Effective Anomaly Detection for Microservice Systems with Real-Time Feature Selection

Siqi Zhou*, Xian Zhan†, Linlin Li‡, and Yepang Liu§
Southern University of Science and Technology, Shenzhen, China
Email: *12032484@mail.sustech.edu.cn, †chichoxian@gmail.com, ‡lill3@mail.sustech.edu.cn, §liuyp1@sustech.edu.cn

***Abstract*—Microservice architecture is getting increasingly popular in recent years for building web-based systems. Finding runtime anomalies in such systems is crucial for improving their reliability. For this purpose, existing AIOps research has proposed various machine learning-based algorithms. However, a common limitation of existing algorithms is that they are sensitive to the settings of thresholds for anomaly identification when dealing with the high-dimensional multivariate time series data collected by monitoring the running instances of microservices. As a result, the performance of anomaly detection can be easily influenced by threshold changes. To tackle this problem, we propose a new anomaly detection framework called COAD (Combinatorial Optimization enhanced Anomaly Detection), which can work with various anomaly detection algorithms and enhance their detection process by performing real-time feature selection via metaheuristic algorithms. We have evaluated our method on three different testbeds based on a representative microservice system open-sourced by Google. The results show that real-time feature selection can significantly reduce the underlying algorithms' sensitivity to threshold settings (142% reduction on average). At the same time, the best anomaly detection performance (evaluated by f1-score) is improved by 5.67% on average. These results demonstrate the effectiveness and potential usefulness of the approach.**

***Index Terms*—Microservice, Anomaly Detection, Time Series, Feature Selection, AIOps**

I. Introduction

Nowadays, microservice architecture is widely adopted for web applications [1, 2]. The idea is to split a single monolithic application into a set of smaller, interconnected services. Deployed independently in their own environments, services communicate with each other using remote call. The advantages are improved productivity and increased scalability. However, the distributed nature of microservice architecture leads to a system based on it being complicated. Hence, each service should be correctly configured and continuously monitored in case of fault occurrence [3].

To reduce the labor cost and enable maintenance automation, AIOps (Artificial Intelligence for IT Operations) [4] is proposed, which adopts machine learning techniques to enhance IT operations. Anomaly detection [5] is an important task of AIOps and microservice systems requires automatic anomaly detection to prevent and report runtime faults. Multivariate time series monitored from microservice systems contain rich information. Each variable represents a feature (or a metric) in the whole system, like CPU utilization of a service instance. However, analyzing such multivariate time series to detect anomalies automatically is non-trivial. Firstly, the monitored multivariate time series easily becomes high-dimensional. Secondly, due to the complex dependencies, the monitored high-dimensional time series could have complicated relationships among its variables.

Existing researches have proposed various anomaly detection techniques for microservice systems [6–17]. However, according to our analysis, previous studies and methods have a common limitation: most anomaly detection methods are highly affected by threshold changes when dealing with high-dimensional time series. Via an empirical study (see Section III), we notice a phenomenon that many true faults (or abnormal states) are given relatively low anomaly scores by all researched detection algorithms [18–26] and are not easily distinguishable from normal states. That means a slight threshold change could lead to huge performance degradation. We further analyze such anomalies with low scores and find out the potential cause for this phenomenon: *anomalies triggered by different true faults might be reflected in different subsets of metrics* (i.e., not all monitored metrics are affected by each true fault). For a true fault, if the size of the affected metrics subset is small, but the detection algorithms consider all metrics, which include many unrelated ones, the anomaly score tends to be low. Then the anomaly can be hard to detect.

To deal with the abovementioned challenges, we propose a general framework to enhance existing anomaly detection algorithms for microservice systems. The proposed framework is called COAD (Combinatorial Optimization enhanced Anomaly Detection). Our basic idea is to combine anomaly detection algorithms with metaheuristic algorithms for real-time feature selection, which identifies related subsets of metrics for potential anomalies. Our approach is inspired by the widely studied topic of feature selection, where metaheuristic algorithms are often adopted for the near-optimal selection of complex features [27]. Since the related metric subsets are identified, the anomaly detection algorithms will not be influenced by the unrelated metrics, thus giving a more accurate anomaly score and showing a better detection performance. It is worth mentioning that different from approaches which simply choose important features before detection using expert knowledge or transform data into principle components

§Yepang Liu is affiliated with both the Department of Computer Science and Engineering and the Research Institute of Trustworthy Autonomous Systems. He is the corresponding author of this paper.

2640-0715/23/$31.00 ©2023 IEEE
DOI 10.1109/APSEC60848.2023.00020

using PCA (Principal Component Analysis) [26], our approach realizes continuous and real-time feature selection during the whole detection process and is able to select a proper metrics subset for each potential anomaly triggered by true faults.

To evaluate COAD, we have conducted experiments on 3 different microservice system testbeds based on a representative cloud-native microservices application open-sourced by Google [28]. We have applied 3 metaheuristic algorithms [29–31] with 3 anomaly detection methods [18, 24, 25] to validate the effectiveness and generalization of COAD. The experiment results show that COAD has the ability to mitigate existing anomaly detection methods' unstable performance related to threshold changes. On average, the performance instability caused by threshold changes is decreased by 142%. Besides, the maximum F1-score is increased by 5.67%, which shows our approach can improve the detection performance under the best threshold.

In summary, our work makes the following contributions:

- We empirically evaluate various anomaly detection algorithms on multivariate time series data from a real-world microservice system dataset [28]. Our study reveals that the existing anomaly detection algorithms suffer from the threshold sensitivity problem, i.e., their performance can be largely affected by threshold changes when dealing with high-dimensional time series.
- We propose a novel anomaly detection framework for microservice systems, which adopts metaheuristic algorithms for feature seletion. Our approach aims to narrow down the gaps among anomaly scores of true faults while keeping the true faults and normal patterns distinguishable, thus alleviating the threshold sensitivity problem and improving the performance.
- We have evaluated the proposed method on 3 testbeds and the experiment results show its expected effect.
- We release all the code, dataset, and experiment results [32] to facilitate further studies.

II. Background and Related work

In this section, we introduce the background of our research and existing studies that are highly related to our work.

A. Anomaly Detection of Microservice Systems

There are many AIOps researches on detecting microservice system anomalies. Microscope [6], for example, adopts the three-sigma rule of thumb to detect if a service instance is abnormal by analyzing univariate time series. MicroRCA [7] employs an online clustering method BIRCH [33] to detect the anomaly in the metric of response time. TraceRCA [8] uses a threshold to detect abnormal service invocations based on univariate time series.

Despite a large amount of AIOps researches, few studies try to analyze multivariate time series. These methods might miss the correlation among different metrics in the whole system and thus fail to detect many real anomalies.

B. Multivariate Time Series Anomaly Detection

We categorize existing multivariate time series anomaly detection methods into two main groups: 1) classical machine learning algorithms and 2) deep neural network-based methods. The former category includes KNN (K-Nearest Neighbors) [24], LOF (Local Outlier Factor) [25], KDE (Kernel Density Estimation) [23], COPOD (Copula-Based Outlier Detection) [21], IForest [22] and GMM (Gaussian Mixture Model) [18]. There is also a study of anomaly detection using PCA (Principal Component Analysis) [26]. The latter category uses deep learning to learn normal patterns and identify abnormal ones. For example, AutoEncoder [18], VAE (Variational Autoencoder) [20] and DeepSVDD (Deep Support Vector Data Description) [19] can capture more complex data patterns with sufficient training, compared with classical machine learning algorithms. However, these deep learning-based methods are usually adopted in an unsupervised way to conduct anomaly detection because the abnormal data is too scarce for supervised learning. Besides, they often require a huge amount of training data, which may hinder their applicability in practice.

C. Feature Selection

Feature selection methods are widely studied to deal with high-dimensional data [34]. They filter out unrelated features or metrics and use those most important ones to conduct data analysis. Metaheuristic algorithms belong to derivative-free algorithms for feature selection and work as a black box to help explore the search space efficiently and effectively [35]. There are plenty of practical metaheuristic algorithms in different areas. For example, PSO (Particle Swarm Optimization) [29] is inspired by schools of fishes and birds; GA (Genetic Algorithm) [31] mimics gene evolution; AEO (Artificial Ecosystem-based Optimization) [30] mimics the energy transfer in the ecosystem.

III. Empirical Study & Motivation

A. Dataset

The dataset for the empirical study is from the 2022 competition of the International AIOps Challenge [28], which is an influential competition in the field of intelligent operation & maintenance. The dataset was produced by Hipstershop [36], a representative cloud-native microservices application open-sourced by Google and it has 9 microservices as shown in Fig. 1. We use this microservice framework in 2 experiments: empirical study (to find the problem) & evaluation (to validate our method). While there is only one framework, it is deployed on 3 testbeds in an industrial environment, to produce the multivariate time series data for our study. Each service has several instances running on machine nodes and we use as much metrics in the system as we can to form the multivariate time series (around 2,000 metrics), such as the CPU load of a machine node or the memory consumption of an instance etc.

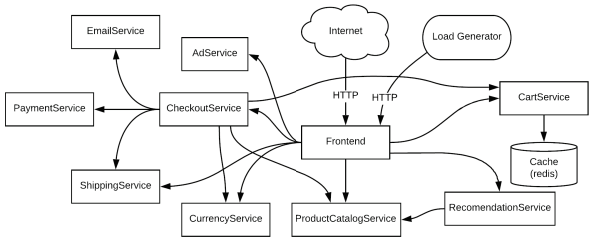


Fig. 1. The microservice system framework of Hipstershop [36].

B. The Threshold Sensitivity Problem

We have evaluated 10 popular anomaly detection algorithms [18–26] on the dataset. Fig. 2 shows an example of the anomaly detection results using GMM [18]. X-axis is the timestamp, and the interval between two consecutive timestamps is one minute. The maximum number on X-axis is the length of a day: $1440m = 24h{*}60m/h$ (*m* is minute and *h* is hour). The blue dots on the vertical Y-axis is the anomaly scores ranging from 0 to 1. The higher anomaly score, the more likely the algorithm believes there is a true fault. To clearly see the detection performance, we also plot the true fault happening time using vertical green lines. It can be seen that many blue dots are on the green lines vertically, which means that the anomaly detection algorithm can successfully pinpoint some true faults. However, the blue dots marked with red rectangles are not outstanding compared with those with yellow rectangles. For each anomaly detection method, a threshold should be set to determine how large the anomaly score of a true fault should be. True faults with low anomaly scores might make it difficult to choose a proper threshold and have a negative effect on the detection performance of an anomaly detection algorithm. More specifically, to catch the true faults with low anomaly scores, the threshold should be set to a small value, but then the algorithm may report many false positives. On the other hand, if the threshold is high, the true faults with low anomaly scores will be missed.

To clearly see how threshold settings may affect the performance of anomaly detection algorithms when there are true faults with low anomaly scores, we measure the following performance indicators with different threshold settings: precision, recall, and F1-score.

$$Precision = \frac{TP}{TP + FP} \quad , \quad Recall = \frac{TP}{TP + FN} \tag{1}$$

$$F1\_score = \frac{2 * Precision * Recall}{Precision + Recall} \tag{2}$$

In formula (1), *TP* means true positive (true faults correctly detected), *FP* means false positive (misjudged normal states), and *FN* means false negative (missed true faults). The main indicator is F1-score in formula (2), which is the harmonic mean of precision and recall. As shown in Fig. 3, the value of the three indicators changes when the threshold ranges from 0 to 1. However, the best threshold where the F1-score reaches the maximum is very close to 0, and the F1-score degrades very quickly when the threshold goes away from the best. This shows that the algorithm performance may vary significantly with different threshold settings (the threshold sensitivity problem).

To further investigate the threshold sensitivity problem, we decide to calculate two derivative indicators. The first indicator is M-value (Maximum), and the second is R-value (Random).

$$M\_value = max(F1\_score_i) \tag{3}$$

$$R\_value = \frac{\sum_{i=1}^{n} F1\_score_i}{n} \tag{4}$$

M-value is the F1-score with the best threshold, and R-value is the average F1-score of various thresholds obtained by Monte Carlo sampling. Their calculation formulas are (3) and (4), where n is the total times of Monte Carlo sampling and $F1\_score_i$ is the F1-score obtained by i-th sampling by choosing a random threshold in $[0, 1]$. In our study, we do the sampling 10,000 times (i.e., $n = 10000$).

TABLE I
THE PERFORMANCE OF THE TEN EVALUATED ALGORITHMS.

Method	M-value	R-value	AUC
AutoEncoder [18]	0.71	0.05	0.83
DeepSVDD [19]	0.63	0.11	0.81
VAE [20]	0.7	0.05	0.83
COPOD [21]	0.63	0.08	0.76
IForest [22]	0.61	0.18	0.74
GMM [18]	0.72	0.14	0.86
KDE [23]	0.66	0.17	0.78
KNN [24]	0.66	0.12	0.79
LOF [25]	0.67	0.11	0.78
PCA [26]	0.71	0.05	0.83

In Table I, we report the M-value and R-value of the ten evaluated algorithms on our dataset. It can be seen that all these algorithms have a relatively high M-value and low R-value, which means that although these methods can have a good performance with the best threshold, they are not robust to threshold changes. It is worth noting that PCA [26], which is a novel anomaly detection scheme based on principal component analysis, also suffers from the threshold sensitivity problem.

C. Motivation of Real-time Feature Selection

To find out the potential cause of the threshold sensitivity problem, we try to figure out the quantity distribution of the abnormal metrics at every timestamp. The three-sigma rule is adopted to roughly identify the abnormal metric. If the value of a metric v is not in $[\mu - 3\sigma, \mu + 3\sigma]$, where μ is the average of normal values and σ is the standard deviation of normal values, then the metric is considered abnormal.

Fig. 4 shows an example of a day's distribution. The X-axis is the timestamp, not sorted by time but by the number of abnormal metrics on the Y-axis. Green bars denote the normal states, and red bars denote the true faults. It is obvious that

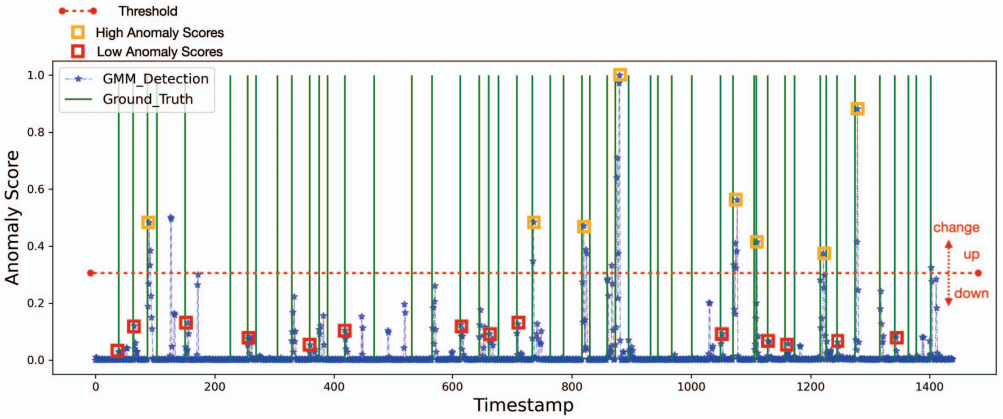


Fig. 2. Anomaly detection results of GMM.

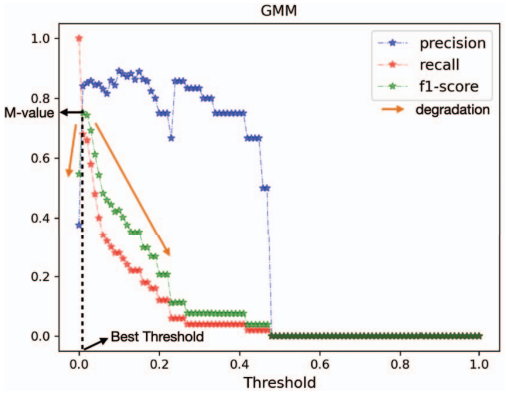


Fig. 3. GMM's detection performance with different threshold settings.

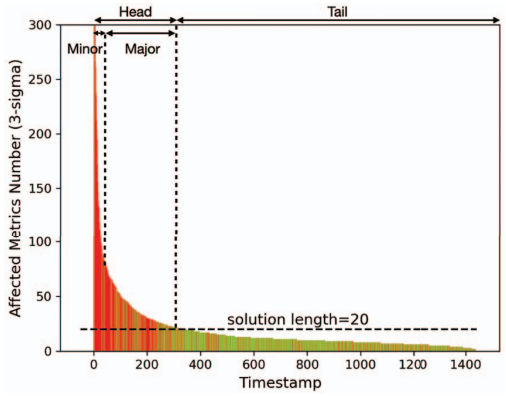


Fig. 4. An example of abnormal metrics quantity distribution.

the quantity of abnormal metrics obeys a long tail distribution. Most of the normal states are on the tail part of the distribution, and most of the true faults are on the head part, which means that the true faults have more abnormal metrics than normal states as judged by the three-sigma rule. The boundary

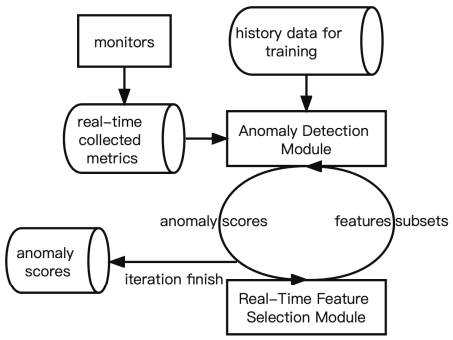


Fig. 5. The workflow of anomaly detection with real-time feature selection.

between the number of abnormal metrics at normal states and when true faults occur is roughly at the horizon line $y = 20$: most normal states have less than 20 abnormal metrics, and most true faults affect more than 20 metrics. The label "solution length" means the integer encoding length of features subset adopted in our framework COAD (see Section IV for more details). We choose 20 to balance the effect of COAD on true faults and normal states. In the head of the distribution, the minor part has much more abnormal metrics than the major part. If the monitored metrics are all considered for anomaly detection, the true faults at the minor part might have high anomaly scores (similar to the yellow rectangles in Fig. 2), and true faults at the major part might have low anomaly scores (similar to the red rectangles). This shows the possibility of improving the performance robustness of existing anomaly detection algorithms by automatically selecting the most relevant metrics at every timestamp, which we call *real-time feature selection*.

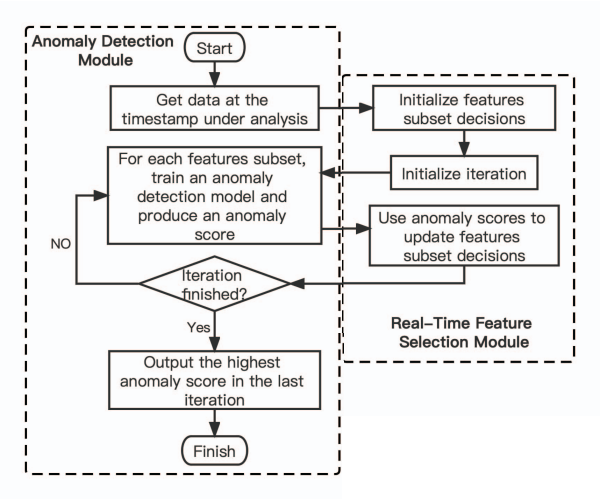


Fig. 6. The interaction between anomaly detection and feature selection.

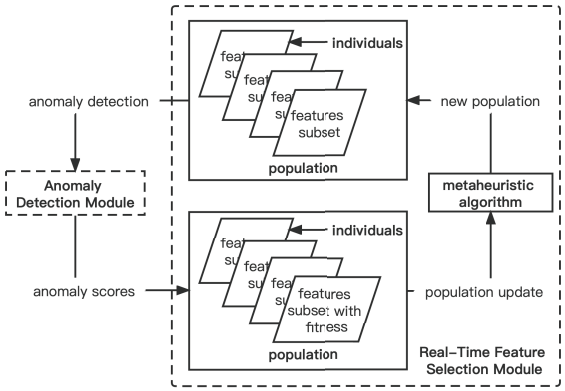


Fig. 7. Realization of real-time feature selection module.

IV. The COAD Framework

A. Overview

As shown in Fig. 5, microservice systems' metrics are continuously collected from monitors and then fed into the *anomaly detection module* to conduct real-time anomaly detection. Our framework COAD (**C**ombinatorial **O**ptimization enhanced **A**nomaly **D**etection) employs a *real-time feature selection module* to work with the existing anomaly detection module. In the workflow, a *features subset* is a combination of features, and COAD always tries to search for the best combination. The feature selection module needs some iterations at each timestamp to decide the final features subset for anomaly detection. During the iterative procedure, history data of system normal states is needed for training anomaly detection models. The real-time feature selection module needs the anomaly scores produced by the anomaly detection module, and the anomaly detection module needs the features subset decisions given by the feature selection module. After enough iterations of such interaction, the final anomaly score for the timestamp is outputted. It is worth noting that the interval between two consecutive timestamps is typically in the order of minutes (e.g., one minute in our dataset), so the time cost for the iterated interaction is affordable as long as it can finish within the interval.

B. The Iterative Interaction between Anomaly Detection and Feature Selection

Fig. 6 depicts the iterative interaction between the anomaly detection module and the feature selection module. The input includes the monitored data of the timestamp under analysis, the iteration times, and the configurations of the two modules. The configurations of the anomaly detection module specify the anomaly detection algorithm (e.g., GMM) and its parameters. The configurations of the feature selection module will be discussed in detail in Section IV-C.

At first, the feature selection module will provide some initial feature selection decisions to the anomaly detection module for producing anomaly scores of each decision. These produced anomaly scores will be used to guide the feature selection module's subsequent decisions. When the anomaly detection module receives the features selection decisions, it needs to train the anomaly detection algorithm's model separately for each decision, for the purpose of multivariate time series anomaly detection with the chosen features subset. After training, the anomaly detection module can use the models to produce anomaly scores for each selection decision. Finally, when the iterations are over, COAD chooses the highest anomaly score produced in the last iteration to output as the final anomaly score for the timestamp.

C. The Real-time Feature Selection Module

Fig. 7 describes how the real-time feature selection module can gradually improve its decisions. As mentioned in Section II-C, metaheuristic algorithms can be used to conduct feature selection by encoding the features subsets as integer vectors. Selected features subsets are called individuals or solutions. A group of individuals is called a population, which can be updated by metaheuristic algorithms towards a better population, which is more likely to get nicer feedback by environment (the anomaly detection module in our framework). The higher the anomaly score produced, the better the individual can be used to detect the potential true fault (the criteria might cause side effects and will be discussed in Section V-B). The feature selection module uses the anomaly scores to improve its feature selection decisions, just as the metaheuristic algorithm uses the fitness of individuals to update the population.

The real-time feature selection module configurations include the metaheuristic algorithm (COAD supports a wide range of nature-inspired metaheuristic algorithms, see Section V-A), the population size, and the individual's encoding length. These three can decide the selection quality and efficiency. Usually, a bigger-sized population and a longer

solution encoding length help generate better results with a higher time and computational cost. To strike a balance between effectiveness and efficiency, we set the encoding length to 20 according to our observation in Section III-C. If the encoding length is too long, the search space of the feature combination will become huge. On the other hand, If the encoding length is too short, normal states may be more likely to be misjudged as anomalies.

D. The Detailed Algorithms and Pseudo Code

Algorithm 1 The overall process of COAD

Input: *TD* (unsupervised training data), *Dt* (the time series data at timestamp t), Θ (metaheuristic algorithm), φ (anomaly detection algorithm), *PL* (population size), *IL* (integer vector length for encoding individuals), *N* (iteration times)

$i \leftarrow 0$
P_i (Population) $\leftarrow$ algorithm 2
repeat
  $i \leftarrow i + 1$
  F_i (Fitness) $\leftarrow$ algorithm 3
  P_i (Population) $\leftarrow$ algorithm 4
until $i = N$
$at \leftarrow max(F_N)$

Output: anomaly score at at timestamp t

Algorithm 2 Random initialization of features subsets

$i \leftarrow 0$
repeat
  $i \leftarrow i + 1$
  $I_i(individual) \leftarrow$ randomly generate an integer vector with length IL, which represents a features subset decision
until $i = PL$

Output: Initialized population $P = \{I_1, I_2, \cdots, I_{PL}\}$

Algorithm 3 Compute the anomaly score (fitness) for each features subset (individual)

$i \leftarrow 0$
repeat
  $i \leftarrow i + 1$
  $score_i \leftarrow$ the anomaly score from φ analyzing *Dt* using features subset I_i
until $i = PL$

Output: Anomaly scores (fitness) for each features subset (individual) $scores = \{score_1, score_2, \cdots, score_{PL}\}$

Algorithm 4 Update the population according to the fitness (anomaly score) of each individual (features subset)

$P = \{I_1, I_2, \cdots, I_{PL}\} \leftarrow$ the offspring of previous population using metaheuristic algorithm Θ

Output: Updated population $P = \{I_1, I_2, \cdots, I_{PL}\}$

The whole process of COAD for anomaly detection at timestamp *t* can be described using Algorithm. 1. First, the unsupervised training data TD, the time series data to be analyzed Dt, the anomaly detection algorithm φ, the metaheuristic algorithm Θ, the solution length (or the encoding length) IL, the population size PL, and the iteration times N is inputted to the algorithm. Then initialization and the iterations start. When the iterations finish, the final anomaly score at timestamp t is outputted. The initialization of features subsets is described in Algorithm 2. The procedure for computing anomaly scores is described in Algorithm 3. The solutions update step is described in Algorithm 4.

V. Evaluation

In this section, we evaluate COAD by investigating the following research questions:

- **RQ1**: Is COAD able to mitigate the threshold sensitivity problem of existing anomaly detection algorithms?
- **RQ2**: Does COAD have a side effect on normal timestamps? If there is a side effect, will it affect the anomaly detection performance?
- **RQ3**: How efficient is COAD? Is the computational and time cost affordable?
- **RQ4**: How does COAD compare with other dimension reduction methods?
- **RQ5**: Can COAD outperform neural network methods?
- **RQ6**: Can COAD generally enhance the performance of various existing multivariate time series anomaly detection algorithms?

A. Implementation and Experiment Design

We have implemented COAD using two popular python packages: PyOD [37] and MEALPY [38]. PyOD is the most comprehensive and scalable Python library for detecting outlying objects in multivariate data. MEALPY is the largest python library for most cutting-edge nature-inspired metaheuristic algorithms. PyOD is used to implement the anomaly detection method φ, and MEALPY is used to implement the metaheuristic algorithm Θ. All experiments are implemented on an iMac machine with 16GB DDR4, Intel Core i9 CPU, and macOS Monterey.

To verify the effectiveness and generalizability of COAD, three classical machine learning-based algorithms of different types are studied: GMM [18], KNN [24], and LOF [25], which are also used in Section III. All of them are fast to train a model. We do not evaluate neural network methods in the current work because directly enhancing them might cost lots of time and computational resources. Instead, We will discuss the idea of applying COAD to neural network methods in Section VI. For metaheuristic algorithms, we also choose three classic ones of different types: GA (Genetic Algorithm) [31], PSO (Particle Swarm Optimization) [29] and AEO (Artificial Ecosystem-based Optimization) [30]. GA is evolution-based, PSO is swarm-based, and AEO is system-based.

Every anomaly detection algorithm "XX" enhanced by a combinatorial optimization algorithm "YY" is denoted as "XX_YY", e.g., "PSO_GMM". The original anomaly detection algorithms are designated as "Original_YY", e.g., "Original_GMM", or directly by their name, e.g., "GMM". We

simulate real-time anomaly detection on three microservice system testbeds from the 2022 International AIOps Challenge. For the sake of fairness, the hyperparameters of each algorithm are the default values in their implementation library. The solution length IL is set to 20 to balance the anomaly score promotion of true faults and normal states, which has also been mentioned in the methodology section. The number of iterations N is set to 50, and the population size PL is set to 20 to balance the search adequacy and time consumption. We ran all experiments three times to eliminate the influence of random factors and report the average results in subsequent discussions.

The metrics to evaluate the experiment results include R-value, M-value, AUC (Area Under Curve), and time cost. R-value (Equation (4)) can evaluate an algorithm's performance robustness against threshold changes. M-value (Equation (3)) can evaluate the upper-bound performance of an algorithm. AUC [39] is widely used to measure the ability of a classifier to distinguish between classes. In our work, it can help measure the detection quality irrespective of the chosen thresholds. Finally, because metaheuristic algorithms are expensive, we also evaluate their time cost to see if COAD is practical.

B. Experiment Results and Analyses

There are plenty of experiment results as we try to enhance three anomaly detection algorithms using three combinatorial optimization methods, and all experiments are run three times. All these results can be obtained from our GitHub repository [40]. Due to the page limit, we only give examples and statistics in our paper in the following.

1) Threshold Sensitivity Problem: Answer to RQ1: Is COAD able to mitigate the threshold sensitivity problem of existing anomaly detection algorithms? Fig. 8 shows the detection result of PSO_GMM using the same data as in Fig. 2. Fig. 9 further shows its performance changes with different thresholds. As can be seen from Fig. 8, PSO_GMM greatly promotes the anomaly scores of those true faults annotated with red rectangles in Fig. 2. If we further compare Fig. 3 with Fig. 9, we can see that the performance degradation of PSO_GMM is much slower than Original_GMM. These results indicate that PSO_GMM is much improved compared with Original_GMM.

To validate whether other anomaly detection algorithms are generally improved, we measure the R-values of these algorithms and their enhanced versions. Because there are three testbeds, we make the statistics separately for each testbed and then calculate the average results among the three testbeds. As shown in Table II, all the anomaly detection algorithms using COAD have a higher R-value than their original version for each testbed. This shows that *COAD can generally mitigate the threshold sensitivity problem of existing machine learning-based anomaly detection algorithms*.

2) Side Effect: Answer to RQ2: Does COAD have a side effect on normal timestamps? If there is a side effect, will it affect the anomaly detection performance? As mentioned in Section IV-C, because the anomaly scores are always promoted due to the optimization (a higher anomaly score means better fitness), COAD might promote the anomaly scores of normal states, causing side effects. According to our experiments, we confirm that the side effect does exist. Here, we give an example. Fig. 10(a) shows the detection results of original_GMM, and Fig. 10(b) shows the detection results of PSO_GMM. The anomaly scores marked with red rectangles are the scores of normal states (not on the green lines indicating true fault happening times). As Fig. 10(b) shows, these normal states get a higher anomaly score by PSO_GMM than Original_GMM, meaning that these normal timestamps might be misjudged as anomalies.

TABLE II
THE R-VALUES OF DETECTION ALGORITHMS WITH AND WITHOUT COAD

Algorithms		R-value			
		Testbed1	Testbed2	Testbed3	Average
LOF	AEO	0.29 ± 0.01	0.28 ± 0.00	0.30 ± 0.01	0.29 ± 0.01
	GA	0.26 ± 0.01	0.29 ± 0.03	0.27 ± 0.03	0.27 ± 0.02
	PSO	0.32 ± 0.10	0.38 ± 0.03	0.32 ± 0.17	0.34 ± 0.04
	Original	0.05 ± 0.01	0.16 ± 0.02	0.12 ± 0.01	0.11 ± 0.01
KNN	AEO	0.23 ± 0.02	0.30 ± 0.01	0.26 ± 0.02	0.26 ± 0.02
	GA	0.31 ± 0.04	0.36 ± 0.03	0.34 ± 0.03	0.34 ± 0.03
	PSO	0.23 ± 0.00	0.29 ± 0.02	0.26 ± 0.00	0.26 ± 0.01
	Original	0.1 ± 0.01	0.13 ± 0.01	0.12 ± 0.01	0.12 ± 0.01
GMM	AEO	0.37 ± 0.03	0.32 ± 0.00	0.31 ± 0.06	0.33 ± 0.01
	GA	0.37 ± 0.02	0.30 ± 0.03	0.28 ± 0.05	0.32 ± 0.03
	PSO	0.52 ± 0.03	0.34 ± 0.05	0.45 ± 0.06	0.44 ± 0.02
	Original	0.24 ± 0.02	0.06 ± 0.01	0.11 ± 0.00	0.14 ± 0.01

TABLE III
THE AUC OF DETECTION ALGORITHMS WITH AND WITHOUT COAD

Algorithms		AUC			
		Testbed1	Testbed2	Testbed3	Average
LOF	AEO	0.84 ± 0.01	0.80 ± 0.03	0.80 ± 0.02	0.81 ± 0.02
	GA	0.83 ± 0.03	0.81 ± 0.01	0.81 ± 0.02	0.81 ± 0.01
	PSO	0.80 ± 0.00	0.82 ± 0.03	0.79 ± 0.02	0.81 ± 0.00
	Original	0.82 ± 0.01	0.77 ± 0.01	0.75 ± 0.01	0.78 ± 0.01
KNN	AEO	0.85 ± 0.01	0.82 ± 0.01	0.73 ± 0.04	0.80 ± 0.02
	GA	0.84 ± 0.00	0.80 ± 0.01	0.79 ± 0.04	0.81 ± 0.01
	PSO	0.84 ± 0.01	0.81 ± 0.01	0.79 ± 0.01	0.82 ± 0.01
	Original	0.85 ± 0.01	0.77 ± 0.00	0.74 ± 0.01	0.79 ± 0.01
GMM	AEO	0.87 ± 0.01	0.83 ± 0.00	0.82 ± 0.01	0.84 ± 0.01
	GA	0.87 ± 0.01	0.83 ± 0.00	0.82 ± 0.01	0.84 ± 0.01
	PSO	0.87 ± 0.01	0.82 ± 0.00	0.83 ± 0.00	0.84 ± 0.00
	Original	0.89 ± 0.00	0.85 ± 0.01	0.84 ± 0.01	0.86 ± 0.01

TABLE IV
THE M-VALUE OF DETECTION ALGORITHMS WITH AND WITHOUT COAD

Algorithms		M-value			
		Testbed1	Testbed2	Testbed3	Average
LOF	AEO	0.72 ± 0.01	0.75 ± 0.03	0.74 ± 0.01	0.74 ± 0.01
	GA	0.69 ± 0.02	0.77 ± 0.00	0.75 ± 0.02	0.74 ± 0.01
	PSO	0.61 ± 0.04	0.71 ± 0.03	0.67 ± 0.04	0.66 ± 0.02
	Original	0.7 ± 0.01	0.66 ± 0.02	0.66 ± 0.01	0.67 ± 0.01
KNN	AEO	0.69 ± 0.00	0.77 ± 0.00	0.68 ± 0.03	0.72 ± 0.01
	GA	0.69 ± 0.02	0.76 ± 0.02	0.70 ± 0.03	0.72 ± 0.02
	PSO	0.70 ± 0.01	0.79 ± 0.01	0.72 ± 0.01	0.73 ± 0.00
	Original	0.65 ± 0.00	0.66 ± 0.01	0.66 ± 0.00	0.66 ± 0.00
GMM	AEO	0.73 ± 0.01	0.78 ± 0.01	0.78 ± 0.01	0.76 ± 0.00
	GA	0.72 ± 0.01	0.80 ± 0.01	0.76 ± 0.00	0.76 ± 0.01
	PSO	0.71 ± 0.01	0.79 ± 0.00	0.78 ± 0.00	0.76 ± 0.00
	Original	0.74 ± 0.01	0.67 ± 0.02	0.75 ± 0.00	0.72 ± 0.01

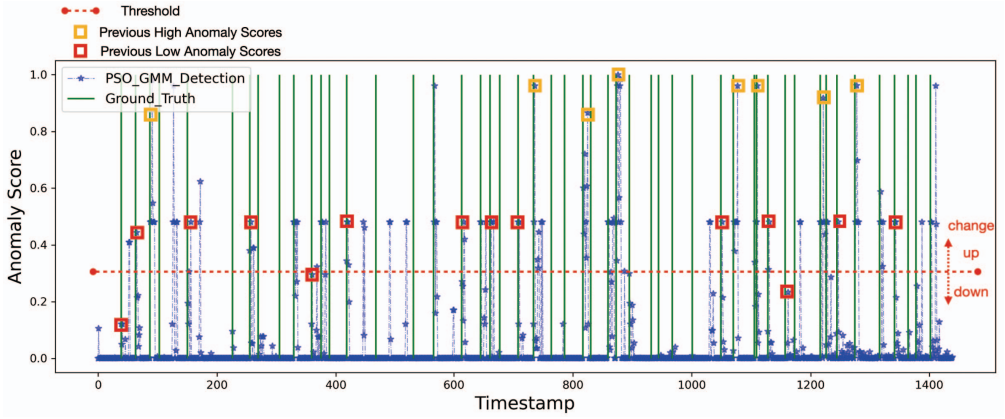


Fig. 8. Anomaly detection results of PSO_GMM.

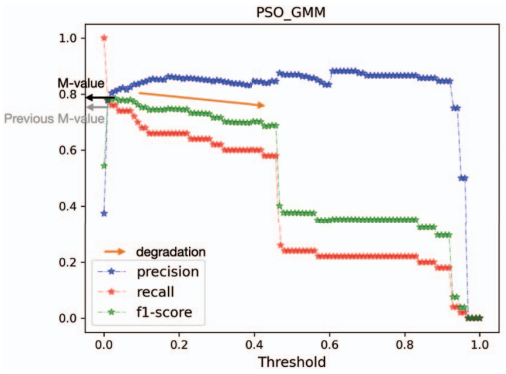


Fig. 9. PSO_GMM's performance changes with different thresholds.

Apart from R-value, we also make statistics of AUC [39] and M-value for different algorithms. Table III shows the statistics of AUC value. It can be seen that, apart from GMM, the other two anomaly detection algorithms have a higher average AUC value after adopting COAD, regardless of which combinatorial optimization algorithm is used. However, the enhanced LOF algorithms have slightly lower AUC values than the original LOF on Testbed1, and the enhanced GMM algorithms have slightly lower AUC values on all testbeds. The results contain little randomness as we have run the experiment three times. We conjecture that the effect of COAD in terms of AUC value may depend on the testbed and the enhanced algorithms. Although the side effect exists, the AUC value changes may be positive (i.e., the values increase) or negative (i.e., the values decrease).

Table IV shows the statistics of M-value. In practice, users may choose the best threshold through various ways to reach the best anomaly detection performance. M-value is hence an important indicator. It can be seen that, apart from PSO_LOF, all other enhanced algorithms have a higher M-value than the original algorithms. It means that in real applications, anomaly detection algorithms enhanced with COAD could generally have a higher upper bound of detection performance.

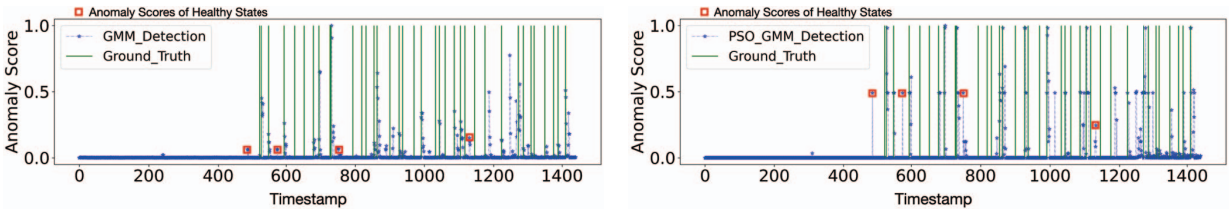
TABLE V
THE TIME COST OF DETECTION ALGORITHMS WITH COAD

	LOF			KNN			GMM		
	AEO	GA	PSO	AEO	GA	PSO	AEO	GA	PSO
Time Cost (s)	74.21	37.04	45.75	59.3	29.42	37.69	30.29	26.51	20.4

To sum up, *although the side effect exists, the enhancement is greater than the degradation*. It is likely because the abnormal features subset of normal states is difficult to search for metaheuristic algorithms, so the promotion of the anomaly scores of normal states might not be too much compared with that of abnormal states.

3) Time Cost: Answer to RQ3: How efficient is COAD? Is the computational and time cost affordable? We make the statistics about the average time cost of every evaluated algorithm. As shown in Table V, other than AEO_LOF, all enhanced algorithms can finish within one minute. So, *although COAD needs several iterations to get the optimal features subset at every timestamp, the time cost for the enhancement is affordable* since the interval between two timestamps is typically in minutes (one minute in our dataset). It is worth mentioning that the time cost might be further reduced if more powerful hardware (compared with our experiment environment as mentioned in Section V-A) is used or better parameters of the metaheuristic algorithms are configured (we did not perform tuning in our experiments).

4) Compared to Other Dimension Reduction Methods: Answer to RQ4: How does COAD compare with other dimension reduction methods? As mentioned in Section II-C, when dealing with high-dimensional multivariate time series, PCA [26] can also be used to project data to a lower dimensional space. Table I shows that the average M-value and R-value of PCA are 0.71 and 0.05, respectively. We also make the statistics about the average AUC value of PCA, and the result is 0.83. Compared with the algorithms enhanced by COAD, it can be seen that the AUC values and M-values do not have



(a) Anomaly detection results of GMM.

(b) Anomaly detection results of PSO_GMM.

Fig. 10. A side effect example in the experiment results.

much difference. However, PCA's R-value, which measures an algorithm's performance robustness against threshold changes, is significantly lower. This shows that COAD has a better ability than PCA to reduce performance instability due to threshold changes.

Neural network methods with feature selection layers also have the ability to perform dimension reduction. Table I lists the results of three neural network methods: AutoEncoder [18], DeepSVDD [19] and VAE [20]. Similar to PCA, they all have a much lower R-value than algorithms enhanced by COAD.

In conclusion, *other dimension reduction methods do not perform well concerning mitigating the threshold sensitivity problem*. This comparison result is understandable since COAD realizes real-time feature selection dynamically at each timestamp. Still, the other methods only perform dimension reduction once (PCA uses the data of normal states to analyze the principle components once, and neural network methods train their feature selection layers before anomaly detection).

5) Compared to Neural Network Algorithms: Answer to RQ5: Can COAD outperform neural network methods? Tables I, IV and II show that almost all anomaly detection algorithms with COAD have a higher M-value and R-value than neural network methods. We further make the statistics about AUC values: the AUC values of VAE and AutoEncoder are both 0.83, and the AUC value of DeepSVDD is 0.81. The results show that *COAD is as competitive as neural network methods in terms of M-value or AUC and is better than neural network methods with respect to R-value*. We have figured out two possible reasons. First, the anomaly detection is based on unsupervised learning. In reality, most monitored data is from normal states, and the anomaly detection cost with supervised learning can be expensive. When learning from a large amount of normal data and a tiny amount of abnormal data, the neural networks can only capture the known knowledge. They are not good at focusing on the small features subsets, which may reflect unknown anomaly patterns. In contrast, COAD is able to focus on these features in a real-time fashion. Second, complex and big neural network models and sufficient training data are often required to capture the complicated patterns in the high-dimensional time series data monitored from a microservice system. In our experiments, the tested neural network models' settings are the default ones in PyOD [37]. Therefore, these methods might not have reached their potential best performance. Nonetheless, our experiment settings are fair to all methods (we do not provide more data or perform tuning for other methods either).

6) General Promotion: Answer to RQ6: Can COAD generally enhance the performance of various existing multivariate time series anomaly detection algorithms? Tables II, III and IV show that all the three different types of anomaly detection algorithms can be enhanced by various metaheuristic algorithms, which shows the generalization capability of COAD. We calculate the average improvement over all the results: the average improvements of R-value, M-value, and AUC value are 142%, 5.67%, and 0.93%. So, we can conclude that *COAD can generally enhance the performance of various anomaly detection algorithms*.

C. Threats to Validity

The validity of our results might be subject to some threats. First, the implementation of COAD might contain bugs and can be further optimized. To mitigate the threat, we implemented COAD using two popular and well-tested python packages MEALPY [38] and PyOD [37]. We have tried our best to review and test the code of COAD. We also release our code for public scrutiny. Second, the parameters of all studied algorithms are not tuned to achieve the best results. However, these parameters are set to default settings for the sake of fair comparisons. Third, there may be randomness in the results of metaheuristic algorithms (e.g., the initialization and update of population have randomness). To reduce the influence of randomness, we ran all experiments three times and studied various algorithms on three different testbeds. The fourth threat is the reliability of the dataset. The dataset is produced by a real deployed microservice system from a well-known AIOps competition. The system adopts the framework Hipstershop [36], which is similar to other widely studied systems: Sock-Shop and Train-Ticket [41, 42].

VI. Conclusion and Future Work

In this paper, we have proposed COAD, a new anomaly detection framework for microservice systems. It enhances classical machine learning-based anomaly detection algorithms with combinatorial optimization algorithms. Via continuously searching for suspicious features subsets, COAD can realize feature selection in a real-time fashion to reduce the existing algorithms' sensitivity to threshold settings and enhance their performance. We have validated COAD's effectiveness on the 2022 International AIOps Challenge dataset based on three

different microservice system testbeds. The results show that COAD is able to reduce the original algorithms' threshold sensitivity by 142% and promote their performance upper bound by 5.67%. There are three aspects of implications in our work:

1) Our work implies that it is possible to combine machine learning and metaheursitic algorithms to accomplish specific tasks. We have successfully leveraged metaheuristic algorithms to address the threshold sensitivity problem of several unsupervised learning methods for anomaly detection in microservice systems. Future research may further explore more possibilities.
2) The subsets of metrics identified by COAD often have the highest anomaly scores. This implies that such identified metrics are likely most related to real anomalies and can assist in root cause analysis, which could significantly reduce human effort in fault diagnosis.
3) Our work also presents a useful paradigm to feature selection. The effectiveness of COAD implies that it is possible to dynamically and continuously select relevant features when analyzing high-dimensional multivariate time series. We hope that our work could shed light on future research in relevant areas.

For future work, we plan to validate the effectiveness of COAD on other datasets and algorithms. As for the application of COAD on neural network methods, we have the preliminary idea of getting features subsets from classical machine learning algorithms and then using the selected features to enhance neural network methods.

ACKNOWLEDGMENT

This work is supported by the National Natural Science Foundation of China (Grant No. 61932021).

REFERENCES

[1] T. Cerný, M. J. Donahoo, and M. Trnka, "Contextual understanding of microservice architecture: current and future directions," *ACM Sigapp Applied Computing Review*, vol. 17, pp. 29–45, 2018.
[2] V. Bushong, A. S. Abdelfattah, A. A. Maruf, D. Das, A. Lehman, E. Jaroszewski, M. Coffey, T. Cerný, K. Frajták, P. Tisnovsky, and M. Bures, "On microservice analysis and architecture evolution: A systematic mapping study," *Applied Sciences*, 2021.
[3] P. D. Francesco, P. Lago, and I. Malavolta, "Migrating towards microservice architectures: An industrial survey," *2018 IEEE International Conference on Software Architecture (ICSA)*, pp. 29–2909, 2018.
[4] "Aiops," https://www.gartner.com/en/information-technology/glossary/aiops-artificial-intelligence-operations.
[5] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Comput. Surv.*, vol. 41, pp. 15:1–15:58, 2009.
[6] J. Lin, P. Chen, and Z. Zheng, "Microscope: Pinpoint performance issues with causal graphs in micro-service environments," in *ICSOC*, 2018.
[7] L. Wu, J. Tordsson, E. Elmroth, and O. Kao, "Microrca: Root cause localization of performance issues in microservices," *NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium*, pp. 1–9, 2020.
[8] Z. Li, J. Chen, R. Jiao, N. Zhao, Z. Wang, S. Zhang, Y. Wu, L. Jiang, L. Yan, Z. Wang, Z. Chen, W. Zhang, X. Nie, K. Sui, and D. Pei, "Practical root cause localization for microservice systems via trace analysis," *2021 IEEE/ACM 29th International Symposium on Quality of Service (IWQOS)*, pp. 1–10, 2021.
[9] C. Zhang, X. Peng, C. Sha, K. Zhang, Z. Fu, X. Wu, Q. Lin, and D. Zhang, "Deeptralog: Trace-log combined microservice anomaly detection through graph-based deep learning," *2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE)*, pp. 623–634, 2022.
[10] Y. Cai, B. Han, J. Su, and X. Wang, "Tracemodel: An automatic anomaly detection and root cause localization framework for microservice systems," *2021 17th International Conference on Mobility, Sensing and Networking (MSN)*, pp. 512–519, 2021.
[11] P. Liu, H. Xu, Q. Ouyang, R. Jiao, Z. Chen, S. Zhang, J. Yang, L. Mo, J. Zeng, W. Xue, and D. Pei, "Unsupervised detection of microservice trace anomalies through service-level deep bayesian networks," *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*, pp. 48–58, 2020.
[12] C. Wu, N. Zhao, L. Wang, X. Yang, S. Li, M. Zhang, X. Jin, X. Wen, X. Nie, W. Zhang, K. Sui, and D. Pei, "Identifying root-cause metrics for incident diagnosis in online service systems," *2021 IEEE 32nd International Symposium on Software Reliability Engineering (ISSRE)*, pp. 91–102, 2021.
[13] J. Audibert, P. Michiardi, F. Guyard, S. Marti, and M. A. Zuluaga, "Usad: Unsupervised anomaly detection on multivariate time series," *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2020.
[14] X. Zhang, Q. Lin, Y. Xu, S. Qin, H. Zhang, B. Qiao, Y. Dang, X. Yang, Q. Cheng, M. Chintalapati, Y. Wu, K. Hsieh, K. Sui, X. Meng, Y. Xu, W. Zhang, S. Furao, and D. Zhang, "Cross-dataset time series anomaly detection for cloud systems," in *USENIX Annual Technical Conference*, 2019.
[15] L. Dai, T. Lin, C. Liu, B. Jiang, Y. Liu, Z. Xu, and Z.-L. Zhang, "Sdfvae: Static and dynamic factorized vae for anomaly detection of multivariate cdn kpis," *Proceedings of the Web Conference 2021*, 2021.
[16] M. Ma, S. Zhang, J. Chen, J. Xu, H. Li, Y. Lin, X. Nie, B. Zhou, Y. Wang, and D. Pei, "Jump-starting multivariate time series anomaly detection for online service systems," in *USENIX Annual Technical Conference*, 2021.
[17] Z. Li, N. Zhao, M. Li, X. Lu, L. Wang, D. Chang, X. Nie, L. Cao, W. Zhang, K. Sui, Y. Wang, X. Du, G. Duan, and D. Pei, "Actionable and interpretable fault localization for recurring failures in online service systems," *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2022.
[18] C. C. Aggarwal, "Outlier analysis," in *Springer New York*, 2013.
[19] L. Ruff, N. Görnitz, L. Deecke, S. A. Siddiqui, R. A. Vandermeulen, A. Binder, E. Müller, and M. Kloft, "Deep one-class classification," in *ICML*, 2018.
[20] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *CoRR*, vol. abs/1312.6114, 2014.
[21] Z. Li, Y. Zhao, N. Botta, C. Ionescu, and X. Hu, "Copod: Copula-based outlier detection," *2020 IEEE International Conference on Data Mining (ICDM)*, pp. 1118–1123, 2020.
[22] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation-based anomaly detection," *ACM Trans. Knowl. Discov. Data*, vol. 6, pp. 3:1–3:39, 2012.
[23] L. J. Latecki, A. Lazarevic, and D. Pokrajac, "Outlier detection with kernel density functions," in *MLDM*, 2007.
[24] F. Angiulli and C. Pizzuti, "Fast outlier detection in high dimensional spaces," in *PKDD*, 2002.
[25] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "Lof: identifying density-based local outliers," in *SIGMOD '00*, 2000.
[26] M.-L. Shyu, S. Chen, K. Sarinnapakorn, and L. Chang, "A novel anomaly detection scheme based on principal component classifier," 2003.
[27] P. Agrawal, H. F. Abutarboush, T. Ganesh, and A. W. Mohamed, "Metaheuristic algorithms on feature selection: A survey of one decade of research (2009-2019)," *IEEE Access*, vol. 9, pp. 26 766–26 791, 2021.
[28] "Aiops challenge," https://aiops-challenge.com/.
[29] R. Poli, J. Kennedy, and T. M. Blackwell, "Particle swarm optimization," *Swarm Intelligence*, vol. 1, pp. 33–57, 1995.
[30] W. guo Zhao, L. Wang, and Z. Zhang, "Artificial ecosystem-based optimization: a novel nature-inspired meta-heuristic algorithm," *Neural Computing and Applications*, vol. 32, pp. 9383–9425, 2019.
[31] L. D. Whitley, "A genetic algorithm tutorial," *Statistics and Computing*, vol. 4, pp. 65–85, 1994.
[32] "Coad," https://github.com/COAD2022/COAD.
[33] T. Zhang, R. Ramakrishnan, and M. Livny, "Birch: an efficient data clustering method for very large databases," in *SIGMOD '96*, 1996.
[34] G. Chandrashekar and F. Sahin, "A survey on feature selection methods," *Comput. Electr. Eng.*, vol. 40, pp. 16–28, 2014.
[35] M. Abdel-Basset, L. Abdel-Fatah, and A. K. Sangaiah, "Metaheuristic algorithms: A comprehensive review," 2018.
[36] "Hipstershop," https://github.com/abruneau/hipstershop.
[37] Y. Zhao, Z. Nasrullah, and Z. Li, "Pyod: A python toolbox for scalable outlier detection," *Journal of Machine Learning Research*, vol. 20, no. 96, pp. 1–7, 2019. [Online]. Available: http://jmlr.org/papers/v20/19-011.html
[38] N. V. Thieu and S. Mirjalili, "MEALPY: a Framework of The State-of-The-Art Meta-Heuristic Algorithms in Python," Jun. 2022. [Online]. Available: https://doi.org/10.5281/zenodo.6684223
[39] "Auc," https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc?hl=en.
[40] "Coad," https://github.com/COAD2022/COAD.
[41] "Sock shop," https://github.com/microservices-demo/microservices-demo.
[42] FudanSELab, "Train ticket," https://github.com/FudanSELab/train-ticket.