Fault Detection and Classification of Time Series Using Localized Matrix Profiles

Jing Zhang, Daniel Nikovski, and Teng-Yok Lee
Mitsubishi Electric Research Labs
Cambridge, MA 02139, USA
Email: {jingzhang,nikovski,tlee}@merl.com

Tomoya Fujino
Mitsubishi Electric Corporation
Kamakura, Kanagawa 247-8501, Japan
Email: Fujino.Tomoya@cj.MitsubishiElectric.co.jp

***Abstract*—We introduce a new primitive, called the Localized Matrix Profile (LMP), for time series data mining. We devise fast algorithms for LMP computation, and propose a fault detector and a fault classifier based on the LMP. A case study using synthetic sensor data generated from a physical model of an electrical motor is provided to demonstrate the effectiveness and efficiency of our approach.**

***Index Terms*—sensors, multivariate time series, localized matrix profile, fault detection, fault classification**

I. Introduction

Time series data generated by sensors in physical systems and data mining on such data are widely used in applications of Prognostics and Health Management (PHM). Recently, a research group from the University of California at Riverside (UCR) have proposed a powerful tool – the Matrix Profile (MP) [1] as a primitive to tackle various time series data mining tasks, including discovery of motifs, discords and shapelets [2], time series joins [3], visualization [4], semantic segmentation [5], etc. They have also devised algorithms to compute MP efficiently [3], making it very scalable.

When only one sensor is used to generate time-dependent sequence data, we would end up with a Univariate Time Series (UTS) that contains one single variable. Letting $\mathbf{x} = (x(t)); [t = 1, 2, \ldots]$ denote such a UTS, the MP of $\mathbf{x}$ is a data structure that annotates the time series $\mathbf{x}$ itself – the MP at the t-th location of time series $\mathbf{x}$ records the distance of the subsequence (with a predesignated length depending on domain knowledge and applications) in $\mathbf{x}$, at the t-th location, to its nearest neighbor in $\mathbf{x}$. Given the MP of $\mathbf{x}$, relatively high values indicate the subsequence in the original time series $\mathbf{x}$ must be unique in its shape (such areas are "discords" or anomalies). This intuitive observation enables us to use the MP for purposes of certain PHM tasks such as fault detection and classification. However, relatively low MP values do not necessarily correspond to non-discords; see, e.g., the "twin freak" issue discussed in [1]. As a matter of fact, for fault detection and/or fault classification tasks, the MP is ideally applicable only for time-stationary physical systems or processes, because it captures variational information of sensor data all the way along the time – if the data themselves are statistically time varying, identifying discords (faults) through the MP would be difficult.

978-1-5386-8357-6/19/$31.00 ©2019 IEEE

In practice, it is broadly seen that time series data are generated from not only multiple sensors but also time-varying systems or processes that have lots of transients [6]. Thus, it is inevitable to operate on Multivariate Time Series (MTS) generated by multiple types of sensors in time-varying systems or processes. Conceptually, it is not hard to adapt the MP of a single UTS to that of a UTS *joining* (i.e., *with respect to*) another UTS or multiple UTS, and even to that of a single MTS or to that of a MTS joining another MTS or multiple MTS. However, such direct adaptations would still not be able to resolve the "twin freak" issue in time-varying systems or processes. Essentially, the MP requires nearest neighbor searching over all available subsequences involved in the UTS or MTS, but such exhaustive searching would lead to false positives and/or false negatives easily, for instance, in a typical fault detection application for a time-varying system. These limitations motivate us to introduce a Localized version of the Matrix Profile (LMP), a new time series primitive which is applicable in time-varying systems or processes for certain PHM tasks including fault detection and classification. A direct brute-force approach computing LMP is inefficient; thus, we also devise a faster algorithm to compute the LMP.

In this paper, when dealing with PHM tasks such as fault detection and classification, we focus on investigating window-based faults that are manifested as subsequences of time series that are significantly different from the subsequences observed in normal conditions. For a particular subsequence length, we can work with a sliding window of that length over the time series, and analyze how different or similar it is from windows in normal time series. Window-based approaches are particularly useful for real-time PHM applications on large-sized streaming data, which are widely observed in all kinds of networked systems; e.g., communication networks [7]–[8], social networks [9], [10], transportation networks [11], [12]. In this work, we propose a fault detector and a fault classifier based on the LMP. To demonstrate the performance of our methods, a case study on fault diagnosis for an electrical motor using simulated sensor data is presented.

The rest of the paper is organized as follows. In Sec. II we define LMP with sufficient preliminaries including an adaptation of UCR's original definition of MP. In Sec. III we devise algorithms for computing LMP, propose a fault detector based on the LMP, and propose a fault classifier

based on the LMP. We present the case study in Sec. IV and conclude in Sec. V. Supplementary materials are contained in the Appendix.

II. Definitions

Throughout the paper, we only consider transient systems (physical devices such as electrical motors) where the runs are correctly time stamped and synchronized, so that we are comparing data from the same time/stage in the process. In particular, we assume that multiple sensors are deployed in the system to generate time series data characterizing the properties of the physical process. Thus, for a typical run of such a system, we end up with a Multivariate Time Series (MTS). As a foundation of certain PHM tasks such as fault detection and classification, we need to collect a number of baseline MTS items corresponding to normal runs of the system. Then, we investigate test MTS items with respect to the baseline ones.

Before proceeding, let us introduce some notation. Let N be the number of variables in the raw MTS data (i.e., number of sensors for collecting MTS data); it is assumed to be the same for all MTS items. Denote by n the variable (sensor) index, where $n = 1, \ldots, N$. Let T be the number of time instances recorded by sensors, which is also assumed to be the same for all MTS items. Denote by t the time instance index, where $t = 1, \ldots, T$. Let L be the number of baseline MTS items, l the MTS item index, and denote by $\mathbf{a}^{(l)} = (a_n^{(l)}(t)); [n = 1, \ldots, N; t = 1, \ldots, T]$, $l = 1, \ldots, L$, the baseline MTS items, where $a_n^{(l)}(t)$ is the value of the n-th variable (sensor) sampled at the t-th time instance in the l-th MTS item. Let $\mathbf{b} = (b_n(t)); [n = 1, \ldots, N; t = 1, \ldots, T]$ be a testing MTS item. Let M be a parameter denoting the query length in nearest neighbor searching of subsequences. The ζ-th subsequence of $\mathbf{a}^{(l)}$ with length M is $\boldsymbol{\theta}_{\mathbf{a}^{(l)},\zeta} = (a_n^{(l)}(t)); [n = 1, \ldots, N; t = \zeta, \ldots, \zeta + M - 1]$, where $\zeta = 1, \ldots, T - M + 1$. The ξ-th subsequence of $\mathbf{b}$ with length M is $\boldsymbol{\theta}_{\mathbf{b},\xi} = (b_n(t)); [n = 1, \ldots, N; t = \xi, \ldots, \xi + M - 1]$, where $\xi = 1, \ldots, T - M + 1$. Noting that different variables (sensors) could contribute differently in fault detection and classification, we introduce a vector $\mathbf{s} = (s_n); [n = 1, \ldots, N]$ representing Variable Importance Scores (VISs), where $s_n \geq 0$ is the importance score of the n-th variable (sensor).

Remark 1 Without loss of generality, assume $\mathbf{b} \notin \{\mathbf{a}^{(l)}; l = 1, \ldots, L\}$; note that in the case where $\mathbf{b} \in \{\mathbf{a}^{(l)}; l = 1, \ldots, L\}$ we could take $\{\mathbf{a}^{(l)}; l = 1, \ldots, L\} \setminus \{\mathbf{b}\}$ as the baseline MTS set.

Remark 2 In this work, for a given physical system, we only consider a global VIS vector $\mathbf{s}$, and assume it is independent of l and t. To determine $\mathbf{s}$, we use a scheme based on Principal Components Analysis (PCA) originally proposed in [13]; see the Appendix for details.

Before adapting UCR's original definition of the Matrix Profile (MP) to our systems settings, we need the following definition of a generalized version of Euclidean distance:

Definition 1 (Weighted Euclidean Distance)
The Weighted Euclidean Distance (WED) $d(\cdot, \cdot)$ between $\boldsymbol{\theta}_{\mathbf{a}^{(l)},\zeta}$ and $\boldsymbol{\theta}_{\mathbf{b},\xi}$ is defined as

$$d\big(\boldsymbol{\theta}_{\mathbf{b},\xi}, \boldsymbol{\theta}_{\mathbf{a}^{(l)},\zeta}\big) = \sum_{n=1}^{N} s_n \times \sqrt{\sum_{\tau=0}^{M-1} \big(b_n(\xi+\tau) - a_n^{(l)}(\zeta+\tau)\big)^2}. \quad (1)$$

Remark 3 Def. 1 generalizes the Euclidean distance used by [1] in terms of dealing with multivariate data and taking the variable importance scores into account. However, we do not z-normalize (i.e., subtract the mean and divide by the standard deviation) the subsequences $\boldsymbol{\theta}_{\mathbf{a}^{(l)},\zeta}$ and $\boldsymbol{\theta}_{\mathbf{b},\xi}$ beforehand, because for fault detection/classification, such normalization would possibly lead to errors; for example, if $(0.1, 0.2)$ is a subsequence being normal, then $(1, 2)$ would be determined also as normal if using the z-normalized Euclidean distance (0), but the latter is actually deviating significantly from the former, thus being a fault.

We are now in a position to adapt the Matrix Profile (MP) to our case and introduce the Localized Matrix Profile (LMP).

Definition 2 (Matrix Profile)
The Matrix Profile (MP) of MTS item $\mathbf{b}$ joining baseline MTS set $\{\mathbf{a}^{(l)}; l = 1, \ldots, L\}$ is a vector $\tilde{\mathbf{g}} = (\tilde{g}_1, \ldots, \tilde{g}_{T-M+1})$ determined by

$$\tilde{g}_\xi = \min_{\substack{\zeta \in \{1,\ldots,T-M+1\} \\ l \in \{1,\ldots,L\}}} d\left(\boldsymbol{\theta}_{\mathbf{b},\xi}, \boldsymbol{\theta}_{\mathbf{a}^{(l)},\zeta}\right), \quad \xi = 1, \ldots, T - M + 1. \quad (2)$$

Definition 3 (Localized Matrix Profile)
The Localized Matrix Profile (LMP) of MTS item $\mathbf{b}$ joining MTS set $\{\mathbf{a}^{(l)}; l = 1, \ldots, L\}$ is a vector $\mathbf{g} = (g_1, \ldots, g_{T-M+1})$ determined by

$$g_\xi = \min_{l \in \{1,\ldots,L\}} d\left(\boldsymbol{\theta}_{\mathbf{b},\xi}, \boldsymbol{\theta}_{\mathbf{a}^{(l)},\xi}\right), \quad \xi = 1, \ldots, T - M + 1. \quad (3)$$

In other words, the ξ-th entry of the MP of $\mathbf{b}$ joining $\{\mathbf{a}^{(l)}; l = 1, \ldots, L\}$ is the Weighted Euclidean Distance (WED) between the ξ-th subsequence of $\mathbf{b}$ and its nearest neighbor over all subsequences of $\{\mathbf{a}^{(l)}; l = 1, \ldots, L\}$, no matter where they start. On the other hand, the ξ-th entry of the LMP of $\mathbf{b}$ joining $\{\mathbf{a}^{(l)}; l = 1, \ldots, L\}$ is the WED between the ξ-th subsequence of $\mathbf{b}$ and its nearest neighbor over all subsequences of $\{\mathbf{a}^{(l)}; l = 1, \ldots, L\}$ with the same starting time (i.e., the ξ-th time instance).

Remark 4 It is seen that, in order to calculate one single entry of the MP (resp., LMP) vector $\tilde{\mathbf{g}}$ (resp., $\mathbf{g}$), one needs to conduct $O(LT)$ (resp., $O(L)$) queries.

III. Algorithms

Since our focus in this paper is on conducting PHM tasks such as fault detection and classification based on the LMP, we will not discuss algorithms for MP computation in detail. Instead, when devising algorithms for LMP computation, we will point out connections to those for MP computation, wherever necessary. In addition, after presenting algorithms for LMP computation, we will propose a fault detector and a fault classifier based on the LMP.

A. Algorithms for Computing LMP

By the definition of LMP and direct searching, it is easy to come up with a brute-force algorithm (see Alg. 1) to obtain the LMP vector $\mathbf{g}$, whose time complexity is $O(MNLT)$ and space complexity is $O(NLT)$. Note that we use "FLT_MAX" to denote a large enough number and, in Line 4 of Alg. 1, the distance $d\left(\boldsymbol{\theta}_{\mathbf{b},\xi}, \boldsymbol{\theta}_{\mathbf{a}^{(l)},\xi}\right)$ is calculated directly by definition; refer to (1).

Algorithm 1 Brute Force Time Series Localized Matrix Profile (BFTLMP)

Input: MTS item $\mathbf{b}$ with length T; L baseline MTS items $\mathbf{a}^{(l)}$, $l = 1, \ldots, L$, each with length T; VIS vector $\mathbf{s}$; query length M

1: **for** $\xi = 1, \ldots, T - M + 1$ **do**
2: $\quad g_\xi \leftarrow$ FLT_MAX
3: $\quad$ **for** $l = 1, \ldots, L$ **do**
4: $\quad\quad g_\xi \leftarrow \min\left\{g_\xi, d\left(\boldsymbol{\theta}_{\mathbf{b},\xi}, \boldsymbol{\theta}_{\mathbf{a}^{(l)},\xi}\right)\right\}$
5: $\quad$ **end for**
6: **end for**

Output: $\mathbf{g} = (g_\xi; \xi = 1, \ldots, T - M + 1)$

On the other hand, given l and ξ, by (1) we can write $d\left(\boldsymbol{\theta}_{\mathbf{b},\xi}, \boldsymbol{\theta}_{\mathbf{a}^{(l)},\xi}\right) = \sum_{n=1}^{N} s_n \times \sqrt{\Delta_\xi^{(n,l)}}$, where $\Delta_\xi^{(n,l)} \stackrel{\text{def}}{=} \sum_{\tau=0}^{M-1} \left(b_n(\xi + \tau) - a_n^{(l)}(\xi + \tau)\right)^2$ satisfies the following recursion:

$$\Delta_{\xi+1}^{(n,l)} = \Delta_\xi^{(n,l)} + \left(b_n(\xi + M) - a_n^{(l)}(\xi + M)\right)^2 - \left(b_n(\xi) - a_n^{(l)}(\xi)\right)^2. \quad (4)$$

From (4), it is seen that once we have computed $\Delta_\xi^{(n,l)}$, then $\Delta_{\xi+1}^{(n,l)}$ can be obtained in $O(1)$ time. The idea makes use of the overlap between consecutive subsequences, and leads to Alg. 2, which has an $O(NLT)$ time complexity. It is worth pointing out that (i) the recursion (4) is similar to [3, Eq. (4)], which paves the way to speed up the computation by a factor M (the query length; this reduction of time complexity is significant especially when M is large), and (ii) the name of Alg. 2 (STOLMP) is partially borrowed from [3], where an algorithm named STOMP is proposed for MP computation.

To better facilitate parallelism, in Alg. 2 we do not directly use the recursion (4), but compute all the prefix sums as an initial scan (see Line 5 of Alg. 2, where we define $h_n^{(l)}(0) = 0, \forall n \in \{1, \ldots, N\}, l \in \{1, \ldots, L\}$), and then extract the partial sums involved in the distance computation by an easy subtraction (see Line 12 of Alg. 2). Note that $h_n^{(l)}(t); [n = 1, \ldots, N; t = 1, \ldots, T]$, $l = 1, \ldots, L$, are temporary variables saving intermediate results. It is worth pointing out that, without much effort, Alg. 2 could be adapted to a GPU (i.e., Graphics Processing Unit) version which would further reduce the execution time by use of the parallel computation mechanism; in an independent paper we will focus on presenting results from extensive numerical experiments regarding efficiency of computing the LMP using GPUs.

Algorithm 2 Scalable Time Series Ordered Localized Matrix Profile (STOLMP)

Input: MTS item $\mathbf{b}$ with length T; L baseline MTS items $\mathbf{a}^{(l)}$, $l = 1, \ldots, L$, each with length T; VIS vector $\mathbf{s}$; query length M

1: **for** $l = 1, \ldots, L$ **do**
2: $\quad$ **for** $n = 1, \ldots, N$ **do**
3: $\quad\quad$ **for** $t = 1, \ldots, T$ **do**
4: $\quad\quad\quad h_n^{(l)}(t) \leftarrow \left(b_n(t) - a_n^{(l)}(t)\right)^2$
5: $\quad\quad\quad h_n^{(l)}(t) \leftarrow h_n^{(l)}(t) + h_n^{(l)}(t-1)$ $\triangleright$ $h_n^{(l)}(0) \stackrel{\text{def}}{=} 0$
6: $\quad\quad$ **end for**
7: $\quad$ **end for**
8: **end for**
9: **for** $\xi = 1, \ldots, T - M + 1$ **do**
10: $\quad g_\xi \leftarrow$ FLT_MAX
11: $\quad$ **for** $l = 1, \ldots, L$ **do**
12: $\quad\quad d \leftarrow \sum_{n=1}^{N} s_n \times \sqrt{h_n^{(l)}(\xi + M - 1) - h_n^{(l)}(\xi - 1)}$
13: $\quad\quad g_\xi \leftarrow \min(g_\xi, d)$
14: $\quad$ **end for**
15: **end for**

Output: $\mathbf{g} = (g_\xi; \xi = 1, \ldots, T - M + 1)$

In the next subsection, we describe a fault detector using the LMP. The basic idea is similar to the MP-based discord discovery in [1]. However, it is worth emphasizing that our LMP-based approach would naturally avoid the "twin freak" issue mentioned earlier.

B. LMP-based Fault Detector

We devise a fault detector based on the LMP, whose structure is shown in Fig. 1, where we propose a semi-supervised approach, i.e., only normal MTS items are used for training. For ease of presentation, we use $\mathbf{a}^{(l)} = (a_n^{(l)}(t)); [n = 1, \ldots, N; t = 1, \ldots, T]$, $l = 1, \ldots, L$, to denote the normal training MTS items, and let $\mathbf{b} = (b_n(t)); [n = 1, \ldots, N; t = 1, \ldots, T]$ denote a MTS item for testing. Henceforth, we use $\langle \mathbf{c}, \{\mathbf{a}^{(l)}; l = 1, \ldots, L\}\rangle_{\text{LMP}}$ to denote the Localized Matrix Profile of any given MTS item $\mathbf{c}$ joining MTS set $\{\mathbf{a}^{(l)}; l = 1, \ldots, L\}$.

We list the stages involved in the detector as follows:

1) Load the normal training MTS items $\mathbf{a}^{(l)} = (a_n^{(l)}(t)); [n = 1, \ldots, N; t = 1, \ldots, T]$, $l = 1, \ldots, L$, into memory.

2) Using Alg. 2, we compute the following LMP vectors:
   - $\langle \mathbf{a}^{(k)}, \{\mathbf{a}^{(l)}; l = 1, \ldots, L\} \setminus \{\mathbf{a}^{(k)}\} \rangle_{\text{LMP}}$, $k = 1, \ldots, L$. Note that these are LMP of the normal training MTS items themselves, which would imply time-dependent ranges of tolerable variations among the normal runs of the system or process and, in turn, provide a guidance for setting the LMP thresholds. Note also that these computations could be done offline.
   - $\langle \mathbf{b}, \{\mathbf{a}^{(l)}; l = 1, \ldots, L\} \rangle_{\text{LMP}}$.
3) For any given $\xi \in \{1, \ldots, T - M + 1\}$, take the maximum of the ξ-th entry in the baseline LMP vectors $\langle \mathbf{a}^{(k)}, \{\mathbf{a}^{(l)}; l = 1, \ldots, L\} \setminus \{\mathbf{a}^{(k)}\} \rangle_{\text{LMP}}$, $k = 1, \ldots, L$, as a rough threshold g_ξ^*.
4) For any given $\xi \in \{1, \ldots, T-M+1\}$, if and only if the ξ-th entry of $\langle \mathbf{b}, \{\mathbf{a}^{(l)}; l = 1, \ldots, L\} \rangle_{\text{LMP}}$ is greater than $\lambda \times g_\xi^*$, we report a fault at time instance ξ for testing MTS item $\mathbf{b}$. Here, $\lambda \geq 0$ is a parameter (default value is 1) which could be tuned, for example, by means of cross-validation.

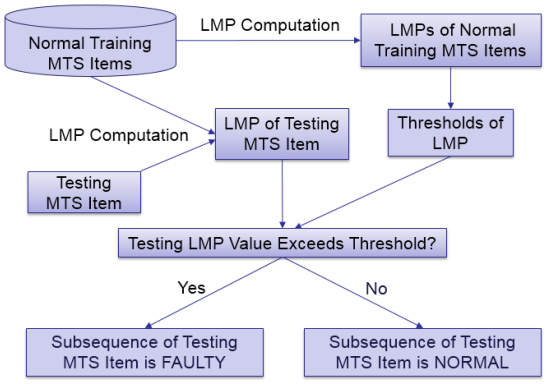

Fig. 1: The structure of the LMP-based fault detector.

In the next subsection, we propose a fault classifier based on the LMP.

C. LMP-based Fault Classifier

From the discussion above, we have seen that the LMPs of original MTS items contain rich variational information which could be utilized for fault detection (discord discovery). In this subsection, we further investigate the possibility of using LMPs to classify faults. As a matter of fact, given a physical system, different types of faulty operation or different degrees of deterioration of certain parts of a device would lead to their own unique LMP values. Based on this intuition, we propose a fault classifier as depicted in Fig. 2.

For convenience, let us denote by $\{\tilde{\mathbf{a}}^{(l)}; l = 1, \ldots, L_0\}$ the normal baseline MTS items, and by $\{\mathbf{b}^{(l)}; l = 1, \ldots, L_1\}$ (resp., $\{\hat{\mathbf{b}}^{(l)}; l = 1, \ldots, L_2\}$) the MTS items for training (resp., testing). We list the stages involved in the classifier as follows:

1) Load the normal baseline MTS items $\{\tilde{\mathbf{a}}^{(l)}; l = 1, \ldots, L_0\}$ into memory.

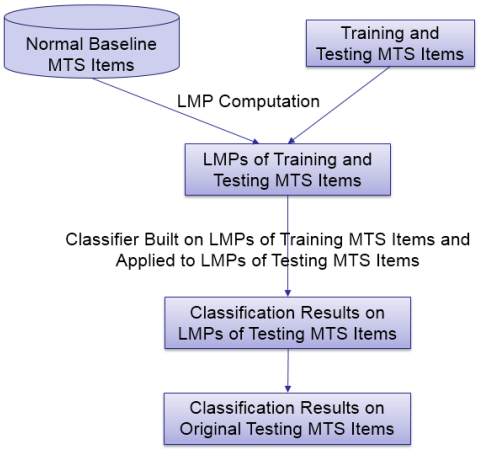

Fig. 2: The structure of the LMP-based fault classifier.

2) Using Alg. 2, we compute the LMP vectors of training MTS items, i.e., $\langle \mathbf{b}^{(k)}, \{\tilde{\mathbf{a}}^{(l)}; l = 1, \ldots, L_0\} \rangle_{\text{LMP}}$, $k = 1, \ldots, L_1$. Note that each and every LMP vector has a label indicating the ground-truth fault type. Note also that the computations for these LMP vectors could be done offline.
3) Build a classifier upon the LMP vectors $\langle \mathbf{b}^{(k)}, \{\tilde{\mathbf{a}}^{(l)}; l = 1, \ldots, L_0\} \rangle_{\text{LMP}}$, $k = 1, \ldots, L_1$. Note that such a classifier could be any general multi-label classifier; e.g., the SVM-based classifier [14], the shapelet based classifier [15], etc.
4) Using Alg. 2, we compute the LMP vectors of testing MTS items, i.e., $\langle \hat{\mathbf{b}}^{(k)}, \{\tilde{\mathbf{a}}^{(l)}; l = 1, \ldots, L_0\} \rangle_{\text{LMP}}$, $k = 1, \ldots, L_2$.
5) Apply the classifier to predict the labels of the LMP vectors of testing MTS items $\langle \hat{\mathbf{b}}^{(k)}, \{\tilde{\mathbf{a}}^{(l)}; l = 1, \ldots, L_0\} \rangle_{\text{LMP}}$, $k = 1, \ldots, L_2$, thus obtaining predicted fault types of the original testing MTS items.

IV. A Case Study

In this section, we present numerical results of a case study using synthetic sensor data generated from a physical model (built with Modelica [16]) that simulates an electrical motor. For LMP computation, we conduct additional experiments to numerically evaluate the performance of STOLMP (Alg. 2) compared to the brute force counterpart (Alg. 1). We run the code on a desktop machine with an Intel(R) Core(TM) i7-4770 CPU and 16 GB of system memory. Unless otherwise specified, we implement all the algorithms in Python.

A. Data Format

We generate simulation data corresponding to the aforementioned electrical motor's operation with different levels of deterioration for a certain component of the motor. There are 6 levels of deterioration in total, and we generate 600 MTS items (indexed by $1, \ldots, 600$) for each level, where each MTS item contains 4900 time instances (the sampling frequency is 1000 Hz) and 5 physical variables denoted by $v_n, n = 1, \ldots, 5$ respectively. Note that the lower the deterioration level, the

healthier the motor's status is. Note that the level 0 data are "normal," and the data whose level exceeds 1 are "faulty;" the data with level 1 are in an "in-between" status. To remove effects of the units of physical variables, we preprocess each and every MTS item by z-normalizing the time series of each and every variable. Henceforth, whenever we mention MTS items in the case study, we mean the preprocessed ones.

B. VIS Computation

To compute the VIS vector $\mathbf{s}$, we use the MTS items indexed $1, \ldots, 200$ for each and every level; thus, we use $200 \times 6 = 1200$ MTS items in total. After obtaining the variable importance scores by applying the algorithms mentioned in the Appendix, we normalize the VIS vector (i.e., $\mathbf{s} \leftarrow \mathbf{s}/\|\mathbf{s}\|_1$) and end up with $\mathbf{s} = (0.2323, 0.2306, 0.2316, 0.0879, 0.2176)$.

C. Fault Detection

To conduct fault detection, we only use data with levels among $\{0, 2, 3, 4, 5\}$; recall that the level 1 data are in an "in-between" status. We take the query length $M = 300$ in LMP computations; this number is determined by a few trials. To obtain the baseline LMPs, we use the MTS items indexed $201, \ldots, 400$ with level 0. The MTS items indexed $401, \ldots, 600$ are used for testing. For economy of space, we only show results for data with level 2; the results for data with levels $3, 4, 5$ are similar and omitted.

In the top subfigure of Fig. 3, the values of a selected variable (i.e., v_1 – the variable with the highest importance score) in the MTS items used for computation of baseline LMPs are plotted as dashed lines; for economy of space, we omit the plots for variables v_2 through v_5. In the bottom subfigure of Fig. 3, we show the values of the baseline LMPs as dashed lines. In each and every subfigure of Fig. 3, we superimpose the corresponding values of the variable or LMP of the testing MTS item indexed 401 with ground-truth level 2 (faulty) as a red solid line. In Fig. 4, we show the LMP values of the same testing item and the thresholds determined by the computed baseline LMPs; we take the factor $\lambda = 1$. It is seen from Fig. 3 that the values of the selected variable in the original baseline MTS items are very close, and they are also close to those of the testing MTS item, while, at certain time instances, the values of the LMP of the testing item are deviating significantly from the baseline LMPs; these deviations indicate faulty windows (i.e., subsequences) (see Fig. 4 for the red curve between time instances 1071 and 1698). This demonstrates the effectiveness of the LMP-based anomaly detector on one single testing item.

To further see the performance of the proposed fault detector, we compare it with a widely used alternative, which is based on One-Class Support Vector Machines (OCSVM) [17], and plot their Receiver Operating Characteristic (ROC) [18] curves as shown in Fig. 5. The True Positive Rate (TPR) and False Positive Rate (FPR) are obtained as follows. The testing MTS items include 200 level 0 (Negative; i.e., "normal") ones and 200 level 2 (Positive; i.e., "faulty") ones. Each MTS item contains $T - M + 1 = 4900 - 300 + 1 = 4601$ windows. For any of the 200 level 0 testing items, if there exists one or more windows (among the 4601 windows) which are detected as faulty, we report a False Positive (FP) for that item. For any of the 200 level 2 testing items, if there exists one or more windows which are detected as faulty, we report a True Positive (TP) for that item. After obtaining the total FPs and TPs, we divide them by 200, thus obtaining FPR and TPR, respectively. For the LMP-based fault detector, we let the threshold factor λ vary among $\{0, 0.2, 0.4, \ldots, 19.8\}$, and attain 100 pairs of (TPR, FPR), which enable us to draw the entire ROC curve as the blue line in Fig. 5. For the OCSVM-based one, we derived (TPR, FPR) pairs by extracting values of the decision functions and varying the binary classification thresholds accordingly. Note that for each and every detection window, we build an OCSVM classifier (with parameter $\nu = 0.01$ and Radial Basis Function (RBF) kernel; refer to [19]) upon the corresponding subsequences of the training MTS items, thus obtaining a decision function. The resulting ROC curve of the OCSVM-based fault detector is shown as the red line in Fig. 5. In our experiments, the Area Under the ROC Curve (AUC) of the LMP-based fault detector is 1.0, while the AUC of the OCSVM-based alternative is 0.975. Thus, in this case study, the LMP-based fault detector is more accurate than its OCSVM-based counterpart.

When comparing the LMP-based and the OCSVM-based fault detectors in our case study, we also record the execution times of the core modules respectively. For the LMP-based one, we record the execution time t_{lmp} of computing the LMPs of the testing MTS items. For the OCSVM-based one, we record the execution time t_{ocsvm} of fitting the OCSVM classifiers (the implementations are based on LIBSVM [20], a highly optimized library) upon windows of training MTS items and calculating the values of decision functions for windows of testing MTS items. It turns out that $t_{\text{lmp}} = 10.277$ seconds, and $t_{\text{ocsvm}} = 81.676$ seconds. Thus, in this case study, the LMP-based fault detector runs about 8 times faster than its OCSVM-based counterpart. It can be foreseen that the larger the query length M, the more significant the advantage of the running time of the former over the latter would be. This is to be expected, since the former makes use of the overlaps between consecutive detection windows while the latter does not.

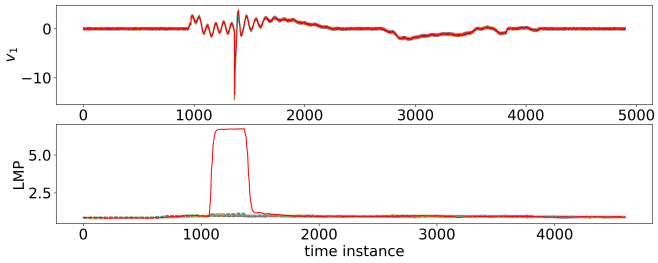

Fig. 3: The values of a selected variable (v_1) in MTS items and LMPs; the red solid curves correspond to a faulty testing item.

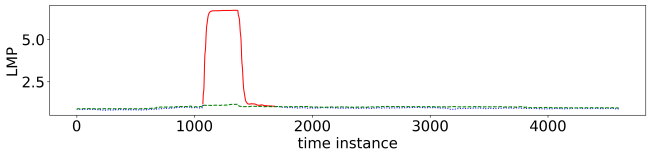

Fig. 4: The LMP values of a faulty testing item and the thresholds determined by baseline LMPs; the green dashed line indicates thresholds, the blue dotted line indicates normal windows in the testing item, and the red solid line indicates faulty windows in the testing item.

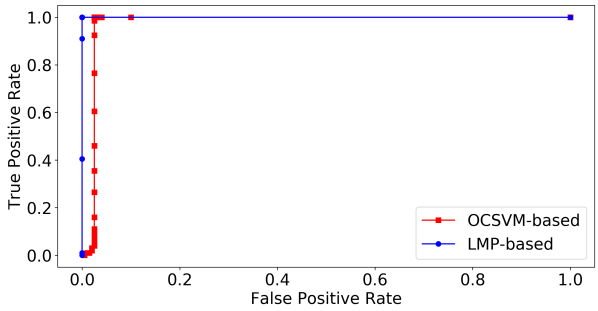

Fig. 5: The ROC curves of the LMP-based and the OCSVM-based anomaly detectors.

D. Fault Classification

To conduct fault classification, we use the MTS items indexed $201, \ldots, 400$ with level 0 as normal baseline MTS items (see Fig. 2). The MTS items indexed $401, \ldots, 500$ (resp., $501, \ldots, 600$) with all 6 levels are used for training (resp., testing). The query length is also taken as $M = 300$. We compute LMPs of all the training and testing MTS items with respect to the 200 baseline MTS items. Thus, we end up with 600 training LMPs (100 for each level) and 600 testing LMPs (100 for each level). Then, we build a shapelet-based multi-label classifier [15] (we use the C++ code downloaded from [21]) upon the 600 training LMPs and apply it to predict the level of each and every testing LMP. The results (wrapped up as a Confusion Matrix) are shown in Tab. I. It is seen that, for items with ground-truth levels greater than or equal to 2, the classification is highly accurate; only one level 5 item is misclassified as level 4. The testing items with ground-truth level 0 are also classified with a high accuracy (96%); only 4 items are misclassified as level 1. On the other hand, the data with ground-truth level 1 are easy to be misclassified as level 0; this is due to the fact that, in an "in-between" status, the original data of level 1 are actually very close to those of level 0.

E. Notes on LMP Computation

We note that, in our fault detector and classifier, the LMP computation submodule is the most costly; we implement STOLMP (Alg. 2) in C++ and make it callable by Python. In our case study, possibly due to the not too large data size (the total size of the raw MTS data is 6.3 GB), we do not encounter challenging computational issues in both CPU speeds and memory occupation. Excluding offline computations (i.e., data preprocessing, computations for LMPs of baseline items and training items, and results saving), running all the fault detection/classification experiments involved in the case study takes about 45 minutes.

TABLE I: The Confusion Matrix of the LMP-based classifier.

prediction \ ground-truth	0	1	2	3	4	5
0	96	96	0	0	0	0
1	4	4	0	0	0	0
2	0	0	100	0	0	0
3	0	0	0	100	0	0
4	0	0	0	0	100	1
5	0	0	0	0	0	99

To explicitly evaluate the speed of STOLMP, we compute the LMP of a randomly generated Univariate Time Series (UTS) joining 3000 randomly generated UTS, and measure how many comparisons (each comparison ends up with a distance value; refer to (1)) can be completed per millisecond. Each UTS has 20000 time instances. The VIS vector $\mathbf{s}$ is set to a scalar of 1. Our implementation is in C++. Table II lists the average numbers of comparisons for STOLMP and the brute force alternative BFTLMP (Alg. 1) corresponding to different query lengths. It is seen that STOLMP is 20–60 times faster than the brute force algorithm, and the speedup is more apparent when the query length increases. This is to be expected, since the time complexity of STOLMP is independent of the query length. It is worth pointing out that, due to subtractions involved (see Line 12 of Alg. 2), it is safer to implement STOLMP in double precision so as to retain numerical accuracy.

V. Conclusions

In this paper, we introduce the Localized Matrix Profile (LMP), a new primitive for time series data mining. We devise fast algorithms for LMP computation, and propose a fault detector and a fault classifier based on the LMP. To show the effectiveness and efficiency of our approach, we provide a case study using synthetic sensor data generated from a physical model of an electrical motor.

As a reminder for practitioners, we point out that our method is more applicable for cases where memory occupation is not an issue. Though LMPs of baseline data and training data could be computed offline, we still need to store the old (baseline) data in memory for the computations of LMPs of new (testing) data.

Appendix: Calculation of the VIS Vector

We refer to [13] about how to calculate the VIS vector $\mathbf{s}$. The idea is based on Common Principal Component Analysis (CPCA). In particular, we use an unsupervised method (i.e., for the purpose of calculating the VIS vector, no labels for the MTS items are needed) which utilizes the properties of the Principal Components (PCs) and the Descriptive Common

TABLE II: Average numbers of comparisons per millisecond for BFTLMP and STOLMP with different query lengths.

number of comparisons per millisecond / algorithm \ query length	250	500	1000
BFTLMP (single precision)	2127	1197	626
STOLMP (double precision)	43472	39211	37875

Principal Components (DCPCs) to preserve the correlation information among variables. The VISs of variables are calculated according to their contribution to the common principal components. The module for calculating the VIS vector is structured as shown in Fig. 6; for details, see [13, Algorithms 1 and 2]. A minor modification is that when applying [13, Alg. 2] (using [13, Alg. 1] as a subroutine), we do not need the input "the number of variables to select" and only [13, Lines 1 through 5, Alg. 2] are executed.

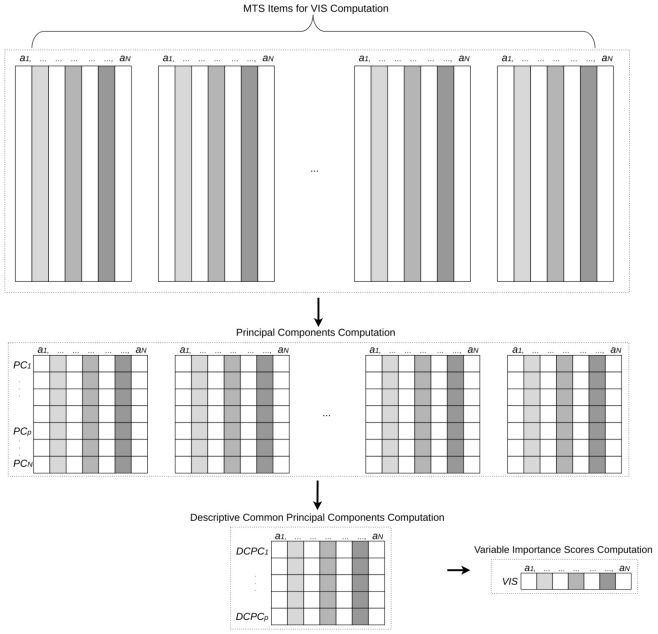

Fig. 6: The structure of the module for VIS calculation modified from [13].

ACKNOWLEDGMENT

The authors would like to thank Ioannis Ch. Paschalidis for helpful discussions.

REFERENCES

[1] A. Mueen, Y. Zhu, M. Yeh, K. Kamgar, K. Viswanathan, C. Gupta, and E. Keogh, "The UCR Matrix Profile Page," 2017, http://www.cs.ucr.edu/~eamonn/MatrixProfile.html.

[2] C.-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 *Data Mining (ICDM), 2016 IEEE 16th International Conference on*. IEEE, 2016, pp. 1317–1322.

[3] Y. Zhu, Z. Zimmerman, N. S. Senobari, C.-C. M. Yeh, G. Funning, A. Mueen, P. Brisk, and E. Keogh, "Matrix Profile II: Exploiting a Novel Algorithm and GPUs to break the one Hundred Million Barrier for Time Series Motifs and Joins," in *Data Mining (ICDM), 2016 IEEE 16th International Conference on*. IEEE, 2016, pp. 739–748.

[4] C.-C. M. Yeh, H. Van Herle, and E. Keogh, "Matrix Profile III: The Matrix Profile Allows Visualization of Salient Subsequences in Massive Time Series," in *Data Mining (ICDM), 2016 IEEE 16th International Conference on*. IEEE, 2016, pp. 579–588.

[5] S. Gharghabi, Y. Ding, C.-C. M. Yeh, K. Kamgar, L. Ulanova, and E. Keogh, "Matrix Profile VIII: Domain Agnostic Online Semantic Segmentation at Superhuman Performance Levels," in *Data Mining (ICDM), 2017 IEEE International Conference on*. IEEE, 2017, pp. 117–126.

[6] S. Nandi, H. A. Toliyat, and X. Li, "Condition monitoring and fault diagnosis of electrical motors – a review," *IEEE transactions on energy conversion*, vol. 20, no. 4, pp. 719–729, 2005.

[7] T. Ahmed, B. Oreshkin, and M. Coates, "Machine learning approaches to network anomaly detection," in *Proceedings of the 2nd USENIX workshop on Tackling computer systems problems with machine learning techniques*. USENIX Association, 2007, pp. 1–6.

[8] J. Zhang, "Detection and optimization problems with applications in smart cities," Ph.D. dissertation, Boston University, 2017.

[9] M. Egele, G. Stringhini, C. Kruegel, and G. Vigna, "Towards detecting compromised accounts on social networks," *IEEE Transactions on Dependable and Secure Computing*, vol. 14, no. 4, pp. 447–460, 2017.

[10] A. Minnich, N. Chavoshi, D. Koutra, and A. Mueen, "Botwalk: Efficient adaptive exploration of twitter bot networks," in *Proceedings of the 2017 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2017*. ACM, 2017, pp. 467–474.

[11] J. Zhang and I. C. Paschalidis, "Statistical anomaly detection via composite hypothesis testing for Markov models," *IEEE Transactions on Signal Processing*, vol. 66, no. 3, pp. 589–602, 2017.

[12] J. Zhang, S. Pourazarm, C. G. Cassandras, and I. C. Paschalidis, "The price of anarchy in transportation networks: Data-driven evaluation and reduction strategies," *Proceedings of the IEEE*, vol. 106, no. 4, pp. 538–553, 2018.

[13] H. Yoon, K. Yang, and C. Shahabi, "Feature subset selection and feature ranking for multivariate time series," *IEEE transactions on knowledge and data engineering*, vol. 17, no. 9, pp. 1186–1198, 2005.

[14] C.-W. Hsu and C.-J. Lin, "A comparison of methods for multiclass support vector machines," *IEEE transactions on Neural Networks*, vol. 13, no. 2, pp. 415–425, 2002.

[15] T. Rakthanmanon and E. Keogh, "Fast shapelets: A scalable algorithm for discovering time series shapelets," in *proceedings of the 2013 SIAM International Conference on Data Mining*. SIAM, 2013, pp. 668–676.

[16] M. Tiller, *Introduction to physical modeling with Modelica*. Springer Science & Business Media, 2012, vol. 615.

[17] S. Mahadevan and S. L. Shah, "Fault detection and diagnosis in process data using one-class support vector machines," *Journal of process control*, vol. 19, no. 10, pp. 1627–1639, 2009.

[18] Wikipedia, "Receiver operating characteristic," 2018, https://en.wikipedia.org/wiki/Receiver_operating_characteristic#Area_under_the_curve.

[19] "sklearn.svm.OneClassSVM," 2018, Python module documentation available at https://scikit-learn.org/stable/modules/generated/sklearn.svm.OneClassSVM.html.

[20] C.-C. Chang and C.-J. Lin, "LIBSVM: A library for support vector machines," *ACM Transactions on Intelligent Systems and Technology*, vol. 2, pp. 27:1–27:27, 2011, software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

[21] T. Rakthanmanon and E. Keogh, "Fast-Shapelets: A Scalable Algorithm for Discovering Time Series Shapelets ," 2013, http://alumni.cs.ucr.edu/~rakthant/FastShapelet/.