Data Min Knowl Disc (2018) 32:1806–1844
https://doi.org/10.1007/s10618-018-0569-7

Exact variable-length anomaly detection algorithm for univariate and multivariate time series

Xing Wang[1] · **Jessica Lin**[1] · **Nital Patel**[2] · **Martin Braun**[2]

Received: 16 October 2017 / Accepted: 22 April 2018 / Published online: 31 July 2018

Abstract The problem of anomaly detection in time series has received a lot of attention in the past two decades. However, existing techniques cannot locate where the anomalies are within anomalous time series, or they require users to provide the length of potential anomalies. To address these limitations, we propose a self-learning online anomaly detection algorithm that automatically identifies anomalous time series, as well as the exact locations where the anomalies occur in the detected time series. In addition, for multivariate time series, it is difficult to detect anomalies due to the following challenges. First, anomalies may occur in only a subset of dimensions (variables). Second, the locations and lengths of anomalous subsequences may be different in different dimensions. Third, some anomalies may look normal in each individual dimension but different with combinations of dimensions. To mitigate these problems, we introduce a multivariate anomaly detection algorithm which detects anomalies and identifies the dimensions and locations of the anomalous subsequences. We evaluate our approaches on several real-world datasets, including two CPU manufacturing

Responsible editor: Eamonn Keogh.

✉ Jessica Lin
jessica@gmu.edu

Xing Wang
xwang24@gmu.edu

Nital Patel
nital.s.patel@intel.com

Martin Braun
martin.w.braunl@intel.com

[1] George Mason University, Fairfax, USA

[2] Intel Corporation, Chandler, AZ, USA

data from Intel. We demonstrate that our approach can successfully detect the correct anomalies without requiring any prior knowledge about the data.

Keywords Anomaly detection · Multivariate time series · Unsupervised learning

1 Introduction

A significant volume of manufacturing data is time series in nature, e.g. sequence of events in an equipment log (1GB/lot), trace data generated by equipment sensors, factory events; test results logged by a tester, etc. This volume of data is overwhelming to the end user. In some cases, the engineers struggled for a year to understand and characterize relationships in the data. Nevertheless, fault detection, or excursion prevention (EP), methodologies—typically developed to characterize and monitor such trace data—have remained static for over two decades. New approaches for large scale data mining and knowledge discovery are not being leveraged. As an example, current fault or excursion detection techniques rely on the analysts to manually segment the time series into several characteristic windows, and in each window, the analysts then set the thresholds to detect deviations. Our work aims to automate the entire process and introduces a new self-learning and online anomaly detection algorithm for time series data to mitigate the aforementioned problems.

Current methods that detect anomalies in time series can be categorized into detection of point anomalies, structural anomalies and anomalous time series. Point anomalies, commonly known as outliers, are data points that are significantly different from others (Hawkins 1980; Senin et al. 2015). Structural anomalies are subsequences whose shape do not conform to the rest of the observed, or expected patterns (Keogh et al. 2005; Zhang et al. 2010; Senin et al. 2015). The detection of anomalous time series, which we refer to as whole time series anomaly, aims to detect time series whose average deviation from other time series is significant (Hyndman et al. 2015; Laptev et al. 2015).

For fault detection in manufacturing data, it is critical to detect not only the anomalous time series, but also the exact locations of the structural anomalies. Existing methods for time series anomaly detection focus on either detecting local (subsequence) anomalies in a long time series, or detecting anomalous time series in a group of time series data.

For example, window-based anomaly detection techniques (Keogh et al. 2005, 2007; Senin et al. 2015) can find local structural anomalies; however, the user must specify the number and length of the expected anomalies. In real-world applications, there may be new anomalies with unpredictable behaviours and unknown length. For window-based techniques to detect all anomalies, they have to try every possible length as input. However, iterating through all possible lengths is inefficient, and requires specific criteria to determine the true anomalies. Such requirement on window length thus limits the application of current methods to only those where users have prior knowledge of the data or the anomalies.

A well-known window-based anomaly detection technique, time series discord discovery (Keogh et al. 2005), defines an anomaly as the subsequence in a long time

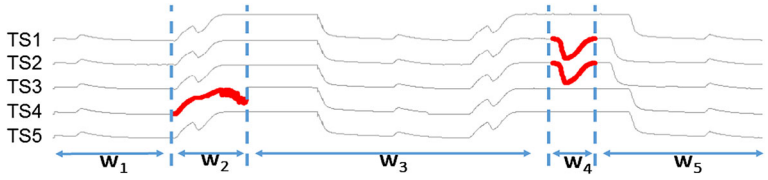

Fig. 1 Example of anomalous subsequences (shown in red and bold) (Color figure online)

series that has the largest distance to its (kth) nearest neighbor (the original algorithm sets $k = 1$). However, the effectiveness of the approach greatly relies on the user-defined window length and parameter k. The technique would fail if there exist clusters of anomalies containing more than k instances, or if there exist multiple clusters of anomalies of varying lengths or sizes. It can be generalized to finding anomalous *whole time series* in a time series dataset (Wei et al. 2006). However, the result is based on overall deviation, according to the distances computed using whole time series, hence it cannot pinpoint the exact locations of the local anomalies without further investigation.

As a concrete example, consider the time series data in Fig. 1, which contains five time series, two different groups of anomalies and three anomalous subsequences (shown as red/bold). These two groups of anomalies occur in window w_2 (length 153) and w_4 (length 89), respectively. Since the anomalies have different lengths, and there exist more than one anomalous instance of the same type, it is hard for current techniques to correctly identify all of them.

In Fig. 1, the time series are segmented into five windows w_{1-5}. Once they are segmented, we can apply a clustering-based anomaly detection algorithm to detect anomalous subsequences within each window. The challenge, then, is how to segment the time series properly into windows. We propose a symbolic clustering algorithm that discretizes time series data, and learns common patterns in the data via grammar induction to find the initial segmenting information, as well as initial clusters in each window. Note that we do not assume any information about the potential anomalies in advance. However, in our manufacturing application, data are expected to behave similarly in the same time window. While we allow some warping in the matching of patterns, in application such as ours, a typical sliding-window based approach that compares subsequences from different parts of time series would not be appropriate.

Furthermore, to alleviate the user's burden of having to set the parameters, we introduce a self-learning dynamic clustering algorithm which learns cluster boundaries and the number of clusters automatically from the input data.

Many multivariate time series anomaly detection algorithms have been introduced recently (Galeano et al. 2006; Baragona and Battaglia 2007; Li et al. 2017; Cheng et al. 2009; Id et al. 2007; Qiu et al. 2012). However, most of them are designed to detect anomalies in a long time series. PCA-based methods (Miljkovi 2011; Ahmed et al. 2012) are often used in the semiconductor sector for fault detection on manufacturing data. However, these methods have to specify the sliding window length in order to compute statistic information. In this work, we introduce algorithms to detect anomalies in both univariate and multivariate times series. The proposed methods can

detect anomalous time series subsequences, as well as the locations of anomalies and the corresponding sensors (dimensions). Specifically, there are three major challenges faced by multivariate time series anomaly detection on manufacturing data. The first challenge is to find the sub-dimensions (i.e. a subset of variables) in which the anomalies occur. If we use data from all dimensions, but the anomalies only exist in a few dimensions, the true anomalies may seem normal because of the irrelevant dimensions. The second challenge is to find the correlation anomalies. The anomalous time series may look normal in each dimension individually, but their combinations may be anomalous. The third challenge is to find the exact locations and dimensions of the problem areas.

To address these problems, we first discuss the univariate version of our anomaly detection algorithm, SLADE-TS (Wang et al. 2016). SLADE-TS automatically segments the time series into variable-length windows, and cluster all subsequences within each window. Subsequences in the same cluster within a window are similar to each other; that is, objects that generated these subsequences behave similarly at least during that time window. On the other hand, subsequences in different clusters have different behaviors. We define the behavior patterns of objects by the cluster centers detected by SLADE-TS. To handle multivariate time series, we repeat SLADE-TS for every dimension. A time series is abnormal if it does not follow the normal patterns discovered in the corresponding dimension and window. These patterns provide information on the dimensions and locations where potential problems occur.

To use the pattern information properly, Step 2 of our multivariate anomaly detection algorithm introduces a data transformation algorithm to transform the original time series into a new feature space in which each feature is the distance to a pattern. The smaller the distance, the more similar the data is to the pattern. Step 3 performs feature selection to select important features. In Step 4, we perform clustering on the transformed data, and assign anomaly score to each time series based on the clustering results and distances to normal cluster. Finally, by a locating strategy we propose, we accurately identify the dimensions and locations of the anomalous time series.

To summarize, our univariate anomaly detection algorithm focuses on the problems of discovering global anomalies as well as local anomalies within individual time series. It can detect variable-length anomalies without prior knowledge or assumptions on anomalies. Our multivariate anomaly detection algorithm can handle multivariate time series data even if there are correlation anomalies. It also detects the exact dimensions and locations of anomalies. Our work has the following significant contributions:

- We propose SLADE-TS (Self-Learning Anomaly DEtection for Time Series), an algorithm that automatically identifies anomalous time series as well as their exact variable-length anomalous subsequences.
- Our algorithm is unsupervised and adaptively learns from data incrementally. It is able to handle streaming time series and identify anomalies in real time.
- We propose SLADE-MTS (Self-Learning Anomaly DEtection for Multivariate Time Series), an algorithm to detect anomalies in multivariate time series data. It detects anomalies even if the anomalies only occur in a subset of the dimensions.
- The algorithm is able to detect correlation anomalies even if they seem "normal" in each individual dimension.

– With the locating strategy, we can pinpoint the exact location(s) and dimension(s) of the problems.
– By using a clustering-based anomaly detection algorithm and introducing an anomaly scoring strategy, we are able to detect anomalies even if there exist more than one type of normal data or anomaly data. We can also categorize different types of anomalies as well as different types of normal.

The rest of the paper is structured as follows. In Sect. 2, we discuss background and related work on time series anomaly detection. Section 3 describes our univariate anomaly detection method. We introduce our multivariate anomaly detection algorithm in Sect. 4. Experiments for both univariate and multivariate methods are presented in Sect. 5. We conclude in Sect. 6.

2 Background and related work

2.1 Definitions

To precisely state the problem at hand, and to relate our work to previous research, we will define the key terms used throughout this paper. We begin by defining our data type, time series:

Time series $TS = t_1, \ldots, t_n$ is a set of scalar observations ordered by time. In this paper, time series arrive in an order of $TS_1, TS_2, \ldots$

Since one of our goals is to find local anomalies, we consider time series subsequences:

Subsequence ss of time series TS is a contiguous sampling $t_i, \ldots, t_{i+m-1}$ of points of length $m < n$, where i is an arbitrary position, such that $1 \leq i \leq n - m + 1$.

Segmented Windows The time series are segmented into a set of non-overlapping windows. The segmentation is data-adaptive rather than based on pre-defined cut-points. Subsequences from the same Segmented Window have the same length. **Anomaly Windows** are Segmented Windows that contain anomalies.

Figure 1 shows examples of *Segmented Windows* and *Anomaly Windows*. Three anomalous subsequences are detected. Two *Anomaly Windows* (w_2, w_4) are identified from five *Segmented Windows* ($w_1 w_2, \ldots, w_5$).

Similarity Thresholds $T = \{T_1, T_2, \ldots, T_N\}$ is a collection of N scalar values for N Segmented Windows. The algorithm learns the value of N, as well as each scalar value T_w, which is used to determine the assignment of subsequences to clusters within a window w. Details about clustering using T will be discussed in Sect. 3.2.
Normalized Similarity Thresholds $T_{norm} = \{T_{norm\_1}, \ldots, T_{norm\_N}\}$ is used for normalized subsequences.

Correlation Anomaly is a type of anomaly in multivariate time series which shows normal behaviors in each individual dimension, but shows anomalous behaviors when we consider the combination of dimensions. An example of such anomaly is shown in Fig. 6.

2.2 Related work

Many time series anomaly detection algorithms have been proposed in recently years (Gupta et al. 2014; Izakian and Pedrycz 2014; Laptev et al. 2015; Xie et al. 2013; Wang et al. 2014), focusing on different types of anomalies, including point outliers, change points, anomalous time series, or structural (subsequence) anomalies. To identify anomalous time series, some works extract features from the original time series and perform anomaly detection in the transformed feature space. For example, one work (Hyndman et al. 2015) extracts commonly known representative features of time series, then perform anomaly detection on the first two principal components. In our previous work (Wang et al. 2015), we extract features that capture the characteristics of medical alarms to detect anomalies in medical time series.

Another type of anomaly detection algorithms use clustering techniques to detect abnormal behaviors (Budalakoti et al. 2006; Pires et al. 2005; Sun et al. 2004). Clustering-based algorithms often have difficulty detecting anomalous time series that contains subtle anomalies, since the algorithms use global measures to determine cluster membership, and deviation from subtle, localized anomalies can be dwarfed by global noises. Even in the case when the algorithms are able to identify an anomalous time series, they cannot locate the exact sections in the time series that contribute most for the abnormality. In order to pinpoint the exact anomalous locations in a time series, analysts may need to visually inspect the detected time series to find the anomalous parts. As will be shown later, our algorithm not only detects the anomalous time series, but also highlights the sections that *explain* the abnormality. Our algorithm outputs the top K (K is determined by the algorithm automatically) subsequences that exhibit abnormal behaviors.

Contrary to anomaly detection based on global measures on whole time series, window-based techniques can achieve better localization of anomalies. Time series discord discovery algorithms (Keogh et al. 2005, 2007) are a type of window-based algorithms that capture the most unusual subsequence(s) within a long time series. While discord discovery has been shown to achieve better performance than other existing techniques (Chandola et al. 2009), to identify a discord, the user must specify its length. There are two limitations with imposing this requirement in real-world applications. First, the user may not know the exact length, or even the best range of lengths for the potential discords in advance. Second, there might exist multiple discords of different lengths in a time series (Senin et al. 2015). It would be extremely cost prohibitive to consider all window lengths. We proposed a solution to mitigate this problem in a past work (Senin et al. 2015), but the algorithm still requires an initial seed length as input.

Another limitation with discord-based approaches is that the definition of discord is tied to the number of nearest neighbors, i.e. the discord is the object with the largest distance to its kth nearest neighbor. Imagine the simplest case where $k = 1$. If an anomaly occurs twice or more, then each anomalous instance would have a match that is very similar, in which case the algorithm would fail to detect them as anomalies. Having a fixed value k is not ideal because we simply do not know how many times an anomaly would occur. In this paper, we address the aforementioned problems. To

the best of our knowledge, our work is the first to solve the problem without any prior knowledge.

Recently there has been considerable interest in developing anomaly detection algorithms for multivariate time series. Some methods utilize time series projection (Galeano et al. 2006), independent component analysis (Baragona and Battaglia 2007) or hidden markov model (Li et al. 2017) to convert the multivariate time series into univariate time series, then perform a univariate method on them. One limitation for this type of methods is the loss of information during the transformation. Some other works use graph-based methods (Cheng et al. 2009; Id et al. 2007; Qiu et al. 2012). In general, nodes of a graph represent subsequences or data points while the weights associated with the edges of the graph are aimed to capture the degree of similarity between the corresponding nodes. Nevertheless, the potential limitation of these methods comes from the fact that more instances imply more time required to estimate the weights of its edges.

3 SLADE-TS method

We start by providing an overview of SLADE-TS, our proposed self-learning online anomaly detection algorithm for univariate time series data.

3.1 Overview of SLADE-TS algorithm

Our algorithm consists of four parts: (a) Adaptive segmentation by symbolic clustering; (b) Dynamic clustering; (c) Parameter self-tuning; and (d) Anomaly Scoring. The work flow is shown in Fig. 2.

We assume that the data come in a streaming fashion. We use a small amount of time series to first segment the data and obtain the positions of the *Segmented Windows* (i.e. via Symbolic Clustering, Sect. 3.3). When a new time series arrives,[1] we segment the time series into subsequences according to the learned *Segmented Windows* positions, and perform dynamic clustering to assign every subsequence to one of the clusters in the corresponding window (Sect. 3.2). The Anomaly Scoring algorithm assigns anomaly scores to subsequences based on the clustering results (Sect. 3.5). By aggregating the anomaly scores of the subsequences, we can identify both global and local anomalies.

SLADE-TS is a self-learning algorithm. When time series TS_i is processed, it updates important information to be used in dynamic clustering subsequently, including (i) *Segmented Windows* W, for cutting time series into subsequences; (ii) Clusters C; and (iii) *Similarity Thresholds* T. We name them WCT. We use the updated WCT to segment and cluster time series TS_{i+1}. Initial values of WCT are determined by the symbolic clustering algorithm using the first p time series. In our experiment, we set p as 5.

[1] We use the whole time series to demonstrate the idea, but our approach also works in the scenario when points of time series come in a streaming fashion.

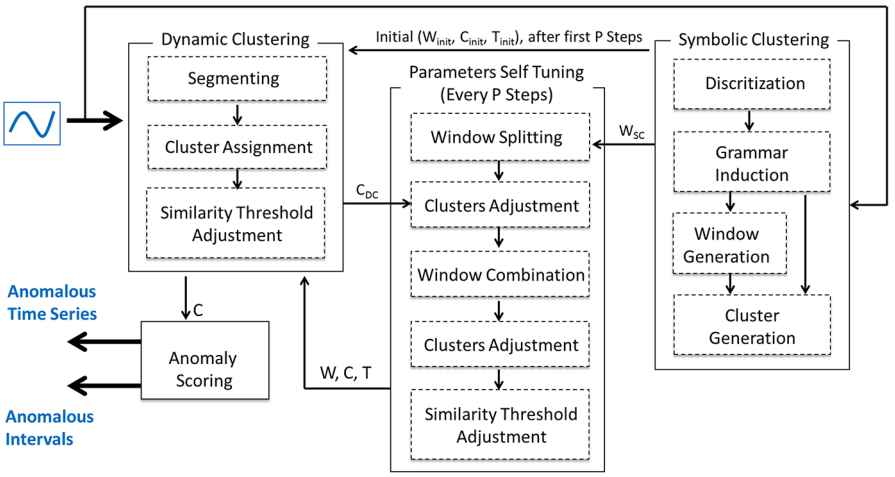

Fig. 2 The work flow of SLADE-TS algorithm

3.2 Dynamic clustering

Dynamic clustering was first introduced to cluster users of computer terminal (Sequeira and Zaki 2002). Then it has been used to detect outliers in data streams (Aggarwal and Yu 2010). In this paper, we use dynamic clustering to cluster subsequences extracted from different time series. Note that the clustering task here is different from the *time series subsequence clustering* problem (Keogh and Lin 2005), in which case subsequences are extracted from the same time series. Also different from the problem of clustering text and categorical data (Aggarwal and Yu 2010), we do not have the instances to be clustered. Specifically, before applying dynamic clustering, we need to cut time series into non-overlapping windows. We can then cluster subsequences that belong in the same window. The segmentation of time series into different windows is an important step for our algorithm. If a window is too large, we might miss some subtle anomalies. If a window is too small, true anomalies could be dominated by noise, or the detected anomalies may be only a subsection of longer anomalies.

In the next two sections, we will formalize an algorithm to determine the initial parameters WCT for our dynamic clustering algorithm, and an algorithm that self-adjusts windows dynamically. We defer the detailed discussion on segmentation until then. To demonstrate the idea of dynamic clustering, we assume for now that the time series is already segmented.

In the previous work of dynamic clustering (Aggarwal and Yu 2010), the process starts with an empty set of clusters. When a new instance arrives, unit cluster that contains single instance is created. Once a maximum number cl of such clusters have been created, the algorithm begins the process of online cluster maintenance. In this work, we do not start from an empty set, as initial clusters C_{init} and *Similarity Thresholds* T_{init} will have been generated by the symbolic clustering algorithm along with the segmentation. Therefore, we start the process of online cluster maintenance

with the initial information when a new time series arrives. For now, we assume that we already have C_{init} and T_{init}.

The dynamic clustering algorithm is outlined in Algorithm 1. There are four input parameters: time series TS, *Segmented Windows* W, current clusters C and *Similarity Threshold* T. We have defined TS, W and T in Sect. 2. Clusters C contain existing clusters learned so far for each window. These parameters are initialized by the symbolic clustering algorithm and updated in a self-learning process. The dynamic clustering algorithm starts by extracting subsequences from the new time series instance based on the *Segmented Windows* W generated by the symbolic clustering algorithm, Line 2. In the nested for-loop in Lines 3–15, we assign the subsequence ss_w from window w of TS to a cluster, by first trying to find an existing cluster close to it (Lines 5–10). If the closest existing cluster is close enough (i.e. within similarity threshold T_w for window w), we assign ss_w to it and update T_w (Lines 14–15). Otherwise, we create a new cluster for ss_w (Line 12). We describe the details in the following subsections.

Algorithm 1 Dynamic clustering algorithm

Input: Time series TS, window information W, current clusters C, similarity threshold for every window T.
Output: New clusters C_{DC}, new threshold T_{DC}

```
1: function DynamicCluster(TS, W, C, T)
2:     S ← CutTS(TS, W)                          ▷ S is a collection of subsequences
3:     for each w in W do
4:         d_min ← INF
5:         for each C_i in C_w do                ▷ C_w is all clusters in w
6:             d_i ← D(ss_w, c_i)                ▷ ss_w ∈ S
7:             nd ← D_norm(ss_w, c_i)
8:             if (d_i ≤ T_w)&(nd ≤ T_norm_w)&(d_i < d_min) then
9:                 d_min ← d_i
10:                C_o ← C_i                     ▷ C_o is the closest cluster to ss_w
11:        if d_min == INF then                  ▷ Create a new cluster
12:            C_w.add(NewCluster(ss_w))
13:        else                                  ▷ Assign a cluster
14:            AssignCluster(ss_w, C_o)
15:            UpdateSimilarityThreshold(d_min)
16:    return (C_DC, T_DC)
```

3.2.1 Finding closest cluster

We define ss_w as the subsequence from the new time series in window w. The distance value $D(ss_w, c_i)$ is computed over all clusters in window w, where c_i is the centroid of cluster C_i. We also compute the distance between normalized ss_w and c_i, in order to capture the shape difference. If $D_{norm}(ss_w, c_o) \leq T_{norm\_w}$, where T_{norm_w} is the similarity threshold for the normalized subsequences in window w, and $D(ss_w, c_o) < D(ss_w, c_j), \forall j$, cluster C_o is chosen as the closest cluster for subsequence ss_w.

The normalized distance gives more weight to shape difference while un-normalized distance focuses on the value difference. While in most time series applications, normalization is important as we want to capture shape (dis)similarity rather than actual

values, in our application we want to detect *both* shape and value differences, so we use both the un-normalized distance and normalized distance. We note that this can be easily changed depending on the application. To find the closest cluster for ss_w, we need to consider three scenarios:

(i) $\forall C_j$ in w, $D(ss_w, c_j) > T_w$. In this case, there is no cluster close to ss_w.
(ii) $\exists C_i$ in w, $D(ss_w, c_i) \leq T_w$, but $D_{norm}(ss_w, c_i) > T_{norm\_w}$. In this case, although cluster C_i is close to ss_w using un-normalized data, the distance for normalized data > threshold. We still do not assign ss_w to it.
(iii) $\exists C_o$ in w, $D(ss_w, c_o) \leq T_w$, $D_{norm}(ss_w, c_o) \leq T_{norm\_w}$ and $D(ss_w, c_o) < D(ss_w, c_j)$, $\forall C_j$ in w. In this case, the closest cluster C_o is found.

In this paper, we define C_o found in case (iii) as the closest cluster. If no such qualified cluster found (situations (i) and (ii)), we conclude there is no suitable cluster for ss_w.

3.2.2 Assigning cluster

If there is no suitable cluster found for ss_w, our algorithm creates a new cluster that contains the solitary instance ss_w. The new cluster is a potential true anomaly or the beginning of a new normal behavior. Further understanding of it may be obtained as more data are collected.

Otherwise, if the closest cluster C_o is found, we assign ss_w to C_o. Meanwhile, the cluster centroid of C_o should be updated by Eq. (1), where CN is the number of instances in cluster C_o before the new instance ss_w is added.

As the centroid of C_o is updated, the distance between the old instances in C_o and the new cluster centroid should be updated as well. However, if we call the distance updating method every time when a new instance is added into a cluster, it will be inefficient and unnecessary. Therefore, we perform batch update when p new time series have arrived. We will describe the details in Sect. 3.4.3.

$$c_{o\_new} = (c_o * CN + ss_w) / (CN + 1) \tag{1}$$

3.2.3 Updating similarity threshold

Similarity threshold T_w is used to determine if there is a suitable cluster for ss_w. Each window has a different threshold. The intuition behind this is that different sections of the time series may have different variance tolerance. *Similarity Thresholds* are dynamically determined based on the distance values of past assignments. The mean μ_t and standard deviation σ_t of the distance values to the assigned cluster centroid in the history of assignments are computed at the time of arrival of the tth instance. The threshold is set as in Eq. (2), which corresponds to the three sigma rule (Pukelsheim 1994) with a normal distribution assumption.

$$T = \mu + 3 * \sigma, (\mu : mean, \sigma : standard\ deviation) \quad (2)$$

Normalized Similarity Thresholds for normalized subsequences are computed separately using the same equation.

3.3 Symbolic clustering algorithm

Dynamic clustering needs WCT as input for further processing. If we do not have this information in advance, or if the parameters are chosen poorly, the clustering results may be undesirable. Therefore, we introduce a symbolic clustering algorithm that provides a heuristic to determine the initial WCT without prior knowledge of the data. Our symbolic clustering algorithm first discretizes continuous time series values into symbolic representation. It then identifies repeated patterns by learning a grammar from the data, in order to approximate the best segmentation, as well as the best clusters and threshold value per window.

In our previous work (Li et al. 2012; Senin et al. 2014), we proposed a grammar-based algorithm for efficient discovery of variable-length frequent patterns. In this work, we introduce a grammar-based segmentation and clustering algorithm. Our approach is built on a two phase process: (i) context-free grammar induction; (ii) clustering using grammar rules. The algorithm is outlined in Algorithm 2.

Algorithm 2 Symbolic clustering algorithm

Input: Time series TS, grammar G (empty if it is the first TS).
Output: Window information W, current clusters C, similarity threshold for every window T, updated grammar G'

```
1: function SymbolicCluster(TS, G)
2:     dTS ← Discretize(TS)                          ▷ Discretize time series TS
3:     G' ← GrammarInduction(dTS, G)
4:     FM ← ComputeFreq(G')                          ▷ Frequency Matrix FM
5:     W ← CutWindow(FM)                             ▷ Cut FM at changing points
6:     for each w in W do
7:         for each g in G' do
8:             if g covers the same location of w then
9:                 C_i ← time series covered by g
10:                C_w.add(C_i)                      ▷ C_i becomes one cluster in w
11:        C.add(C_w)
12:        Des_w ← distances between subsequences and centroid
13:        T_w = μ_w + 3 * σ_w
14:    return (W, C, T, G')
```

The algorithm takes a time series TS and the grammar learned from the past time series as input. Lines 2–5 performs initial segmentation of time series. Details are discussed in the following subsections. In the nested for-loop in Lines 6–13, we cluster subsequences in each segmented window w using the grammar information (details are discussed in 3.3.4). We compute the distances between subsequences and their respective centroids in Line 12. Similarity Threshold T_w for window w is computed in Line 13.

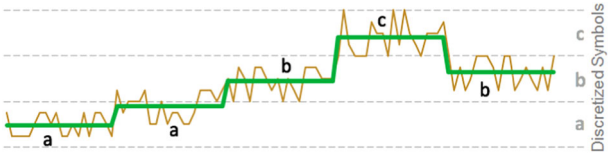

Fig. 3 Piecewise aggre-gate approximation (PAA). Time series TS (gold), its PAA representation (green/bold), and the converted discretized word. The discretized word is *aabcb* (Color figure online)

3.3.1 Discretization

Before applying our grammar induction algorithm, we prepare the data by converting each time series into a sequence of tokens, where each token represents a discretized version of a segment from the time series. In this work, we modify the discretization algorithm slightly: instead of using Symbolic Aggregate approXimation (SAX) (Lin et al. 2007), we use fixed-height discretization because we want to detect both shape and value differences, and SAX only applies to normalized time series.

For each segment, we first reduce the dimensionality with Piecewise Aggregate Approximation (PAA) (Keogh et al. 2001), as shown in Fig. 3. More specifically, PAA divides time series into s equal-sized partitions and computes a mean value for each partition. It then maps these values to symbols according to a pre-defined set of breakpoints that divide the range of values into α equal-height regions, where α is the alphabet size. The parameters of discretization, such as the segment length or PAA size, are set with default values in the experiments. For example, the subsequence length could be set as 5% of the segment length.

3.3.2 Constrained grammar induction

To describe grammar induction in detail, consider the following sequence of *words*. Each word is considered an atomic unit, or a *terminal* in the sequence: $S_1 =$ *aba bac cab acc bac cab*. We feed the string S_1 into a context-free grammar induction algorithm such as Sequitur, a string compression algorithm that infers a context-free grammar in linear time and space (Nevill-Manning and Witten 1997). When applied to a sequence of words, Sequitur treats each word as a token and compresses the sequence by learning a context-free grammar from the input. The algorithm recursively reduces all *digrams* (consecutive pairs of tokens) occurring more than once in the input string to a single new non-terminal symbol.

The following table shows the grammar induced by Sequitur from S_1. The algorithm reduces the length of the input string by creating a grammar whose rules are encoded by *non-terminal* $R1$, which reveals repeated patterns in the input. $R1$ describes a simplified version of the repeated pattern [*bac cab*].

Grammar rule	Expanded grammar rule
R0 → aba R1 acc R1	*aba bac cab acc bac cab*
R1 → bac cab	*bac cab*

Different from the original grammar induction method as shown above, in this work we modify the algorithm so that it is position-aware. That is, we consider the subsequence position information when we generate grammar rules. Doing so, S_1 becomes $S_1 = aba_1\ bac_{151}\ cab_{301}\ acc_{451}\ bac_{601}\ cab_{751}$. The subscript of each word denotes the starting position of the subsequence in the original time series. In this example, the original time series is divided into six non-overlapping segments, each with length 150. Each segment is then discretized, with parameters $s = 3$ (i.e. 3 PAA partitions since each word has three letters), $\alpha = 3$ (cardinality of alphabet is 3: 'a', 'b', 'c'). To determine a match between two digrams, in addition to having identical words, they also must occur at the same position of their respective time series. In other words, we only compare subsequences from different time series if they start at the same positions (with some modification, we could also allow some "warping" between the tokens). In this example, the two occurrences of "bac" are no longer a match because they have different location subscripts, denoting that these patterns are associated with different time intervals. Now suppose we have another sequence from a different time series, $S_2 = aba_1\ bac_{151}\ cab_{301}\ aca_{451}\ cac_{601}\ aaa_{751}$. Between S_1 and S_2, the tokens that can be merged are $aba_1\ bac_{151}\ cab_{301}$ with grammar rules $R1 \rightarrow R2\ cab_{301}$ and $R2 \rightarrow aba_1\ bac_{151}$, since they match on both the words and locations. The expectation is that time series data should have similar value and shape at the same time point. This is important for some real-world applications. For example, in the process of reflow soldering, the temperature is critical to the quality of solder joint.[2] The temperature profile, which shows the transient behavior of the solder reflow oven, should be the same for each item. Otherwise, it could yield a manufacturing defect. Therefore, a subsequence should only be a candidate anomaly if it is different from other subsequences at the same position (e.g. same point in the reflow soldering process).

3.3.3 Generating segmented windows

In most cases, anomalies do not last for the entire time series. If we try to detect anomalies using the whole time series, we may miss subtle anomalies, and we would not be able to identify the exact local anomalies.

Generating *Segmented Windows* is difficult because it requires information about potential anomalies. This is a "deadlock" problem here: we need proper segmentation to detect anomalies, while finding anomalies is the best way to segmenting the windows. We will show how we adapt grammar rules to solve this paradox.

We use frequency coverage of time series points to generate *Segmented Windows*. We define frequency coverage as the number of time series covered by the same grammar rule. With each grammar rule we also store the locations of subsequences that are covered by it, as well as the indices of their respective time series. We can then compute the frequency coverage of each point in a time series. Suppose S_1 and S_2 from the grammar induction example are the discretized string of $T\,S_1$ and $T\,S_2$ respectively. In this example, the frequency coverage for data points in the range 1–450 of $T\,S_1$ and

[2] Reflow soldering: https://en.wikipedia.org/wiki/Reflow_soldering.

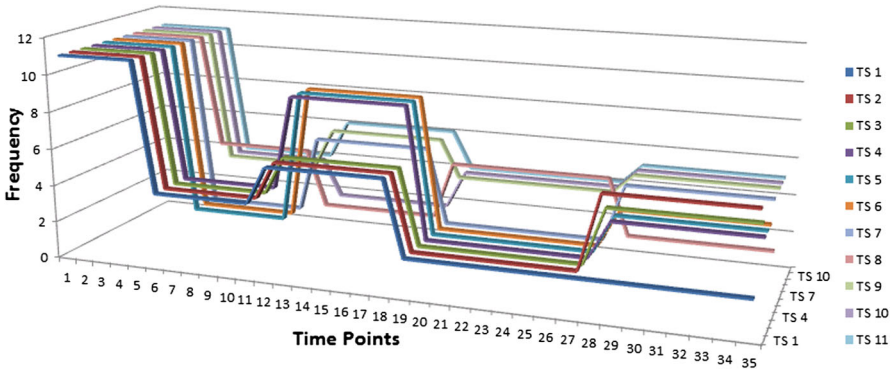

Fig. 4 Frequency coverage of each time point for TS_1–TS_{11}. We generate Segmented Windows by cutting at frequency-change points. In this example, they will be {1–5, 6–11, 12–18, 19–27, 28–35}

TS_2 is 2 because R_1 covers two time series. It means that these two time series share the same pattern at that time interval. The frequency coverage of other data points will be set to 1 since no grammar rule covers them.

Consider another example shown in Fig. 4. The dataset contains 11 time series ($TS_1, TS_2, \ldots, TS_{11}$), each of which has 35 time points. The first 5 data points in every time series are in the rule that covers all the eleven time series. So the frequency of the first 5 time points for all time series should be 11. After computing the frequency for every time point, we can get a frequency matrix that contains frequency for all time points in all time series, as shown in Fig. 4. We set the cut points at the locations where the frequency changes. The intuition behind this is that the changing of frequency suggests the changing of pattern groups. If frequency changes at point A, the group information on the left and right side of point A are different. If a time series starts to form a new group at point A, it possibly starts to show anomalous behavior. In contrast, if it starts to show the same behavior as other time series from point B, then the anomalous behavior only occurs between point A and point B. Thus, the changing points should be the start or end location of anomalous behavior. The collection of all cut points define the *Segmented Windows* W.

3.3.4 Symbolic clustering in each window

After segmenting the time series, we generate initial clusters using grammar rules. In each window, we search for rules that overlap the same window. We assign subsequences into the same cluster if they are covered by same rule. It is possible that a subsequence is (partially or wholly) covered by more than one grammar rule. We merge clusters if they share a large fraction of the same instances. Otherwise, we break tie by assigning a subsequence to the closer cluster.

Because the grammar rules may not cover all subsequences, the clusters generated by symbolic clustering algorithm may not contain all subsequences from each *Segmented Window*. In addition, since the symbolic cluster algorithm works in symbolic space, there may be some information loss due to the discretization. To determine clus-

ter membership for the missed subsequences in window w, we select subsequences that do not get assigned to any cluster, and call the cluster adjustment function described in Sect. 3.4.3 for furthur processing. To refine clusters for a window w, we do the same by selecting subsequences whose distances to their respective centroids are greater than T_w, and calling the cluster adjustment function.

After we obtain clusters for all *Segmented Windows* by symbolic clustering algorithm, we generate cluster centroid for each cluster and compute the distances from the cluster centroid to every instance belonging to that cluster. Since we have the distances of all subsequences to their respective centroids, we can compute the initial *Similarity Thresholds* by calling similarity threshold updating function introduced in Sect. 3.2.3.

3.4 Parameters self tuning

Our algorithm is a self-learning algorithm, that is, the accuracy gets better when more data arrive. The information we get from past data, such as *Segmented Windows*, may only be good for past data but not current data. Thus, we need to update WCT as more data arrive. In our algorithm, when every p new time series arrive, we call symbolic clustering algorithm to get new *Segmented Windows*. However, instead of using the new values directly to replace the old ones, as shown in Fig. 2, we adjust WCT with the following steps: (i) Window Splitting; (ii) Clusters Adjustment; (iii) Window Combination; (iv) Clusters Adjustment; (v) Similarity Threshold Adjustment.

The details of step (ii) and (iv) are introduced in Sect. 3.4.3, while step (v) has been discussed in Sect. 3.2.3. We discuss the detail of step(i) and (iii) here.

3.4.1 Window optimization

The initial *Segmented Windows* are generated by the symbolic clustering algorithm. However, the window information may not be perfect for anomaly detection because: (a) we may lose information by discretization. (b) The grammar induction algorithm in symbolic space requires exact match, which may cause incorrect changes of frequency coverage. (c) The frequency coverage that is used to generate *Segmented Windows* reflects the grouping differences between different sections of time series, but it is not the exact cluster information. In order to get more accurate *Segmented Windows*, we propose two techniques here: window $combination$ and window $splitting$.

3.4.2 Window combination

The constrained grammar induction method uses exact match to find frequent patterns. This strategy may cut a pair of long similar patterns into two pairs of shorter patterns if there are few mismatches in the middle of the long pattern. Therefore, the generated *Segmented Windows* may be smaller than they should be. In addition, the discretization step may produce different symbols for similar values if these values happen to reside on opposite sides of a breakpoint (See Fig. 3. The second 'a' is very close to the breakpoint. A value slightly larger than it would land on the other side of the line, and be assigned to symbol 'b'). Such boundary problem may also cause the breakage

of a long pattern. To obtain appropriate size of windows, we introduce a window combination technique here.

The idea of window combination strategy is straightforward. We combine two consecutive windows if the clustering results in those two windows are similar. The intuition is that, same clustering results in two consecutive windows suggest that the anomalous behaviors, if any, remain the same from one window to the next. In this case, we should extend the length of the anomalies by merging windows.

In order to define similar clusters, we apply Jaccard Coefficient (Jaccard 1912) to the comparison of clustering results. For two consecutive windows, we use the clustering result from the left one as the ground truth. Then we compute the Jaccard distance between the clusters from the two consecutive windows. If the Jaccard distance is large enough, say 80%, the clustering results from the two windows are similar, in which case we should merge the windows into a larger one.

Window Splitting Due to the online property of our algorithm, the current data at some time points may not contain information for future data. For example, the initial p time series that we use to set the parameters could all be normal and similar to each other, in which case we may get a large window with length as long as the whole time series. However, at some time in the future, we may encounter some anomalies in the new data. In this case, we should split the window into smaller ones. The discretization process may also cause window sizes to be larger than they should be since each symbol is an approximation of a range of values in the original time series. To mitigate these problems, we introduce a window splitting technique.

A brute-force way to accomplish this goal is to use a relatively small sliding window to cut the large window into smaller ones and apply clustering algorithm for each small window. We can then call the window merge method introduced previously to get the correct windows. However, this method is inefficient and requires parameters such as the sliding window size.

As mentioned before, our algorithm calls the symbolic clustering algorithm with every p new time series arrived, and finds *Segmented Windows* efficiently. So each time a new set of *Segmented Windows* is generated, we can compare them against the current set of windows. We split an old window into smaller ones if there are several new windows that overlap the old window.

3.4.3 Cluster adjustment

Since clustering is done incrementally, the cluster centroid may move considerably after processing many instances. This may cause two potential problems. (i) Some instances in the cluster may no longer belong to the cluster if their distances to the new centroid are now greater than the threshold. (ii) Clusters may become very close to each other that we should merge them. To solve these two problems, our algorithm updates the clusters when instances are added to clusters.

To determine whether any clusters should be merged, our algorithm computes pairwise distances between centroids. If a distance is smaller than the similarity threshold, we merge the clusters. The centroid of the new cluster should also be updated accordingly.

To determine whether any cluster should be split, when a new instance ss_w is added into a cluster and the cluster centroid has changed, we compute the distance between the new cluster centroid to every instance in the cluster. If the distance between instance ss_{w_i} and its centroid is greater than the similarity threshold, our algorithm removes ss_{w_i} from the cluster, and calls the dynamic clustering algorithm that we introduced in Sect. 3.2 to re-assign it to a new cluster. At the same time, since ss_w is removed from the old cluster, we need to update the centroid of the old cluster.

In practice, we do not need to update distances and perform cluster adjustment every time an instance is added to a cluster, because the change of centroid is likely very insignificant when the number of new instances added is much smaller than the number of instances in the cluster. Thus, we perform batch updates after we have processed p new time series, where p is an integer value between one and the number of total time series.

3.5 Anomaly scoring

After we have the clustering results for each *Segmented Window*, we compute anomaly scores for all windows, and all time series. One way to obtain an anomaly score for a time series is to take the average of anomaly scores from all windows. This method does not work well if most windows do not contain anomalies, since the scores from unrelated subsequences will have negative impact on the final aggregated score. Another approach is to select the top h windows with the largest anomaly scores, and take the average of these h scores. For this method, the parameter h needs to be set properly. The third approach is to set anomaly score as the number of times the running average of the anomaly scores exceeds a threshold. However, this method does not consider the magnitude of the anomaly scores since every score has the same weight if it exceeds the threshold. In this paper, we use a fourth approach, which sets the overall anomaly score as the average of the scores from the anomalous subsequences whose scores exceed a threshold T_{as}. T_{as} is also computed by Eq. (2), here μ and σ are the average and standard deviation of anomaly scores of all subsequences.

Within each window, the anomaly score for each instance is computed based on the clustering result. We can assign score to an instance based on the number of instances in the same cluster. The ones with larger numbers have smaller anomaly scores. We could also compute anomaly score based on density: lower density means higher anomaly score. In our experiments, we modified the scoring method from a existing work (He et al. 2003), which considers both the size of cluster and the distance to cluster centroid, as shown in Eq. (3):

$$AS = D(ss_w, c_i) * (1 - \frac{|C_i|}{|C_{big}|}). \tag{3}$$

where c_i is the centroid of cluster C_i that contains instance ss_w, if cluster C_i contains more than $T_{support}$ instances. Otherwise, c_i is the centroid of cluster i closest to ss_w that has enough number of instances. C_{big} is the cluster that has the largest number

of instances. $|C_i|$ and $|C_{big}|$ are the numbers of instances in clusters C_i and C_{big}, respectively. We define $T_{support}$ as $|C_{big}|/2$.

4 SLADE-MTS method

In previous section, we discussed anomaly detection on univariate time series data. In this section, we extend the algorithm for multivariate time series data. In SLADE-TS, the step that dynamically updates segmented windows for more accurate anomalous subsequences not only provides the windows for potential anomalies, but also windows for patterns that represent the behaviors of the objects generating the data. By performing clustering in each window, objects in the same cluster have the same behaviors in the corresponding location. In the following sections, we will discuss how we use these patterns to detect anomalies in multivariate time series data.

4.1 Overview of SLADE-MTS algorithm

We are now in a position to explain the intuition that allows us to efficiently find anomalies in multivariate time series data. There are three challenges for multivariate time series anomaly detection. First, the accumulative distance over all the dimensions may mask true anomalies. Second, there may exist local correlation between variables in multivariate time series, More specifically, patterns in each individual dimension may seem normal, but abnormal if combined with other dimensions. Third, the locations and the dimensions of the anomalous subsequences need to be detected as well. To solve these problems, we introduce our SLADE-MTS algorithm in this section. As shown in Fig. 5, there are five steps in SLADE-MTS algorithm. The first step is to generate patterns that represent the behaviors of objects that generated time series data. Second, we transform the original time series data into a new feature space in which each feature corresponds to the distance to each pattern. Third, we apply feature selection to select important features. Fourth, we apply cluster based anomaly detection algorithm to detect anomalous time series using the transformed data with a set of concise and important features. Finally, we propose a locating algorithm that identifies the relevant dimensions and locations of anomalous subsequences.

4.2 Pattern generating

In SLADE-TS, we segment the time series into meaningful windows and cluster the subsequences in each window. As discussed before, these clusters represent behaviors of the objects that generate the time series data. Instead of computing an anomaly score for each subsequence, in SLADE-MTS, we generate patterns from these clusters. The first step of SLADE-MTS is to follow the steps in SLADE-TS to detect clusters in each window for each dimension individually. In this paper, we use the center of each cluster as the pattern that represents the behavior of that cluster. For anomalous time series, their behaviors (individual or combination) should be different from others.

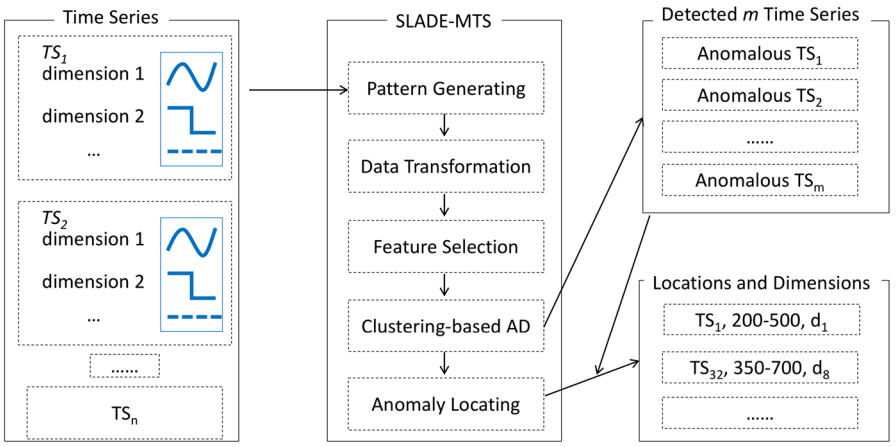

Fig. 5 The work flow of SLADE-MTS algorithm

For example, in Fig. 1, for univariate time series data, the anomalous time series TS_2 has a different behavior in window w_2 compare to other normal time series. Also, in Fig. 6, time series TS_{201} is anomalous time series because it has a different behavior with the combination of behavior in dimensions 1 and 2.

Algorithm 3 Pattern Generating

Input: Multivariate Time Series Dataset D, total number of dimensions $numDim$.
Output: Patterns $ListPatterns$

1: **function** PATTERNGENERATE(D)
2: $ListPatterns \leftarrow \emptyset$
3: **for each** $dimensionI$ in $numDim$ **do**
4: $C \leftarrow \emptyset$ ▷ All clusters in dimension i
5: **for each** ts in $dimensionI$ from D **do**
6: **if** this is p step **then** ▷ For every p step, we optimize parameters
7: $(W, C, T, G') \leftarrow$ SYMBOLICCLUSTER(TS, G)
8: $(W, C, T) \leftarrow$ PARAMETERSELFTUNE(W, C)
9: $C \leftarrow$ DYNAMICCLUSTER(TS, W, C, T)
10: **for each** $cluster$ in C **do**
11: $c \leftarrow cluster$ ▷ c is the cluster center
12: $ListPatterns.add(c)$
13: **return** ($ListPatterns$)

To identify these patterns, in the multivariate method here, as shown in Algorithm 3 from line 6–9, we reuse three algorithms from SLADE-TS: symbolic clustering, dynamic clustering and parameter self tuning. It automatically cuts time series into segmented windows and outputs the clusters of subsequences in each window. We compute the centroid of each cluster to represent the pattern. We repeat these steps for time series in each dimension individually, and collect all patterns (centroids).

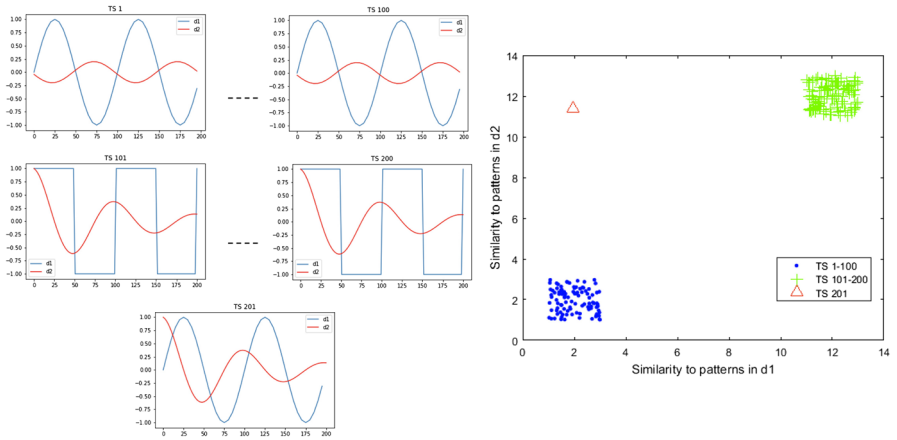

Fig. 6 Correlation anomaly. Time series $TS201$ is a correlation anomaly. In dimension $d1$ it is similar to $TS1$–$TS100$, and in dimension $d2$ it is similar to $TS101$–$TS200$, but it is different from all other time series when we consider both dimensions

4.3 Data transformation

After we have all patterns from all dimensions, the next step is to use them to identify anomalous time series by checking if they contain anomalous patterns. Anomalous patterns could be individual or correlated. Individual anomalous patterns are rare patterns that make the time series different from most other time series. For example, in Fig. 1, time series time series TS_4 has an anomalous pattern in window w_2 while none of the other time series contains that pattern. On the other hand, we can also say that all other time series but TS_4 contain pattern p_2 in window w_2, which makes TS_4 different from others. Combination difference means that each dimension in a time series may seem normal independently for a specific window, but if we consider the combination of patterns in different dimensions, such combination may be abnormal. As an example, in Fig. 6, the dataset contains 201 time series and 2 dimensions. If we consider only dimension d_1, we will find two patterns, one of them has 101 instances (TS_1–TS_{100}, TS_{201}) and the other one has 100 instances (TS_{101}–TS_{200}). Both of them can be considered normal patterns since they have similar numbers of occurrences. Similarly, if we only consider dimension d_2, we will see that the patterns form two (normal) groups as well. However, if we consider both dimensions, we would find that time series TS_{201} is different. In this example, it is easy to say we should use both d_1 and d_2 for anomaly detection task. However, considering all possible subsets of dimensions to find subpsace anomalies is computationally costly if the number of dimensions is high.

To efficiently use the generated patterns to detect both individual and correlation anomalies, we transform the original time series data into a new feature space using the generated patterns, in which each feature represents the distance (or similarity) from a time series to one pattern. The idea is to use detected patterns to transform original time series into a number of features that can be treated as a generic point anomaly detection problem, but it still preserves the contextual information of time series data. An example of the transform process is shown in Fig. 7.

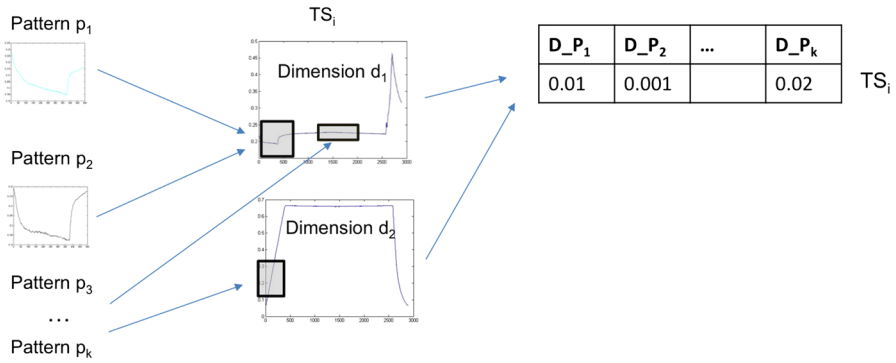

Fig. 7 Data Transformation. Patterns p_1–p_k are patterns generated in *Pattern Generating* step. To transform a time series TS_i, we compute the distances between all the patterns and the corresponding subsequences from TS_i. The output is a data matrix with each time series in a new feature space where each row is a time series, and each column (feature) is the distance to each pattern

The transformation algorithm is outlined in Algorithm 4. The process is carried out using the Euclidean distance between cluster centers and the corresponding subsequence of the current time series. First, a set of patterns P is generated from the cluster centers, as seen in previous section. These patterns are generated from different positions of original time series. The Euclidean distance between pattern p (p is the cluster center c) and time series ts is the distance between pattern p and the corresponding subsequence of ts (subsequence of ts at the some location of p) in the same dimension. For each time series window (segmented from previous steps using SLADE-TS), we compute its distance to all k patterns. The resulting k distances are used to form a new instance of transformed data, where each feature corresponds to the distance from each pattern to the original time series. The value of k is the number of patterns discovered from previous section. It is not a fixed number, but a number chosen by the algorithm based on how many clusters have been discovered in pattern generating.

Algorithm 4 Data Transformation

Input: Time seires dataset D, Generated patterns $ListPatterns$ and the corresponding dimension and location.
Output: Transformed Time Series List $ListNewTS$

```
1: function DataTransformation(D, ListPatterns)
2:     ListNewTS ← ∅
3:     for each ts in D do
4:         transformed ← ∅;
5:         for each p in ListPatterns do
6:             ss_ij ← subsequence of ts at location i and dimension j
7:             dist ← EuclideanDistance(ss_ij, p)
8:             transformed.add(dist)
9:         ListNewTS ← transformed
10:    return (ListNewTS)
```

4.4 Feature selection

Using all k patterns will not necessarily yield the best result for anomaly detection. Some of the patterns are irrelevant to the difference between the normal data and the anomalies. For example, in Fig. 1, in window w_1, every instance belongs to the same cluster. The pattern from that cluster cannot discriminate between the normal and anomalous time series. Besides the irrelevant patterns, some patterns contain redundant information which may already exist in other patterns. Moreover, we may have hundreds or thousands of patterns in the data, and the high dimensionality significantly increases the time and space requirements for detecting anomalies. Therefore, the goal is to find the relevant feature subset from the original features that can facilitate anomaly detection process.

Any feature selection algorithm could be used on the transformed data to select a subset of features. In this paper, we use Multi-Cluster Feature Selection (MCFS) (Cai et al. 2010) to select features that best preserve the multi-cluster structure of the data. MCFS selects features according to spectral clustering. MCFS works in the following steps:

(i) It first constructs a graph with N vertices where each vertex corresponds to a time series in transformed feature space. For each transformed time series x_i, it finds its q nearest neighbors and puts an edge between x_i and its neighbor x_j with weight W_{ij}.
(ii) It defines a diagonal matrix D whose entries are column sums of W and computes the graph Lapalcian $L = DW$. Then it solves the eigen-problem $Ly = \lambda Dy$ to find $Y = [y_1, \ldots, y_K]$ which contains the top K eigenvectors with respect to the smallest eigenvalues.
(iii) Solve K L1-regularized regression problem:

$$\begin{aligned} \min_{a_k} &= \|y_k - X^T a_k\|^2 \\ s.t. \quad & |a_k| \leq \gamma \end{aligned} \tag{4}$$

where a_k contains the coefficients.
(iv) For every feature j, compute the MCFS score for each feature according to the following equation. Select the top features with highest scores:

$$MCFS(j) = \max_k |a_{k,j}| \tag{5}$$

With the feature selection step, we obtain a subset of the original features. It removes irrelevant and redundant features but still preserve the multi-cluster structure of the data. This is useful to choose the patterns that have the abilities to differentiate anomalies from normal data if there are any.

4.5 Anomaly detection on transformed data

After data transformation and feature selection, we have a new data matrix. Each instance in it is a time series in the transformed feature space with concise and important features. The features of each instance is the distance to the selected patterns. In this paper, since our task is unsupervised anomaly detection, we use clustering based anomaly detection on the transformed data. It is a two steps process: (i) the first step is to perform a clustering task on the transformed data; (ii) the second step is to assign anomaly score based on the clustering result.

Any clustering method can be used on the transformed data. In our experiments, we use DBScan (Ester et al. 1996) as the clustering method. For anomaly scoring, the score for each instance is computed based on the clustering result and the distance to normal cluster. One option is to assign score to an instance based on the number of instances in the same cluster. The ones in larger clusters have smaller anomaly scores. Another option is to compute anomaly score based on density: lower density means higher anomaly score. In our experiments, we consider both the size of cluster and the distance to cluster center, as shown in Eq. (6).

$$AS = D(TransformedTS, c_i) * (1 - \frac{|C_i|}{|C_{big}|}). \tag{6}$$

where c_i is the center of cluster C_i that contains instance ss_w, if cluster C_i contains more than $T_{support}$ instances. Otherwise, c_i is the center of cluster i closest to ss_w that has enough number of instances. C_{big} is the cluster that has the largest number of instances. $|C_i|$ and $|C_{big}|$ are the numbers of instances in clusters C_i and C_{big}, respectively. We define $T_{support}$ as $|C_{big}|/2$.

There is an assumption for unsupervised anomaly detection algorithm: normal instances are far more frequent than anomalies in the data. With this assumption, we use three sigma rule to select the anomaly score threshold. The threshold is set as in Eq. (7), which corresponds to the three sigma rule (Pukelsheim 1994) with a normal distribution assumption.

$$T = \mu + 3 * \sigma, (\mu : mean, \sigma : std) \tag{7}$$

To identify anomalous time series, we use the threshold to be the cutting line. Any time series with anomaly score larger than the threshold will be considered as anomalous. Similarly, time series with score below the threshold will be considered as normal.

4.6 Locating anomalies

At this point, we have already detected anomalous time series from multivariate time series data. But yet, we do not know the reason for a time series to be considered as anomaly. More specifically, we do not know in which dimension(s) the time series is anomalous, and also we do not know the locations of subsequences that have the

anomalous behaviors. In this section, we introduce a locating strategy to pinpoint the anomalies with the help of: (i) the detected anomalous time series; (ii) the selected features and (iii) the transformed data.

To determine the reason for a time series to be considered an anomaly, we compare it to the normal data. The comparison is a three step process: (i) we select all anomalous time series of the same type, and the normal time series of the same type. Same type means they have the same cluster label; (ii) With the selected two groups of time series, we compare them on the transformed feature space on each selected feature, as shown in Algorithm 5 line 9. (iii) If the difference between them is larger than the threshold t, we conclude they are different on that feature (pattern). Therefore, we add the corresponding dimension as well as the location on original time series of that feature (pattern) to the result list, line 12. The differences between these two groups are captured by the different patterns between them.

Algorithm 5 Locating Anomalies

Input: Cluster results on transformed time series $Clusters$; Anomalous time series label la; Transformed data TD;
Output: Dimension and location of anomalous subsequences $ListDimLoc$.

1: **function** LOCATEANOMALY($Clusters, la, TD$)
2: $ListDimLoc \leftarrow \emptyset$
3: $ln \leftarrow$ label of the larges cluster ▷ The label of normal cluster
4: **for each** j in feature space of TD **do** ▷ feature j in the transformed feature space
5: $\mu \leftarrow$ MEAN($TD_{ln,j}$) ▷ mean value of normal group in TD in feature j
6: $\delta \leftarrow$ STD($TD_{ln,j}$) ▷ std of normal group in TD in feature j
7: $t \leftarrow \mu + 3 * \delta$
8: $\mu_a \leftarrow$ MEAN($TD_{la,j}$) ▷ mean value of anomaly group in TD in feature j
9: $dist \leftarrow |\mu_a - \mu|$
10: **if** $dist > t$ **then**
11: $loc \leftarrow$ location of feature j int original time series
12: $ListDimLoc.add(j, loc)$
13: **return** ($ListDimLoc$)

A concrete example is shown in Fig. 8. In the transformed feature space, the detected anomalous time series (red) is different from normal time series (gray) in features 2 and 4. In this example, the red time series is anomaly because it does not have pattern 2 and pattern 4, which are two patterns in the gray group.

5 Experimental evaluation

For SLADE-TS, the goal of our experiments is to show that our algorithm not only detects the anomalous time series but also identifies where the anomalous subsequences are. We also show that our algorithm is able to find the correct anomalies without setting parameters like the number of anomalies or the length of anomalies. The parameters we need are only for discretization step, which do not require prior knowledge of dataset. For comparison, we compare our algorithm with a window-based discord technique HOTSAX (Keogh et al. 2005) (other discord algorithms such

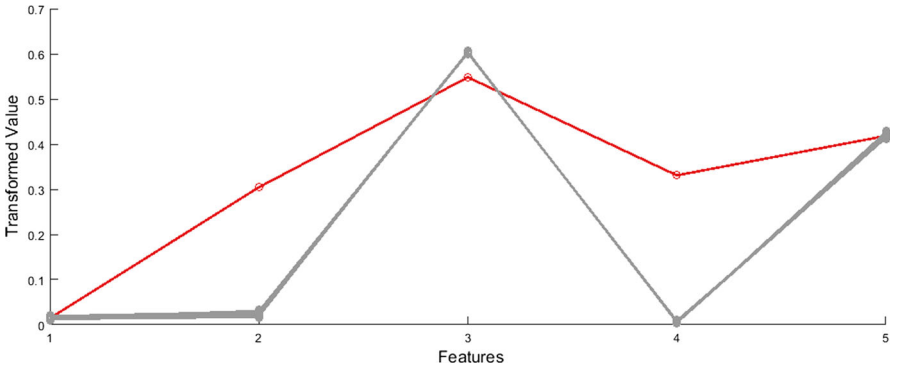

Fig. 8 Two clusters of time series on the transformed spcace with selected features

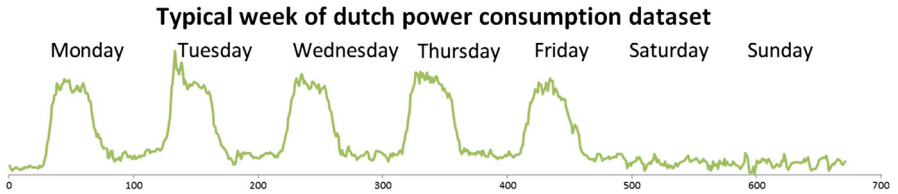

Fig. 9 Typical power consumption in one week

as RRA (Senin et al. 2015) generates similar results). We also compare with anomaly detection algorithm which works on whole time series other than the subsequences.

For SLADE-MTS, in this section, we show that it is able to: (i) detect anomalies from multivariate time series data and categorize different types of normal and anomalous patterns; (ii) successfully detect anomalous time series even if the anomalies only exist on some subsequences in a subset of dimensions, and also detect the corresponding dimensions and locations; (iii) detect correlation anomalies even if they look normal in each individual dimension. In comparison experiment, we compare with a PCA related method (Ahmed et al. 2012). In addition, we demonstrate the utility of our approach on real-world manufacturing data from Intel.

5.1 Dutch power consumption data

The power data contains 52 weeks of power demand by a research facility. The typical weekly power consumption is shown in Fig. 9. The original power demand data is a single time series with the length of one year. In this experiment, since we want to detect anomalies within a week, we pre-process the data by cutting the time series into 52 time series, each one contains one week of data from Monday to Sunday.

Our SLADE-TS algorithm detected eight anomalous time series with the discretization parameters: initial segment length $= 20$, $\alpha = 5$, PAA size $= 3$. The results are shown in the top of Fig. 10. The lengths and locations of anomalous subsequences for each anomalous time series are also identified. Two examples of the detected anomalous time series and their corresponding anomalous subsequences, highlighted in grey, are shown in the bottom of Fig. 10. In the left plot, SLADE-TS detected an anomalous

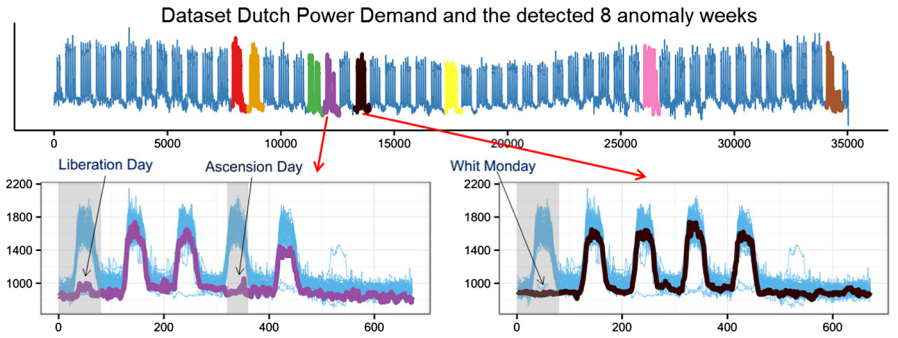

Fig. 10 Top: 8 anomalous weeks detected by SLADE-TS. Bottom: zoomed-in plot of 2 anomalous weeks and the corresponding events. Purple and black time series are detected anomalous weeks while blue time series are other weeks (Color figure online)

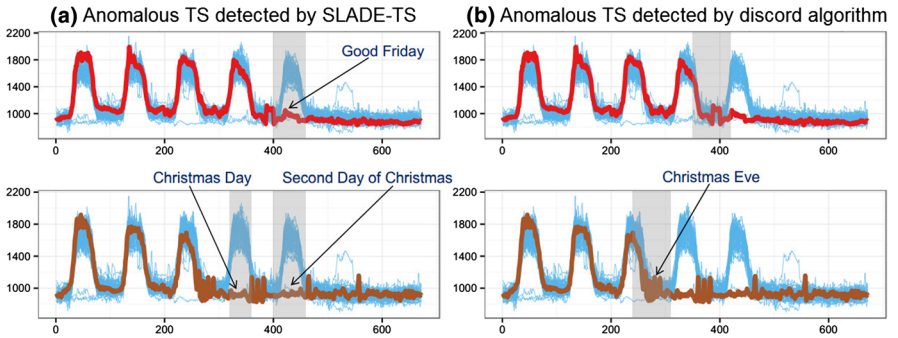

Fig. 11 Detected anomalous subsequences from the same anomalous weeks (top: 12th week, bottom: 51th week) by SLADE-TS and discord algorithm

pattern on Monday (Liberation Day) and Thursday (Ascension Day). The detected anomalous subsequence in the right plot can be interpreted as the national holiday of Whit Monday.

Figure 11 shows the comparison of the anomalies detected by SLADE-TS and discord algorithm (HOTSAX) on the same time series. The discord algorithm requires two parameters as input: length of discords and numbers of discords. In order to compare with them, we set the discord length close to the length of one day, which is 70 in this data set. The number of discords is set to 8, which is the actual number of anomalous time series. The number of nearest neighbors are set to 1. For a fair comparison, we modified the discord algorithm to only compare subsequences that are at the same locations, e.g. Monday to Monday, Tuesday to Tuesday, etc. The discord algorithm detected 5 correct anomalous time series, two of them are shown here for comparison. For the same time series, SLADE-TS and discord algorithm found different anomalous locations. As shown in Fig. 11, SLADE-TS found anomalies at locations matching *Good Friday*, *Christmas Day* and *Dat after Christmas*, while discord algorithm found anomalies at the night before *Good Friday*, and *Christmas Eve*. Note this is a relatively "simple" problem for discord discovery since the length of the anomalies can be pre-determined (i.e. one day).

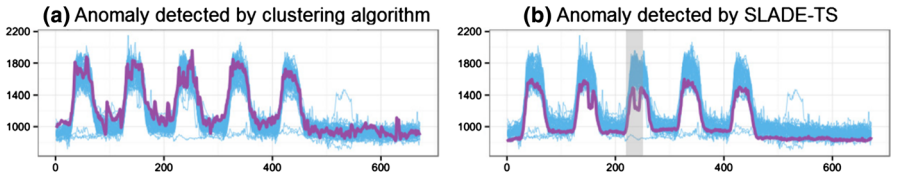

Fig. 12 Different anomalous weeks detected by clustering algorithm and SLADE-TS

We also compare our algorithm clustering-based anomaly detection approach that works with whole time series. We choose a recently proposed clustering algorithm (Begum et al. 2015) based on dynamic time warping (DTW), which outperforms existing clustering algorithms. However, it is not an anomaly detection algorithm itself, so we added an extra step to detect anomalies from the clustering result. This algorithm requires parameter setting of the number of clusters. From the experiments, this clustering algorithm achieves its best performance when the number of cluster is set to 7, so this is what we report. In total, the DTW-based clustering algorithm found eight anomalous weeks, seven of which are the same as anomalous weeks detected by SLADE-TS. The different detected anomalies are shown in Fig. 12.

The clustering-based anomaly detection algorithm uses the whole time series for clustering, so it could miss anomalies that only occur in short sub-sections. Furthermore, since the time series are noisy, the small deviations could add up over time, resulting in large distances. As a result, the time series could be misclassified as anomaly. For example, in Fig. 12, the detected anomalous time series in (a) was detected as anomaly but there is no significant difference. The anomalous time series in (b) was missed by the DTW-based clustering algorithm even though there is an obviously unusual pattern on Wednesday.

Besides the problems of false anomalies and parameter setting, the whole time series clustering based algorithm cannot detect where the anomalies are within time series while SLADE-TS finds the information automatically. Knowing exactly where the anomalies occur is highly valuable in pinpointing problems and looking for solutions.

5.2 Intel production dataset 1

This dataset comprises of normalized temperature profiles observed by thermocouples in an epoxy cure oven. The data has been normalized to rescale the actual temperature/time for proprietary reasons - but it retains the overall transient behaviors. Typical excursion prevention (EP) applications would involve an engineer using his intuition and judgement to setup time windows and various metrics (maximum value, mean, slope etc.) to capture the behavior of the time series within each window in order to ensure the process stays on target and the dynamic response of the oven is captured.

This data set has 8 dimensions and 11 time series of length 3058. There are three anomalies in the data set. These anomalous time series only behave differently in some subsequences from some sub-dimensions. We do not know the lengths of anomalous subsequences nor the dimensions that contain the anomalous subsequences. If we use all dimensions with whole time series data, we cannot find these three anomalous

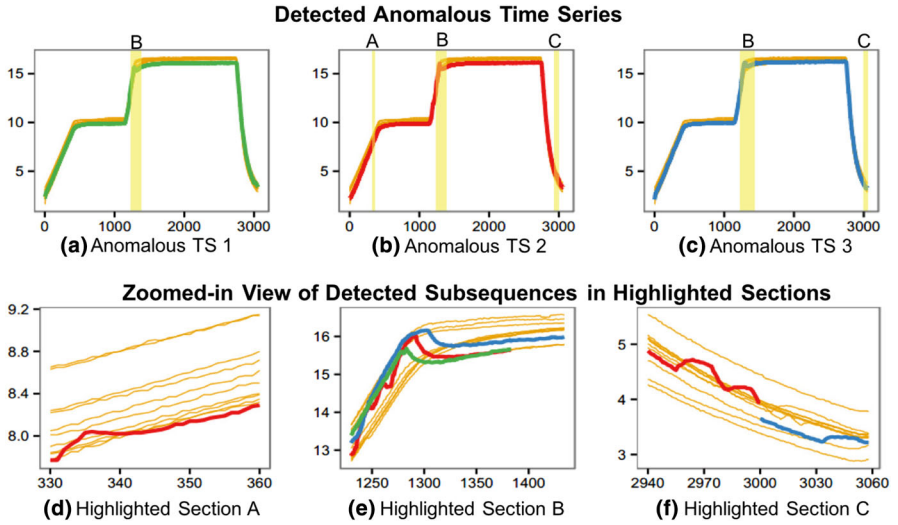

Fig. 13 Top: anomalous time series detected by SLADE-TS algorithm. Highlighted sections indicate locations of the detected anomalous subsequences. Bottom: zoomed-in view of anomalous subsequences

time series. On the other hand, if we try every combinations of dimensions as well as every possible length of subsequences, the cost would be prohibitively expensive. In this section, we will show the results detected by SLADE-TS on one dimension (univariate time series) and the results detected by SLADE-MTS with data from multiple dimensions (multivariate time series).

5.2.1 SLADE-TS results

For SLADE-TS, we use the time series data from dimension 2. Anomalies detected by SLADE-TS and discord algorithm are shown in Figs. 13 and 14 respectively. Our algorithm found 3 anomalous time series (shown as top 3 plots of Fig. 13, detected anomalous time series are shown in bold, the highlighted sections are the corresponding anomalous subsequences) and 6 anomalous subsequences (bottom 3 plots are zoomed-in view of detected anomalous subsequences. Their colors match the colors of their time series). The discretization parameters we used are: initial segment length $= 20$, $\alpha = 6$, PAA size $= 3$. To have discord algorithm also detect 3 anaomalous time series and find as many anomalous subsequences as possible, we set the number of discords as 4. With larger number of discords, discord algorithm returns incorrect anomalous time series in addition to the 3 correct ones, while with smaller number of discord, it finds fewer anomalous subsequences. Also, from prior knowledge of data set, we set discord length as 90 to achieve its optimal performance.

Comparing the results, the discord algorithm detected anomalies with fixed length which is given by the user. SLADE-TS automatically detected anomalies with various lengths without prior knowledge. Besides the variable lengths of anomalous subsequences, the number of anomalies is also automatically detected, as shown in Fig. 13.

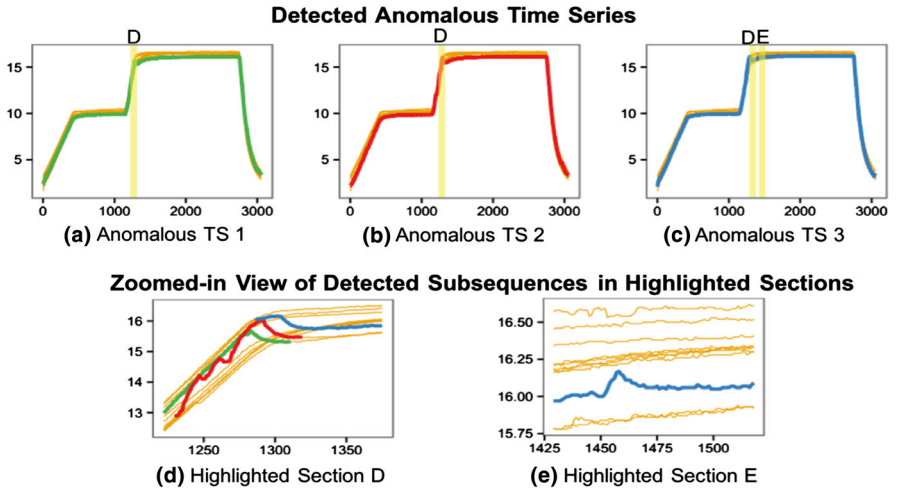

Fig. 14 Top: anomalies detected by discord algorithm. Highlighted sections indicate locations of the detected anomalous subsequences. Bottom: zoomed-in view of anomalous subsequences

Both algorithm detected the same anomalous time series, but some corresponding anomalous subsequences are different. The anomalous subsequences shown in the zoomed-in view of Figs. 13e and 14d are similar. The different anomalies detected are shown in Fig. 14e (by discord algorithm), and Fig. 13d and f (by SLADE-TS). It is hard to say which results are better without knowing the ground truth. However, it is clear that the discord algorithm needs proper input parameters to detect the correct anomalies, and some of these parameters are non-trivial to set, e.g. the number of nearest neighbors. We also tried other parameter settings for the discord algorithm, but the current setting provides the best results. SLADE-TS, on the other hand, learns these critical parameters automatically.

5.2.2 SLADE-MTS results

The detected results by SLADE-MTS are shown in Fig. 15. Three anomalous time series, shown in red color, have been correctly detected. In addition, with the selected features and the locating strategy introduced in Sect. 4.6, we found that the reasons for those time series to be considered anomalies are in dimensions d_2 and d_7. The corresponding anomalous subsequences are shown in the highlighted area in the top two plots in Fig. 15, and the zoom in plot of the highlighted parts are shown in the bottom two plots.

It is important to note that these results contain anomalous subsequences of different lengths. That is to say, our algorithm has the ability to detect how long the anomalies last in different dimensions. The PCA related method (Ahmed et al. 2012) requires the window length as input if we want to detect anomalous subsequences. To compare with PCA related method, we tried two window length, one is 300, another is 1500 (these are the exact lengths of the anomalous subsequences detected by SLADE-MTS). As a result, as shown in Fig. 16, at the window where SLADE-MTS detected anomalies,

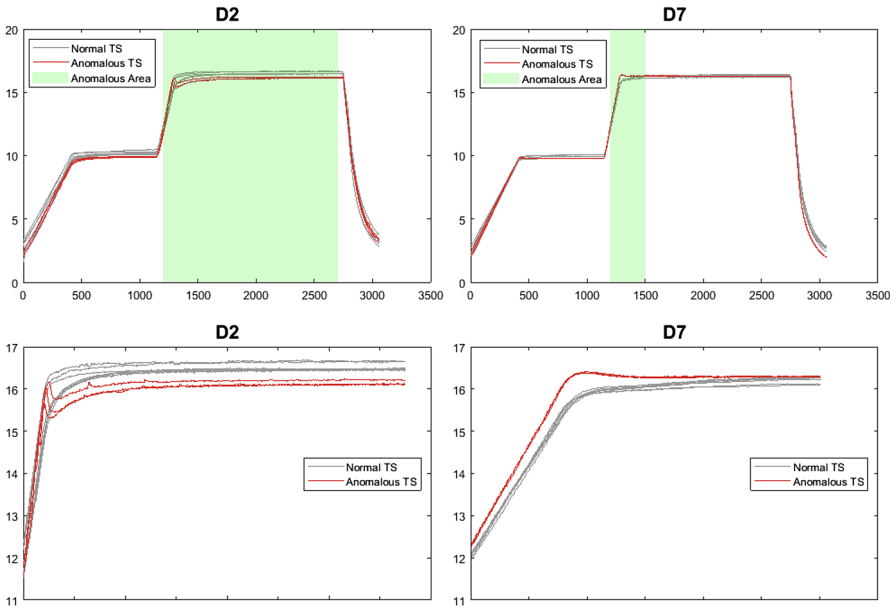

Fig. 15 Detected anomalies. Three anomalous time series, shown in red color, have been detected. Besides, we also located the anomalous subsequences in dimensions d_2 and d_7. The highlighted areas in the top show locations of the anomalous subsequences. The bottom two plots are the zoomed-in view of them

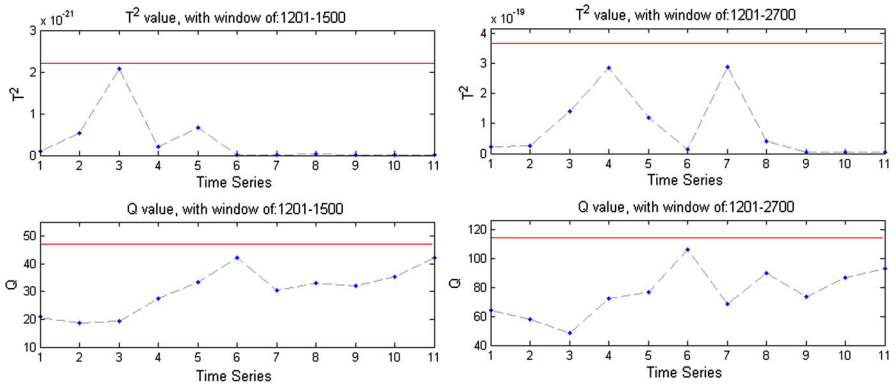

Fig. 16 PCA result on Intel Production Dataset 1. We use the same window size as the SLADE-MTS detected (300 in the left plots, and 1500 in the right plots). Each point represents a time series. The red lines are the threshold for the Hotelling's T-squared statistic (top plots), and the squared residuals Q (bottom plots). As all points are bellow the threshold, there is no anomaly detected (Color figure online)

the PCA method did not detect any anomalies with length 300 (left two plots) and 1500 (right two plots). This is because with both the Hotelling's T-squared statistic, and the squared residuals Q, the values of all time series are below the threshold.

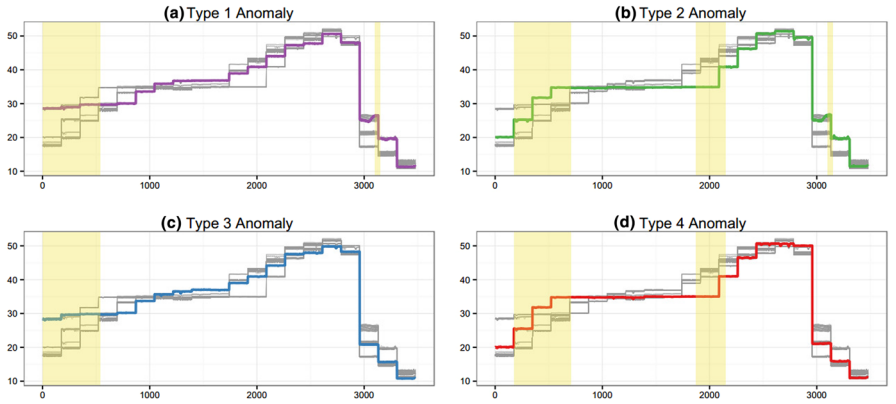

Fig. 17 4 types of anomalies detected by SLADE-TS. Highlighted sections are anomalous subsequences. Colored bold time series are examples of anomalous traces. Time series in gray are other traces (include normal and anomalous traces)

5.3 Intel production dataset 2

This data set comprises of temperature data off the monitor thermocouples in a 20 zone solder reflow oven. In order to ensure a good solder joint, it is imperative that the oven behaves in a repeatable fashion. As trays move through each oven zone, the temperature in that zone is perturbed and the oven PID controller attempts to compensate. By observing the transient behavior of this response, one can detect impeding issues that will impact joint quality. Specifically, the temperature profile has to ensure that the flux is correctly activated without oxidizing the paste and the solder is allowed to melt just enough to attach itself to the pads without damaging components on the package.

The 20 zone data set has 286 time series, each time series has 20 dimension with 174 length. Within the 286 time series, there are 245 normal ones, 41 anomalies. In the following paragraphs, we show the results of using both univariate anomaly detection method SLADE-TS and multivariate method SLADE-MTS on this dataset.

5.3.1 SLADE-TS result

To use univariate method on this dataset, we concatenate the data from different dimension of the same time series. After the concatenation, each time series is 3480 in length. The discretization parameters are: initial segment length $= 50$, $\alpha = 6$, PAA size $= 3$. We correctly detected all 41 anomalies. We also detected four different types of anomalies based on their anomaly scores and anomalous locations. As shown in Fig. 17, our algorithm detected 14 anomalous time series of type 1 (Fig. 17a), 9 of type 2 (Fig. 17b), 10 of type 3 (Fig. 17c) , and 8 of type 4 (Fig. 17d). By further separating the detected anomalies into different types, our algorithm can potentially help domain experts better understand the anomalies.

Besides the anomalous traces, our algorithm also found the anomalous sections within them, shown as the highlighted parts in Fig. 17. For type 1 anomaly, the anomaly

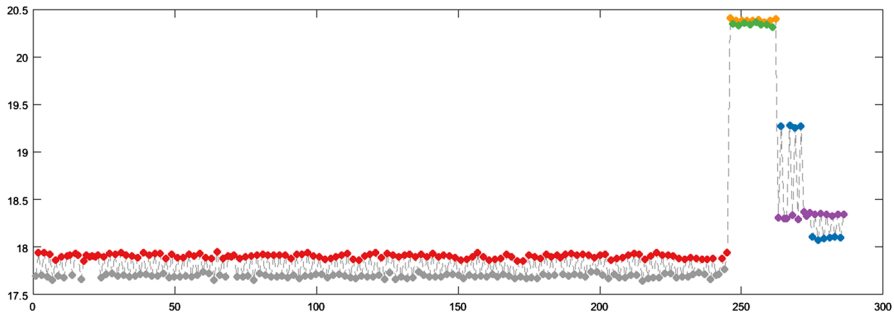

Fig. 18 Ground truth and clustering results of the 20-zone dataset. The x-axis represents the time series; each time series represents an object in the 20-zone dataset. The y-axis represents statistical values computed from proprietary formulae used in Intel to characterize the objects. Intuitively, similar values suggest similar behaviors. Different colors represent different clusters detected by our multivariate algorithm (Color figure online)

locations are 1–535, 3103–3153. As we can see, even for the normal time series, there are some variations. However, these variations are within the normal range. One advantage of our algorithm is that we do not have to set the normal range; the algorithm will find the range automatically. For example, in Fig. 17a, even in the un-highlighted parts, there are still variations between the red anomalous time series and normal time series. Our algorithm is able to correctly differentiates them from the true anomalies.

If we use a clustering algorithm directly on the whole time series, we can also find these 41 anomalies. However, it requires proper setting of parameters, such as the number of clusters or the density threshold of each cluster. With our algorithm, this extra step can be completely omitted. In addition, clustering algorithm is not able to provide the location information of the anomalies, while our algorithm can accurately identify the exact time the anomalies occur and their duration (length). When in real production process, our algorithm can accurately detect when an abnormal behavior happens and how long it lasts. Such additional information will be highly valuable in identifying the production problems and looking for solutions.

5.3.2 SLADE-MTS result

Our multivariate anomaly detection algorithm successfully detected all anomalous and normal time series. In addition, it also detected 6 clusters from the data. Two clusters are normal and 4 clusters are anomalies. This means our algorithm not only identifies anomalies from normal data, but also can tell the differences between different types of normal and anomalous patterns.

Figure 18 contains the ground truth of the 286 time series. The x-axis represents the time series; each time series represents an object in the 20-zone dataset. The y-axis represents statistical values computed from proprietary formulae used in Intel to characterize the objects. Intuitively, similar values suggest similar behaviors. Different colors represent different clusters detected by our multivariate algorithm.

In addition to detecting anomalous time series and differentiating various types of anomalous, as well as normal behaviors, our algorithm also finds the reason for the differences between such behaviors. For example, with the selected features shown in

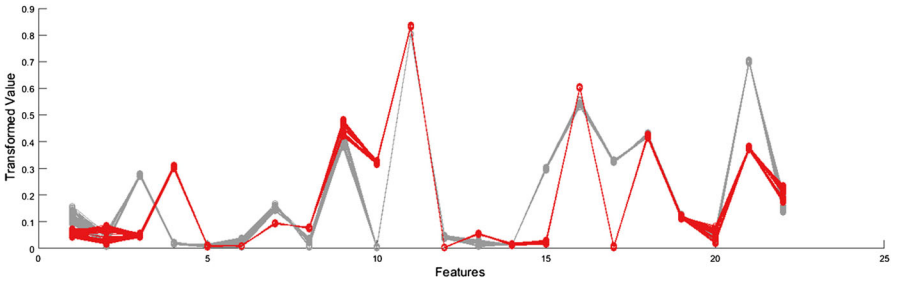

Fig. 19 Difference between red and gray cluster in transformed feature space (Color figure online)

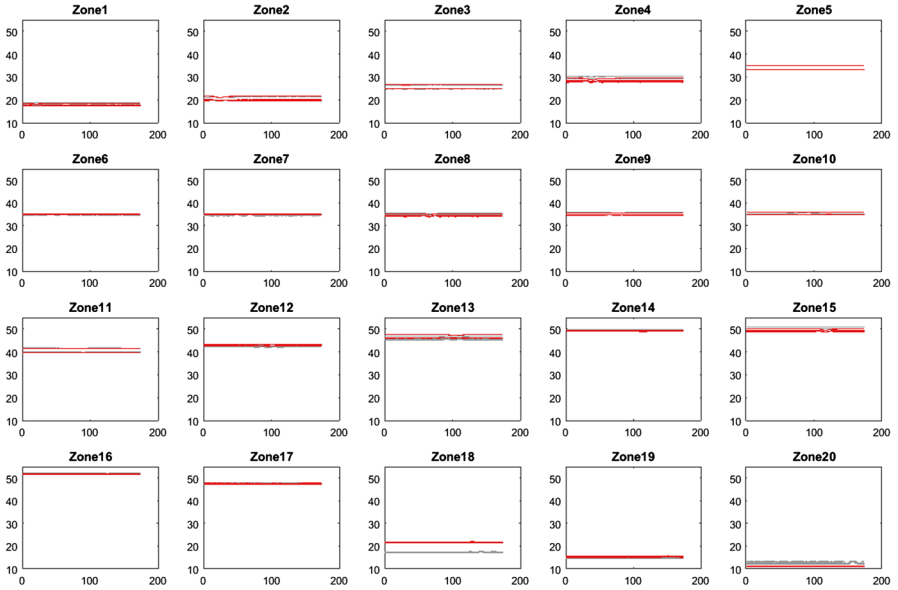

Fig. 20 Difference between red and gray cluster in original space. The the red and gray time series look similar in almost all zones but zone 18 (Color figure online)

Fig. 19 and the analyzing method introduced in Sect. 4.6, we find that the differences between the red and gray cluster are from features 3, 4, 10, 15, 17 and 21. By mapping them back to their original time series, we find that all of them come from zone 18. If we look at Fig. 20, we can see that in all other zones except 18, the red and gray time series look similar, but they have obvious differences in zone 18.

With this advantage of our algorithm, we can also find the problems of anomalous time series by comparing them with normal cluster. Figure 21 shows the transformed data of orange and gray time series. With the strategies introduced in Sect. 4.6, we can find which features cause the differences between these two clusters. If we map them back to their location of the original time series, we can find that they are different in zones 1, 2, 12, 13 and 18, as shown in Fig. 22.

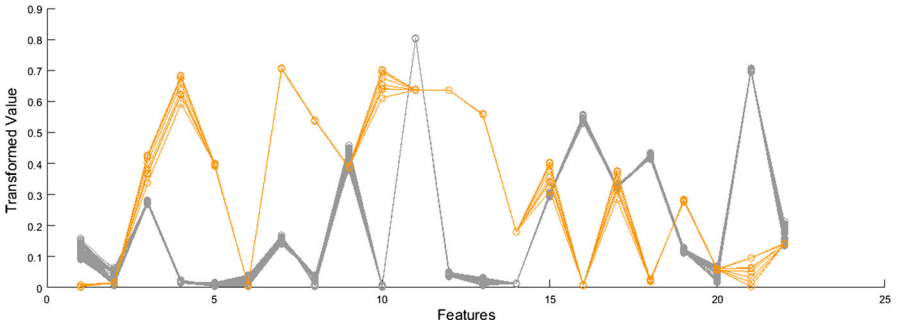

Fig. 21 Time series data from orange and gray cluster in the transformed feature space with the selected features (Color figure online)

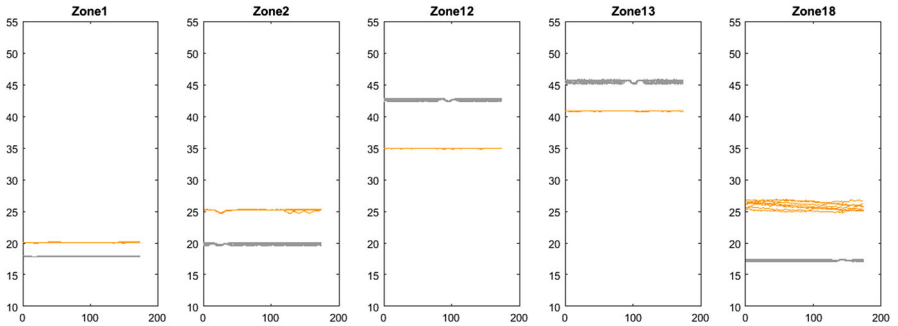

Fig. 22 Difference between orange and gray cluster in the detected dimensions of original time series (Color figure online)

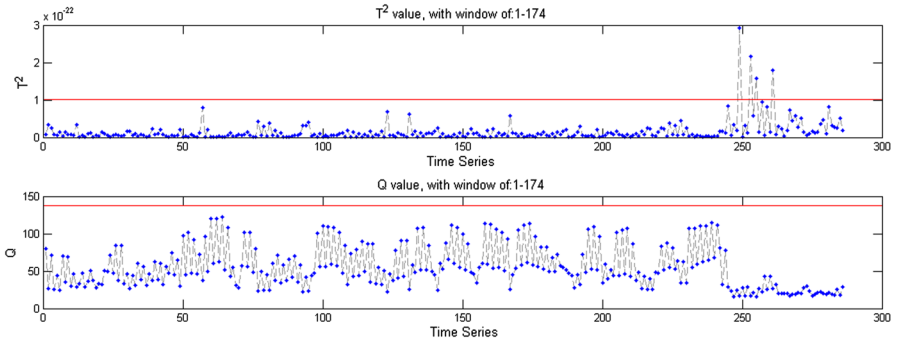

Fig. 23 PCA result on Intel Production Dataset 2. Only 4 out of 41 anomalies has been detected (their T^2 values above the threshold)

As a comparison, with PCA related method, the result is shown in Fig. 23. It detected four anomalous time series out of 41 real anomalies. Also, it cannot tell which dimension(s) the anomalies occur.

As we demonstrate so far, our algorithm is able to: 1. detect anomalous time series; 2. Find clusters of time series which have the same (normal or abnormal) behaviors;

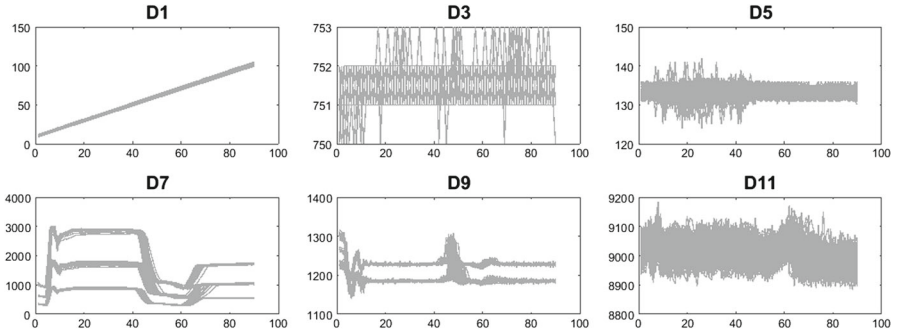

Fig. 24 Etch Data. How the data looks like in some selected dimensions. In different dimensions, the behaviors of time series are quite different. In dimension $D1$, all time series look similar, in dimension $D7$, there are three groups, in dimension $D9$, there are two groups, and the time series are noisy in the rest three dimensions

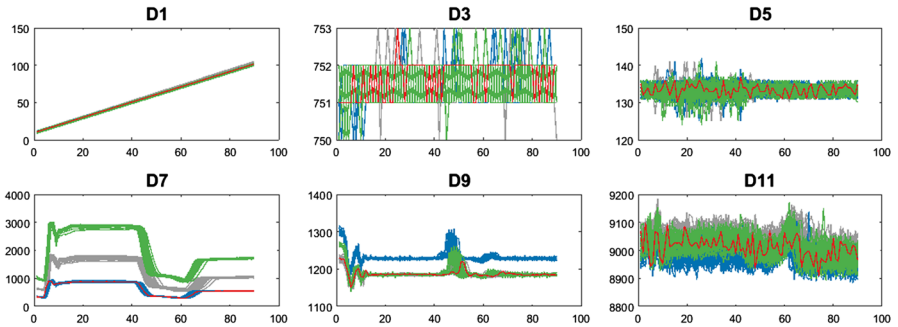

Fig. 25 Detected result in etch data. The red time series is the detected anomaly which is exactly the injected time series, and the gray, green and blue are the detected three normal clusters (Color figure online)

3. Find the reason of the differences between time series (which sensors caused the differences, and which subsequences in those sensors).

5.4 Etch data

For multivariate time series data, in some cases, if we look at the time series in each dimension individually, there is no anomaly. However, if we consider the correlations between dimensions, we may be able to find anomalies (see example shown in Fig. 6).

In this section, we show that our algorithm is able to detect the correlation anomalies. The data set we use is from eigenvector website.[3] The data consists of the engineering variables from a LAM 9600 Metal Etcher over the course of etching wafers. It contains 108 normal wafers taken during 3 experiments. That means there are three groups of normal data. There are 21 variables, and the length of the time series is 90. In order to detect correlation anomalies in this dataset, we injected one anomaly with part of its

[3] http://www.eigenvector.com/data/Etch/index.html.

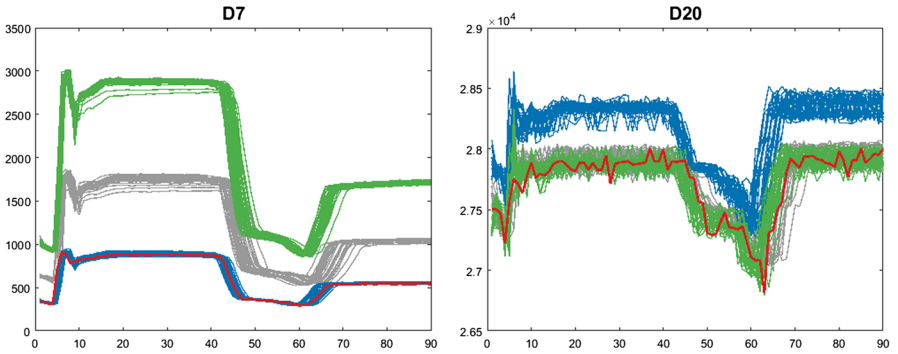

Fig. 26 The dimensions of the detected anomalous time series where caused it to be different with other normal time series

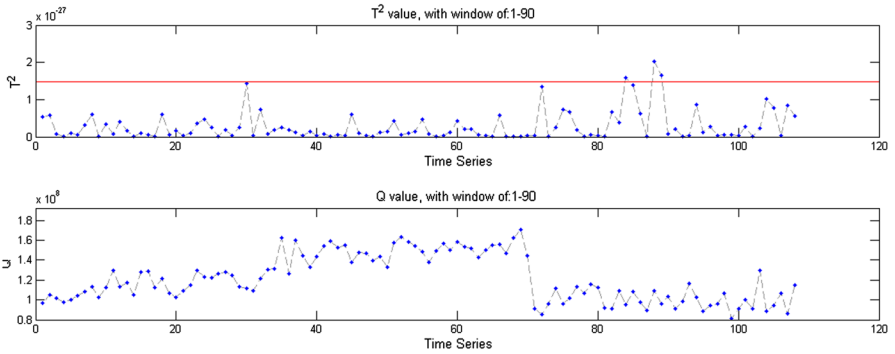

Fig. 27 PCA result on Etch data. Each point represents a time series. In the top plot, there are three time series have T^2 values above the threshold, that means they are anomalies. But none of them are the injected anomaly

data from one group of normal and another part from another group. Figure 24 shows what the data looks like in some of its dimensions.

As a result, our algorithm successfully detected the injected anomaly, and also detected the 3 normal groups. The detected result is shown in Fig. 25, the red time series is the detected anomaly, and the gray, green and blue are the three normal groups. Also, with the locating strategy from Sect. 4.6, we found that the differences of normal groups are mostly in dimension 7. In addition, the detected anomaly shows anomalous combination behavior in dimension 7 and 20. As shown in Fig. 26, the red time series has the behavior of blue group in dimension 7 but it has the behavior of green group in dimension 20.

As a comparison, shown in Fig. 27, the PCA-based algorithm finds three anomalies (Their T-squared values are above the computed threshold), but none of them is a true anomaly.

6 Conclusion and future work

In this paper, we introduce self-learning online anomaly detection algorithms for univariate and multivariate time series data. Our SLADE-TS algorithm detects anomalous time series automatically without any prior knowledge of the data. In addition, it can pinpoint the locations of anomalous subsequences within a time series. Also, it finds the lengths and the number of anomalous subsequences automatically. For multivariate time series, our SLADE-MTS algorithm detects anomalous time series even if the anomalies are from a few subsequences in a subset of dimensions. It is also capable of detecting correlation anomalies even if they look normal in each dimension individually. In addition, it accurately identifies the dimensions and locations of the anomalous subsequences.

One advantage of our algorithm is that it requires minimal configuration. However, for different datasets, the notion of anomaly may differ. In future work, we will extend our algorithm to use the feedback from users to customize the algorithm for different datasets. The distance measure used in this paper is Euclidean Distance (ED). ED is fast, but it requires the time series to be well aligned. For future work, we will incorporate other distance measure such as Dynamic Time Warping to make our algorithm more robust.

References

Aggarwal CC, Yu PS (2010) On clustering massive text and categorical data streams. Knowl Inf Syst 24(2):171–196

Ahmed M, Baqqar M, Gu F, Ball AD (2012) Fault detection and diagnosis using principal component analysis of vibration data from a reciprocating compressor. In: Proceedings of 2012 UKACC international conference on control, pp 461–466

Baragona R, Battaglia F (2007) Outliers detection in multivariate time series by independent component analysis. Neural Comput 19(7):1962–1984

Begum N, Ulanova L, Wang J, Keogh E (2015) Accelerating dynamic time warping clustering with a novel admissible pruning strategy. In: Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining. ACM, New York, KDD '15, pp 49–58

Budalakoti S, Srivastava AN, Akella R, Turkov E (2006) Anomaly detection in large sets of high-dimensional symbol sequences. Tech Rep

Cai D, Zhang C, He X (2010) Unsupervised feature selection for multi-cluster data. In: Proceedings of the 16th ACM SIGKDD international conference on knowledge discovery and data mining. ACM, New York, KDD '10, pp 333–342

Chandola V, Cheboli D, Kumar V (2009) Detecting anomalies in a time series database. University of Minnesota, Tech Rep, Computer Science Department

Cheng H, Tan PN, Potter C, Klooster S (2009) Detection and characterization of anomalies in multivariate time series. In: Proceedings of the 2009 SIAM international conference on data mining, pp 413–424

Ester M, Kriegel HP, Sander J, Xu X (1996) A density-based algorithm for discovering clusters a density-based algorithm for discovering clusters in large spatial databases with noise. In: Proceedings of the second international conference on knowledge discovery and data mining, AAAI Press, KDD'96, pp 226–231

Galeano P, Pea D, Tsay RS (2006) Outlier detection in multivariate time series by projection pursuit. J Am Stat Assoc 101(474):654–669

Gupta M, Gao J, Aggarwal C, Han J (2014) Outlier detection for temporal data. Synth Lect Data Min Knowl Discov 5(1):1–129

Hawkins DM (1980) Identification of outliers, vol 11. Springer, Dordrecht

He Z, Xu X, Deng S (2003) Discovering cluster-based local outliers. Pattern Recogn Lett 24(9):1641–1650

Hyndman RJ, Wang E, Laptev N (2015) Large-scale unusual time series detection. In: 2015 IEEE international conference on data mining workshop (ICDMW), pp 1616–1619
Id T, Papadimitriou S, Vlachos M (2007) Computing correlation anomaly scores using stochastic nearest neighbors. In: Seventh IEEE international conference on data mining (ICDM 2007), pp 523–528
Izakian H, Pedrycz W (2014) Anomaly detection and characterization in spatial time series data: a cluster-centric approach. IEEE Trans Fuzzy Syst 22(6):1612–1624
Jaccard P (1912) The distribution of the flora in the alpine zone. New Phytol 11(2):37–50
Keogh E, Lin J (2005) Clustering of time-series subsequences is meaningless: implications for previous and future research. Knowl Inf Syst 8(2):154–177
Keogh E, Chakrabarti K, Pazzani M, Mehrotra S (2001) Dimensionality reduction for fast similarity search in large time series databases. Knowl Inf Syst 3(3):263–286
Keogh E, Lin J, Fu A (2005) Hot sax: efficiently finding the most unusual time series subsequence. In: Fifth IEEE international conference on data mining (ICDM'05), pp 8
Keogh E, Lin J, Lee SH, Herle HV (2007) Finding the most unusual time series subsequence: algorithms and applications. Knowl Inf Syst 11(1):1–27
Laptev N, Amizadeh S, Flint I (2015) Generic and scalable framework for automated time-series anomaly detection. In: Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining. ACM, New York, KDD '15, pp 1939–1947
Li Y, Lin J, Oates T (2012) Visualizing variable-length time series motifs. In: Proceedings of the 2012 SIAM international conference on data mining, pp 895–906
Li J, Pedrycz W, Jamal I (2017) Multivariate time series anomaly detection: a framework of hidden markov models. Appl Soft Comput 60(Supplement C):229–240
Lin J, Keogh E, Wei L, Lonardi S (2007) Experiencing sax: a novel symbolic representation of time series. Data Min Knowl Disc 15(2):107–144
Miljkovi D (2011) Fault detection methods: a literature survey. In: 2011 Proceedings of the 34th international convention MIPRO, pp 750–755
Nevill-Manning CG, Witten IH (1997) Identifying hierarchical structure in sequences: a linear-time algorithm. J Artif Intell Res 7(1):67–82
Pires AM, Santos-Pereira C (2005) Using clustering and robust estimators to detect outliers in multivariate data. In: Proceedings of the international conference on robust statistics
Pukelsheim F (1994) The three sigma rule. Am Stat 48(2):88–91
Qiu H, Liu Y, Subrahmanya NA, Li W (2012) Granger causality for time-series anomaly detection. In: 2012 IEEE 12th international conference on data mining, pp 1074–1079
Senin P, Lin J, Wang X, Oates T, Gandhi S, Boedihardjo AP, Chen C, Frankenstein S, Lerner M (2014) Grammarviz 2.0: a tool for grammar-based pattern discovery in time series. In: Calders T, Esposito F, Hüllermeier E, Meo R (eds) Machine learning and knowledge discovery in databases: European conference, ECML PKDD 2014, Nancy, France, September 15-19, 2014. Proceedings, Part III. Springer, Berlin pp 468–472
Senin P, Lin J, Wang X, Oates T, Gandhi S, Boedihardjo AP, Chen C, Frankenstein S (2015) Time series anomaly discovery with grammar-based compression. In: Proceedings of the 18th international conference on extending database technology, EDBT 2015, Brussels, Belgium, March 23–27, 2015, pp 481–492
Sequeira K, Zaki M (2002) Admit: anomaly-based data mining for intrusions. In: Proceedings of the Eighth ACM SIGKDD international conference on knowledge discovery and data mining. ACM, New York, KDD '02, pp 386–395
Sun H, Bao Y, Zhao F, Yu G, Wang D (2004) Cd-trees: an efficient index structure for outlier detection. In: Li Q, Wang G, Feng L (eds) Advances in web-age information management: 5th international conference, WAIM 2004, Dalian, China, July 15–17, 2004. Springer, Berlin, pp 600–609
Wang H, Tang M, Park Y, Priebe CE (2014) Locality statistics for anomaly detection in time series of graphs. IEEE Trans Signal Process 62(3):703–717
Wang X, Gao Y, Lin J, Rangwala H, Mittu R (2015) A machine learning approach to false alarm detection for critical arrhythmia alarms. In: 2015 IEEE 14th international conference on machine learning and applications (ICMLA), pp 202–207
Wang X, Lin J, Patel N, Braun M (2016) A self-learning and online algorithm for time series anomaly detection, with application in cpu manufacturing. In: Proceedings of the 25th ACM international on conference on information and knowledge management, ACM, New York, CIKM '16, pp 1823–1832

Wei L, Keogh E, Xi X (2006) Saxually explicit images: Finding unusual shapes. In: Sixth international conference on data mining (ICDM'06), pp 711–720

Xie Y, Huang J, Willett R (2013) Change-point detection for high-dimensional time series with missing data. IEEE J Sel Top Signal Process 7(1):12–27

Zhang Y, Meratnia N, Havinga P (2010) Outlier detection techniques for wireless sensor networks: a survey. IEEE Commun Surv Tutor 12(2):159–170