Pattern-Based Anomaly Detection in Mixed-Type Time Series

Len Feremans*[1], Vincent Vercruyssen*[2], Boris Cule[1], Wannes Meert[2], and Bart Goethals[1,3]

[1] Department of Mathematics and Computer Science, University of Antwerp, Belgium
{firstname.lastname}@uantwerpen.be
[2] Department of Computer Science, KU Leuven, Belgium
{firstname.lastname}@cs.kuleuven.be
[3] Monash University, Melbourne, Australia

Abstract. The present-day accessibility of technology enables easy logging of both sensor values and event logs over extended periods. In this context, detecting abnormal segments in time series data has become an important data mining task. Existing work on anomaly detection focuses either on continuous time series or discrete event logs and not on the combination. However, in many practical applications, the patterns extracted from the event log can reveal contextual and operational conditions of a device that must be taken into account when predicting anomalies in the continuous time series. This paper proposes an anomaly detection method that can handle mixed-type time series. The method leverages frequent pattern mining techniques to construct an embedding of mixed-type time series on which an isolation forest is trained. Experiments on several real-world univariate and multivariate time series, as well as a synthetic mixed-type time series, show that our anomaly detection algorithm outperforms state-of-the-art anomaly detection techniques such as MatrixProfile, Pav, Miffod and Fpof.

Keywords: Anomaly detection, time series, distance measure, pattern-based embedding, frequent pattern mining

1 Introduction

Anomaly detection in time-series is an important real-world problem, especially as an increasing amount of data of human behaviour and a myriad of devices is collected, with an increasing impact on our everyday lives. We live in an "Internet of Things" world, where a network of devices, vehicles, home appliances and other items embedded with software and electronics are connected and exchanging data [9]. Although many organisations are collecting time series data, automatically analysing them and extracting meaningful knowledge, such as an

* These authors contributed equally to the work.

understandable model that automatically flags relevant anomalies, remains a difficult problem, even after decades of research.

Exploring different benchmark datasets for time series anomaly detection, we found that these datasets often consist of univariate time series, where anomalies are local or global extrema or *point anomalies* [2]. In contrast, we focus on *collective and contextual anomalies*: a collection of points in the time series is anomalous depending on the surrounding context. For instance, most smartphones log many *continuous* times series from various sensors, such as the accelerometer, gyroscope, internal thermometer, and battery level. In addition, smartphones log *discrete* events in different operating system logs, such as applications starting or stopping, certain hardware components being turned on or off, or application-specific events. Such events are crucial in determining whether the behaviour observed in the continuous time series, e.g., a spike in power usage, is anomalous. We argue that for many real-world applications one needs to extract information from both types of sources to successfully detect anomalies.

In the active research area for anomaly detection in continuous time series, much attention has been given to finding anomalies using continuous *n-grams*, *dynamic time warping* distance, and similarity to the nearest neighbors [15,18], but not to integrating event log data. On the other hand, pattern mining based techniques for detecting anomalies have been developed for discrete event logs, but not for continuous time series. In this paper, we propose how to circumvent the apparent mismatch between discrete patterns and continuous time series data. We introduce a pattern-based anomaly detection method that can detect anomalies in *mixed-type time series*, i.e., time series consisting of both continuous sensor values and discrete event logs.

Given a time series dataset, the method leverages the mature field of *frequent pattern mining* research [19] to find frequent patterns in the data, serving as a template for the frequently occurring normal behaviour. Then, the frequent patterns are used to map the time series data to a feature-vector representation. This newly found pattern-based embedding of the data combines the information in both the continuous time series and the discrete event logs into a single feature vector. Finally, a state-of-the-art anomaly detection algorithm is used to find the anomalies in the embedded space.

The remainder of this paper is organised as follows. In Section 2, we introduce the necessary preliminaries. In Section 3, we provide a detailed description of our method for pattern mining, feature representation, and anomaly detection. In Section 4, we present an experimental evaluation of our method and compare with state-of-the-art methods. We present an overview of related work in Section 5 and conclude our work in Section 6.

2 Preliminaries

2.1 Time series data

A *continuous time series* is defined as a sequence of real-valued measurements $(\langle x_1, t_1\rangle, \ldots, \langle x_n, t_n\rangle)$, where $x_k \in \mathbb{R}$ and each measurement has a distinct times-

tamp t_k. Although this is not required, we will assume that the continuous time series are sampled regularly, that is $t_{i+1} - t_i$ is constant, and do not contain missing values. A *discrete event log* is a sequence of discrete events $(\langle e_1, t_1 \rangle, \ldots, \langle e_n, t_n \rangle)$ where $e_k \in \Sigma$, with Σ a finite domain of discrete event types. Unlike continuous time series, we assume that multiple events can co-occur at the same timestamp, i.e. $t_i \leqslant t_{i+1}$, and that events can occur sparsely.

In this paper, we consider a *mixed-type time series* $\mathbf{S}$. This is a collection of N continuous time series and M event logs. Thus, $\mathbf{S}$ has $M + N$ dimensions. Typical time series representations are special cases of this: when $N = 1$ and $M = 0$ it is a *univariate time series*; and when $N > 1$ and $M = 0$ it is a *multivariate time series*. A single time series in $\mathbf{S}$ is denoted as S^i and has only one dimension.

A *time series window* $S^i_{t,l}$ is a contiguous subsequence of a time series S^i and contains all measurements for which $\{\langle x_i, t_i \rangle$ or $\langle e_i, t_i \rangle | t \leqslant t_i < t + l\}$. Additionally, we can define a window over all dimensions of $\mathbf{S}$ simultaneously and thus use the same values for timestamp t and length l for all series in $\mathbf{S}$, regardless of whether they are continuous time series or discrete events. In this work, we use *fixed-sized sliding windows*. This means choosing a fixed l given $\mathbf{S}$ (e.g., 1 hour, 5 minutes, ...) and iteratively incrementing t with a fixed value.

2.2 Pattern mining

We provide the following definitions for frequent pattern mining [19], adapted to the context of mixed-type time series.

The first type of pattern we consider is an *itemset*. For an itemset, *no temporal order* between items is required. An itemset X consists of one or more items $x_j \in \Omega$, where Ω is a finite domain of discrete values, that is, $X = \{x_1, \ldots, x_m\} \subseteq 2^{|\Omega|}$. An itemset X *occurs* in, or is covered by, a window $S^i_{t,l}$ if all items in X occur in that window in any order, that is,

$$X \prec S^i_{t,l} \Leftrightarrow \forall x_j \in X : \exists \langle x_j, t_j \rangle \in S^i_{t,l}.$$

Given the set of all windows $\mathcal{S}$ of a time series, we define *cover* and *support* as

$$cover(X, \mathcal{S}) = \{S^i_{t,l} | S^i_{t,l} \in \mathcal{S} \wedge X \prec S^i_{t,l}\} \text{ and } support(X, \mathcal{S}) = |cover(X, \mathcal{S})|.$$

The second type of pattern we consider is a *sequential pattern*. A sequential pattern X_s consists of an ordered list of one or more items, denoted as $X_s = (x_1, \ldots, x_m)$, where $x_j \in \Omega$. A sequential pattern can contain repeating items, and, unlike n-grams, an occurrence of a sequential pattern allows *gaps* between items. We define that a sequential pattern X_s *occurs* in a window $S^i_{t,l}$ using

$$X_s \prec S^i_{t,l} \Leftrightarrow \exists \langle x_1, t_1 \rangle, \ldots, \langle x_p, t_m \rangle \in S^i_{t,l} : \forall i, j \in \{1, \ldots, p\} : i < j \Rightarrow t_i < t_j.$$

The definitions of cover and support are equivalent to those of itemsets. Finally, an itemset or a sequential pattern is *frequent* if its support is higher than a user-defined threshold on *minimal support* (parameter *min_sup*).

Given a set of windows $\mathcal{S}$ and *discretised* continuous values, we can use existing itemset or sequential pattern mining algorithms [19,8] to efficiently mine all *frequent* patterns in both continuous and discrete time series. However, even with the restriction that patterns must occur at least *min_sup* times, it remains a challenge to filter out *redundant* patterns. We will focus on algorithms that mine only *closed* or *maximal* patterns. An itemset X is not closed, and thus redundant, if and only if there exists an itemset Z, such that $X \subset Z$ and $support(X, \mathcal{S}) = support(Z, \mathcal{S})$. Likewise, a sequential pattern X_s is not closed if there exists a sequential pattern Z_s, such that $X_s \sqsubset Z_s$ and $support(X_s, \mathcal{S}) = support(Z_s, \mathcal{S})$, where $\sqsubset$ is used to denote the subsequence relation.

3 Pattern-based anomaly detection

3.1 Problem setting

The problem we are trying to solve can be defined as:

Given: A univariate, multivariate, or mixed-type time series **S**.
Do: Identify periods of abnormal behaviour in **S**.

In this section we outline our proposed *pattern-based anomaly detection* method (PBAD) that computes for each time series window of **S** an anomaly score. The method has four major steps. First, the time series is preprocessed. Second, frequent itemsets and sequential patterns are mined from the individual time series or event logs $S^i \in \mathbf{S}$. Third, the distance-weighted similarity scores between the frequent patterns and each time series window are computed to construct a pattern-based embedding of time series **S**. Fourth, the embedding is used to construct an anomaly detection classifier to detect the anomalous periods of **S**. We use the ISOLATIONFOREST classifier. These steps are illustrated in Figure 1. In the following paragraphs, we outline each step in more detail.

3.2 Preprocessing

Preprocessing is the first step in PBAD shown in Algorithm 1, line 1-7. First, we *normalise* each continuous time series S^i in **S** to values between 0 and 1, because multivariate time series can have widely different amplitudes.

Then, we segment each time series in **S** using *fixed-size sliding windows* of length l. Frequent patterns are often limited in length, i.e., a length of 5 is already quite high. Therefore, it can be useful for certain datasets to also reduce the length of each window such that a frequent sequential pattern or itemset covers a large proportion of each window. For example, we can reduce a continuous window of length 100, using a *moving average*, by storing the average value for every 5 consecutive values. The resulting window now has a length of 20 and subsequently any matching sequential pattern or itemset of size 5 will cover a large part of the window.

Frequent pattern mining only works on *discretised* data. For a continuous time series, each window $S^i_{t,l} = (x_1, \ldots, x_l)$ must be discretised using a function h

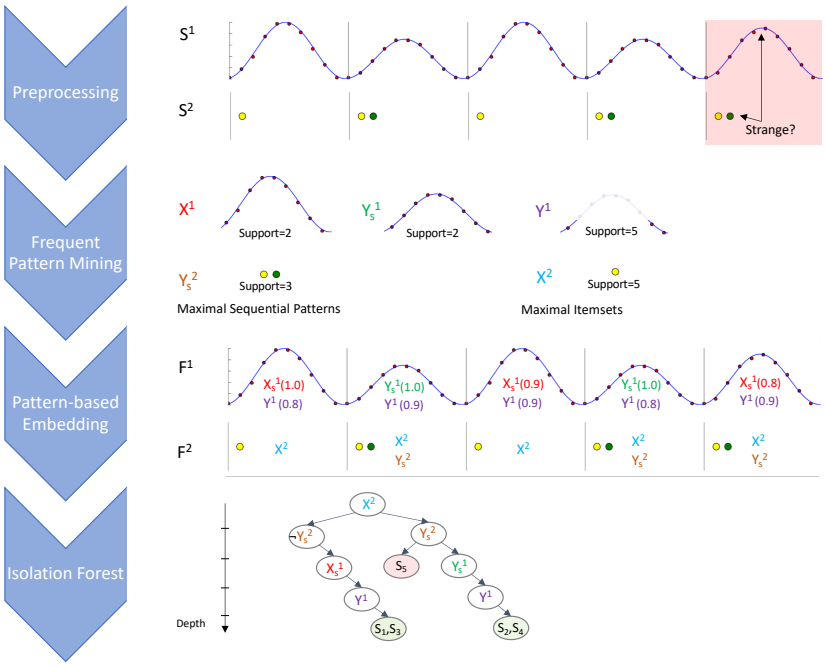

Fig. 1: Illustration of major steps in pattern-based anomaly detection method in mixed-type time series. Note that the 1^{st} and 3^{th} window, and the 2^{nd} and 4^{th} windows match the same set of patterns extracted from both the time series and the event log. The 5^{th} window is anomalous because of the co-occurrence of a peak with a green event. The isolation forest step in PBAD marks it as such since its depth is only 2.

to yield a discrete representation $h(S^i_{t,l}) = (x_1', \ldots, x_l')$ where $x_j' \in \Omega$. Examples of such functions include equal-width or equal-frequency *binning*, or aggregating windows and computing a symbol using Symbolic Aggregate Approximation [13].

3.3 Extracting frequent patterns

After preprocessing, PBAD mines frequent itemsets and sequential patterns from time series $\mathbf{S}$. Given the assumption that anomalous behaviour occurs infrequently in the time series, the frequent patterns would characterise the frequently observed, normal behaviour. To extract the frequent patterns, we leverage the mature field of *frequent pattern mining*. This has two main advantages. First, the existing techniques can be extended to mine patterns in different types of time series data. Second, the mined patterns are easily interpretable and can later be presented to the user to give intuitions as to why the classifier labeled a segment as normal or anomalous.

Extracting frequent itemsets and sequential patterns. After preprocessing, we have created a set of windows for each series. These can trivially be

Algorithm 1: PBAD($\mathbf{S}$, l, *min_sup*, *is_maximal*, *max_sim*) Anomaly detection using late integration

Input : A time series $\mathbf{S}$, window length l, support threshold *min_sup*, *is_maximal* is either `maximal` or `closed`, threshold on max Jaccard similarity *max_sim*

Result: Anomaly scores for each window $S_{t,l}$

`// 1. Preprocessing: create windows and discretise`
$\mathcal{S} \leftarrow \emptyset$
foreach $S^i \in \mathbf{S}$ **do**
  if S^i *is continuous* **then**
    $T^i \leftarrow$ CREATE_WINDOWS(DISCRETISE(NORMALISE(S^i)), l)
  else
    $T^i \leftarrow$ CREATE_WINDOWS(S^i, l)
  $\mathcal{S} \leftarrow \mathcal{S} \cup T^i$
`// 2. Mine maximal/closed frequent itemsets and sequential patterns`
$\mathcal{P} \leftarrow \emptyset$
foreach $T^i \in \mathcal{S}$ **do**
  $\mathcal{P}^i \leftarrow$ MINE_FREQUENT_ITEMSETS(T^i, *min_sup*, *is_maximal*)
  $\mathcal{P}^i \leftarrow \mathcal{P}^i \cup$ MINE_FREQUENT_SEQUENTIAL_PATTERNS(T^i, *min_sup*, *is_maximal*)
  `// Remove redundant patterns using Jaccard`
  $\mathcal{P}^i \leftarrow$ SORT $\mathcal{P}^i$ on descending support
  for $1 \leqslant i < |\mathcal{P}^i|$ **do**
    for $i + 1 \leqslant j \leqslant |\mathcal{P}^i|$ **do**
      if $J(X_i, X_j) \geqslant max\_sim$ **then**
        $\mathcal{P}^i \leftarrow \mathcal{P}^i \backslash X_j$
  $\mathcal{P} \leftarrow \mathcal{P} \cup \mathcal{P}^i$
`// 3. Compute pattern based embedding`
$\mathbf{F} \leftarrow$ matrix of 0.0 values with $|\mathcal{P}|$ columns and $|\mathcal{S}|$ rows for each window $S_{t,l}$
for $1 \leqslant i \leqslant |\mathbf{S}|$ **do**
  for $1 \leqslant j \leqslant |\mathcal{P}^i|$ **do**
    $idx \leftarrow$ global index of X^i_j in $\mathcal{P}$
    for $1 \leqslant t \leqslant |\mathcal{S}|$ **do**
      `// Weighted similarity between pattern` X^i_j `and window` $S^i_{t,l}$ `in dimension` i `for time series`
      $\mathbf{F}_{k,idx} \leftarrow 1.0 - \frac{\text{EXACTMINDIST}(X^i_j, S^i_{t,l})}{|X^i_j|}$
`// 4. Compute anomalies using Isolation Forest`
scores $\leftarrow$ ISOLATION_FOREST($\mathcal{S}, \mathbf{F}$)
scores $\leftarrow$ SORT *scores* descending

transformed to a transaction (or sequence) database, required by existing frequent pattern mining algorithms [19]. Said algorithms generate candidate patterns with growing length. Since support decreases with the length of either the itemset or sequential pattern, it is relatively straightforward for these algorithms to only enumerate patterns that are frequent by pruning the candidate patterns on *min_sup*. We always mine both itemsets and sequential patterns, but filter either on *closed* or *maximal* patterns depending on the status of the parameter

is_maximal (line 9-11). The implementation of both closed and maximal itemsets and sequential pattern mining algorithms is available in the SPMF library [8].

Removing overlapping patterns. Frequent pattern mining algorithms can generate too many redundant patterns. To further reduce the set of patterns, we employ *Jaccard similarity* to remove itemsets and sequential patterns that co-occur in a large percentage of windows. Formally, we use a parameter *max_sim*, and remove all patterns with a high enough Jaccard similarity:

$$J(X_1, X_2) = \frac{|cover(X_1) \cap cover(X_2)|}{support(X_1) + support(X_2) - |cover(X_1) \cap cover(X_2)|}$$

If $J(X_1, X_2) \geqslant max\_sim$, we remove the pattern with the lowest support. We created a straightforward routine that compares all pattern pairs (line 12-17).

Dealing with multivariate and mixed-type time series. For multivariate and mixed-type time series, we consider two strategies for pattern extraction: *early* and *late integration*. Under the *early integration* strategy, the items of all preprocessed series in $\mathbf{S}$ are combined into a single event sequence. The frequent patterns are then mined over this new event sequence, resulting in patterns containing values from multiple dimensions. For example, the windows $S^1_{1,4} = (1, 2, 3, 4)$ and $S^2_{1,4} = (10, 10, 11, 11)$ spanning the same period in two time series S^1 and S^2, can be combined into a single event sequence $E_{1,4} = (\{1^1, 10^2\}, \{2^1, 10^2\}, \{3^1, 11^2\}, \{4^1, 11^2\})$. Frequent patterns can now be mined in this single event sequence, yielding candidate patterns such as the sequential pattern $X_s = (1^1, 2^1, 11^2, 11^2)$, meaning that value 1 followed by 2 in series S^1 is followed by 2 occurrences of 11 in series S^2.

In contrast, the *late integration* strategy mines patterns in each time series of $\mathbf{S}$ separately and takes the union of the resulting set of patterns. Now, each pattern is associated with exactly one time series. While it would be tempting to conclude that early integration is better since it can uncover patterns containing events from different dimensions as well as any order between these events, we prefer *late integration* in our experiments for two reasons. First, in practice, early integration leads to an exponential increase in the search space of possible patterns, i.e., *pattern explosion*, since the pattern mining algorithms consider every possible combination of values in each of the time series. Second, the anomaly detection classifier in PBAD is constructed on the union of pattern-based embeddings of each time series in $\mathbf{S}$. As such, it learns the structure between patterns from the separate time series.

3.4 Constructing the pattern-based embedding

Having obtained a set of patterns for the time series in $\mathbf{S}$, PBAD maps $\mathbf{S}$ to a pattern-based embedding in two steps. First, it computes a similarity score between each window $S^i_{t,l}$ and each pattern X^i mined from the corresponding time series S^i in $\mathbf{S}$. If S^i is continuous, PBAD computes a *distance-weighted similarity score*. If S^i is an event log, PBAD computes the exact match, i.e. 1 if

$X^i \prec S^i_{t,l}$ and 0 otherwise. Second, it concatenates the similarity scores over all dimensions, yielding the feature-vector representation of the window of $\mathbf{S}$. Since this process is repeated for each window in $\mathbf{S}$, we end up with a pattern-based embedding of the full time series (line 18-23). We argue that normal time series windows are more frequent than the anomalous windows and as such normal windows match the set of patterns better. As a result, they will be clustered together in the embedded space whereas the less frequent anomalous windows will have lower similarity scores and will be more scattered in the embedded space.

Computing the distance-weighted similarity score. The intuition behind a weighted similarity score can be illustrated with a simple example. For instance, the sequential pattern $X^1 = (0.1, 0.5)$ clearly matches time series window $S^1_{1,3} = (0.1, 0.55, 1.0)$ better than window $S^1_{4,3} = (0.8, 0.9, 1.0)$. Thus, the similarity between a sequential pattern X^i of length m and a time series window $S^i_{t,l}$ of length l depends on the *minimal* Euclidean distance between the pattern and the window:

$$weighted\_dist(X^i, S^i_{t,l}) = \min_{E \subset S^i_{t,l}} \sqrt{\sum_{j=1}^{m} (E_j - X^i_j)^2} \tag{1}$$

where E is a subsequence of m elements from window $S^i_{t,l}$. The optimisation yields the minimal distance by only observing the best matching elements in the pattern and the window. Given the weighted distance between the sequential pattern and a window, the similarity score is computed as follows:

$$sim(X^i, S^i_{t,l}) = 1.0 - weighted\_dist(X^i, S^i_{t,l})/|X^i|$$

If the distance between the pattern and the time series window is small, the similarity increases. Since the patterns can have different lengths, the distance is normalised for the length of the pattern. Going back to the simple example, the similarity with window $S^1_{1,3}$ is $1.0 - \sqrt{(0.1-0.1)^2 + (0.55-0.5)^2}/2 = 0.975$ while the similarity with window $S^1_{4,3}$ is only $1.0 - \sqrt{(0.8-0.1)^2 + (0.9-0.5)^2}/2 = 0.597$.

Because a sequential pattern imposes a total order on its items, Equation 1 cannot be solved trivially. We design an exact algorithm for computing Equation 1 with the added ordering constraint. Our exact algorithm matches every element in the pattern with exactly one unique element in the window such that the sum of the distances between the matched elements is minimal. The approach is based on the *Smith-Waterman* algorithm for local sequence alignment. However, in contrast, our algorithm ensures that every element in the window and pattern can only be matched once and enforces a match for every element in the pattern. Furthermore, it imposes no gap penalty for skipping elements. Finally, it returns an exact distance between the pattern and the window. Since it is a *dynamic programming algorithm*, it is guaranteed to find the optimal alignment of the

pattern and the segment that minimises the distance. For the sake of brevity, we include the full ExactMinDist algorithm in Appendix A.1.[4]

Dealing with itemsets. Pbad also computes the similarity between itemsets and windows. In contrast to a sequential pattern, an itemset does not impose an order on its elements. We can simply sort the elements of the both the itemset and window before using the ExactMinDist algorithm to obtain the correct weighted distance and compute the similarity score.

Constructing the embedding under the early integration strategy. In case of the early integration strategy, we must deal with patterns with mixed items from different continuous time series and event logs when computing the similarity score. For itemsets, we adapt Equation 1 and compute the minimal distance in each dimension separately and then sum all distances over all dimensions. The distance is computed either weighted, i.e., between an item and a continuous time series value, or binary, i.e., between an item and a discrete event log value. For sequential patterns, we consider the subsequence in each dimension separately and sum all distances. However, in this case we have to satisfy the total order constraint within each time series and between different time series. A brute-force way to compute this, is to generate all possible subsequences (with gaps) over each dimension that satisfy the local and global order constraints, induced by each sequential pattern, and take the subsequence that has the smallest distance. In practice, this is feasible since the length of the time series and patterns is limited to small numbers.

Time complexity. The time complexity of constructing the pattern-based embedding of $\mathbf{S}$ is $\mathcal{O}(|\mathcal{P}|\cdot|\mathcal{S}|\cdot o)$ where $o = \mathcal{O}(l\cdot m)$ is required by the ExactMinDist algorithm, $|\mathcal{P}|$ the number of frequent patterns found, and $|\mathcal{S}|$ the number of windows in the time series. Under the late integration strategy, this complexity increases linearly with the number of dimensions of $\mathbf{S}$.

3.5 Constructing the anomaly classifier

The final step of Pbad is to construct the anomaly detection classifier (lines 24-25 in Algorithm 1). Given the pattern-based embedding of $\mathbf{S}$, any state-of-the-art anomaly detector can be used to compute an anomaly score for each window of $\mathbf{S}$. Pbad uses the IsolationForest classifier [16] since it has been empirically shown to be the state-of-the-art in unsupervised anomaly detection [7]. An isolation forest is an ensemble of decision trees. Each tree finds anomalies by recursively making random partitions of the instance space. Anomalies are isolated quickly from the data, resulting in shorter path lengths in the tree, as illustrated in Figure 1. Usually, the anomaly score is used to rank the segments from most to least anomalous such that the user can easily inspect the most anomalous parts of the data. To generate discrete alarms, one can threshold the score to a specific percentile of the distribution of all scores.

[4] `http://adrem.uantwerpen.be/bibrem/pubs/pbad.pdf`

4 Experiments

In this section, we address the following research questions:

Q1: How does PBAD perform compared to the state-of-the-art pattern based anomaly detection algorithms?
Q2: Can PBAD handle different types of time series data?

We evaluate PBAD on three types of time series: real-world univariate time series, real-world multivariate time series, and synthetic mixed-type time series. Before discussing the results, we lay out the experimental setup.

4.1 Experimental setup

We compare PBAD with following state-of-the-art pattern based anomaly detection methods:

- *Matrix profile* (MP) is an anomaly detection technique based on all-pairs-similarity-search for time series data [18]. The anomalies are the time series discords.
- *Pattern anomaly value* (PAV) is a multi-scale anomaly detection algorithm based on infrequent patterns, specifically bi-grams, for univariate time series [3].
- *Minimal infrequent pattern based outlier factor* (MIFPOD) is an anomaly detection method for event logs [11]. Their outlier factor is based on minimal infrequent, or *rare*, itemsets.
- *Frequent pattern based outlier factor* (FPOF) computes an outlier factor based on the number of frequent itemsets that exactly match the current transaction for transactional databases [10]. We adapt FPOF and compute the outlier factor based on closed itemsets and reduce itemsets further using Jaccard similarity as in PBAD.

Experimental setup. The experimental setup corresponds to the standard setup in time series anomaly detection [11]. Given a time series $\mathbf{S}$ with a number of labelled timestamps: (i) divide the time series into fixed-sized, sliding windows; (ii) each window that contains a labelled timestamp takes its label; (iii) construct the anomaly detection model on the full time series data; (iv) use the model to predict an anomaly score for each window in the time series; (v) evaluate the predictions on the labelled windows by computing the *area under the receiver operating characteristic* (AUROC) and *average precision* (AP).

Parametrisation of methods. Each method has the same preprocessing steps which includes setting an appropriate window size and increment to create the fixed-sized, sliding windows. Continuous variables are discretised using equal-width binning.[5] PAV has no parameters. MATRIXPROFILE has a single parameter,

[5] See Table 4 in Appendix A.2 for details on setting preprocessing parameters.

the window size. The parameters of FPOF and MIFPOD are chosen by an oracle that knows the optimal settings for each dataset. For PBAD, as a rule of thumb, we set minimal support relatively high, that is *min_sup* = 0.01. The Jaccard threshold is set to 0.9. Intuitively, if two patterns cover almost the same windows, e.g., 90 out of 100 windows, using both patterns is both unnecessary and less efficient. For mining *closed itemsets* we use CHARM, and for mining *maximal itemsets* we use CHARM-MFI. For mining *closed* and *maximal sequential patterns* we use CM-CLASP and MAXSP respectively. CHARM and CM-CLASP are based on a vertical representation. MAXSP is inspired by PREFIXSPAN which only generates candidates that have at least one occurrence in the database [19,8]. The sequential patterns should have a minimal length of 2, and by default we set pattern pruning to *closed*. We use the ISOLATIONFOREST classifier implemented in SCIKIT-LEARN with 500 trees in the ensemble. The implementation, datasets and experimental scripts for PBAD are publicly available.[6] We do not report detailed runtime results, however, on the selected datasets PBAD requires less than 30 minutes on a standard PC.

4.2 Anomaly detection in univariate time series

For the univariate test case, we use 9 real-world datasets. Three datasets are from the Numenta time series anomaly detection benchmark [1]. `Temp` tracks the ambient temperature in an office during 302 days where the goal is to detect periods of abnormal temperature. `Latency` monitors CPU usage for 13 days in a data center with the goal of detecting abnormal CPU usage. Finally, `Taxi` logs the number of NYC taxi passengers for 215 days in order to detect periods of abnormal traffic. The 6 remaining datasets are not publicly available. Each tracks on average 2.5 years of `water` consumption in a different store of a large retail company. The company wants to detect periods of abnormal water consumption possibly caused by leaks or rare operational conditions. Domain experts have labelled between 547 and 2 391 hours in each store.

Results and discussion. Table 1 shows the AUROC and AP obtained by each method on each of the 9 univariate time series datasets as well as the ranking of each method. PBAD outperforms the existing baselines for detecting anomalies in univariate time series data in 5 of the 9 datasets.

In the experiments, MP sometimes performs close to random. Because MP detects anomalous windows as those with the highest distance to its nearest neighbour window in the time series, its performance degrades if the data contain two or more similar anomalies. This is, for instance, the case for the `water` consumption data where each type of leak corresponds to a specific time series pattern. A more detailed discussion of the univariate datasets, parameter settings and results is included in Appendix A.2.

We compared the impact of computing the distance-weighted similarity between a pattern and time series window, versus computing an exact match, on the univariate datasets. In this case, using distance-weighted similarity, results in

[6] Implementation of PBAD: `https://bitbucket.org/len_feremans/pbad/`.

	AUROC					AP				
dataset	MP	PAV	MIFPOD	FPOF	PBAD	MP	PAV	MIFPOD	FPOF	PBAD
Temp	0.240	0.590	0.997	**0.999**	0.998	0.014	0.040	0.917	**0.957**	0.917
Taxi	0.861	0.281	0.846	0.877	**0.879**	0.214	0.057	0.300	0.403	**0.453**
Latency	0.599	**0.608**	0.467	0.493	0.553	0.515	0.361	0.255	0.296	**0.382**
Water 1	0.656	0.482	0.514	0.825	**0.884**	0.499	0.301	0.328	0.812	**0.821**
Water 2	0.600	0.520	0.513	0.857	**0.945**	0.353	0.127	0.094	0.688	**0.862**
Water 3	0.536	0.457	0.544	**0.671**	0.605	0.126	0.121	0.079	**0.350**	0.233
Water 4	0.675	0.579	0.548	0.613	**0.721**	0.774	0.687	0.700	**0.817**	0.808
Water 5	0.444	0.581	0.455	0.790	**0.960**	0.199	0.243	0.111	0.671	**0.906**
Water 6	0.682	0.609	0.500	**0.874**	0.752	0.578	0.431	0.228	**0.692**	0.551
Average	0.588	0.523	0.598	0.778	**0.811**	0.364	0.263	0.335	0.632	**0.659**
Ranking	3.333	3.889	4.167	2	**1.611**	3.111	4.111	4.278	1.778	**1.722**

Table 1: The table shows the AUROC and AP obtained by each method on 9 univariate time series. PBAD outperforms the baselines in 5 of the 9 datasets.

a higher AUROC and AP on 8 of the 9 datasets. Using the combination of both frequent itemsets and frequent sequential patterns instead of only itemsets or only sequential patterns results in higher AUROC on 6 of the 9 datasets.

4.3 Anomaly detection in multivariate time series

For the multivariate test case, we use an indoor exercise monitoring dataset [5]. The data contain recordings of 10 people each executing 60 repetitions of three types of exercises: squats (`Sq`), lunges (`Lu`), and side-lunges (`Si`). The (x, y, z) positions of 25 sensors attached to each person were tracked during execution, resulting in a multivariate time series $\mathbf{S}$ of dimension 75.

	AUROC					AP				
Dataset	**MP**	**PAV**	**MIFPOD**	**FPOF**	**PBAD**	**MP**	**PAV**	**MIFPOD**	**FPOF**	**PBAD**
Lu+Si/Sq	0.472	0.571	0.819	0.966	**0.983**	0.283	0.255	0.430	0.862	**0.888**
Lu/Sq	0.604	0.671	0.775	**0.966**	0.940	0.082	0.110	0.131	0.662	**0.737**
Si/Lu	0.471	0.425	0.804	0.864	**0.907**	0.128	0.115	0.444	0.572	**0.573**
Sq/Si	0.484	0.504	0.482	0.903	**0.914**	0.094	0.092	0.087	0.391	**0.707**
Average	0.508	0.542	0.720	0.925	**0.936**	0.147	0.143	0.273	0.622	**0.726**
Ranking	4.5	4	3.5	1.75	**1.25**	4	4.5	3.5	2	**1**

Table 2: The table shows the AUROC and AP obtained by each method on 4 multivariate time series. Each dataset contains tracking of movement during indoor exercises where the normal exercise is listed first, and the anomalous exercise second. For instance, `Lu/Sq` contains 90 repetitions of the *lunge* exercise and 8 repetitions of the *squat* exercise.

We construct 4 multivariate time series datasets containing anomalies by randomly combining around 90 repetitions of one exercise type with 8-12 repetitions of a different type. Then, the goal is to accurately detect the minority exercise. Before applying the algorithms, we use the methodology outlined in [5] to preprocess the raw data and further reduce the number of dimensions of $\mathbf{S}$ to 3. Note that the baseline algorithms are not naturally equipped to deal with multivariate time series. The straightforward solution is to compute an anomaly score for each time series separately and add the scores.

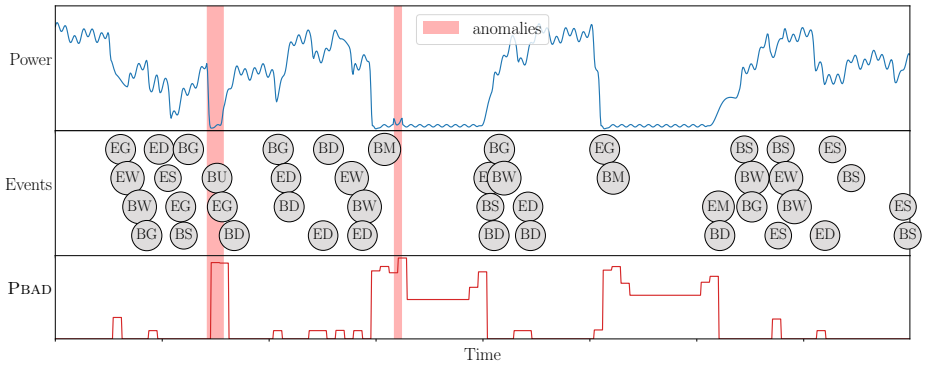

Fig. 2: The figure shows 5 days of synthetic power grid data. The top plot shows continuous power output of the grid. The middle plot shows the discrete events, B and E indicate begin and end respectively, while W, S, D, G, M, and U refer to wind, solar, diesel, gas, maintenance, and shutdown respectively. The bottom plot shows the anomaly score of PBAD. The first anomaly corresponds to a discrete event (BU) that did not generate the expected power response.

Results and discussion. Table 2 shows the AUROC and AP obtained by each method on the 4 multivariate time series datasets as well as the ranking of each method. PBAD and FPOF outperform the other methods, with PBAD improving the AUROC and AP over FPOF with $1.3 \pm 2.7\%$ and $23.8 \pm 33.2\%$ respectively.

4.4 Anomaly detection in mixed-type time series

Due to the lack of publicly-available, labelled mixed-type time series datasets, we construct a realistic synthetic data generator. The generator simulates the electricity production in a microgrid consisting of 4 energy resources: a small wind turbine, a solar panel, a diesel generator, and a microturbine. Each resource has a distinct behaviour. The operator controlling the grid can take 12 discrete actions: turning on and off each of the energy resources, shutting down and starting up the grid, and starting and stopping grid maintenance. Then, a full year of data is generated in three steps. First, a control strategy determines every 5 minutes which actions to take and logs them. It is also possible to take no action. Second, the actions determine the power output of the grid at each time step, forming the continuous time series. Finally, 90 control failures are introduced in the system. These are actions not having the desired effect, e.g., starting the wind turbine does not lead to an increase in electricity production, actions without effect, and effects that are not logged. Using the generator, we generate 45 variations of mixed-type time series, each with different anomalies, and report averaged results. The full details of the generator can be found in Appendix A.3.

Results and discussion. We only ran PBAD on the mixed-type data to detect the control failures. MP and PAV cannot handle event logs, while MIFPOD and

FPOF do not naturally handle mixed-type time series. However, we run PBAD three times: only on the continuous component of the synthetic data (AUROC $= 0.81 \pm 0.13$), only on the event logs (AUROC $= 0.63 \pm 0.07$), and on the full mixed-type series (AUROC $= 0.85 \pm 0.08$). This indicates that PBAD successfully leverages the information in the combined series to detect the anomalous data points. Figure 2 shows 7 days of the power generated by the microgrid and the corresponding event log. The anomaly score generated by PBAD is plotted underneath the data, illustrating that it can accurately detect the anomalies in this case.

5 Related Work

In this section we place our work into the wider context of time series anomaly detection. Existing pattern-based anomaly detection methods each differ in how they define patterns, support, anomaly scores, and what type of input they are applicable to. The original FPOF method [10] mines frequent itemsets for detecting anomalous transactions in a transaction database, using the traditional definition of support. Their outlier factor is defined as the number of itemsets that match the current transaction versus the total number of frequent itemsets. The more recent MIFPOD method [11] mines minimal infrequent, or rare, itemsets for detecting outliers in data streams. Rare itemsets are not frequent, i.e., they do not satisfy the minimal support threshold, but have only subsets that are frequent. They define support as usual, but based on the most recent period, and not necessarily the entire event log. Finally, they compute the outlier factor as the number of rare itemsets that match the current transaction versus the total number of rare itemsets, similar to FPOF, but weighted by the sum of deviation in support of matching rare itemsets. Finally, the PAV method [3] uses linear patterns, i.e., two consecutive continuous values in a univariate time series. The final outlier factor is computed as the relative support of this single pattern in sliding windows of size 2. In contrast to these methods, PBAD considers extensions specific to multivariate and mixed-type time series: it uses a distance-weighted similarity to bridge the gap between a discrete pattern and a continuous signal, employs a late integration scheme to avoid pattern explosion, removes redundant patterns using a threshold on Jaccard similarity, and derives an anomaly score using the ISOLATIONFOREST classifier and the pattern-based embedding of the time series.

The MATRIXPROFILE technique [18] is the state-of-the-art anomaly detection technique for continuous time series. This technique computes for each time series segment an anomaly score by computing the Euclidean distance to its nearest neighbour segment. In contrast, PBAD can also handle the combination of event logs and continuous time series. A host of *time series representation methods* and *similarity measures* have been developed [6] for time series classification. Time series *shapelets* are subsequences from a continuous time series and are used in combination with the dynamic time warping distance to classify time series segments [17]. Sequential patterns used by PBAD are different from shapelets,

because we use non-continuous subsequences with missing elements, or gaps. Itemsets are even more different, because the order of values is discarded. Another key difference is that the enumeration process for shapelets is usually reduced by only considering subsequences of a specific length [12], while we make use of the anti-monotonic property of support to enumerate patterns of varying length without constraints. Finally, itemsets and sequential patterns can also be extracted from discrete event logs.

Another approach, related to classification, is to employ a *minimal redundancy, maximal relevance* strategy to select *discriminative* patterns [4]. Currently, we employ an unsupervised technique, but for future work we could adopt a strategy for selecting patterns that are the most discriminative towards anomalies. Finally, deep learning techniques are becoming a popular choice for time series anomaly detection [14]. For instance, autoencoders could be used to learn an embedding of a mixed-type time series. A key difference with PBAD is that, unlike deep learning techniques, frequent patterns are easily interpretable.

6 Conclusions and future work

Research on anomaly detection in time series so far has prioritised either continuous time series or event logs, but not the combination of both, so-called mixed-type time series. In this paper, we present PBAD, a pattern-based anomaly detection method for mixed-type time series. The method leverages frequent pattern mining techniques to transform the time series into a pattern-based embedding that serves as input for anomaly detection using an isolation forest. An experimental study on univariate and multivariate time series found PBAD to outperform state-of-the-art time series anomaly detection techniques, such as MATRIXPROFILE, PAV, MIFPOD and FPOF. Furthermore, unlike existing techniques, PBAD is able to handle mixed-type time series.

For future research, we see our method as a promising general framework for time series anomaly detection, where certain variations might be more effective in different applications. These include variations on mining a non-redundant and relevant pattern set, on distance measures to match pattern occurrences, and on anomaly detection classification techniques. A useful addition would be an efficient algorithm for computing the similarity between a pattern and window under the early integration strategy. Because the patterns characterise the behaviour of the time series, they can serve as the basis for an anomaly detection classifier that makes explainable predictions. Another possible direction is to adapt the algorithm to work efficiently within a streaming context.

Acknowledgements The authors would like to thank the VLAIO SBO HYMOP project for funding this research.

References

1. Ahmad, S., Lavin, A., Purdy, S., Agha, Z.: Unsupervised real-time anomaly detection for streaming data. Neurocomputing **262**, 134–147 (2017)

2. Chandola, V., Banerjee, A., Kumar, V.: Anomaly detection: A survey. ACM computing surveys (CSUR) **41**(3), 15 (2009)
3. Chen, X.y., Zhan, Y.y.: Multi-scale anomaly detection algorithm based on infrequent pattern of time series. Journal of Computational and Applied Mathematics **214**(1), 227–237 (2008)
4. Cheng, H., Yan, X., Han, J., Hsu, C.W.: Discriminative frequent pattern analysis for effective classification. In: Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on. pp. 716–725. IEEE (2007)
5. Decroos, T., Schütte, K., De Beéck, T.O., Vanwanseele, B., Davis, J.: Amie: Automatic monitoring of indoor exercises. In: Joint European Conference on Machine Learning and Knowledge Discovery in Databases. pp. 424–439. Springer (2018)
6. Ding, H., Trajcevski, G., Scheuermann, P., Wang, X., Keogh, E.: Querying and mining of time series data: experimental comparison of representations and distance measures. Proceedings of the VLDB Endowment **1**(2), 1542–1552 (2008)
7. Domingues, R., Filippone, M., Michiardi, P., Zouaoui, J.: A comparative evaluation of outlier detection algorithms: Experiments and analyses. Pattern Recognition **74**, 406–421 (Feb 2018)
8. Fournier-Viger, P., Lin, J.C.W., Gomariz, A., Gueniche, T., Soltani, A., Deng, Z., Lam, H.T.: The spmf open-source data mining library version 2. In: Joint European conference on machine learning and knowledge discovery in databases. pp. 36–40. Springer (2016)
9. Gershenfeld, N., Krikorian, R., Cohen, D.: The internet of things. Scientific American **291**(4), 76–81 (2004)
10. He, Z., Xu, X., Huang, Z.J., Deng, S.: Fp-outlier: Frequent pattern based outlier detection. Computer Science and Information Systems **2**(1), 103–118 (2005)
11. Hemalatha, C.S., Vaidehi, V., Lakshmi, R.: Minimal infrequent pattern based approach for mining outliers in data streams. Expert Systems with Applications **42**(4), 1998–2012 (2015)
12. Karlsson, I., Papapetrou, P., Boström, H.: Generalized random shapelet forests. Data mining and knowledge discovery **30**(5), 1053–1085 (2016)
13. Lin, J., Keogh, E., Lonardi, S., Chiu, B.: A symbolic representation of time series, with implications for streaming algorithms. In: Proceedings of the 8th ACM SIGMOD workshop on Research issues in data mining and knowledge discovery. pp. 2–11. ACM (2003)
14. Malhotra, P., Vig, L., Shroff, G., Agarwal, P.: Long short term memory networks for anomaly detection in time series. In: 23rd European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning. ESANN (2015)
15. Mueen, A., Keogh, E., Zhu, Q., Cash, S., Westover, B.: Exact discovery of time series motifs. In: Proceedings of the 2009 SIAM international conference on data mining. pp. 473–484. SIAM (2009)
16. Ting, K.M., Liu, F.T., Zhou, Z.: Isolation Forest. In: 2008 Eighth IEEE International Conference on Data Mining(ICDM). pp. 413–422. IEEE (Dec 2008)
17. Ye, L., Keogh, E.: Time series shapelets: a new primitive for data mining. In: Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 947–956. ACM (2009)
18. Yeh, C.C.M., Zhu, Y., Ulanova, L., Begum, N., Ding, Y., Dau, H.A., Silva, D.F., Mueen, A., Keogh, E.: Matrix profile i: all pairs similarity joins for time series: a unifying view that includes motifs, discords and shapelets. In: 2016 IEEE 16th international conference on data mining (ICDM). pp. 1317–1322. IEEE (2016)
19. Zaki, M.J., Meira, W.: Data mining and analysis: fundamental concepts and algorithms. Cambridge University Press (2014)

A Supplementary material

A.1 Efficient algorithm for computing the distance between a pattern and a continuous window

Algorithm 2 outlines the exact distance computation between a pattern X with length m and time series window S with length l. Remark that we use the shorter notation S to denote a window between two timestamps for the i^{th} time series, previously denoted as $S^i_{t,l}$. First, if the pattern is an itemset, the pattern and window are sorted (line 1-2). Next, a scoring matrix with $l+1$ rows and $m+1$ columns is initialized with the first column zeros and the rest of the values ∞ (line 5-6). Intuitively, each row and each column, except the first row and column, correspond to one element in the window and the pattern respectively. Then, each cell in the matrix is updated by setting its value to the minimum of: (1) the sum of the distance between the elements of the pattern and window corresponding to that cell and the value in the cell diagonally up to the left, and (2) the value in the cell above (line 7-9). Finally, the distance that minimizes Equation 1 is stored in the last cell of the matrix.

Algorithm 2: ExactMinDist (X, S)

Input : Pattern X (`itemset` or `sequential pattern`), time series window S
Result: Computed distance d

1 **if** X *is an* `itemset` **then**
2 $\quad X \leftarrow \text{SORT}(X)$; $S \leftarrow \text{SORT}(S)$
3 $m \leftarrow \text{LENGTH}(X)$; $l \leftarrow \text{LENGTH}(S)$
4 $w = l - m + 1$
`// 1. Construct the scoring matrix` **H** `using 0-based indexing:`
5 **H** $\leftarrow$ matrix of ∞ values with $l+1$ rows and $m+1$ columns
`// 2. Fill the first column of the matrix with 0:`
6 $\mathbf{H}_{j0} = 0$ **for** $0 \leqslant j < l+1$
`// 3. Update the values in the matrix:`
7 **for** $0 \leqslant i < m$ **do**
8 $\quad$ **for** $i \leqslant j < i + w$ **do**
9 $\quad\quad \mathbf{H}_{j+1,i+1} = \min \begin{cases} H_{j,i} + (S_j - X_i)^2 \\ H_{j,i+1} \end{cases}$
`// 4. The distance is in the last row and column of the matrix:`
10 $d = \sqrt{\mathbf{H}_{j+1,i+1}}$

The algorithm has a time complexity of $\mathcal{O}(l \cdot m)$. However, we can further reduce the number of computations by only filling in the feasible regions of the matrix. This is possible because there is an ordering to the matched elements. For example, if the pattern has length 5 and the window has length 7, the first element of the pattern can only be matched with one of the first three elements of the window. This is expressed by w in Algorithm 2.

A.2 Additional experimental details

The benchmarks used in this paper consist of 13 different real-world univariate and multivariate datasets. The characteristics of each are outlined in Table 3.

Time series dataset	$\|S\|$	Labelled time period	Labelled normals	Labelled anomalies	Δt	Total length of the series
`Ambient Temperature`	7,267	395 *h*	347 *h*	48 *h*	1 *h*	302 *days*
`Request Latency`	4,017	66 *h*	14 *h*	52 *h*	5 *min*	13 *days*
`New York Taxi`	10,320	356 *h*	236 *h*	120 *h*	30 *min*	215 *days*
`Water use store 1`	292,632	1,217 *h*	829 *h*	388 *h*	5 *min*	1,016 *days*
`Water use store 2`	281,485	2,391 *h*	2,178 *h*	213 *h*	5 *min*	977 *days*
`Water use store 3`	169,253	1,595 *h*	1,488 *h*	107 *h*	5 *min*	587 *days*
`Water use store 4`	292,608	1,821 *h*	615 *h*	1,206 *h*	5 *min*	1,016 *days*
`Water use store 5`	364,032	574 *h*	504 *h*	70 *h*	5 *min*	1,264 *days*
`Water use store 6`	364,032	1,047 *h*	815 *h*	232 *h*	5 *min*	1,264 *days*
`Lu+Si/Sq`	11,152	371.7 *sec*	315.6 *sec*	56.1 *sec*	0.034 *sec*	371.7 *sec*
`Lu/Sq`	17,318	577.2 *sec*	539.7 *sec*	37.5 *sec*	0.034 *sec*	577.2 *sec*
`Si/Sq`	11,114	370.5 *sec*	325.6 *sec*	44.9 *sec*	0.034 *sec*	370.5 *sec*
`Sq/Si`	11,613	387.1 *sec*	349.8 *sec*	37.3 *sec*	0.034 *sec*	387.1 *sec*
`Synthetic`	105,120	8,760 *h*	8,629 *h*	131 *h*	5 *min*	365 *days*

Table 3: Characteristics of all the datasets used in the experiments. The labels and total length of the series are indicated in time units: days, hours, minutes, and seconds.

Before mining the patterns and computing an anomaly score, PBAD and the baselines preprocess the time series by dividing them into *fixed-size sliding windows*. Table 4 shows the values for the window length and increment for each dataset that were used throughout the experiments. If the increment is equal to the window length, the windows do not overlap. The table also shows the number of bins used for discretising each dataset if this is required by the pattern mining algorithms.

Dataset	Window length	Window increment	Nr. bins
`Ambient temperature`	12 *h*	6 *h*	*99*
`Request latency`	1 *h*	30 *min*	*31*
`New York taxi`	6 *h*	3 *h*	*79*
`Water use store 1-6`	1 *h*	1 *h*	*10-50*
`Multivariate indoor exercise`	3 *sec*	0.5 sec	30
`Mixed-type synthetic data`	1 *h*	1 *h*	30

Table 4: The window length and window increment used to divide each time series in fixed-size sliding windows, chosen in function of the sample rate Δt of the underlying dataset. Due to their nature, the water data measurements are already logged as discrete values, similarly for `Temp`, `Latency`, and `Taxi`.

The application of PBAD and the baselines to detect abnormal temperatures in the `Temp` dataset is illustrated in Figure 3. The anomaly scores outputted by PBAD and FPOF peak on the days an actual anomaly was recorded, indicating that these methods successfully detect the abnormal periods. MIFPOD also accurately

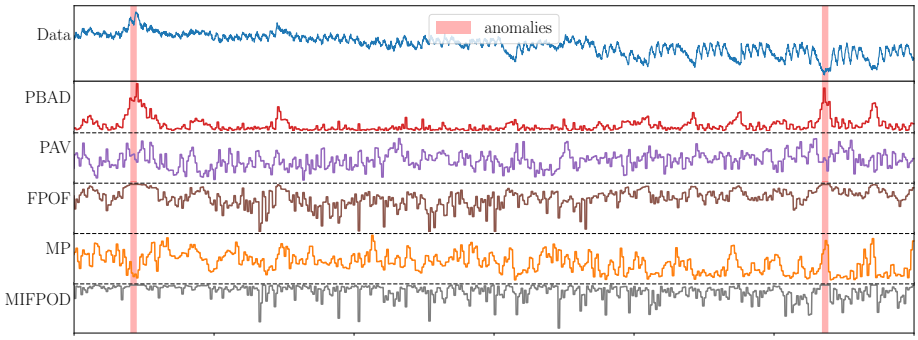

Fig. 3: The figure plots 125 days of the `ambient temperature` time series including 48 hours labelled as anomalous. For each method, the scaled anomaly scores are plotted, a higher score indicating a more anomalous time series window. Both PBAD and FPOF accurately identify the labeled anomalous windows.

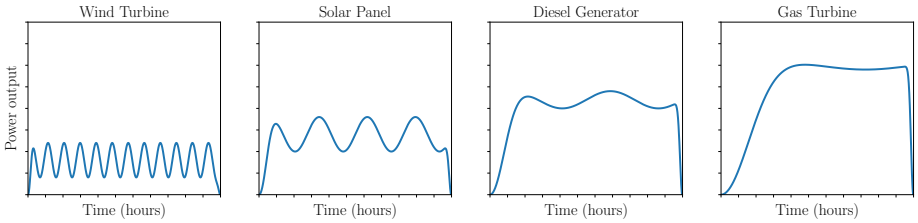

Fig. 4: The figure shows 12 hours of power output for each of the four energy resources, modeled using the variables in Table 5.

finds the anomalies. PAV has highly fluctuating anomaly scores with the scores for the actual anomalies somewhere in the middle of the pack. The AUROC of PAV on this is 0.584 (see Table 1), indicating that its predictions are almost random.

A.3 The mixed-type synthetic data generator

The synthetic data generator simulates the power output of a small electricity grid of distributed energy resources. The grid contains four distinct generators: a wind turbine, a solar panel, a diesel generator, and a gas microturbine. Grid operation is done through 12 distinct actions: turning on and off each of the resources, starting and stopping grid maintenance, shutting down and starting up the grid. The data are generated as follows: (i) a control strategy determines which actions to take at each time step; (ii) the current actions at each time step determine the power output of the grid; (iii) three distinct types of failures are added to the data.

Distributed energy resources. The grid consists of four distinct energy resources. Each of the resources is characterized by five variables: start-up time, shutdown

time, average power output, amplitude of the fluctuation in power output, and speed of the fluctuations. Next, the power output over a period of time is modeled using a sine wave, and the start-up and shutdown are modeled using an exponential squashing function. In addition, the grid can also shutdown or undergo maintenance, leading to six distinct grid behaviors. Table 5 lists all the possible grid behaviors and the variables characterizing each behavior. Figure 4 plots the power output of each of the four energy resources during a period of 12 hours. The power output is modeled with a sine wave to simulate intermittency effects for the renewable resources as well as fluctuations in the energy demand. The start-up and shutdown behavior is modeled by multiplying the sine wave with an exponential squashing function. The variables of the sine wave and squashing function are listed in Table 5.

	Wind turbine	**Solar panel**	**Diesel generator**	**Gas turbine**	**Grid maintenance**	**Shutdown**
Power output	5	10	15	20	2	0
Fluctuation	2	2	1	0.5	0.5	0
Period	1 h	3 h	6 h	12 h	1 h	0 h
Start-up time	10 min	30 min	1 h	2 h	10 min	1 min
Shutdown time	10 min	10 min	10 min	10 min	10 min	10 min

Table 5: The four energy resources and two additional grid behaviors. Each behavior is characterized by five variables.

Control strategy. The generator employs a simple control strategy. Each time step an action is selected from the list of possible actions: *start/stop wind turbine*, *start/stop solar panel*, *start/stop diesel generator*, *start/stop gas turbine*, *start/stop grid maintenance*, *shutdown grid*, *no action*. Each action has a probability to be selected (e.g., *no action* has a probability of 85%). The list of possible actions is updated depending on previous actions. For instance, the wind turbine cannot be started if it is already running. Finally, if an energy resource is started it runs for at least a number of hours randomly selected from the list: *1 hour*, *2 hours*, *6 hours*, *12 hours*, and *a day*. The actions are logged in the event log.

Power output of the grid. The power output of the grid depends on the actions taken by the control strategy. The power output at each time step is the sum of the outputs of each of the energy resources that are starting up or running at that time step. The behaviors *grid maintenance* and *grid shutdown* override the current power output. For instance, if the grid shuts down, there is no power output. The power output is a continuous time series.

Grid anomalies. Three types of failures are added to the mixed-type time series. First, actions that do not lead to a change in power output. Second, changes in the power output that do not correspond to a specific action. Third, actions that lead to an incorrect change in power output. Each failure has a time span of either 1 or 2 hours and none of the failures overlap. In total, 30 failures of each type are added. Combining the event log and the continuous power data results in a fully annotated mixed-type time series.