SAND: Streaming Subsequence Anomaly Detection

Paul Boniol
EDF R&D; Univ. de Paris
paul.boniol@edf.fr

John Paparrizos
University of Chicago
jopa@uchicago.edu

Themis Palpanas
Univ. de Paris; IUF
themis@mi.parisdescartes.fr

Michael J. Franklin
University of Chicago
mjfranklin@uchicago.edu

ABSTRACT

With the increasing demand for real-time analytics and decision making, anomaly detection methods need to operate over streams of values and handle drifts in data distribution. Unfortunately, existing approaches have severe limitations: they either require prior domain knowledge or become cumbersome and expensive to use in situations with recurrent anomalies of the same type. In addition, subsequence anomaly detection methods usually require access to the entire dataset and are not able to learn and detect anomalies in streaming settings. To address these problems, we propose SAND, a novel online method suitable for domain-agnostic anomaly detection. SAND aims to detect anomalies based on their distance to a model that represents normal behavior. SAND relies on a novel steaming methodology to incrementally update such model, which adapts to distribution drifts and omits obsolete data. The experimental results on several real-world datasets demonstrate that SAND correctly identifies single and recurrent anomalies without prior knowledge of the characteristics of these anomalies. SAND outperforms by a large margin the current state-of-the-art algorithms in terms of accuracy while achieving orders of magnitude speedups.

PVLDB Reference Format:
Paul Boniol, John Paparrizos, Themis Palpanas, and Michael J. Franklin. SAND: Streaming Subsequence Anomaly Detection. PVLDB, 14(10): 1717-1729, 2021. doi:10.14778/3467861.3467863

1 INTRODUCTION

Large collections of data series[1] are becoming a reality in a large variety of scientific domains, and there is an increasing demand for developing methods to efficiently and accurately analyze them [5, 38, 40, 43]. Moreover, many domains require methods to deal with streams and adapt to possible drifts in data distribution.

Anomaly Detection: Anomaly, or outlier detection is a well-studied problem [7, 52, 60] relevant to several scientific domains [39], such as cardiology for detecting abnormal heartbeats [22], engineering for identifying wear and tear in bearings of rotating machines [4], manufacturing for product quality assurance [33], data center operation management for hardware and software health monitoring [45], aviation for identifying mechanical failures in helicopter operation monitoring [16], and astrophysics for removing transient noise signals from gravitational wave interferometer measurements [6]. For the specific case of sequences and data series, we are interested in identifying *anomalous subsequences*, where outliers are not single values, but rather a sequence of values. This distinction is important, because even when all values in a subsequence are normal when examined independently from one another (e.g., they all fall inside the normal operation thresholds), the *sequence* of these same values may be anomalous (e.g., the shape of the subsequence may not be normal). Therefore, subsequence anomaly detection enables the early identification of potentially abnormal events that would have been otherwise undetected [4].

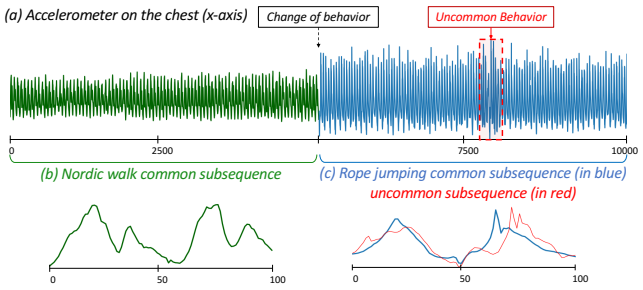


Figure 1: (a) Accelerometer variation positioned on chest (y-axis) while (b) Nordic walking and (c) rope jumping.

Moreover, data measurements arriving continuously in several real-world cases, require anomaly detection to take place in real-time. Because drifts in data distribution are common, the detection needs to be independent of these changes. To illustrate this point, Figure 1(a) depicts acceleration on the x-axis of a device positioned on the chest of a human performing different actions [54]. We observe that the data characteristics corresponding to subsequences are different for Nordic walking (Figure 1(b)) and rope jumping (Figure 1(c)). As changes of actions happen in real-time, the detection of abnormal subsequences (such as the red subsequence in Figure 1(c)) needs to be able to adapt to changes in the data generation process.

Limitations of Previous Approaches: In the non-streaming case, in order to solve the aforementioned task, existing techniques either explicitly search for pre-determined types of anomalies [2, 22], or identify as anomalies the subsequences with the largest distances to their nearest neighbors (termed *discords*) [50, 60]. We observe that these approaches pose limitations to the subsequence anomaly identification task, for several reasons. It has been shown that the State-Of-the-Art (SOTA) algorithms (e.g., [50, 60]) are very successful for datasets that contain a *single* anomaly, or multiple but different (from one another) anomalies [60]. The reason is that these algorithms base their anomaly detection procedure on the distance of a subsequence to its Nearest Neighbor (NN) in the dataset (by selecting as anomalies the subsequences with the farthest NN).

For the case of multiple similar anomalies, the m^{th} *discord* approach has been proposed [59]. This approach takes into account

[1]If the dimensions are ordered by time then we refer to data series as *time series*. We will use the terms *sequence*, *data series*, and *time series* interchangeably.

Proceedings of the VLDB Endowment, Vol. 14, No. 10 ISSN 2150-8097.
doi:10.14778/3467861.3467863

the multiplicity m of the anomalous subsequences that are similar to one another, and computes the m^{th} NN of every subsequence. The subsequences with the furthest m^{th} NN are then selected as anomalies. Nevertheless, this approach assumes that we know the multiplicity m (or a close estimation of it), which is not true in practice. To avoid this situation, approaches that either estimate the isolation of subsequences as multidimensional vectors [28], or estimate and summarize the normal subsequences [10, 11] have been proposed. These methods aim to embed (in trees [28], set [10] or directed graph [11]) the different behaviors of the data series such that anomalies (i.e., rare events) are easy to discriminate. The set-based and the directed graph-based approaches have been shown to outperform the previous discord methods for datasets containing similar anomalies [10, 11]. Nevertheless, in the case of streaming data series, among all previous methods for subsequence anomaly detection, only discords methods (such as Matrix Profile incremental implementation [60]) and tree-based methods [31] can be used. The remaining methods cannot adapt to changes and learn new data characteristics, both of which are required when dealing with data streams, due to their design. In such cases, the methods need to learn and modify their parameters as new data arrive.

Our Approach: To address the aforementioned problems we propose SAND, a novel approach suitable for subsequence anomaly detection in data streams. SAND builds a data set of subsequences representing the different behaviors of the data series. These subsequences are weighted using statistical characteristics such as their cardinality (i.e., how many times the subsequence occurred) and their temporality (i.e., the time difference this subsequence has been detected for the last time). SAND enables this data structure to be updated from one batch to another, while being able to compute an anomaly score at every timestamp. Thus, SAND proposes a solution to the subsequences anomaly detection task on streaming data. SAND benefits from k-Shape [41], a SOTA data-series clustering method, which we extend to enable the clustering result to be updated without storing any of the previous subsequences. We demonstrate experimentally that our method outperforms the current (static and streaming) SOTA approaches.

Contributions: Our contributions are as follows.

- We describe the concepts and ideas used by the SOTA methods on subsequence anomaly detection (static and streaming) and discuss their practical shortcomings.
- We extend k-Shape for streaming scenarios by enabling batch updates of the clustering partition. Our approach avoids entirely the storage of the previous subsequences, a critical step for operating over unbounded data series.
- We present SAND, our subsequence anomaly detection method specifically designed for operation over streaming sequence data. SAND exploits our streaming k-Shape to scale in memory and in execution time for unbounded streams. We propose a new weighting scheme for clusters and an automatic cluster creation procedure to handle distribution drifts. We finally propose a novel anomaly score computation that adapts dynamically to the current batch and gives less importance to old subsequences.
- We perform an experimental analysis using a large data corpus of real datasets (that include a ground truth of annotated anomalies). from different domains. We evaluate both subtle changes in data characteristics (by concatenating datasets from the same domain) and drastic changes (by concatenating datasets from different domains). We empirically evaluate the influence of SAND's parameters on accuracy and execution time. Finally, we compare SAND with several SOTA approaches and show that our method outperforms the strongest competitor up to 27% while executing one order of magnitude faster.

2 PRELIMINARIES

Data Series and Streaming Data: We focus on the analysis of ordered sequences of measurements. We distinguish between sequences with fixed size (data series) and unbounded evolving sequences (data streams). We formally defined them as follow: a data series $T \in \mathbb{R}^n$ is a sequence of real-valued numbers $t_i \in \mathbb{R}$ $[T_0, T_1, ..., T_n - 1]$; $|T|$ is defined as the length of T. We are interested in local section of the data series, called subsequences. A subsequence $T_{i,\ell} \in \mathbb{R}^\ell$ of a data series T is a subset of contiguous values from T of length ℓ (usually $\ell \ll |T|$) starting at position i: $T_{i,\ell} = [T_i, T_{i+1}, ..., T_{i+\ell-1}]$. We define the set of all subsequences of length ℓ in a given data series T: $\mathbb{T}_\ell = \{T_{i,\ell} | \forall i . 0 \leq i \leq |T| - \ell + 1\}$.

In the specific case of data streams, the total size of the data series is not known and potentially infinite. Data points arrive incrementally. Moreover, one can wait for a given number of points before analyzing them. In this case, we define this quantity as a batch. For a given timestamp of arrival t, we note a batch $\mathbb{T}^t_\ell = \{T_{t,\ell}, ..., T_{t+b_{size}-\ell,\ell}\}$ a ordered set of subsequences of length ℓ of size $|\mathbb{T}^t_\ell| = b_{size}$. $\mathbb{T}^0_\ell$ is thus the initial batch.

2.1 Data-Series Clustering

Despite the proliferation of anomaly detection methods, a set of effective methods are those that either determine anomalies by comparing subsequences to previous subsequences or with an estimated normal behavior (as discussed in Section 2.2). Clustering can summarize the underlying patterns in data and, therefore, can be used to extract the recurring behavior in datasets for anomaly detection purposes. Formally, given a set of observations (or subsequences which are the topic of this paper), clustering methods partition this set into k distinct clusters, such that the within-cluster sum of squared distances is minimized. For a given set of subsequences $\mathbb{T}_\ell$, we note $\mathbb{C} = \{C_0, ..., C_k\}$ the optimal partition of k cluster C_i with $\forall C_i, C_j \in C, C_i \cap C_j = \emptyset$. We note $\bar{C}_i$ the centroid of cluster C_i.

k-means and Euclidean Distance: The k-means algorithm solves this partitioning problem using the z-normalized Euclidean Distance, ED. Formally, given two sequences, A and B, of the same length, ℓ, we can calculate their ED, as follows [15, 35, 55, 57, 58]: $ED(A, B) = \sqrt{\sum_i^l (\frac{A_{i,1} - \mu_A}{\sigma_A} - \frac{B_{i,1} - \mu_B}{\sigma_B})^2}$, where μ and σ represent the mean and standard deviation of the sequences. k-means centroids correspond to the arithmetic mean of the subsequences in their corresponding clusters. Other algorithms based on ED have been proposed (such as hierarchical clustering). Nevertheless, only k-means scales linearly with the size of the dataset. Moreover, it is easy to extend k-means to a streaming context [21], since the centroids can be incrementally updated as new points arrive.

k-Shape and Shape-based Distance: ED-based algorithms cannot capture the necessary property of alignment in data series

(i.e., ED is not a competitive distance measure in terms of accuracy [44]). Recently, k-Shape (clustering algorithm based on *Shape-Based-Distance* (*SBD*)) has shown SOTA performance in data series clustering [41, 42]. SBD uses cross-correlation to find the appropriate alignment between two sequences. Formally, given A, B, two sequences of length ℓ, the *SBD* distance is defined as follows:

$$SBD(A,B) = 1 - max_w\left(\frac{R_{w-\ell}(A,B)}{\sqrt{R_0(A,A).R_0(B,B)}}\right)$$
$$\text{with: } R_k(A,B) = \begin{cases} \sum_{i=1}^{\ell-k} A_{i+k}.B_i, & k \geq 0 \\ R_{-k}(B,A), & k < 0 \end{cases}$$

The k-Shape centroid computation corresponds to an optimization problem in which we are computing the minimizer (i.e., sequence) of the sum of squared distances to all other sequences using the *SBD* distance. Formally, as described in Equation 15 in [41], the centroid $\bar{C_j}$ is computed as follows:

$$\bar{C_j} \leftarrow \underset{\bar{C_j}}{argmax} \frac{(\bar{C_j})^T.M.\bar{C_j}}{(\bar{C_j})^T.\bar{C_j}}$$
$$\text{with: } M = Q^T.S_j.Q,\ Q = I - \frac{1}{|\bar{C_j}|}O,\ \text{and } S_j = \sum_{A \in C_j} A.A^T \quad (1)$$

Note that $\bar{C_j}$ is considered as a vector ($(\bar{C_j})^T$ is its transpose) in the above equation. The dot operator is the dot product between two matrices/vectors. Moreover, I is the identity matrix, and O is a matrix with all ones. In practice, the centroid that maximizes Equation 1 corresponds to the eigenvector that corresponds to the largest eigenvalue of the real symmetric matrix M. Moreover, as depicted in Equation 1, the centroid computation requires all the sequences $A \in C_j$ for every cluster C_j in memory and can be used for non-streaming data series only. To alleviate this issue, we propose an extension of k-Shape for data stream applications. This extension stores and incrementally updates only the covariance matrices M for each cluster while new batches of data series arrive. As we are not storing each data series, this implies constant memory usage over time and suits well data stream application. We describe in details the proposed extension in Section 3.3.2.

2.2 Subsequence Anomaly Detection

For the scope of this paper, we are interested in detecting abnormal subsequences within data streams in an unsupervised manner (i.e., no labels for neither normal nor abnormal subsequences). Even though there exists no formal and clear definition of what is an anomaly, we consider abnormal subsequences that are rare (i.e., significantly less frequent than other groups of subsequences). Next, we describe general families of methods and their principles that can be used to detect abnormal subsequences.

Discord-based methods: In the analysis of subsequences and for the purpose of abnormal subsequences detection, previous works [14, 20, 24, 27, 29, 50, 60] focused on the analysis of nearest neighbors distance among subsequences of the data series. Formally, given a subsequence $T_{i,\ell}$ in T, we say that its m^{th} Nearest Neighbor (m^{th} NN) is $T_{j,\ell}$, if $T_{j,\ell}$ has the m^{th} shortest distance to $T_{i,\ell}$, among all the subsequences of length ℓ in T. For accuracy and efficiency reasons, subsequences that highly overlap (i.e., trivial matches [19]) are excluded in the search for nearest neighbors. We define trivial matches of $T_{i,\ell}$ as subsequences $T_{a,\ell}$, with $|i-a| < \ell/4$.

In the case of data series analysis, the state-of-the-art solutions for subsequence anomaly detection use the following definition:

Definition 1 (discord [14, 20, 24, 27, 29, 30, 50, 60]). *Among all subsequences of length ℓ of series T, the subsequence $T_{i,\ell}$ that has the largest distance to its NN is called a (data series) discord.*

This is an intuitive definition: a subsequence is a discord if its NN is very far away. However, this definition fails when we have two neighboring discords, with a small distance to each other, and a very large distance to all the rest of the subsequences. Thus, to capture these situations the m^{th}-discord has been proposed:

Definition 2 (m^{th}-discord [27, 59]). *Among all subsequences of length ℓ of series T, the subsequence $T_{i,\ell}$ that has the largest distance to its m^{th} NN is called an m^{th}-discord.*

This definition skips the nearest neighbors up to the m^{th} one and captures groups of similar discords (i.e., anomalies). Even though discords have been extensively studied in the literature, we make the following observations. First, the presence of similar anomalies may in some situations cause the discord definition to fail in identifying them. Moreover, their multiplicity being difficult to estimate, the m^{th}-discord definition may in some cases not be straight-forward to use in practice.

Proximity based Methods: Instead of measuring the neighbor distance, previous works focused on identifying globally the normal behaviors or the isolation zones inside the subsequence space. Following this idea, general methods for multi-dimensional points outlier detection has been proposed [13, 28, 31]. Nevertheless, methods such as Isolation forest [28] seems to perform well for the specific case of subsequence anomaly detection [11]. The latter aim to model the isolation section of the multidimensional space in which the data series subsequences are. In practice, it builds trees that split randomly the multidimensional space. The depth of these trees is then used to estimate the isolation of a given subsequence (the shorter the depth is, the more isolated the subsequence is).

Moreover, recently proposed methods model the normal behaviors of the data series and has been shown to outperform the previous SOTA approaches. Specifically, NormA [8–10] proposed to model the normal behaviors as a set N_M (named after *Normal Model*) defined as follows: N_M is a set of sequences, $N_M = \{(N_M^0, w_0), (N_M^1, w_1), ..., (N_M^n, w_n)\}$, where N_M^i is the centroid (subsequence) of length ℓ_{N_M} (the same for all N_M^i) that corresponds to a recurring behavior in the data series T, and w_i is its normality score (as we explain later, the highest this score is, the more usual the behavior represented by N_M^i is). In other words, this model averages (with proper weights) the different recurrent behaviors observed in the data, such that all the normal behaviors of the data series will be represented in the normal model, while unusual behaviors will not (or will have a very low weight). In the NormA method $\ell_{N_M} > \ell$ is chosen to make sure that useful subsequences are not missed, i.e., subsequences with a large overlap with an anomalous subsequence. For instance, for a given subsequence of length ℓ, a normal model of length $\ell_{N_M} = 2\ell$ will also contain the subsequences overlapping with the first and last half of the anomalous subsequence. Then, abnormal subsequences are identified by

computing their distance to N_M (specifically defined as in [10]). Subsequences with a large distance to N_M will be considered abnormal. Overall the only essential input parameter of NormA is the length ℓ of the anomaly (which is also one of the inputs in all relevant algorithms in the literature [14, 27, 30, 50, 60]).

Finally, there are alternatives to represent the normal subsequence behavior of a data series. The previous normal model was a set of subsequences that aimed to store both normal and abnormal subsequences into the same set, associated with weights that rank them based on their normality. One can argue that ordering information is missing from this data representation. Series2Graph [11, 12] is an approach for subsequence anomaly detection based on a graph representation in which edges encode the ordering.

Other Related Works: Alternative methods exist for anomaly detection but they are not specifically designed for subsequence anomaly detection [25, 52, 53]. There is also work on point anomaly detection in data streams of wireless sensor networks [37], where the focus is on efficient processing for maximizing the lifespan of the network. Finally, deep learning approaches, e.g., based on recurrent [32] or convolutional [36] neural networks, have been proposed. However, these approaches are supervised, requiring a large amount of labeled data for training, which is usually not available in practice. Previous studies [10, 11] indicate that these methods are not yet competitive to approaches we consider here.

Limitation for Online Operation: Some of the methods discussed above (such as NormA and Series2Graph) suffer from a significant drawback, which is problematic in the context of streaming data: they have not been designed to adapt to and learn from changes in data characteristics (i.e., distribution drifts). In a streaming context, the models need to learn and modify their parameters as new data arrive. In the experimental evaluation, we demonstrate that these methods are not able to adapt to changes of normality (i.e., a person starting to run, walk, or sleep, or a change of state in a machine).

Despite the previously enumerated limitations, discord-based approaches can adapt their operation to data streams. Variants like STAMPI [60] can update the distance profile (which is used to identify discords) incrementally. Other methods, such as Isolation Forest, cannot be used in an online manner but variants such as Isolation Mondrian Forest [31] can be updated incrementally.

2.3 Problem Formulation

The problem we solve in this paper is defined as follows.

Problem 1 (Streaming Subseq. Anom. Detection). *Given a data stream T, arriving in batches $\mathbb{T}_\ell^t$ (with b_{size} the size of the batches) and a targeted anomaly subsequence length ℓ, return $\mathcal{S}_A$, a set containing the η most abnormal subsequences of length ℓ.*

In this work, we focus on the *Top-k* anomalies; using instead a threshold ϵ to detect anomalies is a straightforward extension. Table 1 summarizes the symbols we use in this paper.

3 PROPOSED APPROACH: SAND

In this section, we present SAND, our solution for unsupervised subsequence anomaly detection in data streams.

3.1 Overview

Symbol	Description
T	a data series
$\|T\|$	cardinality of T
ℓ	input subsequence length
$T_{i,\ell}$	Subsequence of T of length ℓ, starting at index i
$\mathbb{T}_\ell^0$	Initial batch of T of length b_{size}
$\mathbb{T}_\ell^t$	batch starting at timestamp t of T of length b_{size}
$\mathbb{C}$	partitioning of a clustering algorithm
C_i	Cluster in $\mathbb{C}$
$\bar{C}_i$	Centroid of C_i
k	number of clusters
η	number of anomalies

Table 1: Table of symbols

Overall, we compute and update a weighted set of subsequences over time. The summary of the computation steps are as follows:

- We start by computing our initial model on the initial batch. We first select subsequence candidates and then perform the k-Shape clustering algorithm. These clusters are then scored and stored in memory (Section 3.2).
- After each new batch, we compute a new clustering on the newest batch. We then match the new cluster with the similar one in the current model stored in memory. The matching procedure is based on a distance threshold that corresponds to the intra-cluster average distance for each existing cluster. If the distance between an existing cluster and a new cluster is smaller than the threshold, we merge the two clusters. Otherwise, we create a new cluster. This procedure is described in Section 3.3.1.
- We then propose a mechanism to compute the centroid of two merged clusters without keeping in memory the raw subsequences of these two clusters. This mechanism is a novel technical extension of k-Shape for streaming scenarios. The matching system and the centroid computation is summarized in Algorithm 2 and in Section 3.3.2.
- We finally update the weights for each cluster (new, merged, or unchanged) based on their previous score. The update procedures is summarized in Algorithm 3 and in Section 3.3.3.
- At any time, one can compute the anomaly score on the current batch using the current model stored in memory. We incrementally learn the mean and the standard deviation to compute the anomaly score such that the anomaly detection is adapted to the current batch subsequences/behaviors (Section 3.4).

Figure 2 depicts the general framework of the model. We now describe in detail the different steps of our approach. Algorithm 1 summarizes the parameters, as well as the update procedures of the set Θ. Next, we describe in detail the procedures.

3.2 Preprocessing Step

We first start by describing the initialization step of the SAND. The initialization step consists on building a set of clusters paired with weights, noted as $\Theta = \{(\bar{C}_0, w_0), (\bar{C}_1, w_1)..., (\bar{C}_k, w_k)\}$ similarly as described in [10] (note that, from a higher perspective, the previously discussed N_M has the same data structure than Θ). In the latter, $\bar{C}_k$ is the centroid of the cluster C_k. The latter is a sequence of length ℓ_Θ. Note that this Θ set will be our main data structure that will evolve through time. For simplicity purposes, in the rest of this paper we say that $\bar{C}_i$ is in Θ if there exists the tuple $(\bar{C}_i, w_i)$ in Θ. We first describe the initialization of this set below.

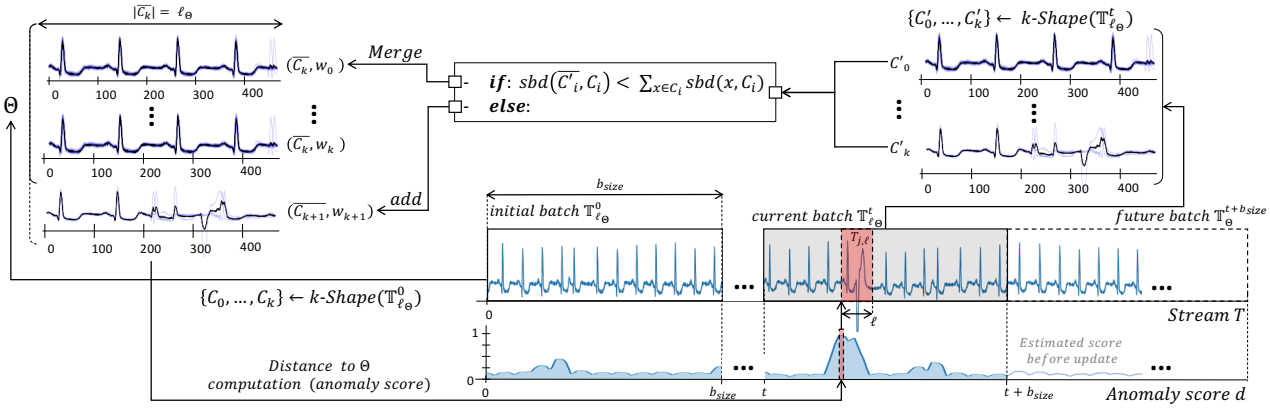


Figure 2: SAND computation framework.

Algorithm 1: SAND

input :
A data stream T: T should be composed of numerical values, evolving on a single dimension.
A subsequence length ℓ_Θ: (with the condition $\ell_\Theta > \ell$). As empirically shown [10], for a given ℓ, $\ell_\Theta = 3 * \ell$ can be used as the default setting.
An initial number of cluster k: It will be used as the number of clusters for the k-Shape algorithm for the initial clustering, and at each new batch. In practice, there is no restriction to use a different k for the initialization and the rest. For the sake of simplicity, we use the same k.
An initial batch $\mathbb{T}^0_{\ell_\Theta}$ **(and a batch size** $b_{size} < |T|$**)**: b_{size} is also used to set the size of every new batch $\mathbb{T}^t_{\ell_\Theta}$. In practice, one can use a different batch size for the initial batch and the others. For the sake of simplicity, we use the same batch size.
A real number $\alpha \in [0, 1]$: a parameter conditioning the rate of change of the centroids, the weights, the estimated mean, and standard deviation of the anomaly score.
output : A set Θ, a data series $score$

// Initialization
$\Theta, \mathbb{C}^0 \leftarrow \{\}, kShape(\mathbb{T}^0_{\ell_\Theta}, k)$;
$\Theta \leftarrow updateCentroid(\Theta, \mathbb{C}^0, init = True)$;
$\Theta \leftarrow updateParam(\Theta, \alpha)$;
$\mu, \sigma \leftarrow 0, 0$;
// Online Update
foreach *in coming batch* $\mathbb{T}^t_{\ell_\Theta}$ **do**
  $\mathbb{C}^t \leftarrow kShape(\mathbb{T}^t_{\ell_\Theta}, k)$;
  $\Theta \leftarrow updateCentroid(\Theta, \mathbb{C}^t, init = False)$// see Alg. 2
  $\Theta \leftarrow updateParam(\Theta, \alpha)$// see Alg. 3
  $score, \mu, \sigma \leftarrow computeScore(\Theta, \mathbb{T}^t_{\ell_\Theta}, \alpha, \mu, \sigma)$ // see Alg. 4

We first select the subsequences in an initial batch before clustering them. Formally, for a given data series T, for a batch of length b_{size} we define $\mathbb{T}^0_{\ell_\Theta} = \{T_{0,\ell_\Theta}, ..., T_{b_{size}-\ell_\Theta,\ell_\Theta}\}$ as the set of all overlapping subsequences in the first initial batch. For efficiency and accuracy matters, one can argue that only a subset of non-overlapping subsequences might be selected. However, using an appropriate clustering algorithm (such as k-Shape clustering algorithm), one can cluster highly overlapping subsequences. We then apply the k-Shape clustering algorithm. For the sake of simplicity, we define k-Shape: $\mathbb{T}_{\ell_\Theta}, k \rightarrow \mathbb{C}$, with k being the number of cluster and $\mathbb{C} = \{C_0, ..., C_k\}$ being the set of clusters. The number of cluster k is a user defined parameter. We evaluate its influence in Section 4.4. Note that the number of cluster k will imply the initial size of the set Θ. A cluster C_i is defined as $C_i \subset \mathbb{T}^0_{\ell_\Theta}$ and $\forall C_i, C_j \in C, C_i \cap C_j = \emptyset$. The initialization step is defined as follows:

$$\{C_0, ..., C_k\} = kShape(\mathbb{T}^0_{\ell_\Theta}, k)$$
$$\forall (\bar{C}_i, w_i) \in \Theta, \begin{cases} w'_i &= \frac{|C_i|^2}{\sum_{\bar{C}_j \in \Theta} sbd(\bar{C}_i, \bar{C}_j)}. \\ w_i &= \frac{w'_i}{\sum_{w'_j} w'_j} \end{cases} \quad (2)$$

As described in the previous equation, we normalize the weight w_i to have their sum equal to 1. In Θ computation, k-Shape algorithm handles internally the realignment of the sequences and thus permits to use a high number of subsequences of T without realigning them beforehand. To be consistent with the SBD distance used in the k-Shape algorithm, we use the SBD distance in the scoring step to measure the isolation of a given cluster to the rest of the clusters. Theoretically, to be able to update sequences (i.e., centroids) in Θ, we have to store in memory the subsequences that were used to compute them. As mentioned earlier, we denote C_i the subsequences set, and $\bar{C}_i$ its centroid. Nevertheless, storing C_i implies an infinite storage need for unlimited streams. Thus, in practice, we do not store C_i, and we describe in Sections 3.3.2 how we update the sequences in Θ without storing their corresponding set C_i. For the sake of simplicity, we still use C_i notation as virtual sets corresponding to $\bar{C}_i$ in Θ.

3.3 Continuous Model Update

Once the initialization is done, the model is ready to receive new subsequences. Let $\mathbb{T}^t_{\ell_\Theta}$ the set of subsequences (of size $|\mathbb{T}^t_{\ell_\Theta}| = b_{size}$) from the current batch arriving at time t. The length of this batch is a user parameter. We evaluate the influence of this parameter in Section 4.4. For every new batch, we perform a k-Shape clustering operation with k clusters (same value of k used in the initialization step), and we note the clustering result $\mathbb{C}^t$.

3.3.1 **Matching Strategy**. We then match $\mathbb{C}^t$ with the sequence in Θ. In practice, we define a threshold $\tau_{c,j}$ for each sequence $\bar{C}_j$ in Θ. We then verify if the distance between a centroid of a new cluster in $\mathbb{C}^t$ and an existing sequence $\bar{C}_j \in \Theta$ is smaller than $\tau_{c,j}$. Formally, given a cluster C^t_i in the clustering result $\mathbb{C}^t = \{C^t_0, ..., C^t_k\}$ on the current batch $\mathbb{T}^t_{\ell_\Theta}$, Θ, and a threshold $\tau_{c,j}$ for each sequence $\bar{C}_j$, the matching process is operated as follows:

Algorithm 2: Θ Centroids Update: *computeCentroid*

input : A set Θ, a clustering partition $\mathbb{C}^t$ on the batch $\mathbb{T}^t_{\ell_\Theta}$, a Boolean *init*.
output: A set Θ

foreach $C_i^t \in \mathbb{C}^t$ **do**
  $newClusters \leftarrow \{\}$;
  if $(\exists(\bar{C}_j, S_j, \tau_{c,j}, w_j, s_j) \in \Theta, sbd(\bar{C}_j, \bar{C}_i^t) < \tau_{c,j}) \wedge (init = False)$ **then**
    // The new cluster C_i^t can be merged with the existing cluster C_j
    $S_j \leftarrow S_j + \sum_{T_{j',\ell_\Theta} \in C_i^t} T_{j',\ell_\Theta}.T^T_{j',\ell_\Theta}$;
    $\bar{C}_j \leftarrow \underset{\bar{C}_j}{argmax} \frac{(\bar{C}_j)^T.Q^T.S_j.Q.\bar{C}_j}{(\bar{C}_j)^T.\bar{C}_j}$;
  else
    // The new cluster C_i^t cannot be merged with any existing cluster
    $S_i \leftarrow \sum_{T_{j,\ell_\Theta} \in C_i} T_{j,\ell_\Theta}.T^T_{j,\ell_\Theta}$;
    $\bar{C}_i \leftarrow Centroid(C_i)$;
    $newClusters \leftarrow newClusters \cup (\bar{C}_i, S_i, None, None, None)$;
return $\Theta \cup newClusters$;

- if $\exists(\bar{C}_j, w_j) \in \Theta, SBD(\bar{C}_j, \bar{C}_i^t) < \tau_{c,j}$: We consider that cluster C_i^t found in the current batch is similar to an existing sequence $\bar{C}_j$ in Θ. We thus update the sequence $\bar{C}_j$, and its corresponding weight w_j using the new cluster C_i^t.
- if $\forall(\bar{C}_j, w_j) \in \Theta, SBD(\bar{C}_j, \bar{C}_i^t) > \tau_{c,j}$: We consider that cluster C_i^t is not similar enough with any of the existing sequences in Θ. Therefore, we include it as a new sequence in the set Θ. We then compute its corresponding score and centroid, and insert it in Θ.

We assume that the cluster quality is a significant property to guarantee an accurate detection of anomalies and the number of clusters has a direct impact on the execution time. As underlined in the previous paragraph, the threshold $\tau_{c,j}$ has a direct impact on the number of new clusters that will be created at each batch and on the cluster quality. On one hand, a high $\tau_{c,j}$ will make the cluster creation rare but might harm the cluster quality. On the other hand, a low $\tau_{c,j}$ will imply a large number of clusters created at each batch. Therefore, an optimal threshold that preserves the quality of the clusters without creating too many clusters is difficult to set for a user. Moreover, one threshold cannot be the same for every cluster. Therefore, to set automatically a threshold that adapts to clusters, we use the intra-cluster distance. For a given subsequence $\bar{C}_j$ in Θ, we compute the intra-cluster distance as follows:

$$\tau_{c,j} = \sum_{T_{i,\ell} \in C_j} SBD(T_{i,\ell}, \bar{C}_j)$$

We then use $\tau_{c,j}$ to decide if a new cluster should be created or not. In practice, we do not store the set C_j. If one cluster changes, we need to update the threshold dynamically. We thus store the size of Θ clusters only. Formally, for a sequence $\bar{C}_j$ in Θ and a given cluster C_i^t to be merged, we update the threshold $\tau^*_{c,j}$ as follows:

$$\tau^*_{c,j} \leftarrow \frac{|C_j| * \tau_{c,j}}{|C_j| + |C_i^t|} + \frac{|C_i^t| * \sum_{T_{m,\ell} \in C_i^t} SBD(T_{m,\ell}, \bar{C}_i^t)}{|C_j| + |C_i^t|}$$

Finally, the matching procedure is executed in Algorithm 2 and the threshold update is executed in Algorithm 3.

Algorithm 3: Θ parameters update: *computeParam*

input : A set Θ, a float α.
output: A set Θ

foreach $(\bar{C}_i, S_i, \tau_{c,i}, w_i, s_i) \in \Theta$ **do**
  $w_i^t \leftarrow \frac{|C_i|^2}{\sum_{\bar{C}_j \in \Theta} sbd(\bar{C}_j, \bar{C}_i)}$;
  if $(\bar{C}_i, S_i, \tau_{c,i}, w_i, s_i)$ *is a new cluster* **then**
    // Initialize the parameters for the new cluster C_i
    $w_i, s_i \leftarrow w_i^t, |C_i|$;
    $\tau_{c,i} \leftarrow \sum_{T_{m,\ell} \in C_i^t} sbd(T_{m,\ell}, \bar{C}_i^t)$;
  else
    // update the parameters for the merged cluster C_i
    $w_i \leftarrow (1-\alpha) * w_i + \frac{\alpha * w_i^t}{max(1, \mathcal{N}_i - b_{size})}$;
    $\tau_{c,i} \leftarrow \frac{s_i * \tau_{c,i}}{|C_i|} + \frac{(|C_i| - s_i) * \sum_{T_{m,\ell} \in C_i} sbd(T_{m,\ell}, \bar{C}_i^t)}{|C_i|}$;
    $s_i \leftarrow |C_i|$;
return Θ;

3.3.2 **Centroids Update**. At this point we matched the arrival subsequences (in the current batch) with existing sequences in Θ. Let's consider that the cluster C_i^t has been matched with the sequence $\bar{C}_j \in \Theta$. We update the sequence $\bar{C}_j$ as described in Equation 1. We note $\bar{C}_j{}^*$ the updated sequence $\bar{C}_j$. As mentioned earlier, we do not store the set C_j in memory. We thus need to compute the shape update process dynamically. As described in Equation 1, the shape update process is computed using the matrix S_j (of size ℓ_Θ^2). S_j is built by computing the dot product between all subsequences in the merged cluster $C_j \cup C_i^t$. Nevertheless, we can split the computation as follows:

$$S_j = \sum_{T_{m,\ell_\Theta} \in C_j} T_{m,\ell_\Theta}.T^T_{m,\ell_\Theta} + \sum_{T_{m,\ell_\Theta} \in C_i^t} T_{m,\ell_\Theta}.T^T_{m,\ell_\Theta}$$

The left part of the above sum is already computed from the previous batch. The only new computation to be performed is the right part. Therefore, we just need to update the matrix S_j by adding the sum of the dot product of the subsequences of the cluster C_i^t to be merged with $\bar{C}_j$. By doing so, updating the cluster shape does not require storing all the subsequences in memory, but just the matrix S_j. This results in a gain in memory space for large data series and execution time. Formally, the matrix S_j (and therefore $\bar{C}_j$) is updated as follows:

$$S_j^* \leftarrow S_j + \sum_{T_{m,\ell_\Theta} \in C_i^t} T_{m,\ell_\Theta}.T^T_{m,\ell_\Theta} \quad (3)$$

Note that we just need to compute the initial S_j for all initial clusters and store them in memory. We then compute the update $\bar{C}_j{}^*$ using the updated S_j^* at every new batch. The centroids computation and update are executed in Algorithm 2.

3.3.3 **Weights Update**. Once the sequences $\bar{C}_j$ are updated, we can update their corresponding weights w_j. One could decide to update the weights like in Equation 2 using their current statistics (cardinality and distance to other sequences in Θ). On the specific case of data series without any changes of normal behaviors, this would be the good decision. However, if a new behavior is detected, one should be able to forget the previous behaviors (or reduce its importance). For that purpose, we dynamically update the weight using its previous value. We introduce a user parameter α with

values between $[0, 1]$, such that it represents the rate of change. We note w_i^* the updated weight w_i, and is computed as follows:

$$w_i^t \leftarrow \frac{|C_i|^2}{\sum_{(C_j, w_j) \in \Theta} sbd(\bar{C_j}, \bar{C_i})}$$
$$w_i^* \leftarrow (1-\alpha) * w_i + \frac{\alpha * w_i^t}{max(1, \mathcal{A}_i - b_{size})} \quad (4)$$
$$\text{with: } \mathcal{A}_i = t - t_{last,i}$$

In the above definition, t is the time index of the current batch, and $t_{last,i}$ is defined as the temporal index of the latest (as regards to the time index) subsequence in cluster C_i. Note that as previously expressed C_i is not stored. However, it is trivial to count the number of subsequences ($|C_i|$), and to compute $t_{last,i}$ without storing C_i. Moreover, note that, at each iteration, we normalize the weight to have their sum equal to 1. As one can see, the new weight w_i^* will be a weighted mean (by alpha) between their old values w_i and their values at the current time w_i^t. If no new subsequences has been added to a cluster, then $w_i^t = w_i^{t-b_{size}}$. However it also means that this cluster might correspond to an old (and potentially irrelevant now) behavior. This has to be taken into account and we include a time decay component in the weight computation (as described in Equation 4). We have two different cases:

- $\mathcal{A}_i \leq b_{size}$: The cluster C_i contains at least one subsequence in the current batch. This means that the cluster is still active. Thus $max(1, \mathcal{A}_i - b_{size}) = 1$, and no time decay is applied.
- $\mathcal{A}_i > b_{size}$: The cluster C_i does not contain any subsequence from the current batch. This mean that the cluster might not be active anymore. We can thus start to apply some decay by dividing the current weight w_i^t by $\mathcal{A}_i - b_{size}$.

Depending on the value of α, the score of the clusters without any new subsequences will converge to zero, giving more importance to the currently activated clusters. In the specific case when an old behavior starts again to happen, its corresponding weight will increase faster (knowing that the cardinality of this cluster is already big). If one does not expect any old behavior to happen, one can decide to remove clusters with scores approximately equal to zero. In practice, this is more efficient in memory. Nevertheless, for the rest of this paper, we do not remove any cluster. The weight update is performed in Algorithm 3.

3.4 Anomaly Scoring

At this point, we can update the set Θ at every new batch. We now describe how we compute the score for all the subsequences inside a given batch. For a given subsequence $T_{j,\ell} \in \mathbb{T}_\ell$ in the current batch (of length $\ell < \ell_\Theta$), we compute the following score:

$$d_j = \sum_{\bar{C_i}} w^i * min_{x \in [0, \ell_\Theta - \ell]} \left\{ dist(T_{j,\ell}, (\bar{C_i})_{x,\ell}) \right\} \quad (5)$$

Even though the weights w^i are adjusted depending on the activity of their related clusters, a certain noise could be observed on the score. Based on $\bar{C_i}$ shape and its possible evolution, the score values distribution might evolve as well. We thus normalize the score at each batch. For a subsequence $T_{j,\ell} \in \mathbb{T}_\ell$ in the current batch $\mathbb{T}^t_{\ell_\Theta}$, we compute the normalization as $d_j^* = \frac{d_j - \mu_t^*}{\sigma_t^*}$, where μ_t and σ_t are the mean and the standard deviation of d_j over the batch $\mathbb{T}^t_{\ell_\Theta}$. We compute the estimated mean and standard deviation μ_t^* and σ_t^* as:

$$\mu_t^* \leftarrow \alpha * \mu_t + (1-\alpha) * \mu_{t-b_{size}}$$
$$\sigma_t^* \leftarrow \alpha * \sigma_t + (1-\alpha) * \sigma_{t-b_{size}} \quad (6)$$

At each batch, we use the previously updated mean and standard deviation to adapt the distance to the set Θ. Otherwise, in the unusual case when a batch contains a higher rate of anomalies than the previous (and future) batches, the normalization (without using the previous batch mean and standard deviation) may result in missing the anomalies in the batch. The score computation procedure is performed in Algorithm 4.

Algorithm 4: Score computation: *computeScore*

input : A set Θ, a subsequence $\mathbb{T}^t_{\ell_\Theta}$, a float α. two floats μ, σ
output: A data series $score$, two floats μ^*, σ^*

```
score ← [ ] foreach T_{j,ℓ} ∈ T^t_{ℓ_Θ} do
    // For each subsequence in the batch
    score[j] ← Σ_{C̄_i} w^i * min_{x∈[0,ℓ_Θ−ℓ]} {dist(T_{j,ℓ}, (C̄_i)_{x,ℓ})};
    score[j] ← (score[j] − μ) / σ;
// Update mean and variance
μ* ← α * μ(score) + (1 − α) * μ;
σ* ← α * σ(score) + (1 − α) * σ;
return score, μ*, σ*;
```

3.5 Execution Time Complexity Analysis

In this section, we analyze the execution time complexity of the various steps of the SAND framework. Note that the time complexity of the k-Shape algorithm is $O(max(|\mathbb{T}^t_{\ell_\Theta}| * k * \ell_\Theta * log(\ell_\Theta), |\mathbb{T}^0_{\ell_\Theta}| * \ell_\Theta^2, k * \ell_\Theta^3))$ per iteration, with $|\mathbb{T}^t_{\ell_\Theta}| = b_{size}$.

Initialisation Step: We now analyze the time complexity of the different steps separately. We first analyze the theoretical execution time needed to perform the initialization. The latter is composed of small computations related to the weights that is using the *sbd* distance between sequence of length ℓ_Θ with complexity $O(k^2 * \ell_\Theta log(\ell_\Theta))$. Thus, the complexity of the initialization step has the k-Shape algorithm as a bottleneck.

Batch Execution Time Complexity Analysis: At each batch, one need to compute the k-Shape algorithm. Then at each step we need to compute *sbd* distance between every new cluster and every sequence in Θ. The *SBD* computation complexity is $O(\ell_\Theta log(\ell_\Theta))$. Thus, the complexity of the first step is $O(|\Theta| * k * \ell_\Theta log(\ell_\Theta))$. As explained in the initialization section, the weight computation complexity is $O(k^2 * \ell_\Theta log(\ell_\Theta))$. Nevertheless, we always have $k \leq |\Theta|$, thus the weights computation step is negligible.

Then for a given cluster C_i, the shape update process require the computation of the matrices S_i and an eigendecomposition. Thus, the shape update operation has complexity $O(max(|C_i| * \ell_\Theta^2, \ell_\Theta^3))$ with $|C_i|$ being the number of subsequences in the cluster we want to extract the shape. We are storing previously computed matrices S_i in memory and we are computing the matrices S_i corresponding to the new clusters C_i^t. For a given time index t, $\sum_{C_i \in C} |C_i| = |\mathbb{T}^t_{\ell_\Theta}| = b_{size}$. Moreover, knowing that only k new clusters need to be merged, the shape update operation cannot be done more than k time. Thus the overall complexity of the shape update is $O(max(b_{size} * \ell_\Theta^2, k * \ell_\Theta^3))$. Therefore, this complexity does

not depend on the time evolution, and the execution time needed remains constant for the entire stream.

Scoring Execution Time Complexity Analysis: The anomaly distance between subsequences of length ℓ and the set Θ computation is defined by the computation of the equation in Equation 5, which is bounded by $O((b_{size} - \ell + 1) * \ell_\Theta * |\Theta|)$ using the Fourier transform to compute efficiently the correlation and distances over overlapping subsequences.

Overall complexity: We now analyze the overall complexity of the batch operation. Note that some operations share the same complexity, so can be grouped. Overall, the time complexity of the batch update operation is defined as follows:

$$SAND = max \left\{ \begin{array}{l} b_{size} * k * \ell_\Theta * log(\ell_\Theta) \\ 2 * b_{size} * \ell_\Theta^2 \\ 2 * k * \ell_\Theta^3 \\ |\Theta| * k * \ell_\Theta * log(\ell_\Theta) \\ (b_{size} - \ell + 1) * \ell_\Theta * |\Theta| \end{array} \right\} \quad (7)$$

One should note that the only parameter that varies while time evolves is the size of the set Θ (involved in the two last lines of Equation 7). Nevertheless, this evolution is most likely to be slow. Thus, the overall complexity does not depend on the time evolution and remains constant regardless of the increasing number of batches. The important parameters on the execution time are the batch size b_{size}, the number cluster k and the length ℓ_Θ. Moreover, we note that the complexity is linear to the size of the batch, which implies that SAND scales gracefully to large batches.

4 EXPERIMENTAL EVALUATION

In this section, we compare the performance of SAND against the SOTA subsequence anomaly detection methods in terms of accuracy and efficiency. We measure the ability of SAND to detect abnormal subsequences in a benchmark of real datasets. We confirm that SAND, despite operating in a streaming setting, performs similarly to SOTA (non-streaming) subsequence anomaly detection methods that operate over the entire dataset. We then demonstrate the shortcomings of the non-streaming methods, as well as the ability of SAND to adapt to changes of normality. Overall SAND outperforms existing streaming and non-streaming methods. We compare the scalability of SAND to SOTA streaming methods for anomaly detection when we vary different parameters, and show that SAND is an order of magnitude faster.Finally, we evaluate the influence of SAND's parameters on accuracy and execution time.

4.1 Experimental Settings

We implemented our algorithms Python 3.5, and used a server with Intel Xeon CPU E5-2650 2.20GHz with 250GB RAM. For reproducibility purposes, our code is available on this webpage [1].

Measures: We use the Precision@k as the accuracy measure. The latter is the ratio of correctly identified anomalies in the η subsequences corresponding to the η highest anomaly score. We set η as the number of anomalies in the datasets (as depicted in Table 2). Note that this parameter η is only used for evaluation purposes, and is not required for practical usage. We then use the throughput metric and the execution time in seconds to evaluate the scalability. The throughput is defined as the number of subsequences that can be handled in one second and corresponds to an upper-bound of the data acquisition speed of the method. The higher the throughput, the better the model will handle high-frequency data streams.

Datasets: We benchmark our system using real and synthetic datasets, for all of which a ground truth of annotated anomalies is available. We simulate streams by selecting static data series. We carefully select two specific types of data series. We first select several real data series. The first data series is a Simulated Engine Disks data (SED) from the NASA Rotary Dynamics Laboratory [3] representing disk revolutions recorded over several runs (3K rpm speed). We also included MIT-BIH Supraventricular Arrhythmia Database (MBA) [18, 34], which are electrocardiogram recordings from 4 patients, containing multiple instances of two different kinds of anomalies (either supraventricular contractions or premature heartbeats). All the previously enumerated dataset and their characteristics are grouped in Table 2.

To evaluate the capacity of our method to adapt to changes over time, we create synthetic datasets that contain more than one specific normality. We note them *Double*-Normality, *Triple*-Normality, and so on. We build them by concatenating our real single normality datasets. We thus perform two kinds of concatenation. We first concatenate different datasets from the same domain (i.e. two Electro-Cardiogram from two different patients), to evaluate methods to adapt to subtle changes. We then concatenate datasets from different domains (i.e. Electro-Cardiogram with SED) to evaluate methods to adapt to drastic changes.

Baselines: We first compare with four SOTA static methods (i.e., methods that take as input the entire data series): Isolation Forest (IF) [28], NormA [10], Series2Graph (S2G) [11], and STOMP [60]; as detailed below, we use the parameters suggested in the original papers. IF is a density/isolation-based method that aims to randomly isolate multi-dimensional points (subsequences in our case) by using a collection of random split trees: a point is considered abnormal if the distance to the root of these trees is short. We use 100 trees as explained in [28]. NormA is a distance-based method that aims to first build a set of subsequences that summarize typical subsequences in the dataset. It then computes distances to this set, and considers abnormal the subsequences with large distances. As described in [10], we use the default parameters for sampling rate, $r = 0.4$, and subsequence length, $\ell_{N_M} = 4 * \ell$. S2G detects unusual behavior by embedding a series in a directed graph, and evaluating the trajectories of subsequences on this graph. It uses as parameters local convolution $\lambda = 1/3 * \ell$, bandwidth h (set using Scott's rule [49]), and number of angles $r = 50$ [11]. STOMP is computing the nearest neighbor distance for every subsequence and use it to identify anomalies [60]. We finally compare SAND to NormA-mn, a variation of NormA, where we adapted the computation of the anomaly score based on the average anomaly score in a given window length: we compute the anomaly score of a subsequence $T_{i,\ell}$, and we subtract the average anomaly score of all subsequences within the interval $[i - 2 * \ell_\Theta, i + 2 * \ell_\Theta]$. This adaptation of the scoring step enables NormA to operate on multi-normality datasets.

We then compare SAND to dynamic methods (i.e., methods that receive subsequences of the data series incrementally). We first build baselines from the SOTA static methods called NormA-batch and S2G-batch, which operate locally (and independently) on each new arriving batch. We also compare SAND to two SOTA dynamic methods: IMondrian Forest [31] and STAMPI [60]. The first method

Datasets	Length
SED	100K
MBA (803)	100K
MBA (804)	100K
MBA (805)	100K
MBA (806)	100K
MBA (820)	100K

Table 2: Dataset characterist number of annotated anom

is an alternative to Isolation F (called Mondrian tree, origina pose [26]) with the characteri while new points (subsequences in our case) arrives. Similarly to Isolation Forest, we use 100 different trees. Similar to STOMP, the second approach is using the nearest neighbor distance to identify abnormal points. On the contrary to STOMP, this method has the specificity of being updatable incrementally. One can either keep track of all the previous points and update the distance profile until the end of the stream or can keep track of the distance profile over a fixed window length (called batch size in our case). We consider using a fixed window length equal to the batch size for the following reason: (i) keeping track of the entire distance profile requires a large amount of computation and make the latency increase quadratically. (ii) STAMPI (that is using the discord definition) can suffer from the fact that similar anomalies can happen in a stream. Therefore, keeping the old distance profile might lead to a higher rate of false-negative than keeping only a fixed window length. The second point is confirmed by Table 3. For SAND, we set the additional parameters as follows: $\alpha = 0.5$, $\ell_\Theta = 4 * \ell$ and $k = 6$.

4.2 Accuracy Evaluation

4.2.1 **Clustering accuracy evaluation**. In this section, we evaluate the clustering accuracy of our extension of the k-Shape algorithm. For that purpose, we use all the UCR datasets [17]. We run the original k-Shape on the entire dataset, and we run incrementally our extended k-Shape on 1/10th of the dataset at each new batch. Figure 3(a) depicts the rand score [41] accuracy comparison between the original and our extended k-Shape algorithm. We observe that our extended k-Shape has a similar rand score on average as the usual k-Shape. Thus our extended version of k-Shape algorithm provides a way to use the original k-Shape for streaming scenarios with the same accuracy. Moreover, Figure 3(b) depicts the average execution time (in seconds) for a simple solution for incremental k-Shape (i.e., storing all subsequences to compute the centroids from scratch at every new batch), and our proposed solution for incremental k-Shape (i.e., without storing all the subsequences). The results confirm the benefits of our proposed solution for an incremental k-Shape algorithm.

4.2.2 **SAND accuracy evaluation**. We now compare the Precision@k of our method with several other methods, both static (without any update mechanism at every new batch or point) and dynamic (that includes an update mechanism at every new batch or point). All methods share the subsequence length ℓ as the main parameter. For each dataset, we set the subsequence length $\ell = \ell_A$ as shown in Table 2. We use a batch size of 5000 points for IMondrian

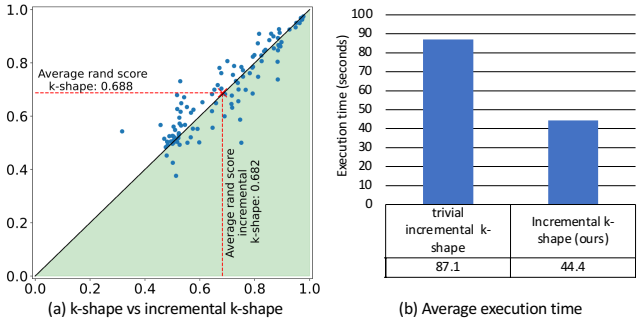


Figure 3: Comparison between the original and our extended k-Shape on rand score (a), and execution time (b).

Forest, STAMPI, and our approach. Table 3 depicts the Precision@k accuracy of the aforementioned methods for our datasets corpus. One can notice that for single normality datasets, static methods NormA, Isolation Forest, and Series2Graph (that are using the entire series to build their model), have good performances. Nevertheless, online methods IMondrian Forest and SAND are still performing well by being slightly less significant than the static method. STAMPI has a medium Precision@k due to the limitations caused by similar anomalies in the single normality datasets. Nevertheless, STAMPI has better performance than STOMP, which confirms that using a fixed sliding window limits the number of similar anomalies and provides better accuracy. Finally, we note that SAND outperforms both STAMPI and IMondrian Forest, and performs equal or very close to Series2Graph and NormA. Moreover, one can notice that the online adaptation of NormA and S2G (NormA-batch and S2G-batch) are not performing as well as their respective static version for the single normality datasets.

Regarding double normality datasets, observe that the static methods suffer from a significant drop in accuracy, while the online methods have on average similar Precision@k accuracy to the single normality datasets. However, we note that both NormA-batch and S2G-batch are performing better than NormA and S2G for double normality datasets from different domains, but not for datasets from the same domain. Only NormA-mn is robust to multiple normality datasets, but SAND is still more accurate. Moreover, SAND outperforms both IMondrian Forest and STAMPI. More precisely, SAND significantly outperforms the competitors (both static and online) for double normality datasets created by concatenated datasets of different domains. This underlines the superior adaptability of SAND, regardless of the dataset composition.

The same observation can be made for triple, quadruple, and quintuple normality datasets, for which the Precision@k contrast between SAND and the other state-of-the-art methods is even stronger. More generally, one should underline that static methods (Isolation Forest, Series2Graph, and NormA) see their Precision@k dropping while the number of different normality increase, in opposite to online methods (IMondrian Forest, STAMPI, and SAND) that seems to be more stable. Only STOMP seems to have a stable accuracy while the number of normality sections increases.

A careful look can be addressed to Series2Graph performance on multiple normality datasets. Similar to NormA and Isolation Forest, one can notice an accuracy drop. Nevertheless, this drop

	Static					*Streaming*				
Data Series	**NormA**	**IF**	**STOMP**	**S2G**	**NormA-mn**	**NormA-batch**	**S2G-batch**	**IMondrian F**	**STAMPI**	**SAND**
Single Normality (100,000 points)										
MBA(803)	0.99(0.01)	**1.00(0.00)**	0.72	**1.00**	0.97(0.01)	0.70(0.07)	0.97	0.99(0.01)	0.46	0.97(0.02)
MBA(805)	**0.99(0.00)**	**0.99(0.01)**	0.10	**0.99**	**0.99(0.00)**	0.82(0.05)	0.90	0.96(0.02)	0.35	0.98(0.01)
MBA(806)	0.86(0.02)	0.75(0.06)	0.59	**1.00**	0.88(0.02)	0.74(0.07)	0.70	0.85(0.03)	0.66	0.80(0.03)
MBA(820)	**0.98(0.01)**	0.92(0.03)	0.90	0.91	0.97(0.00)	0.74(0.08)	0.71	0.95(0.02)	0.84	0.96(0.01)
SED	0.92(0.05)	0.65(0.02)	0.57	**1.00**	0.98(0.01)	0.80(0.06)	0.90	0.31(0.05)	0.87	0.96(0.00)
Average	0.95	0.86	0.58	**0.98**	0.96	0.76	0.84	0.81	0.64	0.93
Double Normality (200,000 points)										
Same Domain										
MBA(803 + 805)	0.91(0.10)	0.53(0.03)	0.32	**0.99**	0.95(0.02)	0.76(0.03)	0.94	0.97(0.01)	0.40	0.94(0.01)
MBA(803 + 806)	0.70(0.01)	0.75(0.00)	0.58	0.75	0.89(0.03)	0.68(0.02)	0.84	0.87(0.02)	0.52	**0.96(0.02)**
MBA(803 + 820)	0.83(0.27)	0.75(0.05)	0.78	0.76	0.92(0.01)	0.66(0.04)	0.81	**0.93(0.01)**	0.67	0.88(0.00)
MBA(805 + 806)	0.74(0.00)	0.68(0.04)	0.20	0.73	0.85(0.01)	0.76(0.06)	0.41	0.81(0.04)	0.44	**0.95(0.01)**
MBA(805 + 820)	0.76(0.03)	0.49(0.04)	0.51	0.51	**0.97(0.01)**	0.71(0.02)	0.71	0.94(0.01)	0.61	0.90(0.02)
MBA(806 + 820)	0.77(0.02)	0.78(0.00)	0.83	0.81	0.92(0.01)	0.75(0.02)	0.20	0.51(0.04)	0.79	**0.93(0.01)**
Average	0.78	0.66	0.54	0.76	0.92	0.72	0.65	0.84	0.57	**0.93**
Different Domains										
MBA(803) + SED	0.56(0.19)	0.45(0.00)	0.67	0.06	0.60(0.14)	0.72(0.02)	0.84	0.65(0.08)	0.64	**0.96(0.01)**
MBA(805) + SED	0.69(0.20)	0.37(0.01)	0.30	0.11	0.87(0.05)	0.84(0.03)	0.41	0.68(0.09)	0.57	**0.95(0.02)**
MBA(806) + SED	0.74(0.05)	0.57(0.01)	0.62	0.07	**0.84(0.01)**	0.79(0.02)	0.72	0.46(0.04)	0.79	0.80(0.03)
MBA(820 + SED	0.91(0.03)	0.38(0.01)	0.82	0.10	**0.92(0.02)**	0.72(0.06)	0.40	0.52(0.01)	0.85	0.88(0.00)
Average	0.72	0.44	0.60	0.09	0.81	0.77	0.59	0.58	0.71	**0.90**
Triple Normality (300,000 points)										
Same Domain										
MBA(803 + 805 + 806)	0.84(0.00)	0.43(0.02)	0.37	0.82	0.84(0.01)	0.71(0.04)	0.59	0.82(0.01)	0.44	**0.92(0.03)**
MBA(803 + 805 + 820)	0.60(0.23)	0.37(0.02)	0.54	0.63	0.86(0.06)	0.66(0.02)	0.79	**0.95(0.00)**	0.56	0.88(0.02)
MBA(803 + 806 + 820)	0.83(0.00)	0.68(0.05)	0.74	0.67	0.82(0.04)	0.67(0.06)	0.47	0.69(0.06)	0.66	**0.91(0.01)**
MBA(805 + 806 + 820)	0.60(0.12)	0.41(0.02)	0.53	0.44	0.85(0.02)	0.65(0.05)	0.25	0.70(0.02)	0.61	**0.94(0.00)**
Average	0.72	0.47	0.54	0.64	0.84	0.69	0.52	0.79	0.57	**0.91**
Different Domains										
MBA(803 + 805) + SED	0.60(0.13)	0.26(0.00)	0.41	0.10	0.60(0.12)	0.77(0.03)	0.52	0.75(0.02)	0.53	**0.95(0.00)**
MBA(803 + 806) + SED	0.67(0.14)	0.34(0.01)	0.62	0.06	0.67(0.04)	0.70(0.03)	0.73	0.67(0.05)	0.64	**0.88(0.00)**
MBA(803 + 820) + SED	0.60(0.12)	0.26(0.00)	0.75	0.10	0.64(0.02)	0.64(0.06)	0.52	0.66(0.01)	0.72	**0.87(0.01)**
MBA(805 + 806) + SED	0.75(0.11)	0.31(0.00)	0.35	0.09	**0.79(0.02)**	0.78(0.02)	0.40	0.66(0.05)	0.59	0.76(0.01)
MBA(805 + 820) + SED	0.62(0.06)	0.24(0.00)	0.54	0.09	**0.94(0.01)**	0.71(0.03)	0.26	0.73(0.01)	0.67	0.91(0.02)
MBA(806 + 820) + SED	0.37(0.11)	0.31(0.00)	0.78	0.10	0.82(0.02)	0.74(0.05)	0.39	0.40(0.08)	0.81	**0.86(0.00)**
Average	0.60	0.28	0.58	0.09	0.74	0.72	0.47	0.65	0.66	**0.87**
Quadruple Normality (400,000 points)										
Same Domain										
MBA(803 + 805 + 806 + 820)	0.86(0.01)	0.32(0.02)	0.53	0.57	0.86(0.03)	0.67(0.05)	0.42	0.70(0.01)	0.57	**0.95(0.00)**
Different Domains										
MBA(803 + 805 + 806) + SED	0.47(0.03)	0.23(0.00)	0.44	0.10	0.74(0.06)	0.74(0.01)	0.50	0.72(0.01)	0.55	**0.91(0.01)**
MBA(803 + 806 + 820) + SED	0.30(0.10)	0.23(0.00)	0.71	0.09	0.60(0.27)	0.67(0.04)	0.50	0.58(0.02)	0.71	**0.85(0.04)**
MBA(805 + 806 + 820) + SED	0.54(0.01)	0.21(0.00)	0.55	0.09	0.55(0.30)	0.66(0.03)	0.28	0.60(0.01)	0.67	**0.90(0.02)**
Average	0.43	0.22	0.57	0.09	0.63	0.69	0.43	0.63	0.64	**0.89**
Quintuple Normality (500,000 points)										
Different Domains										
MBA(803 + 805 + 806 + 820) + SED	0.40(0.01)	0.16(0.05)	0.55	0.08	0.81(0.05)	0.67(0.04)	0.38	0.69(0.02)	0.62	**0.90(0.00)**

Table 3: Precision@k accuracy for NormA (and NormA-batch), Isolation Forest (IF), STOMP, S2G (and S2G-batch), IMondrian Forest, STAMPI, and SAND applied to our datasets corpus (including concatenations of different datasets from same and different domains). The standard deviation of 10 runs is reported in parentheses.

is significantly bigger for datasets concatenated from different domains. This underlines a limitation of Series2Graph to data series that have a strong heterogeneous range of values through time. The embedding space needs to be changed to adapt to this specific case.

To conclude, SAND has equivalent accuracy for single normality datasets with state-of-the-art static methods and significantly outperforms both static and online state-of-the-art methods for datasets containing changes of normality. To assess the significance of these differences, Figure 4 depicts a critical difference diagram computed using a Wilcoxon pair-wised signed-rank test (with $\alpha = 0.05$) on single and multiple normality datasets. Overall, Figure 4(a) underlines that SAND achieves the highest rank of all methods and statistically outperforms all previous static state-of-the-art methods (i.e., thick lines connect methods performing similarly; SAND outperforms methods with statistically significant differences). Similarly, Figure 4(b) confirms that SAND outperforms online state-of-the-art methods as well.

4.3 Time Performance Evaluation

In this section, we evaluate the scalability of our methods and the streaming state-of-the-art methods analyzed in the previous section. For that purpose, we first evaluate the global execution time (in seconds) needed to perform the update operation of the model, and the subsequences scoring (in batch). We then measure the throughput when we vary different parameters.

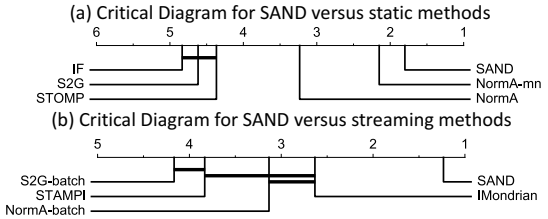


Figure 4: Critical difference diagrams using a Wilkoxon pairwised signed rank test (with $\alpha = 0.05$) on both single and multiple normality datasets.

We first measure the execution time needed for the model update and batch scoring. Figure 5 presents results on the double normality datasets (results with single normality datasets exhibit similar trends; we omit them for brevity). Figure 5(d) depicts the global execution time for the model update and the batch scoring for SAND (in blue), IMondrian Forest (in red), and STAMPI (in green). One should note that STAMPI performs both the model update and the batch scoring at the same time. We thus report zero as the model update execution time of STAMPI. Nevertheless, SAND is significantly faster than IMondrian Forest for the model update operation, up to three orders of magnitude faster for the batch scoring operation. Overall, the total execution time (represented by the dotted lines) of SAND is significantly lower than the two other methods.

We now measure the influence of the batch size on throughput (using a fixed subsequence length of 75). Figure 5(a) illustrates with the dotted lines the standard deviation envelope over all the double normality datasets. SAND throughput remains stable as the batch size increase. This confirms our expectation (cf. Section 3.5). On the contrary, STAMPI and IMondrian forest throughput is decreasing. Thus, SAND has a significantly higher throughput for large batches (>5000 subsequences).

We then measure the influence of the subsequence length on the throughput (while keeping an almost constant batch size of approximately 20,000 subsequences). The latter is illustrated in Figure 5(b). We notice that the throughput of SAND and IMondrian is reducing when the subsequence length is increasing. This has been predicted by the high importance of the length in Equation 7. On the contrary, STAMPI throughput is constant whatever the value of the subsequence length. We notice that the throughput of IMondrian Forest is equivalent to the throughput of STAMPI for subsequence length of 1000 points for double normality datasets. Overall, SAND throughput is significantly higher for small subsequence length (up to 200 points) and remains higher for subsequence length up to 1000 points. We observe that most of the subsequence anomalies are of moderate size (e.g., for cardiology datasets [18, 34], abnormal heartbeats are usually shorter than 200 points), and rarely measure up to 1000 points (e.g., electrical consumption datasets, patient respiration, and Space Shuttle Marotta Valve [23, 51]).

We finally investigate the evolution of throughput as the stream evolves (we use 10,000 as batch size, and 75 as subsequence length). Figure 5(c) shows the throughput of SAND, IMondrian Forest, and STAMPI at each batch (represented with its index in time on the x-axis). We note that, despite the variability between different datasets (represented by the dotted line envelope), SAND's throughput is relatively constant (this is true for both multi-normality and single normality datasets). This confirms the statement made in Section 3.5, and proves that the size evolution of $|\Theta|$ does not affect execution time. Similarly, STAMPI throughput is constant over time. On the contrary, IMondrian Forest's throughput is reducing over time. We observe a perturbation on its throughput for double normality datasets, taking place when the normality changes (at index 100,000). At that point, the insertion of elements in the Mondrian trees is faster, because these elements are significantly different (larger distances) than current elements and, therefore, the IMondrian Forest throughput is increasing. Nevertheless, it starts rapidly to decrease on average. Overall, SAND throughput is constantly one order of magnitude larger than IMondrian Forest and STAMPI.

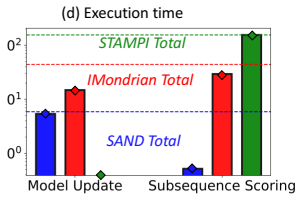
4.4 Parameter Tuning

Recall that there are only four main parameters (ℓ_Θ,k, b_{size}, and α) that may affect the anomaly detection accuracy. In this section, we evaluate and analyze their behavior. Note that two parameters can *jointly* influence the accuracy and, therefore, we vary two parameters simultaneously. We first start by analyzing parameters influences independently. Figure 7 depicts the Precision@k (Fig. 7(1)), the execution time in seconds to compute a batch (Fig. 7(2)) and the final number of clusters created (Fig. 7(3)) with a color range between black and yellow (with black the lowest and yellow the highest) for the double-normality datasets (results for single-normality are omitted for brevity).

Influence of the centroids length, ℓ_Θ: We first evaluate the parameter ℓ_Θ as regards to the parameter ℓ. We define $\ell_\Theta = a * \ell$. Figure 6 depicts the Precision@k and the execution time when a varies between 1 and 10 for double normality datasets (results for single-normality datasets follow similar trends and are omitted for brevity). For accuracy purposes, ℓ_Θ should be greater than two times ℓ. Above $a > 2$, the Precision@k is reaching its maximum value and stays constant. Thus, for a value of a above a given value, the length of the centroids does not have a significant impact on the accuracy. Moreover, as explained in Section 3.5, the centroids length does have an impact on the execution time. Thus, one needs to choose a large enough length to maximize the accuracy without increasing too much the execution time. We pick $\ell_\Theta = 4 * \ell$.

Influence of Initial Number of Clusters, k: We analyze the influence of the SAND initial number of cluster k (independently) on the detection accuracy and the execution time per batch. The y-axis of Figure 7(b) depicts the evolution of the accuracy and execution time per batch when we vary the initial number of clusters k. Note that the other parameter is set to its default value ($b_{size} = 5000$). We notice that the parameter k does not have any impact on the detection accuracy. Nevertheless, increasing k leads to a higher number of clusters after the final batch and a higher execution time per batch (as theoretically explained in Section 3.5). Therefore, a low initial number of clusters seems to be an optimal choice. In our experiments, we pick $k = 6$.

Influence of Batch Size: We then measure the influence of the batch size on accuracy and execution time. The y-axis of Figure 7(c) depicts the accuracy and execution time when we vary the batch size from 2000 points (i.e., subsequences) to 40,000. We note that a bigger batch requires more execution time. Nevertheless, we show



(**with fixed batch size** b_{size} =

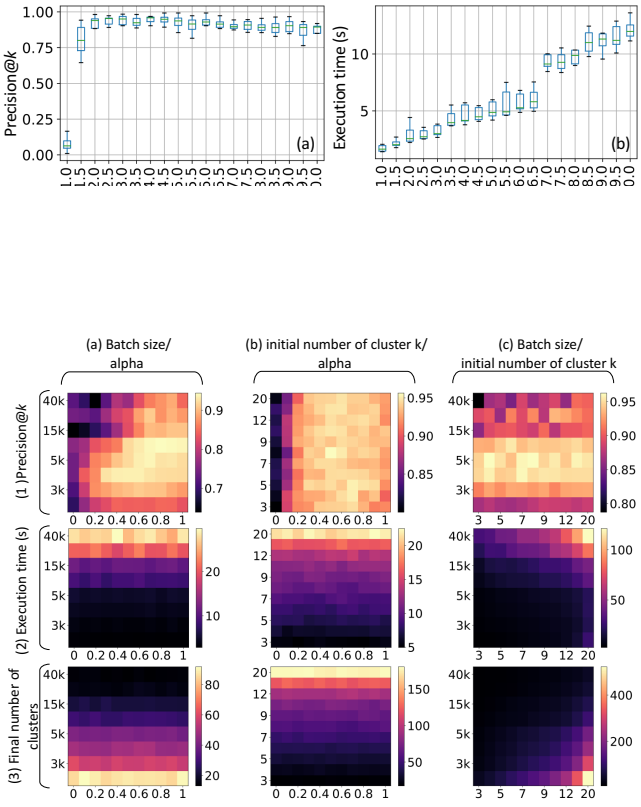


Figure 7: Influence of batch size b_{size}, rate of change α, and initial number of clusters k on accuracy (1st line), execution time (2nd line) and final number of clusters created (3rd line), over all double normality datasets.

in Section 4.3 that throughput remains constant to this parameter. We observe a drop in accuracy for small batches, but we also notice a slow reduction of accuracy while the batches increase. In this case, there is a change of normality in the middle of the data series, thus small batches are more able to adapt to this change. We select the nearly optimal choice of b_{size} = 5000 for our experiments.

Influence of α: We measure the impact of α on accuracy and execution time (see Figure 7(b)). As expected, α does not have an impact, neither on the number of clusters created, nor on execution time. However, when normality changes, the model needs to adapt fast enough. Thus, a high α leads to a more accurate result (for the single normality datasets, the impact on accuracy slowly decrease as α increases, since no adaptation is needed for these datasets).In our experiments, we use $\alpha = 0.5$, which provides good accuracy in both single and double normality datasets.

Influence of Batch Size and α: We now evaluate the influence of the batch size joined with α. As previously underlined, the execution time and the number of clusters created are independent to α, but only dependent on the batch size. We note that a high values of batch size joined with low values of α implies low accuracy for double normality datasets (Fig. 7(a.1)). On the contrary, we observe that a high value of alpha joined with a low value of batch size implies a lower accuracy for single normality datasets. Overall, parameters that are on the diagonal (such as b_{size} = 5000 and $\alpha = 0.5$) are optimal.

Influence of Initial Number of Clusters k and α: We then evaluate the initial number of clusters k joined with α (Figure 7(b)). As mentioned earlier, the execution time and the number of clusters created are independent of α. Moreover, for the double normality datasets, the initial number of clusters k does not have any impact on accuracy, while α does.

Influence of Batch Size and Initial Number of Clusters k: Finally, we measure the batch size joined with the initial number of cluster k influence (see Figure 7(c)). One can see that the combination of small-batch size and a high k leads to a significant number of clusters, and large batch sizes combined with a high initial number of clusters leads to a high execution time per batch. With regards to accuracy, only the batch size has an impact.

5 CONCLUSIONS

We described SAND, a novel unsupervised approach for subsequence anomaly detection in streaming sequences. SAND is based on a set representation of the subsequences in a data stream, and can detect both single and recurrent anomalies. Experiments with several synthetic and real datasets demonstrate the benefits of our approach in terms of efficiency and accuracy.

In future work, we plan to investigate techniques that can improve the scalability of our approaches, such as employing parallel [46–48], or deep-learned [56] solutions.

ACKNOWLEDGMENTS

We thank M. Meftah and E. Remy, from EDF R&D, for their support and advice, and the reviewers for the numerous constructive comments. Work supported by EDF R&D, ANRT French program, NetApp, Cisco, and Exelon Utilies.

REFERENCES

[1] 2020. *SAND Webpage: http://helios.mi.parisdescartes.fr/~themisp/SAND/*.

[2] D. Abboud, M. Elbadaoui, W.A. Smith, and R.B. Randall. 2019. Advanced bearing diagnostics: A comparative study of two powerful approaches. *MSSP* 114 (2019).

[3] Ali Abdul-Aziz, Mark R Woike, Nikunj C Oza, Bryan L Matthews, and John D lekki. 2012. Rotor health monitoring combining spin tests and data-driven anomaly detection methods. *Structural Health Monitoring* (2012). https://doi.org/10.1177/1475921710395811

[4] Jerome Antoni and Pietro Borghesani. 2019. A statistical methodology for the design of condition indicators. *Mechanical Systems and Signal Processing* (2019).

[5] Anthony J. Bagnall, Richard L. Cole, Themis Palpanas, and Konstantinos Zoumpatianos. 9(7), 2019. Data Series Management (Dagstuhl Seminar 19282). *Dagstuhl Reports* (9(7), 2019).

[6] S. Bahaadini, V. Noroozi, N. Rohani, S. Coughlin, M. Zevin, J. R. Smith, Vicky Kalogera, and Aggelos K Katsaggelos. 2018. Machine learning for Gravity Spy: Glitch classification and dataset. *Information Sciences* 444 (1 5 2018).

[7] V. Barnet and T. Lewis. 1994. *Outliers in Statistical Data*. John Wiley and Sons, Inc.

[8] Paul Boniol, Michele Linardi, Federico Roncallo, and Themis Palpanas. 2020. Automated Anomaly Detection in Large Sequences. In *36th IEEE International Conference on Data Engineering, ICDE 2020, Dallas, TX, USA, April 20-24, 2020*.

[9] Paul Boniol, Michele Linardi, Federico Roncallo, and Themis Palpanas. 2020. SAD: An Unsupervised System for Subsequence Anomaly Detection. In *ICDE*.

[10] Paul Boniol, Michele Linardi, Federico Roncallo, Themis Palpanas, Mohammed Meftah, and Emmanuel Remy. 2021. Unsupervised and scalable subsequence anomaly detection in large data series. *The VLDB Journal* (2021).

[11] Paul Boniol and Themis Palpanas. 2020. Series2Graph: Graph-based Subsequence Anomaly Detection for Time Series. *PVLDB* 13, 11 (2020).

[12] Paul Boniol, Themis Palpanas, Mohammed Meftah, and Emmanuel Remy. 2020. GraphAn: Graph-based Subsequence Anomaly Detection. *Proc. VLDB Endow.* 13, 12 (2020), 2941–2944.

[13] Markus M. Breunig, Hans-Peter Kriegel, Raymond T. Ng, and Jörg Sander. 2000. LOF: Identifying Density-based Local Outliers. In *SIGMOD*.

[14] Yingyi Bu, Oscar Tat-Wing Leung, Ada Wai-Chee Fu, Eamonn J. Keogh, Jian Pei, and Sam Meshkin. 2007. WAT: Finding Top-K Discords in Time Series Database. In *SIAM*.

[15] Bill Yuan Chiu, Eamonn J. Keogh, and Stefano Lonardi. 2003. Probabilistic discovery of time series motifs. In *SIGKDD 2003*. 493–498.

[16] Nassia Daouayry, Ammar Mechouche, Pierre-Loic Maisonneuve, Vasile-Marian Scuturici, and Jean-Marc Petit. 2019. Data-Centric Helicopter Failure Anticipation: The MGB Oil Pressure Virtual Sensor Case. IEEE BigData.

[17] H. A. Dau, A. Bagnall, K. Kamgar, C. M. Yeh, Y. Zhu, S. Gharghabi, C. A. Ratanamahatana, and E. Keogh. 2019. The UCR time series archive. *IEEE/CAA Journal of Automatica Sinica* 6, 6 (2019), 1293–1305. https://doi.org/10.1109/JAS.2019.1911747

[18] Goldberger et al. [n.d.]. PhysioBank, PhysioToolkit, and PhysioNet. *Circulation* ([n. d.]). http://circ.ahajournals.org/content/101/23/e215

[19] Yan Zhu et al. [n.d.]. Matrix Profile II: Exploiting a Novel Algorithm and GPUs to Break the One Hundred Million Barrier for Time Series Motifs and Joins. In *ICDM 2016*.

[20] Ada Wai-Chee Fu, Oscar Tat-Wing Leung, Eamonn J. Keogh, and Jessica Lin. 2006. Finding Time Series Discords Based on Haar Transform. In *ADMA*.

[21] Sudipto Guha, Adam Meyerson, Nina Mishra, Rajeev Motwani, and Liadan O'Callaghan. 2003. Clustering Data Streams: Theory and Practice. *TKDE* 15, 3 (2003).

[22] Medina Hadjem, Farid Naït-Abdesselam, and Ashfaq A. Khokhar. 2016. ST-segment and T-wave anomalies prediction in an ECG data using RUSBoost. In *Healthcom*.

[23] E. Keogh, J. Lin, and a. Fu. 2005. HOT SAX: Efficiently Finding the Most Unusual Time Series Subsequence. *(ICDM)* (2005). https://doi.org/10.1109/ICDM.2005.79

[24] Eamonn Keogh, Stefano Lonardi, Chotirat Ann Ratanamahatana, Li Wei, Sang-Hee Lee, and John Handley. 2007. Compression-based data mining of sequential data. *Data Mining and Knowledge Discovery* (2007).

[25] M. Kontaki, A. Gounaris, A. N. Papadopoulos, K. Tsichlas, and Y. Manolopoulos. 2011. Continuous monitoring of distance-based outliers over data streams. In *2011 IEEE 27th International Conference on Data Engineering*. 135–146. https://doi.org/10.1109/ICDE.2011.5767923

[26] Balaji Lakshminarayanan, Daniel M. Roy, and Yee Whye Teh. 2015. Mondrian Forests: Efficient Online Random Forests. arXiv:1406.2673 [stat.ML]

[27] Michele Linardi, Yan Zhu, Themis Palpanas, and Eamonn J. Keogh. 2020. Matrix Profile Goes MAD: Variable-Length Motif And Discord Discovery in Data Series. In *DAMI*.

[28] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. 2008. Isolation Forest. In *ICDM (ICDM)*.

[29] Yubao Liu, Xiuwei Chen, and Fei Wang. 2009. Efficient Detection of Discords for Time Series Stream. *Advances in Data and Web Management* (2009).

[30] Wei Luo and Marcus Gallagher. 2011. Faster and Parameter-Free Discord Search in Quasi-Periodic Time Series. In *Advances in Knowledge Discovery and Data Mining*.

[31] Haoran Ma, Benyamin Ghojogh, Maria N. Samad, Dongyu Zheng, and Mark Crowley. 2020. Isolation Mondrian Forest for Batch and Online Anomaly Detection. arXiv:2003.03692 [cs.LG]

[32] Pankaj Malhotra, Lovekesh Vig, Gautam Shroff, and Puneet Agarwal. 2015. Long Short Term Memory Networks for Anomaly Detection in Time Series. (2015).

[33] Katsiaryna Mirylenka, Alice Marascu, Themis Palpanas, Matthias Fehr, Stefan Jank, Gunter Welde, and Daniel Groeber. 2013. Envelope-Based Anomaly Detection for High-Speed Manufacturing Processes. *European Advanced Process Control and Manufacturing Conference* (2013).

[34] G. B. Moody and R. G. Mark. 2001. The impact of the MIT-BIH Arrhythmia Database. *IEEE Engineering in Medicine and Biology Magazine* (2001).

[35] Abdullah Mueen, Eamonn J. Keogh, Qiang Zhu, Sydney Cash, and M. Brandon Westover. [n.d.]. Exact Discovery of Time Series Motifs. In *SDM 2009*.

[36] M. Munir, S. A. Siddiqui, A. Dengel, and S. Ahmed. 2019. DeepAnT: A Deep Learning Approach for Unsupervised Anomaly Detection in Time Series. *IEEE Access* 7 (2019), 1991–2005. https://doi.org/10.1109/ACCESS.2018.2886457

[37] Themis Palpanas. 2013. Real-Time Data Analytics in Sensor Networks. In *Managing and Mining Sensor Data*. 173–210.

[38] Themis Palpanas. 2015. Data Series Management: The Road to Big Sequence Analytics. *SIGMOD Rec.* 44, 2 (2015), 47–52.

[39] Themis Palpanas and Volker Beckmann. 2019. Report on the First and Second Interdisciplinary Time Series Analysis Workshop (ITISA). *SIGMOD Rec.* 48, 3 (2019).

[40] John Paparrizos and Michael J Franklin. 2019. GRAIL: Efficient Time-Series Representation Learning. *PVLDB* 12, 11 (2019), 1762–1777.

[41] John Paparrizos and Luis Gravano. 2015. k-Shape: Efficient and Accurate Clustering of Time Series. In *SIGMOD*. 1855–1870.

[42] John Paparrizos and Luis Gravano. 2017. Fast and accurate time-series clustering. *TODS* 42, 2 (2017), 1–49.

[43] John Paparrizos, Chunwei Liu, Bruno Barbarioli, Johnny Hwang, Ikraduya Edian, Aaron J. Elmore, Michael J. Franklin, and Sanjay Krishnan. 2021. VergeDB: A Database for IoT Analytics on Edge Devices. In *CIDR*.

[44] John Paparrizos, Chunwei Liu, Aaron J. Elmore, and Michael J. Franklin. 2020. Debunking Four Long-Standing Misconceptions of Time-Series Distance Measures. In *SIGMOD*. 1887–1905.

[45] Tuomas Pelkonen, Scott Franklin, Paul Cavallaro, Qi Huang, Justin Meza, Justin Teller, and Kaushik Veeraraghavan. 2015. Gorilla: A Fast, Scalable, In-Memory Time Series Database. *PVLDB* 8, 12 (2015).

[46] Botao Peng, Panagiota Fatourou, and Themis Palpanas. 2021. Fast Data Series Indexing for In-Memory Data. *VLDBJ* (2021).

[47] Botao Peng, Panagiota Fatourou, and Themis Palpanas. 2021. SING: Sequence Indexing Using GPUs. In *ICDE*.

[48] Botao Peng, Themis Palpanas, and Panagiota Fatourou. 2020. ParIS+: Data Series Indexing on Multi-core Architectures. *TKDE* (2020).

[49] D. W. Scott. 1992. *Multivariate Density Estimation. Theory, Practice, and Visualization*. Wiley.

[50] Pavel Senin, Jessica Lin, Xing Wang, Tim Oates, Sunil Gandhi, Arnold P. Boedihardjo, Crystal Chen, and Susan Frankenstein. 2015. Time series anomaly discovery with grammar-based compression. In *EDBT*.

[51] Pavel Senin, Jessica Lin, Xing Wang, Tim Oates, Sunil Gandhi, Arnold P. Boedihardjo, Crystal Chen, and Susan Frankenstein. 2018. GrammarViz 3.0: Interactive Discovery of Variable-Length Time Series Patterns. *TKDD* (2018).

[52] Sharmila Subramaniam, Themis Palpanas, Dimitris Papadopoulos, Vana Kalogeraki, and Dimitrios Gunopulos. 2006. Online Outlier Detection in Sensor Data Using Non-Parametric Models. In *Proceedings of the 32nd International Conference on Very Large Data Bases, Seoul, Korea, September 12-15, 2006*.

[53] Luan Tran, Liyue Fan, and Cyrus Shahabi. 2016. Distance-Based Outlier Detection in Data Streams. *Proc. VLDB Endow.* 9, 12 (Aug. 2016), 1089–1100.

[54] Dafne van Kuppevelt, Vincent van Hees, and Christiaan Meijer. 2017. *PAMAP2 dataset preprocessed v0.3.0*. https://doi.org/10.5281/zenodo.834467

[55] J. Wang, A. Balasubramanian, L. Mojica de la Vega, J. Green, A. Samal, and B. Prabhakaran. [n.d.]. Word recognition from continuous articulatory movement time-series data using symbolic representations. In *SLPAT* (2013).

[56] Qitong Wang and Themis Palpanas. 2021. Deep Learning Embeddings for Data Series Similarity Search. In *SIGKDD*.

[57] CW Whitney, DJ Gottlieb, S Redline, RG Norman, RR Dodge, E Shahar, S Surovec, and FJ Nieto. 1998. Reliability of scoring respiratory disturbance indices and sleep staging. *Sleep* (November 1998).

[58] Dragomir Yankov, Eamonn J. Keogh, Jose Medina, Bill Yuan-chi Chiu, and Victor B. Zordan. [n.d.]. Detecting time series motifs under uniform scaling. In *ACM*.

[59] Dragomir Yankov, Eamonn J. Keogh, and Umaa Rebbapragada. 2007. Disk Aware Discord Discovery: Finding Unusual Time Series in Terabyte Sized Datasets. In *ICDM*.

[60] Chin-Chia Michael Yeh, Yan Zhu, Liudmila Ulanova, Nurjahan Begum, Yifei Ding, Hoang Anh Dau, Diego Furtado Silva, Abdullah Mueen, and Eamonn J. Keogh. 2016. Matrix Profile I: All Pairs Similarity Joins for Time Series. In *ICDM*.