Robust and Rapid Clustering of KPIs for Large-Scale Anomaly Detection

Zhihan Li†, Youjian Zhao†, Rong Liu‡, Dan Pei†*
†*Tsinghua University* ‡*Stevens Institute of Technology*
†*Beijing National Research Center for Information Science and Technology (BNRist)*

***Abstract*—For large Internet companies, it is very important to monitor a large number of KPIs (Key Performance Indicators) and detect anomalies to ensure the service quality and reliability. However, large-scale anomaly detection on millions of KPIs is very challenging due to the large overhead of model selection, parameter tuning, model training, or labeling. In this paper we argue that KPI clustering can help: we can cluster millions of KPIs into a small number of clusters and then select and train model on a per-cluster basis. However, KPI clustering faces new challenges that are not present in classic time series clustering: KPIs are typically much longer than other time series, and noises, anomalies, phase shifts and amplitude differences often change the shape of KPIs and mislead the clustering algorithm.**

To tackle the above challenges, in this paper we propose a robust and rapid KPI clustering algorithm, *ROCKA*. It consists of four steps: preprocessing, baseline extraction, clustering and assignment. These techniques help group KPIs according to their underlying shapes with high accuracy and efficiency. Our evaluation using real-world KPIs show that *ROCKA* gets F-score higher than 0.85, and reduces model training time of a state-of-the-art anomaly detection algorithm by 90%, with only 15% performance loss.

I. Introduction

Internet-based service companies (*e.g.*, online shopping, social networks, search engine) monitor thousands to millions of *KPIs* (*Key Performance Indicators*, *e.g.* CPU utilization or # of queries per second) of their applications and systems in order to keep their service reliable. Anomalies on KPIs (*e.g.*, a spike or dip) often indicate potential failures on relevant applications [1–3], such as server failures, network overload, external attacks, *etc*. Thus, anomaly detection techniques have been widely used to detect anomalous events timely to minimize the loss caused by such events [2, 4, 5].

However, most anomaly detection algorithms (*e.g.*, Opprentice [2], EDAGS [4], DONUT [5]) assume that an individual model is needed for each KPI. Thus, large-scale anomaly detection on thousands to millions of KPIs is very challenging due to the large overhead of model selection, parameter tuning, model training, or anomaly labeling. Fortunately, many KPIs are similar due to their implicit associations and similarities. If we can identify homogeneous KPIs (*e.g.*, number of queries per server in a well-loaded balanced server cluster) based on their similarities and group them into a few clusters, perhaps only one anomaly detection model is needed per cluster, thus significantly reducing the various overhead aforementioned.

* Dan Pei is the corresponding author.

978-1-5386-2542-2/18/$31.00 ©2018 IEEE

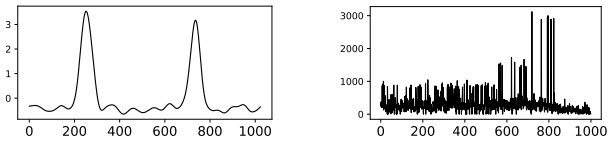
KPIs are streaming data aggregated at pre-defined time intervals (*e.g.*, 1 minute or 10 seconds), thus are essentially time series. Time series clustering, as a popular topic in data mining, has been studied for over 20 years [6]. Most published papers [7, 8] focus on clustering methods and similarity measures with an assumption of idealized time series data. These idealized time series are often low-dimensional (usually less than 1000 data points), and the curves are smooth without abnormal patterns, as shown in Fig. 1a.

However, KPI clustering faces two major new challenges that are not present in classic time series clustering. First, **noises**, **anomalies**, **phase shifts** and **amplitude differences** often change the shape of KPIs, as shown in Fig. 2. These shape variations (*e.g.*, huge or small spikes) on time series curves often distort KPI similarities, making it difficult to use traditional methods to cluster KPIs accurately. Second, a KPI usually contains tens of thousands of data points because it spans from several days to weeks in order to fully capture its patterns (*e.g.* periodicity, seasonality). Unfortunately, there is little work in the literature on clustering high dimensional time series data with anomalies and noises.

In this paper, we tackle above challenges by proposing a robust and rapid time series clustering algorithm, *ROCKA*, which can cluster large-scale real-world KPIs with high accuracy and fast speed. First, for meaningful comparison, KPIs are normalized to eliminate amplitude differences. Then, *ROCKA* extracts the underlying shape of each KPI, called a baseline, to further reduce the impact of noises and anomalies. *ROCKA* adopts shape-based distance (SBD) [9] as distance measure, which is robust to phase shift and achieves high computational efficiency when dealing with high-dimensional time series data. Then *ROCKA* uses an efficient density-based algorithm to create clusters based on shape similarities of baselines. Finally, a representative KPI is identified as a centroid in each cluster and new KPIs can be assigned by their distances to centroids.

The contributions of this paper are summarized as follows:

- To the best of our knowledge, this is the first work that focuses on the clustering of a special type of time series data, KPIs, which are widely used in large Internet companies, and *ROCKA*, is the first reported algorithm for robust and rapid clustering of long time series.
- We propose or adopt three effective techniques in *ROCKA*, baseline extraction, shape-based similarity measure and density-based clustering method, the combina-



(a) Idealized time series data (b) Real-world KPI data

Fig. 1: Idealized time series is smooth without much noise and shape variations, which is common in public datasets. Real-world KPIs often have noises and anomalies.

tion of which obtain underlying shapes of KPIs for clustering and reduce the impact of various shape variations.

- We use public time series datasets and real-world KPI datasets to validate *ROCKA*'s robustness and efficiency. It outperforms a state-of-the-art clustering algorithm YADING [10] on all these datasets and gets F-score higher than 0.85 for the real-world KPIs.
- *ROCKA* reduces the model training time of a state-of-the-art anomaly detection algorithm [5] by 90%, with only 15% performance loss. This is the first reported study that uses clustering to reduce the training overhead of anomaly detection.

II. Challenges and *ROCKA* Overview

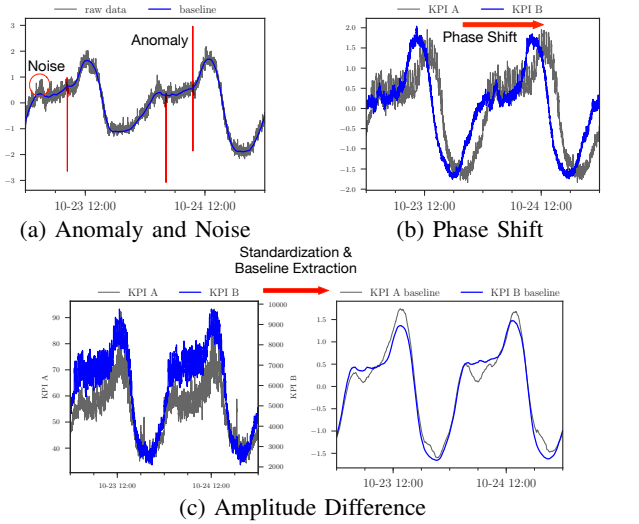
A KPI, as a time series, can be denoted as $T = (x_1, x_2, \cdots, x_m)$, where x_i is the metric value at time index i for $i \in 1, 2, ..., m$, and m is the length of T.

A. Challenges

KPIs are usually *periodic*, and their periods can vary from days to weeks depending on applications and systems. Thus, each KPI usually has thousands of points to fully capture its behavior within periods. Fig. 2 shows a few real-world KPI samples. Compared to the idealized time series shown in Fig. 1a, they have a few notable shape variations.

- *Noises* and *anomalies*: Noises and anomalies (Fig. 2a) are quite common in KPIs. Noises usually refer to small random fluctuations around the expected values in a KPI, while anomalies are significant fluctuations, usually larger than 2 standard deviations [11]. Noises and anomalies may mislead clustering methods since they can distort similarities between KPIs.
- *Amplitude differences*: KPIs can be in different scales. For example, QPS on two closely related but different modules of the same service might look like the left part of Fig. 2c, but if we remove amplitude differences (shown on the right of Fig. 2c), these KPIs have similar patterns and can be analyzed as a group.
- *Phase shifts*: A phase shift refers to a global horizontally shift between two KPIs (Fig. 2b). It's quite common to find the KPIs which have explicit or implicit association shift in phases. *e.g.*, a group of KPIs on the same system call chain may be in similar shape with a time lag. Phase shifts can make it difficult to find similar KPIs.

These variations pose major challenges for clustering KPIs because they may change KPI shapes and clustering algorithms need to be robust to these variations. Previous clustering



(a) Anomaly and Noise (b) Phase Shift

(c) Amplitude Difference

Fig. 2: Four types of common shape variations on KPIs. These KPIs are similar in their underlying shape, and thus can be clustered into the same cluster.

methods [7, 8] usually calculate similarity directly based on the raw data and hence cannot detect coherent clusters for real-world KPIs. In addition, KPIs are often high-dimensional time series data. Algorithms should have low complexity with regard to KPI lengths.

B. ROCKA overview

In this paper, we address these challenges by clustering KPIs based on similarities in their underlying shapes despite noises, anomalies, amplitude differences, and phase shifts. This clustering can be extremely useful in KPI analysis and large-scale anomaly detection. Typically there are a vast number of KPIs in a large-scale internet-based service company. It is impossible for operators to carefully analyze each KPI individually. With clustering, they can analyze KPIs per cluster and create an anomaly detection model for each cluster, significantly reducing modeling cost and improving their efficiency.

We propose a novel time series clustering algorithm, *ROCKA*, as shown in Fig. 3. Overall, *ROCKA* consists of four steps: *preprocessing*, *baseline extraction*, *clustering* and *assignment*. Preprocessing is conducted on the raw KPI data to remove amplitude differences and standardize data. In baseline extraction step, we reduce noises, remove the extreme values (which are likely anomalies), and extract underlying shapes, referred to as *baselines*, of KPIs. Clustering is then conducted on the baselines of sampled KPIs, with robustness against phase shifts and noises. Finally, we calculate the centroid of each cluster, then assign the unlabeled KPIs by their distances to these centroids.

In order to cluster a large number of KPIs, for the sake of efficiency, we create a clustering model using a subset of randomly sampled KPIs, and then assign the rest of KPIs to the resulting clusters. As discussed in [10], a small sample dataset is enough for clustering even if the number of KPIs is really large. *e.g.*, for a dataset with more than 9000 time series, sample 2000 of them is enough for clustering.

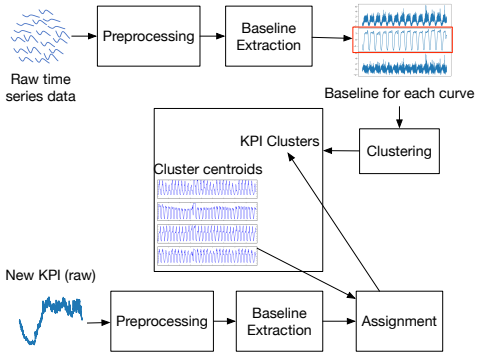


Fig. 3: Overall framework of *ROCKA*.

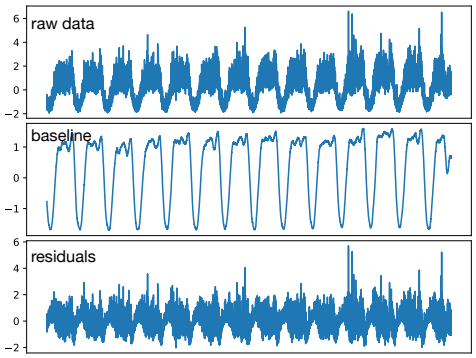


Fig. 4: Extract the baseline of a real-world KPI data. The baseline preserves the underlying shape of original data.

III. Algorithm

After giving an overview of *ROCKA*, in this section, we introduce each component in detail.

A. Preprocessing

It is common that KPIs have missing values. Fortunately, according to our observation, the percentage of missing values in a KPI usually is very small. We simply use linear interpolation to fill them based on their adjacent data points. This enables us to calculate point-wise distance between two KPIs.

Another important preprocessing technique is standardization: $\hat{x_t} = \frac{x_t - \mu_x}{\sigma_x}$, where x_t is the raw data, μ_x and σ_x are the mean and standard deviation of x_t, separately. As discussed in [12], time series must be standardized in order to make meaningful comparison between them, because KPIs, sourced from different applications and systems, vary in their amplitudes. Standardization can remove the differences in amplitude and help calculate similarities between KPIs.

B. Baseline Extraction

Noises and anomalies, as demonstrated by the examples in Fig. 2a, can significantly change the shapes of KPI curves and mislead shape-based similarity. We need a simple but effective method to remove those extreme values which are likely anomalies, such that a rough baseline can be extracted to represent the underlying structure of a KPI.

*1) **Smoothing Extreme Value**:* After standardization, each KPI is normalized to have zero mean and unit variance. Intuitively, anomalies deviate the most from the mean value. In general, the ratio of anomaly points in a time series is less than 5% [5]. As such, we simply remove the top 5% data which deviate the most from the mean value, and then use linear interpolation to fill them. Extreme anomalies (which are often huge spikes or dips) are removed and replaced with their neighboring normal observations. In case that a KPI has few anomalies, normal data may be removed by mistake, but they are interpolated still by normal observations without changing the underlying shape of the KPI much.

*2) **Extract Baseline**:* Besides anomalies, noises also distort the shape-based similarity between KPIs. Thus we try to extract a rough baseline to represent each KPI. Generally, a KPI curve can be regarded as a smooth baseline (*i.e.*, normal patterns that indicate its expected behavior) with many random noises. A simple but effective method is to apply moving average with a small sliding window on the KPI, separating its curve into two parts: baseline and residuals. Specifically, for a KPI, T, with a sliding window of length W, stride = 1, for each point x_t, the corresponding point on the baseline, denoted as x_t^*, is the mean of vector $(x_{t-W+1}, \cdots, x_t)$. Then the difference between x_t and x_t^* is called a residual. The baseline B and residuals R can be computed as:

$$\begin{aligned} T &= (x_1, x_2, \cdots, x_m) \\ x_t^* &= \frac{1}{W}\sum_{i=1}^{W} x_{t-i+1} \\ B &= (x_W^*, x_{W+1}^*, \cdots, x_m^*) \\ R &= (x_W - x_W^*, \cdots, x_m - x_m^*) \end{aligned} \tag{1}$$

Fig. 4 shows baseline extraction for a real-world KPI example. The baseline extracted by our algorithm removes most of the anomalies and noises while preserving its underlying shape. The residuals contains random noises, and are not considered in clustering. After that, we apply standardization again to get a standardized baseline. Such a baseline is used as the input of our clustering algorithm.

C. Density-based Clustering

We perform density-based clustering on the baselines of sampled KPIs based on their shape similarities. We first review cross-correlation, a widely-used shape-based similarity measure, and then describe our density-based clustering algorithm.

*1) **Shape-based Similarity Measure**:*
We briefly review some popular similarity measures. L_p norms [13] are a group of widely-used distance measures due to their simplicity and efficiency. However, they are quite sensitive to noises, amplitude differences and phase shifts. DTW (Dynamic Time Warping) [14] is well known for its invariance to phase shifts, scaling distortion, and other shape variations, but has high computation complexity. Generally, it takes $O(m^2)$ to compute the dissimilarity between two time series of length m. This makes DTW impractical to deal with high dimensional KPIs. Besides, correlation-based metrics (*e.g.*, cross-correlation) [7, 9, 15] are also used as similarity measure in recent years. They are often used in signal processing and can natively handle the phase shifts between

time series. Moreover, the calculation complexity of cross-correlation can be reduced to $O(m\log(m))$ using convolution theorem and Fast Fourier Transform. Thus, correlation-based metrics can be suitable similarity measures for our KPIs.

Cross-correlation is a well-accepted similarity measure of time series in the signal processing field. It calculates the sliding inner-product of two time series, which is natively robust to phase shifts. [9] proposed a shape-based distance (SBD) on the basis of cross-correlation and applied it on *idealized* time series data. In this paper, we use SBD to measure the similarity of our baselines, which have higher dimensional and more phase shifts than idealized time series.

Since baselines are standardized time series without amplitude differences, the comparison between them is now meaningful. For two time series $\vec{x} = (x_1, \cdots, x_m)$ and $\vec{y} = (y_1, \cdots, y_m)$, cross-correlation slides $\vec{y}$ over $\vec{x}$ to compute the inner-product for each shift s of $\vec{y}$, as defined by Eq. (2).

$$\vec{x}_{(s)} = (x_1, x_2, \cdots, x_m)$$
$$\vec{y}_{(s)} = \begin{cases} (\overbrace{0, \cdots, 0}^{|s|}, y_1, y_2, \cdots, y_{m-s}), & s \geq 0 \\ (y_{1-s}, \cdots, y_{m-1}, y_m, \underbrace{0, \cdots, 0}_{|s|}), & s < 0 \end{cases} \tag{2}$$

For all possible shifts $s \in [-m+1, m-1]$, we can calculate the inner-product $CC_s(\vec{x}, \vec{y})$ as the similarity between time series $\vec{x}$ and $\vec{y}$ with a phase shift s. It is defined as Eq. (3).

$$CC_s(\vec{x}, \vec{y}) = \begin{cases} \sum_{i=1}^{m-s} x_{s+i} \cdot y_i, & s \geq 0 \\ \sum_{i=1}^{m+s} x_i \cdot y_{i-s}, & s < 0 \end{cases} \tag{3}$$

The cross-correlation is the maximized value of $CC_s(\vec{x}, \vec{y})$, which means the similarity between $\vec{x}$ and $\vec{y}$ at the optimal phase shift s. Intuitively, at the optimal shift, the similar patterns in $\vec{x}$ and $\vec{y}$ are properly aligned such that the inner-product is the maximum (*i.e.*, the peaks in $\vec{x}$ are aligned with similar peaks in $\vec{y}$ with the optimal shift). Thus, the cross-correlation measure can overcome the phase shift and represent shape similarity between two time series. In practice, a **normalized version of cross-correlation** (NCC) is often used to limit the values to be within $[-1, 1]$, where 1 shows a perfect similarity and -1 indicates the two time series are completely opposite. NCC is defined as follows:

$$NCC(\vec{x}, \vec{y}) = \max_s \left(\frac{CC_s(\vec{x}, \vec{y})}{\|\vec{x}\|_2 \cdot \|\vec{y}\|_2}\right) \tag{4}$$

$$SBD(\vec{x}, \vec{y}) = 1 - NCC(\vec{x}, \vec{y}) \tag{5}$$

Then we can define the shape-based distance (SBD) according to NCC, as discussed in [9]. SBD ranges from 0 to 2, where 0 means two time series have exactly the same shape. A smaller SBD means higher shape similarity.

Since the SBD is a point-wise similarity measure, extreme anomalies might be mistaken for peaks or troughs, misleading the similarity calculation. Thus, the baseline extraction step (Section III-B) plays an important role in finding the shape similarity between two KPIs. As shown in Fig. 5, the SBD of the two baselines is 0.0208, while that of the raw KPIs is 0.2802, over 10 times bigger. In fact, these two KPIs are very similar in their underlying shapes, but anomalies and noises can cause inaccurate similarity measurement.

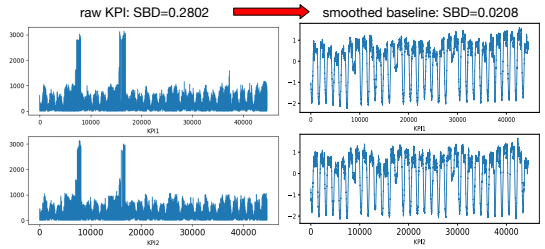


Fig. 5: Calculate SBD on raw KPIs and their baselines.

*2) **Density-based Clustering with Parameter Estimation**:* According to [8], clustering algorithms can be broadly classified into six groups. Here we briefly discuss three that are related to our work. Partitioning methods, like K-Means [16] and K-medoids [17], are the most widely used ones due to their simplicity and effectiveness. However, the number of clusters, *k*, as well as the initial partition of each cluster, need to be predetermined. Feature- and model-based methods [7] try to transform time series into several features or fit them to predefined models to extract more information about data. These algorithms often make strong assumptions (*e.g.*, the time series can be modeled using Gaussian mixture [18], ARIMA [19], *etc*), which barely hold in complicated datasets. Density-based methods, like DBSCAN [20], finds dense regions separated by low-density areas to form clusters. A cluster is expanded if its neighbors are dense, *i.e.*, others that are similar to its core will be absorbed into it. In this way, density-based methods can identify clusters in arbitrary shape and size. Moreover, these methods can work with most similarity measures.

We decide to adopt DBSCAN, a density-based clustering method in our work for two reasons. First, since KPIs are collected from various applications and systems, it is difficult to predetermine the number of clusters. Density-based methods form clusters in dense regions, which can be in arbitrary shapes and size. Second, since SBD gives shape similarities between KPIs, naturally we can leverage the transitivity of shape similarities to expand clusters. For example, three KPIs, named a, b, c, measuring the performance of machines used by the same application. a is similar to b in shape, and so is b to c. Intuitively, a and c are also similar in their shapes. Therefore, they can be assigned into the same cluster.

The main idea of DBSCAN is to *find some cores in dense regions, and then expand the cores by transitivity of similarity to form clusters*. A core p is defined as an object that has at least *minPts* objects within a distance of ϵ from it (excluding p). Only cores can be used to expand clusters. In other words, only objects that are within the distance of ϵ from a core can be absorbed into a cluster. ϵ, also called density radius, is the maximum distance of these objects from p. ϵ indicates the

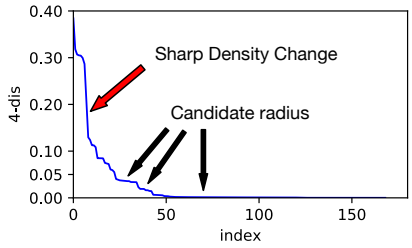
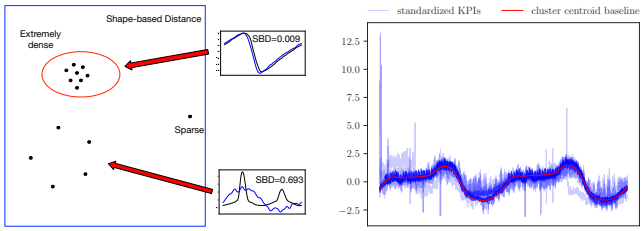


Fig. 6: k_{dis} curve ($k = 4$) with SBD. Flat parts are candidate radiuses. The largest candidate $\leq$ *max_radius* is selected as density radius. The steep parts indicate sharp density changes.



(a) (b)

Fig. 7: (a) With SBD, extremely dense regions contain similar objects that form clusters, while large density radius indicates dissimilar objects. (b) A cluster with 18 standardized KPIs and its centroid capturing the underlying shape of cluster.

density of a dense region. We set $minPts = 4$, as suggested by the authors of DBSCAN in [20]. Next we discuss how to determine the key parameter, density radius ϵ.

For $minPts = k$, let k_{dis} be the SBD between an object and its k-Nearest-Neighbor (KNN). If the k_{dis} value of each object is plotted in a descending order, as shown in Fig. 6, a flat portion of the curve indicates the density around a number of objects is consistent, while a steep part shows significant density changes. As suggested by [10, 20], k_{dis} on a flat portion of the curve can be considered as candidate density radiuses, since it ensures a good number of cores to expand clusters.

However, a candidate density radius should be small enough to ensure a core and its neighbors are similar to each other. A large candidate value actually corresponds to a sparse region where the objects are dissimilar with each other (Fig. 7a). An object with a large k_{dis} is actually considered as an **outlier**, *i.e.*, it is not similar to most other objects and should not be clustered into any cluster. In practice, a threshold *max_radius* sets the upper bound of the density radius.

Motivated by [10], we designed a heuristic algorithm, as shown in Algorithm 1, to find candidate density radiuses. Given a $k_{\text{dis}} \leq$ *max_radius*, it first checks whether the curve's slopes on the left and right sides of the point are smaller than a threshold (*slope_thresh*, *e.g.*, 0.01) to avoid search in a steep area. Then the left and right slopes are compared to see if the difference is less than a small threshold (*slope_diff_thresh*, *e.g.*, 10^{-3}). If so, this k_{dis} is considered as a candidate radius. A divide and conquer strategy is used to find all candidate radiuses. In our implementation, we empirically set *max_radius* to 0.05, and the final density radius is the largest candidate. This setup works well on all our datasets.

In practice, *max_radius* can be adjusted by operators to obtain clusters in different granularity. *i.e.*, a large *max_radius* leads to a few coarse-grained clusters, while a small value results in many fine-grained clusters and a large fraction of outliers.

With the estimated parameters, we use DBSCAN to do clustering. The complexity of DBSCAN is $O(n \log(n))$, for an input dataset with n time series data.

Algorithm 1: Density Estimation

Input: k_{dis}: k_{dis} curve sorted in descending order;
start: the smallest index that has k_{dis}[start] $\leq$ *max_radius*;
end: the last index of k_{dis};
len_thresh: minimum length of a flat part;
slope_thresh: small value preventing search in steep area;
slope_diff_thresh: small value indicates a flat part.
Output: a list of candidate radiuses
Function `FindCandidateRadius`(*k_{dis}, start, end, candidates*) **:**
  if *end*−*start* $\leq$ *len_thresh* **then**
    return ▷ Search area contains few points;
  end
  r ← -1, *diff* ← 2;
  for *i* ← *start to end* **do**
    leftslope ← (k_{dis}[*i*] − k_{dis}[*start*])/(*i*-*start*);
    rightslope ← (k_{dis}[*end*] − k_{dis}[*i*])/(*end*-*i*);
    if *leftslope*> *slope_thresh or rightslope*> *slope_thresh* **then**
      continue ▷ Search area is steep;
    end
    if |*leftslope*−*rightslope*| < *diff* **then**
      diff ← |left−right|, *r* ← *i*;
    end
  end
  if *diff* < *slope_diff_thresh* **then**
    add *r* to candidates; ▷ Find a flat part near point *i*;
  end
  ▷ Find all candidates using divide and conquer;
  FindCandidateRadius(k_{dis}, *start*, *r*-1, *candidates*);
  FindCandidateRadius(k_{dis}, *r*+1, *end*, *candidates*);

D. Assignment

After creating a clustering model using sampled KPIs, now we calculate the centroid of each cluster and then assign the rest of KPIs to the clusters based on centroids. Generally, in each cluster, the object which has the smallest sum of squared similarity distance to all the others is regarded as the cluster centroid. The centroid in a cluster can be calculated using Eq. (6). Fig. 7b shows a centroid of a cluster. Since we use SBD as our distance measure, the centroid can be used as a representative KPI that captures the common underlying shape of KPIs in the cluster. Thus, for anomaly detection, as will be discussed in Section V, we can train anomaly detection models on centroids only. In practice, operators can gain intuitive understanding of KPIs in each cluster through its centroid.

$$\text{centroid} = \arg \min_{\vec{x} \in \text{cluster}_i} \sum_{\vec{y} \in \text{cluster}_i} SBD(\vec{x}, \vec{y})^2 \qquad (6)$$

Once we get the centroid of each cluster, the assignment step is straightforward. For each unlabeled KPI, preprocessing and baseline extraction are applied to get its baseline time series $\vec{x}$. Then we calculate the SBD between $\vec{x}$ and each cluster centroid, assigning it into the cluster with the nearest centroid.

Furthermore, we should consider that some unlabeled KPIs may not be similar to any clusters, which should be marked as outliers. In general, it is believed that two time series with NCC smaller than 0.8 do not have strong positive correlation [21] (*i.e.*, they are not similar in shape). Thus, an unlabeled KPI whose SBD to its nearest cluster centroid larger than 0.2 is considered as an outlier. Certainly, in different applications, one may use a smaller threshold to ensure higher similarity in KPIs assigned to a cluster.

For a dataset with k clusters, the complexity of assigning each KPI is $O(km\log(m))$, where m is the length of the KPI. Therefore, our algorithm can assign clusters to a large number of high-dimensional KPIs very efficiently. In the clustering step, it takes $O(n^2 m\log(m))$ to calculate the SBD between each pair of baselines in the sampled dataset of size n. Then it costs $O(n\log(n))$ to estimate the candidate radiuses, and $O(n\log(n))$ for DBSCAN clustering. For a dataset with N KPIs, we sample n of them for clustering($n \ll N$), and assign the rest by cluster centroids. The total time complexity of *ROCKA* is $O(Nm\log(m) + n^2 m\log(m) + n\log(n))$.

IV. Evaluation

In this section, we conduct extensive experiments to evaluate the performance of *ROCKA*. First, we use three public time series datasets that are considered "idealized". Then we use two real-world KPI datasets from large Internet companies to show *ROCKA*'s performance in practice. We also compare *ROCKA* with a state-of-the-art time series clustering algorithm, YADING [10], to demonstrate *ROCKA*'s accuracy and robustness. Finally, a group of experiments are conducted to show the effectiveness of each key technique used in *ROCKA*.

A. Baseline Algorithm: YADING and YADING'

YADING [10] is a new method for large-scale time series data. It uses a multi-density clustering algorithm with L_1 distance as similarity measure. It automatically selects several density radiuses and conducts density-based clustering with each of them. On StarLightCurve dataset in UCR time series archive [22], YADING outperforms DENCLUE and DBSCAN algorithms in accuracy, and outperforms CLARANs in speed.

However, since there are too many shape variations in real-world KPI data, L_1 distances between KPIs fall into a big range, causing some difficulties in finding appropriate density radiuses. According to our experiments, more than 95% real-world KPIs are marked as outliers using the original YADING, which is unacceptable. Therefore, in this paper, when using real-world KPIs, we use an improved version of YADING, called YADING', by using our Algorithm 1 (without the constraint of *max_radius*) to fine-tune density radiuses for better clustering results in YADING.

B. Results on Public Datasets

We first evaluate *ROCKA* on three public datasets and compare our results with YADING.

Name	# of time series	dimensionality	# clusters
StartLightCurves	9236	1024	3
MALLAT	2400	1024	8
CinC_ECG_torso	1420	1639	4

TABLE I: Description of three UCR datasets

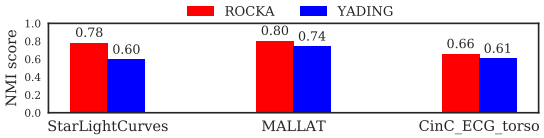


Fig. 8: Performance on three public datasets.

1) ***Datasets:*** We use three public time series datasets from UCR time series archive [22]. The details of the three datasets are shown in Table I. These time series are often considered as idealized data with very few anomalies and only slight noises. In addition, all the three datasets are in large size and have relatively high dimensionality than other public datasets, making algorithms with high computation complexity unsuitable. The data are also labeled with clustering ground truth, so we can easily evaluate the performance of our algorithm.

2) ***Evaluation Metrics:*** As suggested by previous experiments on the datasets [10, 23], we use a well-accepted metric, Normalized Mutual Information (**NMI**) [24], to evaluate clustering accuracy. NMI measures the mutual dependence between ground-truth labels and obtained clusters. It ranges from 0 to 1, where 1 means the clusters perfectly match the labels and 0 indicates they are completely irrelevant.

3) ***Results:*** Since the data on public datasets are smooth, for *ROCKA*, we set the sliding window size $W = 1$ in baseline extraction. The clustering results are mapped to the ground-truth to calculate the NMI score. The performance is shown in Fig. 8. *ROCKA* outperforms YADING on all the three datasets. Specifically, there are quite a few phase shifts and anomalies in dataset StartLightCurves. YADING is confused by these variations since it directly applies L_1 distance on the raw data without necessary alignment. On the other hand, the techniques in *ROCKA* enhance its robustness against shifts and anomalies. In addition, as discussed in Section III-D, *ROCKA* is computationally efficient when dealing with high-dimensional time series. Our experiments were conducted on a machine with Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40GHz and 64GB RAM. With the pre-computed distance matrix and density radius, it takes less than 1 second to do clustering. It uses less than 0.05 second to assign an unlabeled 1000-dimensional time series, only slightly higher than YADING.

C. Results in Practice

Now, in a similar fashion, we evaluate the overall performance of *ROCKA* and *YADING'* on two real-world KPI datasets obtained from several large Internet companies.

1) ***Datasets:*** As shown in Table II, DS1 is a TPS/QPS (transaction/query per second) dataset from different clusters of machines. DS2 contains machine-level metrics (*e.g.*, Search Response Time, CPU utilization) collected from a large number of machines. All KPIs in these two datasets are collected at one-minute time interval for about one month.

Name	# of KPIs	dimensionality	# of KPI types
DS1	212	43200	4
DS2	2010	44640	7

TABLE II: Description of real KPI datasets

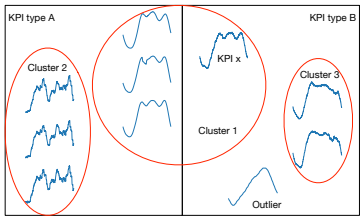


Fig. 9: Demonstrate the evaluation method for real-world KPIs. Each curve is a baseline extracted from the raw KPI.

The KPIs in DS1 and DS2 are grouped into four and seven types respectively by the engineers of data providers. These types are regarded as the ground truth for clustering. However, according to these engineers, *KPIs that belongs to the same type (e.g., CPU utilization) can be clustered into several clusters due to their differences in shapes, but KPIs in different types (e.g., CPU utilization and memory utilization) cannot belong to the same cluster*. We will calculate our evaluation metrics based on this rule.

*2) **Evaluation Metrics**:* In practice, **F-score** is a widely-used intuitive metric for operators to evaluate clustering. Hence, we decide to use F-score as our evaluation metric in this experiment. F-score is the harmonic average of precision and recall. F-score has its best value at 1 (perfect precision and recall) and worst at 0. For the multi-class problems, the final F-score is the average F-score over all classes.

Since a cluster should only contain KPIs of one type, after clustering, we determine the KPI type for each cluster by majority vote rule. Then we calculate precision and recall according to the assigned KPI types. Fig. 9 illustrates the evaluation method. For example, most KPIs in Cluster 1 belongs to type A, then the entire cluster is labeled as type A. Then KPI x, which actually belongs to type B, is counted as a False Positive when calculating F-score on type A, and as a False Negative when calculating F-score on type B. Note that, as shown in Fig. 9, it is legitimate to group KPIs of type A into two clusters, because these KPIs vary in shapes.

It is worth mentioning that, KPIs that are not similar to any cluster are marked as outliers by density-based algorithms (*e.g. ROCKA* and YADING'). Since it is hard for operators to identify outlier KPIs in the large-scale datasets, we simply ignore them while calculating the F-score (*i.e.*, we calculate F-score only for KPIs that have been assigned to clusters). Instead, we use a metric, **fraction of outliers**, to show the percentage of KPIs considered as outliers by each algorithm. Generally, it is quite common to have some outliers in real-world KPI datasets, since restart or service changes on machines may change the shapes of some KPIs. However, algorithms that are sensitive to noises and anomalies may mistakenly mark a large number of KPIs as outliers despite underlying shape invariance. In contrast, a robust algorithm will have lower fraction of outliers.

	DS1		DS2	
	ROCKA	**YADING'**	***ROCKA***	**YADING'**
F-score	1.00	0.98	0.85	0.99
fraction of outliers	0.04	0.18	0.17	0.49
# clusters	6	7	29	33
avg distance calculation (ms)	53	0.205	58	0.226
avg assignment time (ms)	411	54	1350	99

TABLE III: Performance on real KPI datasets

*3) **Results**:* The results are shown in Table III. Compared with the UCR datasets, the real-world KPIs are more challenging for clustering because they have complicated shape variations, *e.g.*, phase shifts, anomalies. We can see that *ROCKA* achieves good accuracy on both datasets. For DS1, *ROCKA* accurately clusters all KPIs with a small number of outliers. DS1 contains KPIs about the same metric from different groups of machines. Generally, machines from the same group have direct or indirect associations with each other, and as such, their corresponding KPIs roughly follow the same distribution. *ROCKA*, equipped with techniques to reduce the impact of shape variations (*e.g.*, noises, phase shifts, *etc*), successfully clusters the KPIs according to their underlying shapes. In comparison, although YADING' also obtains good accuracy on DS1, it marks a much larger fraction of outliers due to its sensitivity to shape variations.

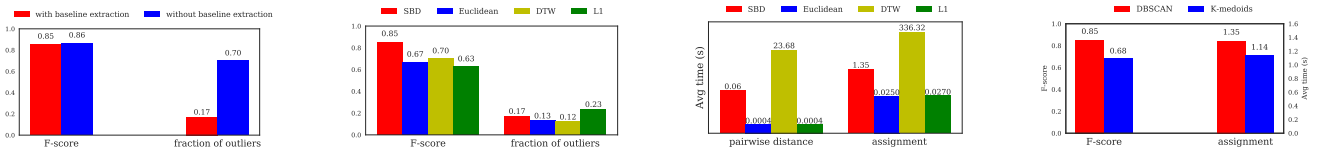
DS2 contains metrics from a large number of machines, in which the association between KPIs is much weaker than in DS1. However, *ROCKA* still obtains good accuracy and marks only a small fraction of outliers thanks to its robustness. Although YADING' achieves high F-score on the clustered data, half of the KPIs are marked as outliers. In fact, it only clusters KPIs that are very similar (with an extremely small L_1 distance among raw data) into small clusters. Others are mistakenly marked as outliers by YADING' due to its sensitivity to shape variations.

Moreover, *ROCKA* is efficient even when dealing with the high-dimensional KPIs. The experiments are run on the same hardware as before. For a large number of KPIs, it takes only one second to assign cluster for each KPI. The assignment step is easy to be parallelized since it only depends on the pre-computed cluster centroids. Compared with YADING', *ROCKA* takes more time mainly because SBD is more time consuming than L_1 distance. However, the simple L_1 distance is unable to handle KPIs with complicated shape variations.

D. The effects of techniques in ROCKA

We mainly use three techniques to make *ROCKA* more robust and rapid: baseline extraction, shape-based distance measure, and a density-based clustering method. We conducted additional experiments to demonstrate the effectiveness of each technique with dataset DS2, where KPIs present complicated shape variations.

*1) **Baseline extraction**:* Fig. 10a compares the F-score and fraction of outliers of *ROCKA* with and without the baseline extraction. Although the F-score does not drop, the fraction of outliers increases dramatically without baseline extraction.



(a) Performance with and without baseline extraction

(b) Performance with different similarity measures

(c) Calculation time (log scale) with different similarity measures

(d) Performance with different clustering methods

Fig. 10: The effects of techniques in *ROCKA*

Because of excessive anomalies and noises, most KPIs in DS2 appear in different shapes with little similarity. Without baseline extraction, *ROCKA* can only accurately cluster a small number of KPIs that are less affected by noises and anomalies, but regard the majority as outliers.

*2) **Similarity measures**:* In this experiment, we replace SBD by other popular distance measures, L_1 norm, Euclidean distance or DTW. The F-score and computation time using these four similarity measures are shown in Fig. 10b and Fig. 10c. We can see that *ROCKA* with SBD excels the other options by a large margin. Since the KPIs are high dimensional and have frequent phase shifts, the simple L_1 and Euclidean cannot obtain accurate similarities.

DTW, as explained in Section III-C, is invariant to phase shifts and shape scaling. As a result, it can accurately capture similarities between KPIs with phase shifts. However, it can also perfectly align two KPIs with different periodicities or seasonalities to consider them similar to each other. For example, KPIs about Page Views and KPIs about Search Response Time may have similar shapes but differ in seasonality (*e.g.*, a day vs. a week). They are similar under DTW distance, but KPIs with different types cannot be clustered into the same cluster per the operators' rule described before. As a result, DTW has lower F-score than SBD. Moreover, DTW takes much longer than others to calculate pairwise distance.

*3) **Clustering methods**:* In this experiment, we replace DBSCAN by K-medoids [17]. Fig. 10d shows the performance comparison. *ROCKA* with DBSCAN achieves better F-score and the time consumed by assignment is only slightly higher. Note that, although K-medoids is usually used with Euclidean distance as similarity measure, here we still use SBD when clustering by K-medoids, because KPIs have frequent phase shifts. In each iteration of K-medoids, for each cluster, an object with the minimal average distance to all the objects is identified as a medoid. However, when KPIs are not perfectly aligned, this method cannot guarantee a medoid that can represent the cluster. However, DBSCAN expands clusters by similarity between objects, which can be obtained by SBD, L_p norm, or any other similarity measures, and flexibly detects clusters in arbitrary shape and size. Thus, DBSCAN is suitable for our algorithm.

V. Clustering for KPI Anomaly Detection

In this section, we combine *ROCKA* and a state-of-the-art anomaly detection algorithm, DONUT [5], to demonstrate how *ROCKA* can assist in anomaly detection.

A. ROCKA for KPI Anomaly Detection

Anomaly detection algorithms are often designed to have a model trained for each individual time series. With a large number of KPIs, it takes a prohibitive amount of time to train all the models. *ROCKA* clusters KPIs similar in underlying shapes into a cluster. Thus, we can train a model on each cluster centroid using an anomaly detection algorithm, and then directly use the model to detect anomalies on other KPIs in the same cluster. In addition, this method can be extremely useful when a KPI does not have enough labeled anomalies for individual model training or threshold selection.

Moreover, in some anomaly detection algorithms, a threshold needs to be fine-tuned by the ground-truth anomaly labels for optimal performance. With clusters created by *ROCKA*, the threshold selected for a cluster centroid can be used by other KPIs in the same cluster, simplifying parameter tuning.

B. Overview of DONUT

This very recent algorithm, DONUT [5], is an unsupervised anomaly detection algorithm for seasonal KPIs. In [5], DONUT greatly outperforms the popular algorithm Opprentice [2], which has excelled other traditional anomaly detection algorithms in the past. DONUT trains a deep generative model to reconstruct the KPI data and output an indicator (anomaly score) for each point to show the severity of anomaly. Specifically, DONUT applies sliding windows over the KPI to get short series $\mathbf{x}$ and tries to recognize what normal patterns $\mathbf{x}$ follows. The indicator is then calculated by the difference between reconstructed normal patterns and $\mathbf{x}$ to show the severity of anomalies. In practice, a threshold should be selected for each KPI. A data point with an indicator value larger than the threshold is regarded as an anomaly.

We choose DONUT as the anomaly detection algorithm in our experiments not just for its excellent performance. DONUT identifies anomalies by extracting normal patterns from KPIs. This design fits well with *ROCKA* since *ROCKA* groups KPIs based on their underlying shapes, *i.e.* normal patterns. When DONUT reconstructs a normal pattern for a centroid KPI, this pattern approximates the normal patterns of other KPIs in the cluster, making it possible to share anomaly detection models among KPIs. Thus, we believe *ROCKA* can work with most anomaly detection algorithms that are based on recognition of normal patterns of time series [25, 26].

C. Experiment Setup

We use a real-world KPI dataset DS3 to evaluate the performance of anomaly detection. DS3 contains 48 6-month-

long KPIs collected from different machines in a large Internet company. Experienced operators has labeled anomalies on these KPIs according to their domain knowledge to provide a ground truth for anomaly detection.

To show how *ROCKA* can assist in anomaly detection, we conduct the following experiments (E1-E3):

- **E1**: DONUT only. As a baseline experiment, we use DONUT to train an anomaly detection model for each of the 48 KPIs. Then, the threshold is fine-tuned for best F-score for each KPI, as suggested in [5].
- **E2**: *ROCKA* + DONUT. Here we first apply *ROCKA* to cluster the 48 KPIs into clusters. Then, we use DONUT to train an anomaly detection model *only* on the centroid KPI in each cluster, and select the best threshold according to the ground-truth labels on the centroid. The model and threshold are then used to detect anomalies in other KPIs of the same cluster.
- **E3**: *ROCKA* + DONUT + KPI-specific threshold. Certainly, for a KPI, we can fine-tune the threshold for the best performance, if the KPI has sufficient ground-truth labels. In this experiment, the threshold of each KPI, except centroids, is reestimated by its ground-truth labels.

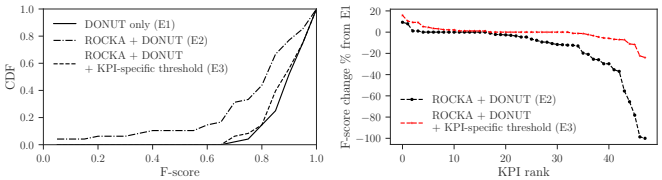
All these anomaly detection experiments are run on a server (NVIDIA GeForce GTX 1080 Ti graphics cards with 11GB GDDR5X memory). Next we describe our experiment results.

D. Experiment Results

Table IV shows training and testing time consumed by each experiment. We can see that it takes *1075 seconds* to train a DONUT model for a KPI. The total model training time will become prohibitive when dealing with a large number of KPIs. With *ROCKA*, KPIs are clustered into 5 clusters (the total cluster time is only few seconds, which is negligible) and we only need to train 5 anomaly detection models, reducing 90% model training time. The advantage in time reduction becomes even more significant with larger KPI datasets, where a cluster may contain more KPIs.

Now we turn to F-score. Table V shows the average F-score of the three experiments for each KPI cluster. Fig. 11a gives the cumulative distribution function (CDF) of F-score on each KPI. We can see that, in E1, DONUT achieves high F-score on most of the KPIs with an average of 0.89. About 90% KPIs has a F-score over 0.8. In E2, the cluster-based approach also obtains decent results, with an average F-score of 0.76. This is because KPIs with a similar underlying shape tends to have implicit associations in practice (*i.e.*, belong to the same cluster of machines, measure similar metrics, *etc*). In this way, KPIs in the same cluster also have similar normal patterns. As a result, they can share an anomaly detection model and threshold.

E2 is very useful when dealing with large-scale KPIs. First, this cluster-based approach is very efficient in model training. More importantly, we do not need ground-truth anomaly labels for each KPI to calculate the threshold. Instead, operators only need to label the centroid KPIs, and the threshold can be used on other KPIs in the same cluster.



(a) (b)

Fig. 11: (a) CDF of F-score on each KPI. (b) The F-score change while using *ROCKA*+DONUT, compared to raw DONUT result (E1).

algorithm	total train	avg. train	avg. test
DONUT only (E1)	51621	1075	345
ROCKA+DONUT (E2)	5145	1029	345
ROCKA+DONUT+KPI-specific threshold (E3)	5145	1029	345

TABLE IV: Time Consumption of DONUT on DS3 (seconds)

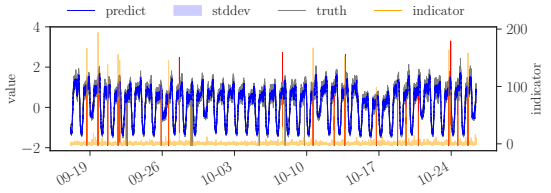
However, as shown in Fig. 11b, compared to E1, the F-score of most KPIs in E2 is reduced slightly, by less than 15%. The main reason is that KPIs vary by their anomaly severity levels, and a uniform threshold cannot be the optimal for every KPI. This can be explained by an example shown in Fig. 12. The KPI (say A) shown in Fig. 12b only has a few slight anomalies, but its cluster centroid suffers with more severe anomalies (Fig. 12a). With the centroid's threshold (15.35), which is too high for A, anomalies on A are missed (Fig. 12b). With the same centroid's anomaly detection model, we reestimate the threshold based on A's ground-truth anomaly labels. With a new threshold (10.01), we reach a perfect F-score of 1.0.

Indeed, a few KPIs need KPI-specific thresholds. In E3, we fine-tune the threshold for each KPI based on its ground-truth labels. Then, as shown in Table V and Fig. 11a, the cluster-based approach achieves similar F-score as E1, but with significant reduction on model training time. Compared to E1, the F-score of most KPIs drop less than 5% in E3 (see Fig. 11b). This further demonstrates that the anomaly detection model can be shared in the same cluster regardless of different anomaly severity levels.

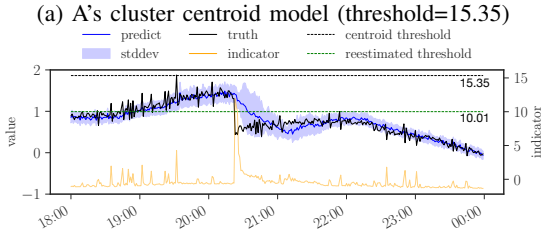
Interestingly, we find that the F-score on two particular KPIs actually arise in E2 compared to E1. Fig. 13a shows one KPI which has a few anomalies and some normal slight perturbations. The DONUT model (Fig. 13c) in E1 marks these perturbations as anomalies. In contrast, the centroid's model (Fig. 13b) makes fewer mistakes. Generally, the anomaly detection model might be overfitting on some particular KPIs, making it sensitive to small fluctuations. In turn, the cluster centroid model learns less details of the particular KPI, which actually prevents overfitting and gets higher F-score.

Cluster	E1	E2	E3	# KPIs
A	0.88	0.66	0.86	18
B	0.79	0.78	0.79	6
C	0.95	0.81	0.95	12
D	0.87	0.86	0.87	4
E	0.90	0.83	0.88	8
Overall	0.89	0.76	0.88	

TABLE V: Average F-score for anomaly detection on DS3 (E1: DONUT only, E2: *ROCKA* + DONUT, E3: *ROCKA* + DONUT + KPI-specific threshold)

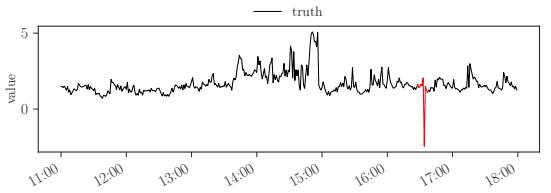


(a) A's cluster centroid model (threshold=15.35)

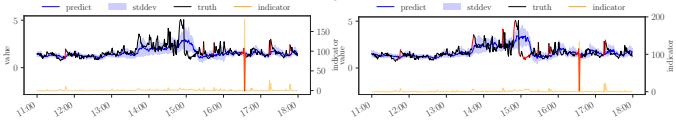


(b) KPI A with *ROCKA* + DONUT

Fig. 12: Orange line is anomaly indicator at each point and red line is the anomalies detected by algorithm. The best threshold on KPI A's centroid is 15.35, which is larger than the indicator of the most significant anomaly on A (11.90). However, with the reestimated threshold (10.01), we can detect all anomalies on A accurately.



(a) KPI B ground-truth



(b) KPI B with *ROCKA* + DONUT(c) KPI B with raw DONUT (E1)

Fig. 13: The raw DONUT model on KPI B is a bit overfitting and sensitive to small fluctuations. The cluster centroid model is more robust and gets higher F-score.

VI. Conclusion

In this paper, we propose a robust and rapid time series clustering algorithm, *ROCKA*, to cluster a large number of KPIs which are a special type of time series, with noises, anomalies, phase shifts, amplitude differences, and high dimensionality. This is the first work to study this problem, to the best of our knowledge. To tackle challenges of KPI clustering, we propose or creatively integrate several effective techniques to achieve high robustness and efficiency in *ROCKA*. Our evaluation using real-world KPIs show that *ROCKA* gets F-score higher than 0.85, significantly outperforming a state-of-the-art time series clustering algorithm YADING. *ROCKA* also reduces the model training time of a state-of-the-art anomaly detection algorithm DONUT by 90%, with only 15% performance loss. This is the first reported study that uses clustering to reduce the training overhead of KPI anomaly detection.

We believe *ROCKA* is an important first step towards the direction of using KPI clustering to enable (previously infeasible) large-scale KPI anomaly detection, a key to ensure service reliability in the Internet.

VII. Acknowledgements

We thank Rui Ding for sharing us more details about YADING.

References

[1] Y. Chen, R. Mahajan, B. Sridharan, and Z.-L. Zhang, "A provider-side view of web search response time," *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4, pp. 243–254, 2013.

[2] D. Liu, Y. Zhao, H. Xu, Y. Sun, D. Pei, J. Luo, X. Jing, and M. Feng, "Opprentice: towards practical and automatic anomaly detection through machine learning," in *Proceedings of the 2015 ACM Conference on Internet Measurement Conference*. ACM, 2015, pp. 211–224.

[3] S. Zhang, Y. Liu, D. Pei, Y. Chen, X. Qu, S. Tao, and Z. Zang, "Rapid and robust impact assessment of software changes in large internet-based services," in *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies*. ACM, 2015, p. 2.

[4] N. Laptev, S. Amizadeh, and I. Flint, "Generic and scalable framework for automated time-series anomaly detection," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2015, pp. 1939–1947.

[5] H. Xu, D. Pei *et al.*, "Unsupervised anomaly detection via variational auto-encoder for seasonal kpis in web applications," in *Proceedings of the 27th International Conference on World Wide Web*. ACM, 2018. [Online]. Available: https://arxiv.org/abs/1802.03903

[6] A. Debrégeas and G. Hébrail, "Interactive interpretation of kohonen maps applied to curves." in *KDD*, vol. 1998, 1998, pp. 179–183.

[7] T. W. Liao, "Clustering of time series data–a survey," *Pattern recognition*, vol. 38, no. 11, pp. 1857–1874, 2005.

[8] S. Aghabozorgi *et al.*, "Time-series clustering–a decade review," *Information Systems*, vol. 53, pp. 16–38, 2015.

[9] J. Paparrizos and L. Gravano, "k-shape: Efficient and accurate clustering of time series," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. ACM, 2015, pp. 1855–1870.

[10] R. Ding, Q. Wang, Y. Dang, Q. Fu, H. Zhang, and D. Zhang, "Yading: Fast clustering of large-scale time series data," *Proceedings of the VLDB Endowment*, vol. 8, no. 5, pp. 473–484, 2015.

[11] S.-B. Lee, D. Pei, M. Hajiaghayi, I. Pefkianakis, S. Lu, H. Yan, Z. Ge, J. Yates, and M. Kosseifi, "Threshold compression for 3g scalable monitoring," in *INFOCOM*. IEEE, 2012, pp. 1350–1358.

[12] T. Rakthanmanon, B. Campana, A. Mueen, G. Batista, B. Westover, Q. Zhu, J. Zakaria, and E. Keogh, "Searching and mining trillions of time series subsequences under dynamic time warping," in *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2012, pp. 262–270.

[13] B.-K. Yi and C. Faloutsos, "Fast time sequence indexing for arbitrary lp norms." VLDB, 2000.

[14] H. Sakoe and S. Chiba, "Dynamic programming algorithm optimization for spoken word recognition," *IEEE transactions on acoustics, speech, and signal processing*, vol. 26, no. 1, pp. 43–49, 1978.

[15] X. Golay, S. Kollias, G. Stoll, D. Meier, A. Valavanis, and P. Boesiger, "A new correlation-based fuzzy logic clustering algorithm for fmri," *Magnetic Resonance in Medicine*, vol. 40, no. 2, pp. 249–260, 1998.

[16] J. MacQueen *et al.*, "Some methods for classification and analysis of multivariate observations," in *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, vol. 1, no. 14. Oakland, CA, USA., 1967, pp. 281–297.

[17] L. Kaufman and P. J. Rousseeuw, *Finding groups in data: an introduction to cluster analysis*. John Wiley & Sons, 2009, vol. 344.

[18] C. Biernacki *et al.*, "Assessing a mixture model for clustering with the integrated completed likelihood," *IEEE transactions on pattern analysis and machine intelligence*, vol. 22, no. 7, pp. 719–725, 2000.

[19] K. Kalpakis, D. Gada, and V. Puttagunta, "Distance measures for effective clustering of arima time-series," in *ICDM 2001, IEEE International Conference on Data Mining*. IEEE, 2001, pp. 273–280.

[20] M. Ester, H.-P. Kriegel, J. Sander, X. Xu *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise." in *Kdd*, vol. 96, no. 34, 1996, pp. 226–231.

[21] K. H. Zou, K. Tuncali, and S. G. Silverman, "Correlation and simple linear regression," *Radiology*, vol. 227, no. 3, pp. 617–628, 2003.

[22] Y. Chen, E. Keogh, B. Hu, N. Begum, A. Bagnall, A. Mueen, and G. Batista, "The ucr time series classification archive," *URL www. cs. ucr. edu/˜ eamonn/time_series_data*, 2015.

[23] Y. Zhong, S. Liu, X. Wang, J. Xiao, and Y. Song, "Tracking idea flows between social groups." in *AAAI*, 2016, pp. 1436–1443.

[24] L. Danon, A. Diaz-Guilera, J. Duch, and A. Arenas, "Comparing community structure identification," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2005, no. 09, p. P09008, 2005.

[25] M. Sölch, J. Bayer *et al.*, "Variational inference for on-line anomaly detection in high-dimensional time series," *International Conference on Machine Laerning Anomaly detection Workshop*, 2016.

[26] M. Amer, M. Goldstein, and S. Abdennadher, "Enhancing one-class support vector machines for unsupervised anomaly detection," in *Proceedings of the ACM SIGKDD Workshop on Outlier Detection and Description*. ACM, 2013, pp. 8–15.