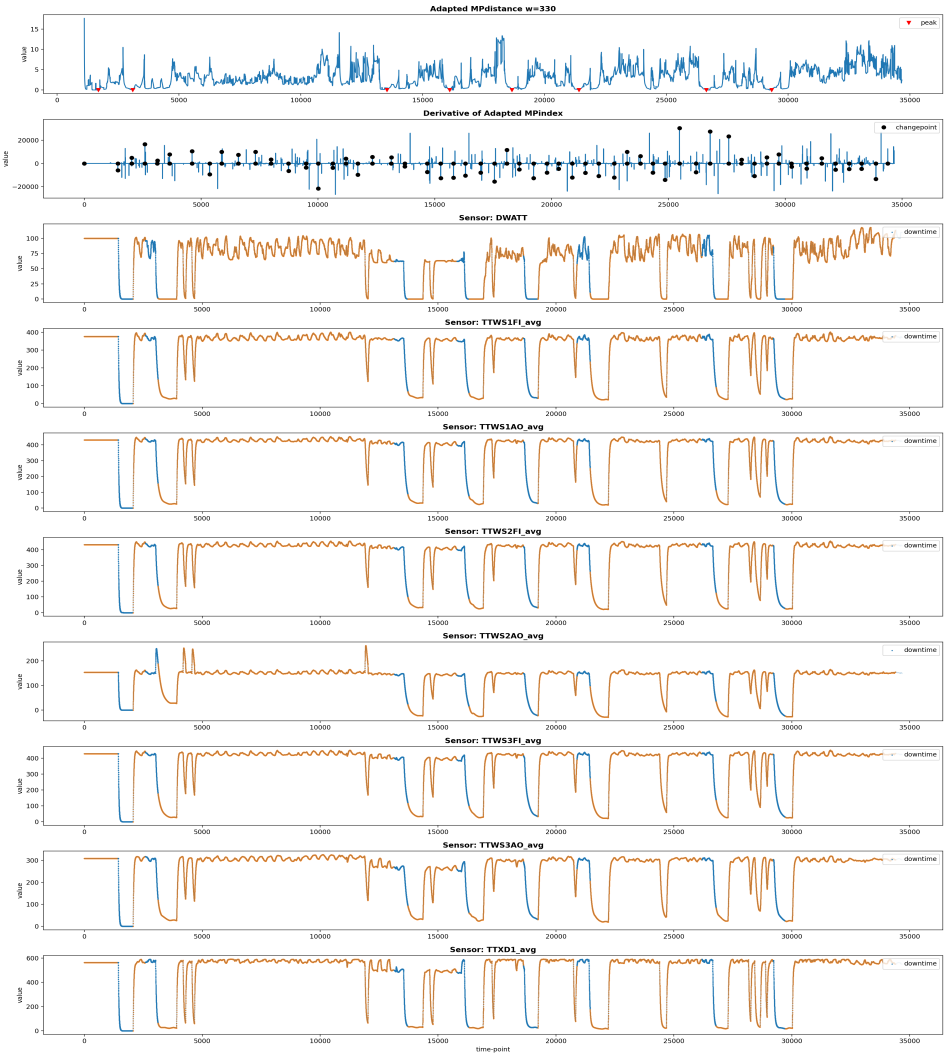


Figure A.6.: Downtime in idling phase detection (blue mark) in Gas Turbine (power and temperature) sensors, period:2018-07 to 2018-10. [setup: sampling 20s, EMA=20, mp_window=330, f_window=101, sigma=1, cp_gap=500, cp_peak_t=2500]

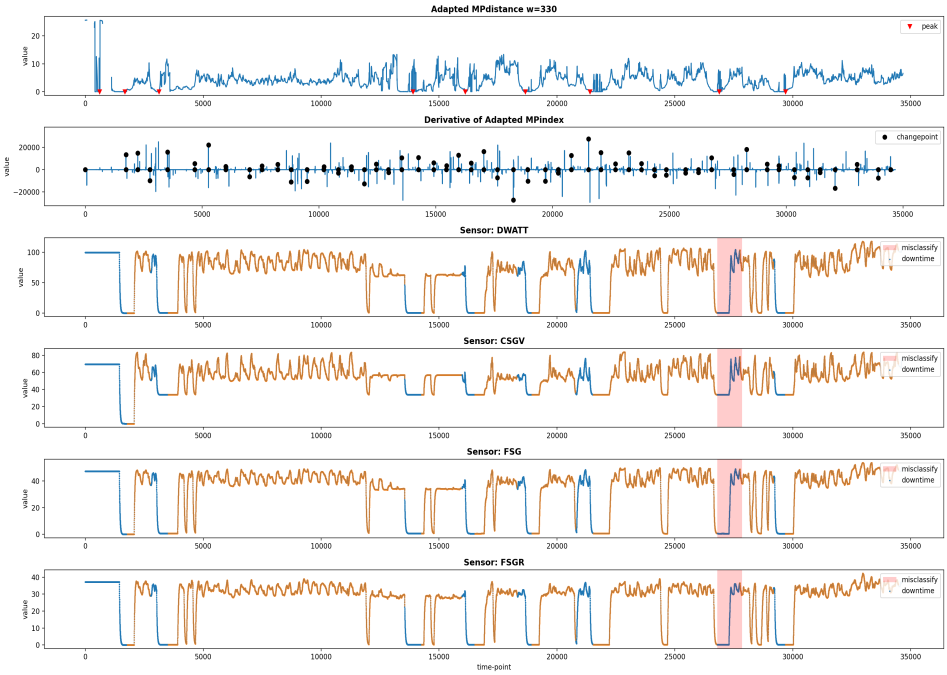


Figure A.7.: Downtime in idling phase detection (blue mark) in Gas Turbine (power and valve) sensors, period:2018-07 to 2018-10. [setup: sampling 20s, EMA=20, mp_window=330, f_window=101, sigma=1, cp_gap=500, cp_peak_t=2500]

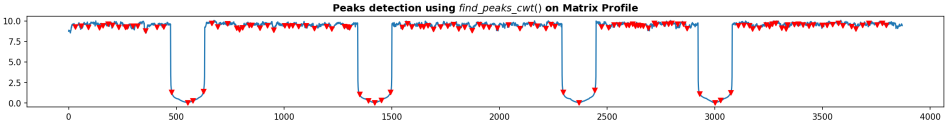


Figure A.8.: Peaks detection using scipy *find_peaks_cwt* of MP*distance* Figure 5.4

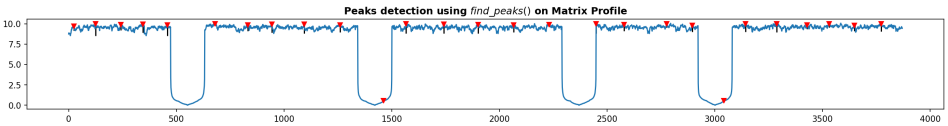


Figure A.9.: Peaks detection using scipy *find_peaks* of MP*distance* Figure 5.4

A.2. Experiment Setup Terms Summary

Variable name	Description
mp_window	window size for mSTAMP input
f_window	moving average window in EMA (Exponentialy Moving Average)
sigma	standard deviation, related to noise filtering in EMA
cp_gap	distance between a pair of changepoint in adapted_mpi derivative, related to the minimum window size of a segment
cp_peak_t	minimum threshold peak in adapted_mpi derivative
mpindex_var_t	tolerable pointers variance of a segment in adapted_mpi
adapted_mpi	MP*index* returned from mSTAMP that has been modified
adapted_mp	MP*distance* returned from mSTAMP that has been modified

Table A.1.: Experiment Setup Terms Summary.

A.3. DCDB sensors grouping for top-down perf. counters analysis

DCDB Sensor	Middle Layer	Top Layer
MemAvailable Vmalloc chunk Vmalloc total Vmalloc used alloc_stall cached	MemBound	BackendBound
col_idle col_iowait col_nice col_system col_user	CoreBound	
branch_misses	Branch Misspredicts	Bad Speculation
cpu_cycles ref_cycles instructions	Base	Retiring
branch_instructions branch_misses instructions	Fetch Latency	FrontendBound
Cached Buffers	Fetch Bandwidth	

Table A.2.: Grouping of DCDB sensors

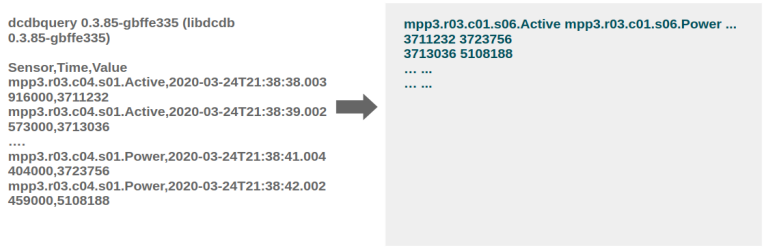
A.4. Gas Turbine sensors grouping

Sensors	Type
TNH_RPM	Speed
DWATT	Power
AFPAP *avg*(AFPCS,CPD) FPG3	Pressure
avg(CTIF1A,CTIM) *avg*(TTWS1FI1,TTWS1FI2) *avg*(TTWS1AO1,TTWS1AO2) *avg*(TTWS2FI1,TTWS2FI2) *avg*(TTWS2AO1,TTWS2AO2) *avg*(TTWS3FO1,TTWS3FO2) *avg*(TTWS3AO1,TTWS3AO2) *avg*(TTXD1_1,7,13,19,24)	Temperature
CSGV FSG FSGR	Valve Position
DLN_MODE_GAS L28FD	Gas and Flame

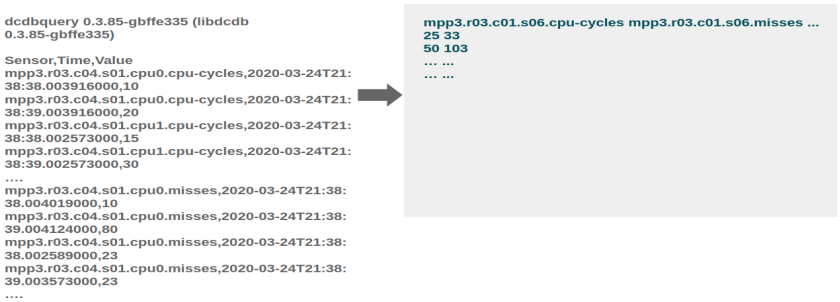
Table A.3.: Grouping of Gas Turbine Sensors

A.5. DCDB data preparation

Data acquisition needs to be prepared in order to fits the input for the algorithm and matrix profile. Therefore, conversion to a defined CSV format can be described as follows:



(a) Data Format for sensors with no-core decomposition.



(b) Data Format for sensors with cores decomposition. All cores values in a single sensor are summed up.

Figure A.10.: DCDB Data Format conversion in second precision timestamp.

A.6. LRZ CoolMUC-3 Hardware and Software Specifications

The experiments mostly ran in parallelized version of matrix profile. Jupyter notebook for data analysis and data acquisition of performance counters (via DCDB) were deployed in SuperMUC Linux Cluster (CoolMUC-3).

Hardware	Specification
CPU	1x Intel Xeon Phi CPU 7210F
Number of Nodes	148
Number of Cores	9472
Cores per Node	64
Hyperthreads per Core	4
Main Memory per Node	96 GB
Software	**Specification**
Operating System	SUSE Linux Enterprise Server (SLES12 SP2)
Batch System	SLURM
MPI	Intel MPI 2018 or 2019
Python	Python3.6
C++ Compilers	Intel icc, icpc

Table A.4.: Configuration of LRZ CoolMUC-3

List of Figures

List of Tables

Bibliography

[1] P. Esling and C. Agon. "Time-series data mining". In: *ACM Computing Surveys* 45.1 (2012), p. 12. DOI: 10.1145/2379776.2379788. URL: https://hal.archives-ouvertes.fr/hal-01577883.

[2] Y. Zhu, C.-C. M. Yeh, Z. Zimmerman, K. Kamgar, and E. Keogh. "Matrix Profile XI: SCRIMP++: Time Series Motif Discovery at Interactive Speeds". In: Nov. 2018, pp. 837–846. DOI: 10.1109/ICDM.2018.00099.

[3] G. Pfeilschifter. "Time Series Analysis with Matrix Profile on HPC Systems". Masterarbeit. Technische Universität München, 2019.

[4] M. Ali, M. Jones, X. Xie, and M. Williams. "TimeCluster: dimension reduction applied to temporal data for visual analytics". In: *The Visual Computer* (May 2019), pp. 1–14. DOI: 10.1007/s00371-019-01673-y.

[5] M. Nielsen. "High-Dimensional Data Visualization". In: 2017.

[6] S. Liu, D. Maljovec, B. Wang, P.-T. Bremer, and V. Pascucci. "Visualizing High-Dimensional Data: Advances in the Past Decade". In: Jan. 2015. DOI: 10.2312/eurovisstar.20151115. URL: https://ieeexplore.ieee.org/document/7784854.

[7] R. Sicat, J. Li, J. Choi, M. Cordeil, W. Jeong, B. Bach, and H. Pfister. "DXR: A Toolkit for Building Immersive Data Visualizations". In: *IEEE Transactions on Visualization and Computer Graphics* 25.1 (Jan. 2019), pp. 715–725. ISSN: 2160-9306. DOI: 10.1109/TVCG.2018.2865152.

[8] C. Ratanamahatana and E. Keogh. "Making Time-Series Classification More Accurate Using Learned Constraints." In: Apr. 2004. DOI: 10.1137/1.9781611972740.2.

[9] J. Serrà and J. L. Arcos. "An Empirical Evaluation of Similarity Measures for Time Series Classification". In: *Knowledge-Based Systems* 67 (Jan. 2014). DOI: 10.1016/j.knosys.2014.04.035.

[10] C. Cassisi, P. Montalto, M. Aliotta, A. Cannata, and A. Pulvirenti. "Similarity Measures and Dimensionality Reduction Techniques for Time Series Data Mining". In: Sept. 2012. DOI: 10.5772/49941.

[11] D. Guijo-Rubio, A. M. Durán-Rosal, P. A. Gutiérrez, A. Troncoso, and C. Hervás-Martínez. "Time-Series Clustering Based on the Characterization of Segment Typologies". In: *IEEE Transactions on Cybernetics* (2020), pp. 1–14.

[12] A. J. Bagnall, A. Bostrom, J. Large, and J. Lines. "The Great Time Series Classification Bake Off: An Experimental Evaluation of Recently Proposed Algorithms. Extended Version". In: *CoRR* abs/1602.01711 (2016). arXiv: 1602.01711. URL: http://arxiv.org/abs/1602.01711.

[13] E. Keogh, S. Chu, D. Hart, and M. Pazzani. "An online algorithm for segmenting time series". In: *Proceedings 2001 IEEE International Conference on Data Mining*. Nov. 2001, pp. 289–296. DOI: 10.1109/ICDM.2001.989531.

[14] S. Gharghabi, C.-C. M. Yeh, Y. Ding, W. Ding, P. Hibbing, S. LaMunion, A. Kaplan, S. E. Crouter, and E. Keogh. "Domain agnostic online semantic segmentation for multi-dimensional time series". In: *Data Mining and Knowledge Discovery* 33.1 (Jan. 2019), pp. 96–130. ISSN: 1573-756X. DOI: 10.1007/s10618-018-0589-3. URL: https://doi.org/10.1007/s10618-018-0589-3.

[15] C. F. Jekel and G. Venter. *pwlf: A Python Library for Fitting 1D Continuous Piecewise Linear Functions*. 2019. URL: https://github.com/cjekel/piecewise_linear_fit_py.

[16] E. Keogh and J. Lin. "Clustering of time-series subsequences is meaningless: Implications for previous and future research". In: *Knowledge and Information Systems* 8 (Aug. 2005), pp. 154–177. DOI: 10.1007/s10115-004-0172-7.

[17] D. Shipmon, J. Gurevitch, P. Piselli, and S. Edwards. "Time Series Anomaly Detection; Detection of anomalous drops with limited features and sparse examples in noisy highly periodic data". In: (Aug. 2017).

[18] M. Verleysen and D. François. "The Curse of Dimensionality in Data Mining and Time Series Prediction". In: vol. 3512. June 2005, pp. 758–770. DOI: 10.1007/11494669_93.

[19] C.-C. M. Yeh, H. Herle, and E. Keogh. "Matrix Profile III: The Matrix Profile Allows Visualization of Salient Subsequences in Massive Time Series". In: Dec. 2016, pp. 579–588. DOI: 10.1109/ICDM.2016.0069.

[20] D. Jäckle, F. Fischer, T. Schreck, and D. Keim. "Temporal MDS Plots for Analysis of Multivariate Data". In: *IEEE Transactions on Visualization and Computer Graphics* 22 (Nov. 2015), pp. 1–1. DOI: 10.1109/TVCG.2015.2467553.

[21] C. M. Yeh, Y. Zhu, L. Ulanova, N. Begum, Y. Ding, H. A. Dau, D. F. Silva, A. Mueen, and E. Keogh. "Matrix Profile I: All Pairs Similarity Joins for Time Series: A Unifying View That Includes Motifs, Discords and Shapelets". In: *2016 IEEE 16th International Conference on Data Mining (ICDM)*. Dec. 2016, pp. 1317–1322. DOI: 10.1109/ICDM.2016.0179.

[22] Y. Tanaka, K. Iwamoto, and K. Uehara. "Discovery of Time-Series Motif from Multi-Dimensional Data Based on MDL Principle". In: *Machine Learning - ML* 58 (Feb. 2005), pp. 269–300. DOI: 10.1007/s10994-005-5829-2.

[23] S. Spiegel, B. Jain, E. De Luca, and S. Albayrak. "Pattern recognition in multivariate time series: Dissertation proposal". In: Oct. 2011. DOI: 10.1145/2065003.2065011.

[24] W. Aigner, S. Miksch, W. Müller, H. Schumann, and C. Tominski. "Visual Methods for Analyzing Time-Oriented Data". In: *IEEE Transactions on Visualization and Computer Graphics* 14.1 (Jan. 2008), pp. 47–60. ISSN: 2160-9306. DOI: 10.1109/TVCG.2007.70415.

[25] Z. Zimmerman, K. Kamgar, N. Shakibay Senobari, B. Crites, G. Funning, P. Brisk, and E. Keogh. "Matrix Profile XIV: Scaling Time Series Motif Discovery with GPUs to Break a Quintillion Pairwise Comparisons a Day and Beyond". In: Nov. 2019, pp. 74–86. ISBN: 978-1-4503-6973-2. DOI: 10.1145/3357223.3362721.

[26] A. Raoofy, R. Karlstetter, D. Yang, C. Trinitis, and M. Schulz. "Time Series Mining at Petascale Performance". In: *High Performance Computing*. Ed. by P. Sadayappan, B. L. Chamberlain, G. Juckeland, and H. Ltaief. Springer International Publishing, 2020, pp. 104–123.

[27] URL: https://www.cs.ucr.edu/~eamonn/MatrixProfile.html.

[28] C.-C. M. Yeh, N. Kavantzas, and E. Keogh. "Matrix Profile VI: Meaningful Multidimensional Motif Discovery". In: Nov. 2017, pp. 565–574. DOI: 10.1109/ICDM.2017.66.

[29] V. Pavlovic, J. Rehg, and J. Maccormick. "Learning Switching Linear Models of Human Motion". In: vol. 2000. Jan. 2000, pp. 981–987.

[30] URL: https://stumpy.readthedocs.io/en/latest/Tutorial_Semantic_Segmentation.html.

[31] M. Hao, M. Marwah, H. Janetzko, R. Sharma, D. Keim, U. Dayal, D. Patnaik, and N. Ramakrishnan. "Visualizing Frequent Patterns in Large Multivariate Time Series". In: (Jan. 2011). DOI: 10.1117/12.872169.

[32] D. A. Keim, J. Schneidewind, and M. Sips. "Scalable Pixel Based Visual Data Exploration". In: *Pixelization Paradigm*. Ed. by P. P. Lévy, B. Le Grand, F. Poulet, M. Soto, L. Darago, L. Toubiana, and J.-F. Vibert. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 12–24. ISBN: 978-3-540-71027-1.

[33] M. Wattenberg. "Arc diagrams: visualizing structure in strings". In: *IEEE Symposium on Information Visualization, 2002. INFOVIS 2002.* 2002. URL: http://ieg.ifs.tuwien.ac.at/~aigner/teaching/ws06/infovis_ue/papers/arcdiagram_01173155.pdf.

[34] D. Smilkov, N. Thorat, C. Nicholson, E. Reif, F. B. Viégas, and M. Wattenberg. "Embedding Projector: Interactive Visualization and Interpretation of Embeddings". In: *ArXiv* abs/1611.05469 (2016).

[35] URL: https://projector.tensorflow.org/.

[36] C. Donalek, S. G. Djorgovski, S. Davidoff, A. Cioc, A. Wang, G. Longo, J. S. Norris, J. Zhang, E. Lawler, S. Yeh, A. Mahabal, M. J. Graham, and A. J. Drake. "Immersive and Collaborative Data Visualization Using Virtual Reality Platforms". In: *CoRR* abs/1410.7670 (2014). arXiv: 1410.7670. URL: http://arxiv.org/abs/1410.7670.

[37] V. Kraft. "Efficient rendering of massive and dynamic point cloud data in state-of-the-art graphics engines On the example of the Unreal Engine". In: (2019). URL: https://cgvr.cs.uni-bremen.de/theses/finishedtheses/render/thesis_FINAL_WEB.pdf.

[38] URL: https://asc.llnl.gov/coral-2-benchmarks/.

[39] A. Netti, M. Mueller, A. Auweter, C. Guillen, M. Ott, D. Tafani, and M. Schulz. "From Facility to Application Sensor Data: Modular, Continuous and Holistic Monitoring with DCDB". In: *CoRR* abs/1906.07509 (2019). arXiv: 1906.07509. URL: http://arxiv.org/abs/1906.07509.

[40] In: (). URL: https://computing.llnl.gov/tutorials/mpi/.

[41] A. Yasin. "A Top-Down method for performance analysis and counters architecture". In: *2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. 2014, pp. 35–44.

[42] T. A. Runkler. *Data Mining: Modelle und Algorithmen intelligenter Datenanalyse*. 2nd ed. Wiesbaden, 2015. DOI: 10.1007/978-3-8348-2171-3.

[43] N. Tatbul, T. J. Lee, S. Zdonik, M. Alam, and J. Gottschlich. "Precision and Recall for Time Series". In: *Advances in Neural Information Processing Systems 31*. Ed. by S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett. Curran Associates, Inc., 2018, pp. 1920–1930. URL: http://papers.nips.cc/paper/7462-precision-and-recall-for-time-series.pdf.

[44] S. Liu, D. Maljovec, B. Wang, P. Bremer, and V. Pascucci. "Visualizing High-Dimensional Data: Advances in the Past Decade". In: *IEEE Transactions on Visualization and Computer Graphics* 23.3 (2017), pp. 1249–1268.

[45] H. Edelsbrunner and J. Harer. *Computational Topology: An Introduction*. Jan. 2010. ISBN: 978-0-8218-4925-5. DOI: 10.1007/978-3-540-33259-6_7.

[46] A. Gogolou, T. Tsandilas, T. Palpanas, and A. Bezerianos. "Comparing Similarity Perception in Time Series Visualizations". In: *IEEE Transactions on Visualization and Computer Graphics*. TVCG 2019 (InfoVis 2018) 25.1 (Oct. 2018), pp. 523–533. URL: https://hal.inria.fr/hal-01845008.