IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received May 30, 2018, accepted June 30, 2018, date of publication July 10, 2018, date of current version July 30, 2018.

Digital Object Identifier 10.1109/ACCESS.2018.2854836

A Parameter Space Framework for Online Outlier Detection Over High-Volume Data Streams

GUANZHE ZHAO[1], YANWEI YU[1], (Member, IEEE), PENG SONG[1], (Member, IEEE), GENG ZHAO[2], AND ZHE JI[3]

[1]School of Computer and Control Engineering, Yantai University, Yantai 264005, China
[2]College of Communication and Art Design, University of Shanghai for Science and Technology, Shanghai 200093, China
[3]China Mobile Group Jiangsu Company Ltd., Changzhou Branch, Changzhou 213000, China

Corresponding author: Yanwei Yu (yuyanwei@ytu.edu.cn)

This work was supported in part by the National Natural Science Foundation of China under Grant 61773331, Grant 61403328, Grant 61703360, and Grant 61572419, and in part by the Project of Shandong Province Higher Educational Science and Technology Program under Grant J17KA091.

ABSTRACT In diverse applications ranging from social networks to location-based online services to traffic monitoring, data streams are continuously monitored by multiple outlier analysts customized with different parameter settings. Real-time response to such complex outlier analytics in high-speed streaming data has been recognized as critical for many domains. In this paper, we propose a parameter space framework, called PSOD, for online outlier detection over sliding window streams to support a large variety of query requests in *parameter space* with both diverse pattern and window parameter settings. First, we design an ingenious neighbor table that records the neighbors for each point in different distance intervals and different slides, which enables us to maximally reuse the already acquired neighbor information across the entire parameter space. In addition, we propose a series of shared strategies in sliding window environment to minimize processing cost by eliminating the redundant query requests. Moreover, the PSOD effectively transforms the query group in 4-D parameter space into a periodic query group in 3-D parameter space to minimize the number of queries. Our experimental study on three real-world steaming data demonstrates that our PSOD successfully drives down the CPU costs by more than 100 folds compared with the state-of-the-art method.

INDEX TERMS Outlier detection, data streams, multi-query, parameter space.

I. INTRODUCTION

Nowadays, diverse high-speed huge-volume data streams are being generated from a variety of sources including social networks (e.g., Facebook and Twitter), location-based online services (e.g., DiDi and Uber), online shopping systems (e.g., Taobao and Amazon), mobile payments (e.g., Alipay and Apply pay), stock-trading systems (e.g., Robinhood and Acorns), and other smartphone applications. The discovery of abnormal patterns (i.e., outliers) from such high-speed huge-volume streaming data in a near real-time fashion has been recognized as critical for many domains [1]–[6].

In recent years, distance-based outlier detection [7], [8] have been widely applied in many applications due to their simplicity and insensitivity to concept drift [2]. The basic notion of anomaly detection in data can be traced back to the initial work by Hawkins [9]. The distance-based outlier model is initially proposed in [7] to capture outliers from multidimensional datasets. More specifically, an object o with fewer than k neighbors in the dataset D is defined as an outlier, where a neighbor of object o can be defined to be any other object in D that lies less than distance r from o. The model is consistent with Hawkins' definition and suitable for any situations where the distribution does not fit any standard distribution. As described above, to detect the outliers from streaming data using the distance-based outlier model, we need to first determine a set of input pattern parameters (i.e., *neighbor count threshold* k and *distance threshold* r) for processing system. For example, in credit card fraud monitoring system, analyst would continuously monitor the credit card transactions to find the unusual records that significantly differ from the majority of previous transactions made by card-holder or persons with similar income levels. Therefore, how to set the r that measures the notion of significant difference in transaction values and the k

that determine the majority of normal transaction records. In addition, for streaming data applications, sliding window are widely adopted to continuously detect the outliers on the most recent portion of data stream where people is more interested in. Therefore, the sliding window parameters, i.e., *window size* and *slide size*, also need to be provided using priori knowledge.

However, determining appropriate parameters is a critical challenge for many real-time data applications. This is because 1) data distribution may rapidly change with time, resulting in selected parameters in historical data may not fit, and 2) it is difficult to predict the data distribution in advance in streaming environment. Therefore, we may need to conduct multiple outlier detection queries with different parameter settings on a same data stream to discover outliers under different settings, which is utilized to understand the meaning of each parameter value in measuring the anomaly situation. For example, in credit card fraud monitoring system, how many neighbors being considered for each transaction record can define 80% of the majority. Moreover, outlier conditions for different situations on the same data stream may be considered by different user requirements in many applications. Since the continuous nature of streaming data and real-time response requirement, we have to handle a large number of outlier detection queries with various parameter settings within a large range on a data stream simultaneously. Namely, for an effective and efficient streaming outlier detection system, fast processing capacity to answer thousands or more of outlier queries covering a large space simultaneously is very necessary. However, to answer such a large number of query requests on a same data stream in real time, designing effective and tactfully shared processing strategies for outlier detection is very challenging.

In literature, several studies [10]–[13] have been done to discover distance-based outliers from data streams. Although these methods are efficient for handing a single outlier detection query, they fail to answer a large number of outlier detection queries simultaneously in real time because they only process each query independently. The most recent work [14] focuses on distance-based outlier analytics for multiple query requests with different parameter settings. They aim to transform the multi-query outlier problem into a single-query skyband problem. Although their proposed framework can support multiple queries with varying pattern-related parameters and window-related parameters, they only consider shared execution strategies when varying one specific parameter, ignoring the opportunities of reuse already acquired information across the parameter space.

In this work, we study shared execution strategies for supporting multiple outlier detection queries in *parameter space* to further improve the efficiency of outlier detection. First, we formally define the problem of distance-based outlier detection queries in parameter space with both pattern and window parameters in the stream contexts. Second, we propose an unified detection framework to support distance-based outlier queries in various dimensional parameter spaces over data streams. Specifically, we design an ingenious neighbor table that records the neighbors for each object in different distance intervals and different slides, which enable us to maximumly reuse the already acquired neighbor information across entire parameter space. We also propose a series of shared strategies in sliding window environment to minimize processing cost by eliminating the redundant query requests. Third, to minimize processing cost for supporting varying slide sizes, we propose a novel method that effectively transforms a outlier query group in four-dimensional parameter space into a periodic query group in three-dimensional parameter space. Finally, we conduct extensive experiments on three real-world datasets to demonstrate the efficiency of our proposed framework to answer huge numbers of query requests in parameter space.

To summarize, we make the following contributions:

- We propose a unified detection framework to support multiple distance-based outlier queries in various dimensional parameter spaces with both pattern and window parameters over data streams.
- we design an ingenious neighbor table that records the neighbors for each data point in different distance intervals and different slides, which enable us to maximumly reuse the already acquired neighbor information across entire parameter space.
- We propose a novel method that effectively transforms a group of outlier queries in four-dimensional parameter space into a group of queries with a periodically variable slide size in three-dimensional parameter space.
- Our experimental studies on three real-world datasets demonstrate that our framework significantly outperforms the state-of-the-art method by over 100-fold in CPU time.

II. RELATED WORK

In this section, we briefly review the related work in three areas: distance-based outlier detection in static datasets, in streaming data, and multi-query outlier detection optimization.

A. DISTANCE-BASED OUTLIERS IN STATIC DATASETS

Knorr and Ng [7] first propose the definition of distance-based outlier detection. In a follow-up work, Knorr *et al.* [8] apply the distance-based outlier model into outlier detection on spatiotemporal data. Ramaswamy *et al.* [15] propose an anomaly definition based on k-nearest neighbors. According to the distance from each data point to its k-*th* nearest neighbor, the top-n data points with the largest kNN distance are detected as the outliers. Subsequently, there have been many studies on outlier detection based on the distance-based outlier model [16], [17].

B. DISTANCE-BASED OUTLIERS OVER DATA STREAMS

Several methods have been proposed to detect distance-based outliers on streaming data [11], [12], [18]–[20].

Yang *et al*. [20] adopt a sliding window method to mine neighbor-based patterns on data streams, including density-based clustering and distance-based outlier detection. Specifically, they pre-compute the number of neighbors for each data point for each future window that the data points will participate in, which improves processing performance at the expense of a huge memory consumption. Angiulli and Fassetti [13], [19] optimize the detection method by analyzing whether data points are *safe inlier* on the data streams. If the number of neighbors that enter the sliding window after p is not less than k, then p is defined as *safe inlier*. Kontaki *et al*. [11] further improve the concept of *safe inlier* in [19]. More specifically, they employ the *safe inlier* concept to design a time-triggered event detection by scheduling the necessary checks according to the expiration time of preceding neighbors of unsafe inliers when the window slides. Cao *et al.* [14] propose an outlier detection framework over data streams, which supports two types of outlier detection models based on distance and kNN. They aim to find safe inliers as early as possible during the detection process by leveraging the temporal relationships among streaming data points to avoid full range query searches. Bu *et al*. [21] combine sliding window technology with a distance-based outlier definition to process outlier detection for trajectory data streams. The distance calculation between the sub-track is transformed into the cluster connection operation between local clusters, reducing a large amount of distance calculations. Yu *et al*. [22], [23] extend the distance-based outlier model to neighbor-based outlier detection in spatiotemporal trajectory streams by considering both spatial and temporal constrains. However, all these streaming outlier detection methods do not support multiple outlier detection requests.

C. MULTI-QUERY OUTLIER DETECTION

Recently, a line of research work focuses on multiple outlier detection queries over data streams. This line of research is the most relevant to ours. Multiple query sharing has been widely studied as a general optimization problem in streaming environments. Yang *et al*. [24], [25] propose a shared execution strategy based on predictive view growth theory and hierarchical clustering structure for multiple neighbor-based pattern mining requests with varying input parameters in a sliding window environment. In addition to focusing on distance-based outlier detection for a singe query request, [19] also discusses multiple-query outlier detection problem. However, they only consider outlier queries with different pattern parameter in the same sliding window. More recently, Cao *et al*. [14] present a solution for efficient shared processing of a large number of distance-based outlier detection requests with diverse parameter instantiations over sliding window streams. The method transforms multiple outlier detection queries into a single-query skyline by leveraging the domination relationships among the streaming data points. Although the method supports multiple outlier requests on data streams, it neglects the opportunities of sharing execution across the parameter space of both pattern and window parameters varying within a large range.

III. PROBLEM DEFINITION

In this section, we first define the key data structures and notations used in the paper. Then we formally state the focal problem to be solved. We now first review the notion of distance-based outliers [7], [8], [12]. In this work, we use the term *data point* to refer to a multi-dimensional tuple in a data stream, which is defined as follows:

Definition 1 (Data Point): A data point p_i is a triple $\langle i, t, atts\rangle$ that represents tuple p_i arriving at time t in the data stream, where atts denotes the multi-dimensional attributes of p_i.

A data stream DS is defined as an infinite sequence of data points ordered by time with $DS = \{p_1, p_2, \ldots, p_i, \ldots\}$ $(p_i.t \leq p_{i+1}.t,\ i = 1, 2, \ldots)$.

We use the function $dist(p_i, p_j)$ to denote the distance between two data points p_i and p_j. Without loss of generality, we utilize Euclidean distance as the distance measure, though any other distance measure could equally be plugged in.

We next define the notion of neighbor between any pairs of data points as follows:

Definition 2 (Neighbor): Given a distance threshold r, for two data points p_i and p_j, if $dist(p_i, p_j) \leq r$, then p_i and p_j are neighbors to each other.

For point p_i, all neighbors of p_i with regard to the distance threshold r comprise the neighbor set of p_i, denoted as $\mathcal{N}(p_i, r)$.

Definition 3 (Distance-based Outlier): Given a distance threshold r and a neighbor count threshold k, for any data point p_i, if $|\mathcal{N}(p_i, r)| < k$, then p_i is regarded as a distance-based outlier.

To capture the distance-based outliers in data streams, we use the periodic sliding window semantics as proposed by [26] for defining the sub-stream of the infinite data stream. The sliding window can be either time or count-based. In both cases, each sliding window has a window size w and a slide size s. For time-based sliding windows, each window W has an starting time $W.T_{start}$ and an ending time $W.T_{end} = W.T_{start} + w$. Periodically, the current window W_c slides, causing $W_c.T_{start}$ and $W_c.T_{end}$ to increase by s time points. For count-based sliding windows, the window size w corresponds to a fixed number of data points, which means each window contains w data points. Periodically the current window W_c slides s new data points, and $W_c.T_{start} \leq p_i.t < W_c.T_{end}$ for all $p_i \in W_c$.

We use $\mathcal{N}(p, r, W_c)$ to denote the set of neighbors of p in the current window W_c, i.e., $\mathcal{N}(p, r, W_c) = \{q | q \in \mathcal{N}(p_i, r) \wedge W_c.T_{start} \leq q.t < W_c.T_{end}\}$. We now define the distance-based outliers in sliding windows as follows:

Definition 4 (Distance-Based Outlier in Sliding Windows): Given a data stream DS, a distance threshold r, a neighbor count threshold k, for any data point p_i in the current window W_c, if $|\mathcal{N}(p, r, W_c)| < k$, then p_i is regarded as a distance-based outlier in window W_c.

Outliers in the current window W_c will be detected by Definition 4. However, the points in W_c may have a different status in the next windows over the data stream if it is still alive. Therefore, we need to continuously detect the statuses of data points as window slides.

Definition 5 (Outlier Detection Query in Sliding Windows): Given a data stream DS, a distance threshold r, a neighbor count threshold k, and the sliding window with window size w and slide size s, the outlier detection query $Q(r, k, w, s)$ continuously detects and outputs the distance-based outliers in each window in stream DS as the window slides.

Next we define the *parameter space* as a model for managing, exploring and optimizing multiple outlier detection queries on the same data stream as follows:

Definition 6 (Parameter Space): Given a data stream DS, a distance threshold set R, a neighbor count threshold set K, sliding window size set W, and slide size set S, we use a four-dimensional parameter space $R \times K \times W \times S$ to organize the parameter settings for the outlier query group $\mathbb{Q}$ that includes $Q(r, k, w, s)$ for all $(r, k, w, s) \in R \times K \times W \times S$ on stream DS.

Finally, we formally state our problem solved in this work as follows:

Problem 1 (Outlier Detection Queries in Parameter Space): Given a data stream DS, and a parameter space $R \times K \times W \times S$, our goal is to answer all outlier detection queries in $\mathbb{Q}$ for parameter space $R \times K \times W \times S$ using minimum processing time and memory consumption.

IV. OUTLIER DETECTION FRAMEWORK

In this section, we present our proposed parameter space framework for online outlier detection over data streams. We first introduce our shared execution strategy for detecting outliers in the two-dimensional parameter space $R \times K$. Then we present how our method works in the three-dimensional parameter spaces $R \times K \times W$. Next we introduce our method that transforms query group in the four-dimensional parameter space $R \times K \times W \times S$ into a periodic query group in the three-dimensional parameter spaces $R \times K \times W$. Finally we show our parameter space framework to handle outlier query group in $R \times K \times W \times S$.

A. OUTLIER DETECTION IN PARAMETER SPACE R × K

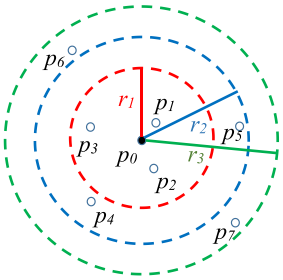
We first explore the sharing-aware outlier detection scheme in the two-dimensional parameter space $R \times K$, namely, we assume all queries share the same sliding window parameter w and s. So we omit w and s in this section. We can easily get the following two lemmas.

Lemma 1: Given a parameter space $R \times K$, $R = \{r_1, r_2, \ldots, r_m\}$ $(r_1 < r_2 < \cdots < r_m)$, $K = \{k_1, k_2, \ldots, k_n\}$ $(k_1 < k_2 < \cdots < k_n)$, for any data point p and $k \in K$, if $|\mathcal{N}(p, r_i)| \geq k$, p is a distance-based inlier w.r.t. parameter setting (r_j, k) in $R \times K$ for all $j \geq i$.

Lemma 2: Given a parameter space $R \times K$, $R = \{r_1, r_2, \ldots, r_m\}$ $(r_1 < r_2 < \cdots < r_m)$, $K = \{k_1, k_2, \ldots, k_n\}$ $(k_1 < k_2 < \cdots < k_n)$, for any data point p and $r \in R$, if $|\mathcal{N}(p, r)| < k_i$, then p is a distance-based outlier w.r.t. parameter setting $(r, k_j) \in R \times K$ for all $j \geq i$.

Lemma 1 and Lemma 2 are intuitive. Next we illustrate these two observations with an example.

Example 1: Given a query group $\mathbb{Q}$ in parameter space $\{r_1, r_2, r_3\} \times \{k_1, k_2, k_3\}$ as shown in Fig. 1, here we set $k_1 = 2$, $k_2 = 4$, and $k_3 = 8$.



Distance Interval	Neighbor Set
$[0,r_1]$	p_1,p_2,p_3
$(r_1,r_2]$	p_4,p_5
$(r_2,r_3]$	p_6,p_7

FIGURE 1. p_0 and its neighbor points.

We now describe how our shared execution strategy detects the statuses of p_0 in parameter space $\{r_1, r_2, r_3\} \times \{k_1, k_2, k_3\}$. We first introduce the data structure that enable our strategy to reuse the already found neighbors. We use a neighbor table, denoted as $p.NT$, to record the neighbor information in different distance intervals. As show in Fig 1, there are three distance intervals, i.e., $[0, r_1]$, $(r_1, r_2]$, $(r_2, r_3]$, for each data point according to the set of distance thresholds $\{r_1, r_2, r_3\}$.

We first use a range query operation to search neighbors for data point p_0 in the current window, and record the neighbor information in $p_0.NT$ according to the distance ranges. Thus the status of p_0 for parameter setting (r_1, k_1) can be determined. Since $\mathcal{N}(p_0, r_1) = \{p_1, p_2, p_3\}$, i.e., $|\mathcal{N}(p_0, r_1)| > k_1(2)$, p_0 is an inlier w.r.t. (r_1, k_1). By Lemma 1, we now directly get the statuses of p_0 for parameter settings (r_2, k_1) and (r_3, k_1) as shown in Table 1. Then we can quickly determine the status of p_0 w.r.t. (r_1, k_2). According to $\mathcal{N}(p_0, r_1)$, p_0 is an outlier for parameter setting (r_1, k_2) because $|\mathcal{N}(p_0, r_1)| < k_2(4)$. Therefore, the status of p_0 in the case of (r_1, k_3) can be directly detected by Lemma 2.

Next we examine the status of p_0 for (r_2, k_2). Obviously, $\mathcal{N}(p_0, r_2)$ is sum of neighbors in distance interval $[0, r_1]$ and $(r_1, r_2]$. So p_0 is an inlier w.r.t. (r_2, k_2) because $|\mathcal{N}(p_0, r_2)| > k_2(4)$. Similarly, p_0 is also an inlier for (r_3, k_2) by Lemma 1. Since $\mathcal{N}(p_0, r_2)$ is already recognized, the status of p_0 for (r_1, k_1) is also quickly detected.

For parameter setting (r_3, k_3), we only need to find out $\mathcal{N}(p_0, r_3)$ through $\mathcal{N}(p_0, r_2)$ plus the neighbors in distance interval $(r_2, r_3]$. Thus p_0 is an outlier in the case of (r_3, k_3) because $\mathcal{N}(p_0, r_3) < k_3(8)$.

In Table 1, we use the orange font to indicate the cases that can be directly determined without examination, the cyan font to the cases that can be quickly detected via checking on already recognized neighbor information, and the black font to the cases that need to look-up $p.NT$ and examine the neighbor condition. Therefore, our shared execution can

TABLE 1. The status of p_0 in parameter space (R, K).

Parameter space	$k_1 = 2$	$k_2 = 4$	$k_3 = 8$
r_1	inlier $[0, r_1]$	outlier	outlier
r_2	inlier	inlier $(r_1, r_2]$	outlier
r_3	inlier	inlier	outlier $(r_2, r_3]$

detect the statuses for each data points in parameter space $R \times K$ only performing a range query operation and limited number of table look-up operations.

However, to minimize the computation cost, we do not perform complete range query for searching neighbors of each data point in our method. nimum neighbor searching mechanism in Section IV-B and Section IV-D.

B. OUTLIER DETECTION IN PARAMETER SPACE $R \times K \times W$

Next we discuss the case that all detection queries in the three-dimensional parameter space $R \times K \times W$ have the same slide size s. All queries require to output the statuses of all data points in the windows that end with the current time. Namely, all queries share the same ending time $W_c.T_{end}$. This case is more complicated than the previous $R \times K$ because the status of each data point need to be considered in streaming context.

We now first introduce the status of data point in the context of streaming data. Traditional detection methods only divide a data point into two statuses: "inlier" and "outlier". However, in sliding window, the status of the data points can be further subdivided to reduce the distance calculation. To quickly determine the status of data points, we divide the neighbors of a data point p in the current window $\mathcal{N}(p, r, W_c)$ into two parts: neighbors before p and neighbors after p, denoted as $\mathcal{N}_{before}(p, r, W_c)$ and $\mathcal{N}_{after}(p, r, W_c)$ respectively. That is, $\mathcal{N}(p, r, W_c) = \mathcal{N}_{before}(p, r, W_c) \cup \mathcal{N}_{after}(p, r, W_c)$. Therefore, the status of point p in the current window W_c can be subdivided into:

- **Safe inlier (*s-inlier*).** If $|\mathcal{N}_{after}(p, r, W_c)| \geq k$, no matter how W_c slides, the number of neighbors in the sliding window for p must be greater than or equal to k before p expires. Thus the status of p can be determined as an inlier in the current window and subsequent windows before it expires, denoted as *s-inlier*.
- **Unsafe inlier (*u-inlier*).** If $|\mathcal{N}(p, r, w)| \geq k$, but the number of neighbors after p is less than k, i.e., $|\mathcal{N}_{after}(p, r, W_c)| < k$, then p is an inlier in the current window. However, the neighbors before p may slide out of the window as the window slides, so the status of p may become an outlier in the subsequent windows. Therefore, we consider p as an unsafe inlier in the current window, denoted as *u-inlier*.
- **Outlier.** If $|\mathcal{N}(p, r, w)| < k$, i.e., $|\mathcal{N}_{before}(p, r, W_c)| + |\mathcal{N}_{after}(p, r, W_c)| < k$, then p is regarded as an outlier in the current window, denoted as *outlier*. However, as the window slides, p may acquire k neighbors in the new window, thus the state of p may become *u-inlier* or *s-inlier*.

As the window slides, the neighbors in $\mathcal{N}_{before}(p, r, W_c)$ gradually slides out of the window, and the neighbors in $\mathcal{N}_{after}(p, r, W_c)$ would stay in the window until p slides out of the window. Therefore, the longer the effective lifetime of the later neighbor is, while for the previous neighbors, the closer to p, the longer survival period is. When searching for a neighbor of p, the neighbors behind it can be preferentially searched for. Based on this observation, in the window slides, the neighbors are searched in order from the last timestamp to previous ones, in this way, it can satisfy the priority search $\mathcal{N}_{after}(p, r, W_c)$, and at the same time, realizing the result that to search $\mathcal{N}_{before}(p, r, W_c)$ from the nearest to the farthest.

When detecting the point p, it is not necessary to search out all the neighbors. When k neighbor is acquired, the status of the p can be determined, so we use a minimum neighbor search mechanism to minimize the processing time. Given p and a new point p_{new} that just enters the window, if $|\mathcal{N}_{after}(p, r, W_c)| \geq k$ and also $|\mathcal{N}_{before}(p_{new}, r, W_c)| \geq k$, then there is no need to calculate the distance between p and p_{new}. At this point, p has been determined to be a *s-inlier*, and p_{new} has also been determined to be an *u-inlier*. Otherwise, it will continue to calculate the distance between p_{new} and other points in the current window till to determine that p_{new} is an *u-inlier* or an *outlier*. This mechanism not only satisfies the detection for all points but also achieves the minimum distance calculation.

Lemma 3: Given a parameter space $R \times K \times W$, $R = \{r_1, r_2, \ldots, r_m\}$ $(r_1 < r_2 < \cdots < r_m)$, $K = \{k_1, k_2, \ldots, k_n\}$ $(k_1 < k_2 < \cdots < k_n)$, $W = \{w_1, w_2, \ldots, w_x\}$ $(w_1 < w_2 < \cdots < w_x)$, for any data point p, $r \in R$ and $k \in K$, if $|\mathcal{N}_{after}(p, r, W_i)| \geq k$ (the window size of W_i corresponds to w_i), p is a s-inlier w.r.t. parameter setting $(r, k, w_j) \in R \times K \times W$ for all $j >= i$.

Lemma 4: Given a parameter space $R \times K \times W$, $R = \{r_1, r_2, \ldots, r_m\}$ $(r_1 < r_2 < \cdots < r_m)$, $K = \{k_1, k_2, \ldots, k_n\}$ $(k_1 < k_2 < \cdots < k_n)$, $W = \{w_1, w_2, \ldots, w_x\}$ $(w_1 < w_2 < \cdots < w_x)$, for any data point p, $r \in R$ and $k \in K$, if $|\mathcal{N}(p, r, W_i)| \geq k$ but $|\mathcal{N}_{after}(p, r, W_i)| < k$ (the window size of W_i corresponds to w_i), p is an u-inlier w.r.t. parameter setting $(r, k, w_j) \in R \times K \times W$ for all $j >= i$.

Lemma 3 and Lemma 4 are easily proved. We omit the proofs due to the limited space.

Next, we use an example to illustrate our shared execution strategy for outlier detection in parameter space $R \times K \times W$.

Example 2: Given a query group $\mathbb{Q}$ in parameter space $\{r_1, r_2, r_3\} \times \{k_1, k_2, k_3\} \times \{w_1, w_2, w_3\}$ as shown in Fig. 2. s_i $(i = 1, 2, \ldots, 7)$ denotes a slide with size s. We set $k_1 = 2$, $k_2 = 4$, $k_3 = 8$. For window size, we set $w_1 = 3*s$, $w_2 = 5*s$ and $w_3 = 6*s$. W_i corresponds to parameter w_i $(i = 1, 2, 3)$, and all windows share ending time and slide size s.

Similar to Example 1, we first introduce our designed neighbor table $p_0.NT$ that enables our strategy to reuse

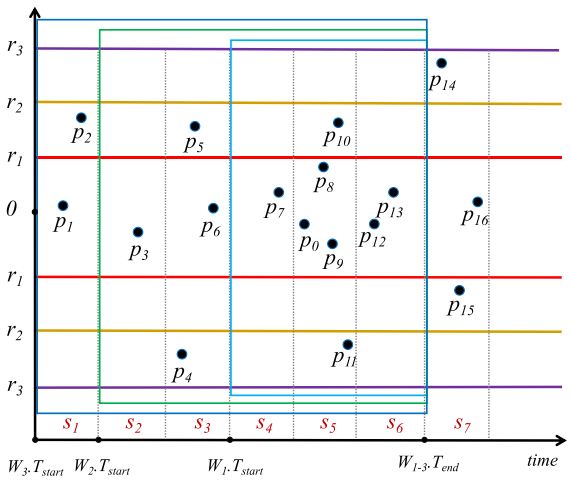


FIGURE 2. p_0 and its neighbor points in different windows.

the already found neighbors. To support neighbor reuse in sliding window, we extend our neighbor table to record the neighbor information in different distance ranges for each slide. As shown in Table 2, each column records the neighbors of p_0 in each slide for three distance intervals, i.e., $[0, r_1]$, $(r_1, r_2]$, $(r_2, r_3]$.

Suppose that the current time is t_6, namely, $W_1.T_{end} = W_2.T_{end} = W_3.T_{end} = t_6$. We first detect the statuses of p_0 in the current time for parameter space $\{r_1, r_2, r_3\} \times \{k_1, k_2, k_3\} \times \{w_1, w_2, w_3\}$. To obtain the neighbors of p_0, we first search neighbors for data point p_0 in each slide, and record the neighbor information in $p_0.NT$, shown as columns $s_1 - s_6$ in Table 2.

TABLE 2. Neighbor table of p_0.

Dist. Int.	s_1	s_2	s_3	s_4	s_5	s_6	s_7
$[0, r_1]$	p_1	p_3	p_6	p_7	p_8, p_9	p_{12}, p_{13}	p_{16}
$(r_1, r_2]$	p_2		p_5		p_{10}		p_{15}
$(r_2, r_3]$			p_4		p_{11}		p_{14}

Since $\mathcal{N}_{after}(p_0, r_1, W_1) = \{p_8, p_9, p_{12}, p_{13}\}$, i.e., $|\mathcal{N}_{after}(p_0, r_1, W_1)| > k_1(2)$, thus p_0 is a *s-inlier* for parameter setting (r_1, k_1, w_1). By Lemma 1, we now directly determine that p_0 is also a *s-inlier* for parameter settings (r_2, k_1, w_1), (r_3, k_1, w_1), (r_1, k_2, w_1), (r_2, k_2, w_1) and (r_3, k_2, w_1). We can also quickly detect that p_0 is a outlier for (r_1, k_3, w_1). By Lemma 3, we can know that the statuses of p_0 are all *s-inlier*s for parameter settings (r_1, k_1, w_2), (r_1, k_1, w_3), (r_2, k_1, w_2), (r_2, k_1, w_3), (r_3, k_1, w_2), (r_3, k_1, w_3), (r_1, k_2, w_2), (r_1, k_2, w_3), (r_2, k_2, w_2), (r_2, k_2, w_3), (r_3, k_2, w_2) and (r_3, k_2, w_3).

For parameter setting (r_1, k_3, w_2), we only need to find $\mathcal{N}(p_0, r_1, W_2)$ through $\mathcal{N}(p_0, r_1, W_1)$ plus the neighbors in slides s_2 and s_3, thus $\mathcal{N}(p_0, r_1, W_2) = \{p_3, p_6, p_7, p_8, p_9, p_{12}, p_{13}\}$, i.e., $|\mathcal{N}(p_0, r_1, W_2)| < k_3(8)$. Therefore, p_0 is an outlier for (r_1, k_3, w_2). Similarly, p_0 is an *u-inlier* for (r_1, k_3, w_3) because $|\mathcal{N}(p_0, r_1, W_3)| > k_3(8)$ but $|\mathcal{N}_{after}(p_0, r_1, W_3)| < k_3(8)$.

Next we examine the status of p_0 for (r_2, k_3, w_1). Since $|\mathcal{N}(p_0, r_2, W_1)| = |\{p_7, p_8, p_9, p_{10}, p_{12}, p_{13}\}| < k_3(8)$, thus p_0 is an outlier. Then we can determine the status of p_0 w.r.t. (r_2, k_3, w_2) by querying the neighbors in s_2 and s_3. p_0 is an *u-inlier* for parameter setting (r_2, k_3, w_2) because $|\mathcal{N}(p_0, r_2, W_2)| > k_3(8)$ but $|\mathcal{N}_{after}(p_0, r_2, W_2)| < k_3(8)$. Therefore, p_0 can be directly detected as an *u-inlier* in the case of (r_1, k_3, w_3) by Lemma 4.

Similarly, we can further detect the status of p_0 by exploring the neighbors in slides $[s_4, s_6]$ of distance interval $(r_2, r_3]$ in the basis of $\mathcal{N}(p_0, r_2, W_1)$. Since $|\mathcal{N}(p_0, r_3, W_1)| = |\{p_7, p_8, p_9, p_{10}, p_{11}, p_{12}, p_{13}\}| < k_3(8)$, p_0 is an outlier for parameter setting (r_3, k_3, w_1). For parameter setting (r_3, k_3, w_2), we can quickly get the status of p_0, i.e., *u-inlier*, because $|\mathcal{N}(p_0, r_2, W_2)| > k_3$ and $|\mathcal{N}_{after}(p_0, r_3, W_1)| < k_3$. Finally, p_0 is directly regraded as an *u-inlier* for (r_3, k_3, w_3) by Lemma 4.

Like Table 1, we use the orange font to indicate the cases that can be directly determined without examination, the cyan font to the cases that can be quickly detected via checking on already recognized neighbor information, and the black font to the cases that need to look-up $p.NT$ and examine the neighbor condition in Table 3.

TABLE 3. The status of p_0 in parameter space (R, K, W).

Param.	$w_1 = 2$	$w_2 = 4$	$w_3 = 8$
k_1=2, r_1	*s-inlier* $[0, r_1]$,$[s_5, s_6]$	*s-inlier*	*s-inlier*
k_2=4, r_1	*s-inlier*	*s-inlier*	*s-inlier*
k_3=8, r_1	*outlier* $[0, r_1]$,$[s_4]$	*outlier* $[0, r_1]$,$[s_2, s_3]$	*u-inlier* $[0, r_1]$,$[s_1]$
k_1=2, r_2	*s-inlier*	*s-inlier*	*s-inlier*
k_2=4, r_2	*s-inlier*	*s-inlier*	*s-inlier*
k_3=8, r_2	*outlier* $(r_1, r_2]$,$[s_4, s_6]$	*u-inlier* $(r_1, r_2]$,$[s_2, s_3]$	*u-inlier*
k_1=2, r_3	*s-inlier*	*s-inlier*	*s-inlier*
k_2=4, r_3	*s-inlier*	*s-inlier*	*s-inlier*
k_3=8, r_3	*outlier* $(r_2, r_3]$,$[s_4, s_6]$	*u-inlier*	*u-inlier*

Next, we illustrate how our strategy updates the statuses of data points with windows slide. We now suppose all windows slide to s_7. According to our previous analysis, we know that if the status of p_0 is a *s-inlier*, the status of p_0 will not change as windows slides until data point p_0 expires. As shown in Table 3, we only need to re-detect the statuses for p_0 in the cases of *u-inlier* and *outlier*.

For parameter setting (r_1, k_3, w_1), we only need to compare the number of neighbors in new slide s_7 with that in expired slide s_4 w.r.t distance interval $[0, r_1]$. Because $|\mathcal{N}^{new}(p_0, r_1, W_1)| = |\mathcal{N}^{old}(p_0, r_1, W_1)| - |s_4([0, r_1])| + |s_7([0, r_1])| = |\{p_8, p_9, p_{12}, p_{13}, p_{16}\}| < k_3(8)$, p_0 is still an outlier w.r.t. (r_1, k_3, w_1) in the current time. Similarly, we can update the statuses of p_0 for parameter settings (r_1, k_3, w_2) and (r_1, k_3, w_3).

TABLE 4. The updated status for p_0 in new sliding windows.

Parameters	$w_1 = 2$	$w_2 = 4$	$w_3 = 8$
$k_1 = 2, r_1$	×	×	×
$k_2 = 4, r_1$	×	×	×
$k_3 = 8, r_1$	*outlier* $[0, r_1]$,$[s_4][s_7]$	*outlier* $[0, r_1]$,$[s_2]$	*outlier* $[0, r_1]$,$[s_1]$
$k_1 = 2, r_2$	×	×	×
$k_2 = 4, r_2$	×	×	×
$k_3 = 8, r_2$	*outlier* $(r_1, r_2]$,$[s_4][s_7]$	*u-inlier* $(r_1, r_2]$,$[s_2]$	*u-inlier*
$k_1 = 2, r_3$	×	×	×
$k_2 = 4, r_3$	×	×	×
$k_3 = 8, r_3$	*s-inlier* $(r_2, r_3]$,$[s_4][s_7]$	*s-inlier*	*s-inlier*

Then we examine the new status of p_0 for (r_2, k_3, w_1). p_0 is till an outlier because $\mathcal{N}^{new}(p_0, r_2, W_1) = \{p_8, p_9, p_{10}, p_{12}, p_{13}, p_{15}, p_{16}\}$, i.e., $|\mathcal{N}^{new}(p_0, r_2, W_1)| < k_3(8)$. For parameter setting (r_2, k_3, w_2), although $|\mathcal{N}^{new}(p_0, r_2, W_2)| > k_3(8)$, $|\mathcal{N}^{new}_{after}(p_0, r_2, W_2)| < k_3(8)$. Thus p_0 is still an *u-inlier*. By Lemma 4, we can know p_0 is an *u-inlier* for parameter setting (r_2, k_3, w_3) in the current time.

At last, we update the status of p_0 w.r.t. (r_3, k_3, w_1). Due to $|\mathcal{N}^{new}_{after}(p_0, r_3, W_1)| = |\{p_8, p_9, p_{10}, p_{11}, p_{12}, p_{13}, p_{14}, p_{15}, p_{16}\}| > k_3(8)$, p_0 becomes a *s-inlier*. By Lemma 3, we directly update the status of p_0 to *s-inlier* for parameter settings (r_3, k_3, w_2) and (r_3, k_3, w_3).

In Example 2, the sizes of all windows are set to integer multiples of slide size s. However, in reality, the sizes of some windows in W may not be divisible by the slide size, e.g., $w = 5, s = 2$. To support the outlier examination for such queries using above method, we set a minimum slide size s_{min} to the greatest common divisor on all windows sizes in W and slide size s. We can record the neighbors for each data point in each minimum slide, and each time all windows slide s/s_{min} minimum slides.

C. OUTLIER DETECTION IN PARAMETER SPACE R × K × S

In this section, we present our shared execution strategy for examining outliers in the three-dimensional parameter spaces $R \times K \times S$. In this case, all detection queries have the same window size w in streaming context.

Similar to Section IV-B, for all slide sizes in S, to share the acquired neighbors in the sliding window, we need to find the greatest common divisor s_{min} on all slide sizes and the window size. We divide the sliding window into the minimum slides with length s_{min}. For example, in Fig. 3, $s^i_m (i = 1, 2, \ldots, 9)$ is a minimum slide with length s_{min}.

Example 3: Given a query group $\mathbb{Q}$ in parameter space $\{r_1, r_2, r_3\} \times \{k_1, k_2, k_3\} \times \{s_1, s_2, s_3\}$ as shown in Fig. 3. We set $k_1 = 2, k_2 = 4, k_3 = 8$. Window size is set to 5, i.e., $w = 5$. Suppose $s_1 = 2$, $s_2 = 3$, $s_3 = 4$. We find the greatest common divisor of the window size w and all slide sizes is 1. Thus we divide the sliding window into five equal parts with length 1. Since $s_3 = 2 * s_1$, as for s_3, it is equivalent to the window sliding s_1 twice. We can reuse the query results of s_1 for s_3 thus omit the queries for parameter s_3.

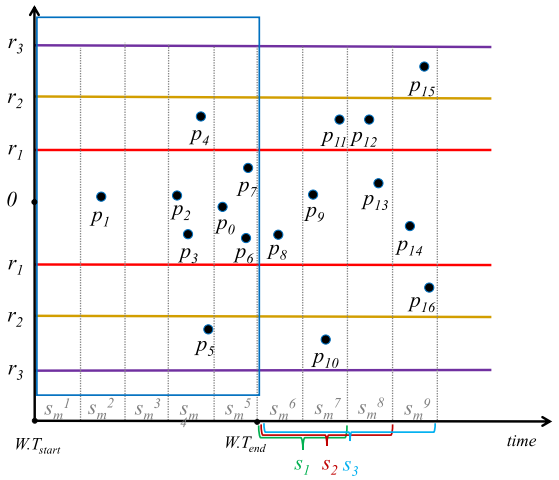


FIGURE 3. p_0 and its neighbor points in sliding window with different slides.

As shown in Fig. 3, the query group $\mathbb{Q}$ in $\{r_1, r_2, r_3\} \times \{k_1, k_2, k_3\} \times \{s_1, s_2, s_3\}$ is reduced to $\mathbb{Q}'$ in $\{r_1, r_2, r_3\} \times \{k_1, k_2, k_3\} \times \{s_1, s_2\}$. Next we discuss how our method reuse the execution for slide sizes s_1 and s_2 to each other.

First, we can examine the statuses of p_0 for parameter settings $\{r_1, r_2, r_3\} \times \{k_1, k_2, k_3\} \times \{s_1\}$. Using the method in last section, we determine the statuses of p_0 shown as column s_1 in Table 5.

TABLE 5. The status of p_0 in parameter space (R, K, S).

Parameters	$s_1 = 2$	$s_2 = 3$
$k_1 = 2, r_1$	*s-inlier*	*s-inlier*
$k_2 = 4, r_1$	*s-inlier* $[0, r_1]$, $[s^1_m, s^2_m], [s^6_m, s^7_m]$	*s-inlier*
$k_3 = 8, r_1$	*outlier*	*outlier* $[0, r_1]$, $[s^3_m], [s^8_m]$
$k_1 = 2, r_2$	*s-inlier*	*s-inlier*
$k_2 = 4, r_2$	*s-inlier*	*s-inlier*
$k_3 = 8, r_2$	*u-inlier* $(r_1, r_2]$, $[s^1_m, s^2_m], [s^6_m, s^7_m]$	*u-inlier* $(r_1, r_2]$, $[s^3_m], [s^8_m]$
$k_1 = 2, r_3$	*s-inlier*	*s-inlier*
$k_2 = 4, r_3$	*s-inlier*	*s-inlier*
$k_3 = 8, r_3$	*s-inlier* $(r_2, r_3]$, $[s^1_m, s^2_m], [s^6_m, s^7_m]$	*s-inlier*

Then we detect the statuses of p_0 for parameter settings $\{r_1, r_2, r_3\} \times \{k_1, k_2, k_3\} \times \{s_2\}$. Obviously, we can reuse the execution of s_1 for s_2. Namely, we can get the detection result for s_2 by sliding the current window a minimum

slide. Therefore, we can quickly obtain the statuses of p_0 in $\{r_1, r_2, r_3\} \times \{k_1, k_2, k_3\} \times \{s_2\}$ by checking the neighbors in the new minimum slide s^8_{min}.

Next we again consider the question that how to reuse the current examination for next queries in $\{r_1, r_2, r_3\} \times \{k_1, k_2, k_3\} \times \{s_1\}$. Therefore, we can transform the query group in $\{r_1, r_2, r_3\} \times \{k_1, k_2, k_3\} \times \{s_1, s_2\}$ into a query group in $\{r_1, r_2, r_3\} \times \{k_1, k_2, k_3\} \times \{s_i^{var}\}$, where s_i^{var} is a variable slide size.

To resolve the execution reuse for the slide sizes in S, we first unfold the queries $\mathbb{Q}'$ w.r.t $\{r_1, r_2, r_3\} \times \{k_1, k_2, k_3\} \times \{s_1, s_2\}$ on the axis of slide size. Obviously, given the current window W_c, the set of slide sizes is periodic. We can easily find the first round of slide sizes with respect to the current window: $S = \{s_1, 2 * s_1, \ldots, s_2 * s_1, s_2, 2 * s_2, \ldots, s_1 * s_2\}$. Thus the smallest slide size is s_1, and the largest slide size is the least common multiple on s_1 and s_2. By removing duplicates and sorting by size, we can determine the periodic slide sizes, e.g., $\{s_1, s_2, 2 * s_1, s_1 * s_2\} = \{2, 3, 4, 6\}$. Therefore, the variable slide size s_i^{var} follows: $s_1^{var} = 2, s_2^{var} = 1, s_3^{var} = 1, s_4^{var} = 2$. Namely, the sliding window periodically slides from s_1^{var} to s_4^{var} to maximally reuse the execution to answer the detection query group $\mathbb{Q}$.

We call the set of periodic slide sizes for the variable slide size s_i^{var} with respect to S **periodic slide size series**, denoted as $P_s(S)$.

By the above observation, next we define our slide-sharing detection optimization based on this transformation.

Principle 1 (Slide-sharing Detection): Given a parameter space $R \times K \times W \times S$, $R = \{r_1, r_2, \ldots, r_m\}$ $(r_1 < r_2 < \cdots < r_m)$, $K = \{k_1, k_2, \ldots, k_n\}$ $(k_1 < k_2 < \cdots < k_n)$, $W = \{w_1, w_2, \ldots, w_x\}$ $(w_1 < w_2 < \cdots < w_x)$, $S = \{s_1, s_2, \ldots, s_y\}$ $(s_1 < s_2 < \cdots < s_y)$, the query group $\mathbb{Q}$ in parameter space $R \times K \times W \times S$ is equivalent to the query group $\mathbb{Q}'$ in parameter space $R \times K \times W \times \{s_i^{var}\}$, where s_i^{var} periodically follows the periodic slide size series $P_s(S)$.

By the slide-sharing detection principle, the query group in four-dimensional parameter space $R \times K \times W \times S$ are effectively transformed into a periodic query group in three-dimensional parameter space $R \times K \times W \times \{s_i^{var}\}$.

D. PSOD FRAMEWORK FOR OUTLIER DETECTION

Next we show how our PSOD (Parameter Space Framework for Outlier Detection) detects the outliers in four-dimensional parameter space $R \times K \times W \times S$. As shown in Algorithm 1, take data point p for example, PSOD updates the status of p for each query in parameter space with time. $f_{st}(p, r_j, k_x, w_y)$ denotes the status of p with respect to $Q(r_j, k_x, w_y)$.

First, PSOD transforms the query group in four-dimensional parameter space $R \times K \times W \times S$ into a query group in three-dimensional parameter space $R \times K \times W$ by applying our proposed Principle 1 (line 1). Second, PSOD directly eliminates the queries $Q(r, k, w)$ for p if p have already been a *s-inlier* in last window (lines 5-6). Then, PSOD will explore the neighbor points for p in the new slide S_{new} in order from the last timestamp to previous ones. For any neighbor

Algorithm 1 PSOD Framework

Input: Data Stream DS, Query group $\mathbb{Q}$, R, K, W and S.
Output: The status of p in parameter space $R \times K \times W \times S$

```
1:  for s_i^var ∈ P_s(S) do
2:      for r_j ∈ R do
3:          for k_x ∈ K do
4:              for w_y ∈ W do
5:                  if f_st(p, r_j, k_x, w_y) == s-inlier then
6:                      break;
7:                  else if f_st(p, r_j, k_x, w_y) has been updated
    then
8:                      break;
9:                  else
10:                     for p_new^{l-1} from S_new.T_end to S_new.T_start
    do
11:                         if dist(p, p_new) ∈ (r_a, r_b] then
12:                             p.NT.add(p_new, (r_a, r_b]);
13:                             p_new.NT.add(p, (r_a, r_b]);
14:                         if N_after(p, r_j, W_c) ≥ k_x then
15:                             f_st(p, r_≥j, k_x, w_≥y)←s-inlier;
16:                             mark p_new^l; break;
17:                     if N_after(p, r_j, W_c) < k_x then
18:                         if N(p, r_j, W_c) ≥ k_x then
19:                             f_st(p, r_j, k_x, w_≥y)←u-inlier;
20:                         else
21:                             f_st(p, r_j, k_≥x, w_y)←outlier;
```

points of p in distance interval $(r_a, r_b]$, we put them into corresponding row of $p.NT$. Once enough neighbors in new slide are acquired (i.e., $|\mathcal{N}_{after}(p, r_j, W_c)| \geq k_x$), PSOD then early terminates the searching because the *s-inlier* status of p have already been proved in the current window with respect to $Q(r_j, k_x, w_y)$. Therefore, as shown in line 15, the status of p for all queries $Q(r_{\geq j}, k_x, w_{\geq y})$ is set to *s-inlier* by Lemma 1 and Lemma 3. Note that $f_{st}(p, r_{\geq j}, k_x, w_{\geq y})$ denotes the status of p with respect to all queries with parameter settings $r \geq r_j, k = k_x$ and $w \geq w_y$. To avoid double counting the distance between two points, we mark the location of p^l_{new} for p in the new slide. If p needs more neighbors for larger k, our PSOD would continue to search neighbors for p starting at p^{l-1}_{new}. Hence PSOD at most executes a complete range query to search neighbors of p for supporting different distance thresholds.

Next, if $|\mathcal{N}_{after}(p, r_j, W_c)| < k_x$ but $\mathcal{N}(p, r_j, W_c) \geq k_x$, then p is a *u-inlier* for queries with parameter settings $r = r_j, k = k_x$ and $w \geq w_y$ by Lemma 4 (line 19). If $\mathcal{N}(p, r_j, W_c) \geq k_x$, then p is a *outlier* for queries with parameter settings $r = r_j, k \geq k_x$ and $w = w_y$ by Lemma 2 (line 21).

V. EXPERIMENT

In this section, we use three real-world streaming data to evaluate the efficiency of our framework.

Datasets. (1) Gowalla dataset. Gowalla[1] [27] is a real location-based social network service. In Gowalla dataset, the friendship network consists of 196,591 nodes and 950,327 edges, and a total of 6,442,890 check-in records of these users are collected over the period of February 2009 - October 2010. (2) YearPredictionMSD (YearMSD) dataset. YearMSD[2] [28] is a subset of the Million Song Dataset, which is a freely-available collection of audio features and metadata for a million contemporary popular music tracks. In YearMSD dataset, there are total 515,345 instances with 90 dimensional attributes. (3) GeoLife Dataset. GeoLife[3] [29] is a location-based social-networking service, which enables users to share life experiences and build connections among each other using human location history. This dataset collects 17,621 GPS trajectories of 182 users in a period of over four years (from April 2007 to October 2012), including a total of 23,667,828 GPS points.

Baselines. We compare our proposed PSOD framework with alternative methods LEAP [12] and SOP [14]. SOP is the state-of-the-art distance-based outlier detection solution that supports multiple query requests in steaming data. LEAP is the state-of-the-art single query strategy for distance-based outlier detection over data streams. Multiple queries are supported by applying LEAP independently to process each query in the query group.

Evaluation metrics. We measure two metrics common for data streams, namely the average processing time (CPU time) per one thousand data points and the peak memory consumption. The CPU time metric is the total amount of running time utilized to process the queries on one thousand data points. The peak consumed memory metric indicates the memory required to store the information for each active object and the outliers of all queries over data streams. We report the average value over all queries processed in the given experiments. All experiments are conducted using the count-based window.

Experimental Setting. We conduct all experiments on a computer with 4.0GHz i7-6700k processor and 8GB of memory, running Windows 7 operating system. All algorithms are implemented by java.

TABLE 6. Sizes of parameter spaces used in experiment.

Size of parameter space	$\|R\| \times \|K\| \times \|W\| \times \|S\|$
A	$6 \times 6 \times 6 \times 6$
B	$8 \times 8 \times 8 \times 8$
C	$10 \times 10 \times 10 \times 10$
D	$12 \times 12 \times 12 \times 12$
E	$14 \times 14 \times 14 \times 14$

We evaluate the scalability of all methods with an increasing parameter space. Table 6 shows the five sizes of parameter spaces used in our evaluation. $|X|$ denotes the cardinality of set X.

[1]http://snap.stanford.edu/data/loc-gowalla.html
[2]https://archive.ics.uci.edu/ml/datasets/yearpredictionmsd
[3]https://www.microsoft.com/en-us/download/details.aspx?id=52367

A. PERFORMANCE W.R.T. SIZE OF PARAMETER SPACE

In this experiment, we evaluate the performance of our PSOD framework compared with SOP and LEAP under different sizes of parameter space (from size A to E). Specifically, we randomly select the corresponding number of r from 50 to 2K for Gowalla and GeoLife (4K to 6K for YearMSD), k from 10 to 500, w from 10K to 100K, and s from 100 to 10K for each size of parameter space, respectively.

The results on Gowalla, YearMSD, and GeoLife are shown in Figures 4, 5 and 6, respectively. From Figures 4(a), 5(a) and 6(a), our PSOD framework is consistently superior to SOP and LEAP in term of CPU time in all tested cases on three datasets. More specifically, PSOD significantly outperforms SOP and LEAP up to two orders and four orders of magnitude in CPU time, respectively. This is because LEAP needs to repeatedly detect the statuses of data points for each query from scratch when any parameter is varied. As the size of parameter space increases, the number of queries increases. When the size of parameter space is larger

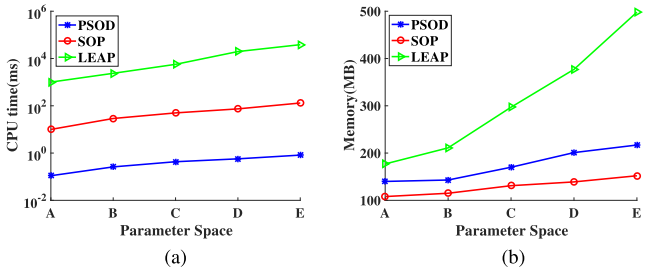


FIGURE 4. Varying size of parameter space on Gowalla. (a) CPU (log scale). (b) Memory.

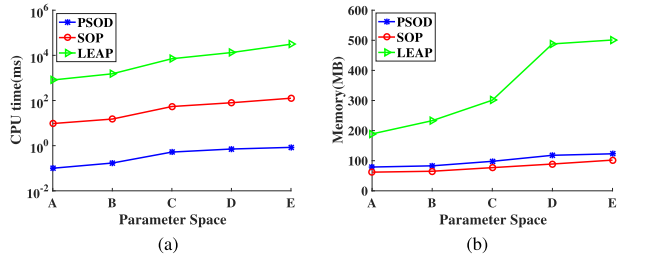


FIGURE 5. Varying size of parameter space on YearMSD. (a) CPU (log scale). (b) Memory.

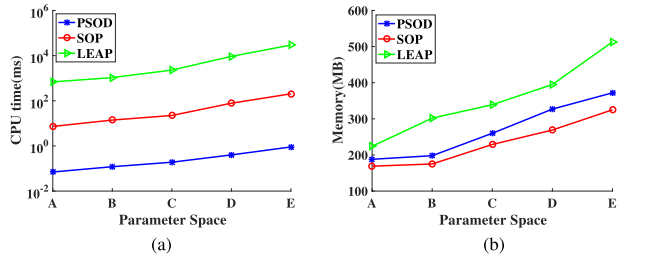


FIGURE 6. Varying size of parameter space on GeoLife. (a) CPU (log scale). (b) Memory.

than C, the number of query requests in the parameter space reaches more than ten thousand, thus the utilized CPU time of LEAP increases quickly. Although SOP collects minimum information to prove the outlier status of each point with respect to all queries, and maps the multiple outlier queries to a single skyband query problem, its shared-execution strategy only supports varying one parameter each time. Therefore, SOP equivalently needs to process multiple skyband queries with respect to different query groups for outlier detection in parameter space. When the size of parameter space increases, the number of skyband queries also increases, thus the CPU time of SOP gradually increases. In our PSOD framework, we use a flexible neighbor table to store the neighbor information for each data point, which can be reused for outlier detection queries with different pattern parameters and window parameters. Namely, the already acquired neighbor information can be reused across entire parameter space. In addition, we propose a series of pruning rules to eliminate the redundant query requests. Although the size of parameter space increases up to ten thousand, lots of query requests may be eliminated based on the statuses of data points (e.g., *s-inlier*). More importantly, PSOD successfully transforms query group in four-dimensional parameter space into a query group in three-dimensional space by sharing slide, which significantly reduces the number of queries and maximumly reuses already acquired resources, resulting in significant higher CPU efficiency compared to SOP and LEAP.

As shown in Figures 4(b), 5(b) and 6(b), SOP and PSOD significantly outperform LEAP in term of memory consumption. This is because LEAP does not consider the sharing opportunities across multiple queries, thus needs to maintain the neighbor information of each data point for each query independently. Therefore, the utilized memory resource of LEAP increases dramatically as the size of parameter space grows. SOP algorithm consumes the least amount of memory. This is because SOP only collects minimum neighbor information of each point to verify the status for each skyband query, which avoids to store the redundant neighbor information. Compared to SOP, PSOD stores detailed necessary neighbor information in the designed neighbor table of each data point for supporting different parameter settings, which benefits the outlier detection in parameter space. Hence our PSOD uses a litter more memory than SOP. In particular, our PSOD only uses 21% more memory on three datasets, on average. However, this extra memory leads to huge gains in CPU processing resources (at least 90 folds faster than SOP).

B. PERFORMANCE W.R.T. THE SCALE OF PARAMETERS

In this section, we study the performance of our PSOD with respect to the scale of parameters in parameter space on three datasets. We fix the size of parameter space to C, i.e., $|R| \times |K| \times |W| \times |S| = 10 \times 10 \times 10 \times 10$. Except for the specified parameter, other parameters vary in the same range with last experiment by default.

1) VARYING THE SCALE OF PARAMETER r

First, we evaluate the impact of the scale of distance threshold r in parameter space on our framework. We vary r from 50 to 200 for case C_1, 50 to 1K for case C_2, and 50 to 2K for case C_3 on Gowalla and GeoLife data (4K to 4.5K for case C_1, 4K to 5K for case C_2, and 4K to 6K for case C_3 on YearMSD data).

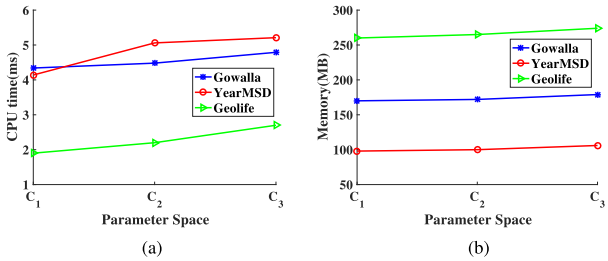


FIGURE 7. Varying the scale of distance parameter r. (a) CPU. (b) Memory.

Figure 7 shows the results of three cases on the three datasets. As shown in Figure 7(a), the CPU time utilized by our PSOD slightly increases with the scale of parameter r in the parameter space. This is expected. In parameter space, if we only increase the scale of parameter r, for the majority inliers in C_1, same computation time would be used in C_2 and C_3, because our PSOD also only acquires the minimum neighbor information to verify the *s-inlier* or *u-inlier* status for each point. For the outliers in C_1, more neighbors may be explored in the larger distance range until to determine their status. However, the general ratio of outliers in the real scenario is very small. Therefore, the impact of the scale of parameter r on the CPU time is small.

From Figure 7(a), PSOD nearly keeps stable in term of memory consumption. As explained above, the inliers nearly store the same neighbor information in their neighbor table in the three cases. Only very few outliers in the case C_1 need more memory to record their newly collected neighbors in the cases C_2 and C_3.

2) VARYING THE SCALE OF PARAMETER k

In this experiment, we vary the scale of neighbor parameter k in parameter space to analyze the performance of our PSOD. Specifically, we randomly select ten values of k from 10 to 50 for case C_1, 10 to 200 for case C_2, and 10 to 500 for case C_3 on three datasets.

As shown in Figure 8, the CPU time of our PSOD linearly increases as the scale of parameter k in parameter space increases. This is because PSOD needs to explore more neighbors for each data point to prove their outlier statuses as the requirements of neighbor count k increase. However, our PSOD framework uses a neighbor table that records all explored neighbors in different distance intervals and different minimum slides, thus it never repeatedly verify the same potential neighbor points for each data point. Namely, PSOD at most performs a complete range query for determining the status of a data point for all distance thresholds in

parameter space. Hence the utilized CPU time for our PSOD would not increase sharply with respective to the scale of parameter k.

As expected, the memory consumed by our PSOD also increase slightly as shown in Figure 8. The reason is that PSOD also needs more memory to store more explored neighbor information to meet the requirement of neighbor count k for each query in the cases C_2 and C_3 compared to case C_1.

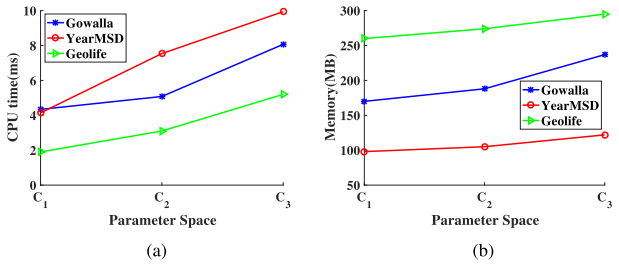


FIGURE 8. Varying the scale of neighbor parameter k. (a) CPU. (b) Memory.

3) VARYING THE SCALE OF PARAMETER W

Next, we explore the impact of window-specific parameters on the performance of our proposed PSOD framework. In particular, window size w in parameter space is varied from 10K to 20K for case C_1, 10K to 50K for case C_2, and 10K to 100K for case C_3 on three datasets. Figure 9 depicts the performance of PSOD framework in terms of CPU time and memory consumption with respect to the scale of window size w in parameter space.

We observe that PSOD keeps linear increasing in term of CPU cost as the scale of window size w increases. This is because as the window size w increases, the number of data points contained in the window increases for each query. That is, the number of points that need to be processed increases for each query. In addition, the search space that PSOD needs to verify the potential neighbors for each data point also increases in a larger scale sliding widow. However, because of the minimum neighbor searching and designed zone-stored neighbor table, PSOD will terminate early neighbor searching once the *s-inlier* or *u-inlier* status of data point is determined, which significantly saves lots of CPU cost.

Since the number of data points greatly increases in a larger scale sliding window, our framework needs to use more memory to store them and their neighbor tables. Hence the memory consumed by PSOD increases as the scale of window size w grows in parameter space as show in Figure 9(b).

4) VARYING THE SCALE OF PARAMETER s

Finally, we test the performance of our proposed PSOD framework with respective to the scale of slide size s in parameter space. We vary slide size s from 100 to 1K for case C_1, 100 to 5K for case C_2, and 100 to 10K for case C_3 on three datasets.

Figure 10 shows the results of CPU time and memory consumed by PSOD with respect to the scale of slide size s in

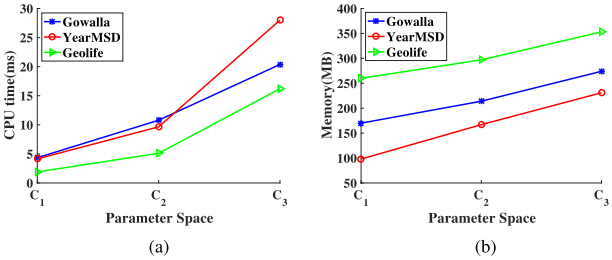


FIGURE 9. Varying the scale of window size parameter w. (a) CPU. (b) Memory.

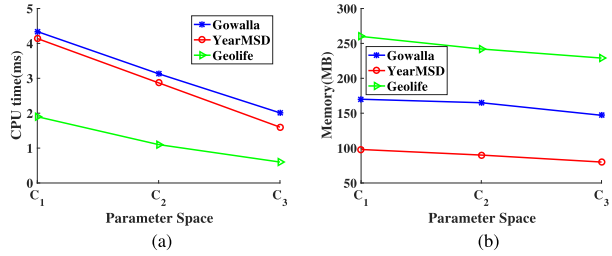


FIGURE 10. Varying the scale of slide size parameter s. (a) CPU. (b) Memory.

parameter space. As we can see, both CPU time and memory consumption of our PSOD decrease as the scale of slide sizes in parameter space increase. This is because we treat the neighbor points of each point in the same slide as a part of $\mathcal{N}_{after}$, as the slide size increases, more data points can be quickly proved as *s-inlier*s in the slides where they are. Hence those queries for these *s-inlier*s would be eliminated in the subsequent windows. In addition, when the window sizes are fixed in parameter space, larger slide size reduces the number of slides on data streams, which also reduces the total number of queries in the streaming data.

For memory consumption, we store the neighbor information for each data point in different minimum slides, thus the total number of active slides for each point decreases as the scale of slide size increases. Accordingly, the memory used by our PSOD is reduced.

In summary, our PSOD shows excellent scalability with respective to the size of parameter space as well as the scale of pattern and window parameters in parameter space.

VI. CONCLUSION

In this work, we present a solution, called PSOD, for supporting multiple distance-based outlier detection queries in parameter space with both pattern and window parameters over data streams. PSOD uses an ingenious neighbor table that records the neighbors for each point in different distance intervals and different slides, which enables us to maximumly reuse the already acquired neighbor information across entire parameter space. Moreover, PSOD successfully transforms the query group in four-dimensional parameter space into a periodic query group in three-dimensional parameter space.

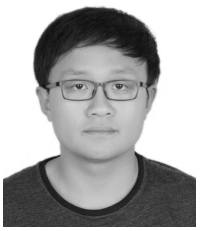
Extensive experiments on three real datasets demonstrate that our PSOD outperforms the state-of-the-art method by more than 100 folds in CPU time, while only using 20% more memory space.

ACKNOWLEDGMENTS

The authors are very grateful to the anonymous reviewers for their insightful comments and suggestions.

REFERENCES

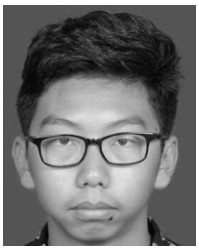
[1] M. Gupta, J. Gao, C. C. Aggarwal, and J. Han, "Outlier detection for temporal data: A survey," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 9, pp. 2250–2267, Sep. 2014.
[2] C. C. Aggarwal, "Outlier analysis," in *Data Mining*. New York, NY, USA: Springer, 2015, pp. 237–263.
[3] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Comput. Surv.*, vol. 41, no. 3, p. 15, 2009.
[4] Z. Cai, Z. He, X. Guan, and Y. Li, "Collective data-sanitization for preventing sensitive information inference attacks in social networks," *IEEE Trans. Depend. Sec. Comput.*, vol. 15, no. 4, pp. 577–590, Jul./Aug. 2018.
[5] X. Zheng, Z. Cai, J. Yu, C. Wang, and Y. Li, "Follow but no track: Privacy preserved profile publishing in cyber-physical social systems," *IEEE Internet Things J.*, vol. 4, no. 6, pp. 1868–1878, Dec. 2017.
[6] Y. Wang, Z. Cai, X. Tong, Y. Gao, and G. Yin, "Truthful incentive mechanism with location privacy-preserving for mobile crowdsourcing systems," *Comput. Netw.*, vol. 135, pp. 32–43, Apr. 2018.
[7] E. M. Knorr and R. T. Ng, "Algorithms for mining distance-based outliers in large datasets," in *Proc. Int. Conf. Very Large Data Bases*, 1998, pp. 392–403.
[8] E. M. Knorr, R. T. Ng, and V. Tucakov, "Distance-based outliers: Algorithms and applications," *Very Large Data Bases J.*, vol. 8, nos. 3–4, pp. 237–253, 2000.
[9] D. M. Hawkins, *Identification of Outliers*. London, U.K.: Chapman & Hall, 1980.
[10] K. Mouratidis, K. Mouratidis, D. Papadias, and D. Papadias, "Continuous nearest neighbor queries over sliding windows," *IEEE Trans. Knowl. Data Eng.*, vol. 19, no. 6, pp. 789–803, Jun. 2007.
[11] M. Kontaki, A. Gounaris, A. N. Papadopoulos, K. Tsichlas, and Y. Manolopoulos, "Continuous monitoring of distance-based outliers over data streams," in *Proc. IEEE Int. Conf. Data Eng.*, Apr. 2011, pp. 135–146.
[12] L. Cao, D. Yang, Q. Wang, Y. Yu, J. Wang, and E. A. Rundensteiner, "Scalable distance-based outlier detection over high-volume data streams," in *Proc. IEEE Int. Conf. Data Eng.*, Mar./Apr. 2014, pp. 76–87.
[13] F. Angiulli and F. Fassetti, "Distance-based outlier queries in data streams: The novel task and algorithms," *Data Mining Knowl. Discovery*, vol. 20, no. 2, pp. 290–324, 2010.
[14] L. Cao, J. Wang, and E. A. Rundensteiner, "Sharing-aware outlier analytics over high-volume data streams," in *Proc. Int. Conf. Manage. Data*, 2016, pp. 527–540.
[15] S. Ramaswamy, R. Rastogi, and K. Shim, "Efficient algorithms for mining outliers from large data sets," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, 2000, pp. 427–438.
[16] S. D. Bay and M. Schwabacher, "Mining distance-based outliers in near linear time with randomization and a simple pruning rule," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2003, pp. 29–38.
[17] F. Angiulli and C. Pizzuti, "Fast outlier detection in high dimensional spaces," in *Proc. Eur. Conf. Princ. Data Mining Knowl. Discovery*, 2002, pp. 15–27.
[18] S. Subramaniam, T. Palpanas, D. Papadopoulos, V. Kalogeraki, and D. Gunopulos, "Online outlier detection in sensor data using non-parametric models," in *Proc. Int. Conf. Very Large Data Bases*, Seoul, South Korea, Sep. 2006, pp. 187–198.
[19] F. Angiulli and F. Fassetti, "Detecting distance-based outliers in streams of data," in *Proc. 16th ACM Conf. Conf. Inf. Knowl. Manage.*, 2007, pp. 811–820.
[20] D. Yang, E. A. Rundensteiner, and M. O. Ward, "Neighbor-based pattern detection for windows over streaming data," in *Proc. Int. Conf. Extending Database Technol., Adv. Database Technol.*, 2009, pp. 529–540.
[21] Y. Bu, L. Chen, A. W. C. Fu, and D. Liu, "Efficient anomaly monitoring over moving object trajectory streams," in *Proc. ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2009, pp. 159–168.
[22] Y. Yu, L. Cao, E. A. Rundensteiner, and Q. Wang, "Detecting moving object outliers in massive-scale trajectory streams," in *Proc. 20th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2014, pp. 422–431.
[23] Y. Yu, L. Cao, E. A. Rundensteiner, and Q. Wang, "Outlier detection over massive-scale trajectory streams," *ACM Trans. Database Syst.*, vol. 42, no. 2, p. 10, 2017.
[24] D. Yang, E. A. Rundensteiner, and M. O. Ward, "A shared execution strategy for multiple pattern mining requests over streaming data," *Proc. VLDB Endowment*, vol. 2, no. 1, pp. 874–885, 2009.
[25] D. Yang, E. A. Rundensteiner, and M. O. Ward, "Shared execution strategy for neighbor-based pattern mining requests over streaming windows," *ACM Trans. Database Syst.*, vol. 37, no. 1, pp. 1–44, Feb. 2012.
[26] A. Arasu, S. Babu, and J. Widom, "The CQL continuous query language: Semantic foundations and query execution," *VLDB J.*, vol. 15, no. 2, pp. 121–142, 2006.
[27] E. Cho, S. A. Myers, and J. Leskovec, "Friendship and mobility: User movement in location-based social networks," in *Proc. 17th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2011, pp. 1082–1090.
[28] T. Bertin-Mahieux, D. P. Ellis, B. Whitman, and P. Lamere, "The million song dataset," in *Proc. 12th Int. Conf. Music Inf. Retr. (ISMIR)*, vol. 2, no. 9, 2011, p. 10.
[29] Y. Zheng, X. Xie, and W.-Y. Ma, "GeoLife: A collaborative social networking service among user, location and trajectory," *IEEE Data Eng. Bull.*, vol. 33, no. 2, pp. 32–39, Jun. 2010.



GUANZHE ZHAO was born in Jining, China, in 1993. He received the B.S. degree in computer science from Yantai University, China, in 2016, where he is currently pursuing the M.S. degree in computer science, under the supervision of Prof. Y. Yu. His main research interests include outlier detection and data stream mining.

YANWEI YU (M'16) was born in Heze, China, in 1986. He received the B.S. degree from Liaocheng University, China, in 2008, and the Ph.D. degree from the University of Science and Technology Beijing, China, in 2014. He was a Visiting Scholar with the Department of Computer Science, Worcester Polytechnic Institute, from 2012 to 2013. He is currently an Associate Professor with the School of Computer and Control Engineering, Yantai University. His research interests include data mining, machine learning, and database systems.

PENG SONG (M'16) was born in Yantai, China, in 1983. He received the B.S. degree from the Shandong University of Science and Technology, China, in 2006, and the M.S. and Ph.D. degrees from Southeast University, China, in 2009 and 2014, respectively. From 2009 to 2011, he was a Software Engineer at Motorola. He is currently an Associate Professor with the School of Computer and Control Engineering, Yantai University. His research interests include affective computing, speech signal processing, and machine learning.



GENG ZHAO was born in Laiwu, China, in 1997. He is currently pursuing the B.S. degree with the University of Shanghai for Science and Technology, China. He is also a Research Assistant with the College of Communication and Art Design, University of Shanghai for Science and Technology. His research interests include data mining and information systems.

ZHE JI was born in Changzhou, China, in 1989. He received the B.S. degree from the Nanjing University of Posts and Telecommunications, China, in 2011, and the M.S. degree from the Worcester Polytechnic Institute, USA, in 2013. He is currently the Director of the Wireless Business Division, China Mobile Group Jiangsu Company Ltd., Changzhou Branch. His research interests include big data and business intelligence.