IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received November 30, 2019, accepted December 15, 2019, date of publication December 24, 2019, date of current version January 3, 2020.

Digital Object Identifier 10.1109/ACCESS.2019.2962066

PTAOD: A Novel Framework for Supporting Approximate Outlier Detection Over Streaming Data for Edge Computing

RUI ZHU[1], TIANTIAN YU[1], ZHIYUAN TAN[2], WEI DU[3], LIANG ZHAO[1], JIAJIA LI[1], AND XIUFENG XIA[1]
[1]College of Computer Science, Shenyang Aerospace University, Shenyang 110136, China
[2]School of Computing, Edinburgh Napier University, Edinburgh EH11 4DY, U.K.
[3]Chinese People's Liberation Army Force, Benxi 32673, China

Corresponding author: Liang Zhao (lzhao@sau.edu.cn)

This work was supported in part by the National Natural Science Foundation for Young Scientists of China under Grant 61702344, Grant 61701322, and Grant 61502317, in part by the Key Projects of Liaoning Natural Science Foundation under Grant 20170540700, in part by the Natural Science Foundation of Liaoning Province under Grant 2019-MS-253, and in part by the Liaoning Provincial Department of Education Science Foundation under Grant L201722 and Grant JYT19052.

ABSTRACT Outlier detection over sliding window is a fundamental problem in the domain of streaming data management, which has has been studied over 10 years. The key to supporting outlier detection is to construct a neighbour list for each object, which is used for predicting which objects may become outliers or are impossible to become outliers. However, existing work ignores the fact that, outliers amount is usually small, in which it is unnecessary to construct neighbour-list for all objects when they arrive in the window. It causes both high space and computational cost, which turns the solution infeasible for working under edge computation environment. In this paper, we propose a novel framework named PTAOD (Probabilistic Threshold-based Approximate Outlier Detection). Firstly, we propose an algorithm for evaluating the probability of a newly arrived object becoming an outlier before it expires from the window, using evaluating result for avoiding unnecessary candidate maintenance. In addition, we introduce a novel index namely ZHB-Tree (Z-order-based Hash B-Tree) to maintain streaming data. Last of all, we propose a novel algorithm to maintain candidate outliers. Theoretical analysis and extensive experimental results demonstrate the effectiveness of the proposed algorithms.

INDEX TERMS Outlier detection, streaming data, probability guarantee, index.

I. INTRODUCTION

Continuous outlier detection over sliding window [1], [2] is a fundamental problem, which has been deeply studied over ten years. It has various applications, ranging from fraud detection, geological disasters warming to network traffic analysis and health data monitoring. According to the description of Hawkins [3], an object is described as an outlier if it behaves not in accordance with the expectation, which should be paid more attention.

Various of definitions [4] could be used for evaluating whether a given object is an outlier. Among all of them, distance-based outlier is one of the most widely used definition. Specially, let k and r be two parameters. An object o is labelled as an outlier if there are less than k objects within the range of the maximum r from o.

In the edge computation environment [5], streaming data only can stay in memory for a short time due to the limitation of memory space. In addition, it is impossible to use much space cost to support outlier detection. Therefore, in this paper, we use sliding window model to depict the lifetime of streaming data, and study approximate outlier detection over streaming data. Without loss of generality, this window can be either time- or count-based. In either case, the query window has a fixed window size and a fixed slide (either a time interval or an object count). Formally, in a *count-based window*, it returns outliers in the query window that contains n objects when the window slides; in a *time-based window*, it returns outliers in the last n time units whenever the window slides [6]. In other words, a continuous outlier

The associate editor coordinating the review of this manuscript and approving it for publication was Honghao Gao.

detection could be expressed by the tuple $\langle n, s, k, r\rangle$, where parameter s represents the number of objects that arrive whenever the window slides or the duration in terms of time units between two adjacent windows sliding [6]–[8]. The continuous outlier detection returns objects satisfying that the number of objects within a distance r from them are smaller than another parameter k.

Due to the importance of outlier detection over streaming data, many efforts have been proposed [9]–[13]. Their key idea is using an index $\mathcal{I}$ to maintain streaming data in the window. Based on $\mathcal{I}$, when an object o arrives in the window, they submit a range query on $\mathcal{I}$, find objects within the range of r, and check whether it is an outlier via counting the number of objects contained in the query region. If the answer is yes, these algorithms construct an inverted-list for it, and use k last arrived neighbours as the element of inverted-list. After other objects arrive in the window, these algorithms update inverted-list for o. In particularly, if elements in the inverted-list are all arrived later than o, these algorithms regard o as a safe object. The benefit is that only part of objects in the window should be monitored, i.e., having chance to become outliers. However, the algorithm has to spend high cost both in space and computational in maintaining neighbours for each object.

Besides exact algorithms, Angiulli et al [11].proposed an approximate algorithm named Approx-storm. It makes a tradeoff between space requirement and detection accuracy. In this way, the algorithm could effectively reduce the memory cost, and return approximate results. However, this algorithm still has to spend relatively high cost in processing newly arrived objects. Note that, in most applications, only a small number of objects have chance to become outliers, it is unnecessary to spend relatively high cost in processing each newly arrived cost especially under edge computing environment.

In a nutshell, both exact and approximate algorithms have to spend high cost in evaluating which objects have chance to become outliers. All of the above algorithms are not suitable for the situations where the capability of calculation and space have limitation. They cannot effectively work under edge computation environment. [14], [15].

Therefore, in this paper, we propose an approximate outlier detection framework named PTAOD(short for probabilistic threshold based approximate outlier detection). The key idea behind it is, given a newly arrived object o and a threshold ρ, we evaluate the probability of o becoming an outlier. If the probability is lower than ρ, we regard it as a ρ-safe object, and do not monitor it at all. Otherwise, we should construct an inverted-list for it. Compared with other algorithms, we only construct inverted-lists for a small number of objects. Obviously, both the space and computational cost could be reduced a lot.

Challenges: However, it meets the following challenges. For the first one, how to find a suitable threshold ρ. Note that, if the threshold ρ is low, most objects could be pruned directly, leading that fewer outlier could be found. Otherwise, we still have to construct inverted-list for many objects, and the space cost could not be effectively reduced. For the second one, it is also difficult to evaluate which objects have chance to become outlier when they are arriving in the window.

TABLE 1. The summary of notations.

Notation	definition
W	the sliding window
o_{in}	newly arrived object
o_{exp}	expired object
$z(o)$	the Z-address of o
$NL(o)$	the neighbour list of o
$\lvert e\rvert$	the number of objects contained in the node e

To deal with the challenges above, the contributions of this paper are as follows.

- We define a novel query called ρ-approximate continuous outlier detection over sliding window. ρ is a threshold specified by users. It is used for bounding the probability of identifying an outlier as inlier.
- We propose a novel index named ZHB-Tree. It is a two level index. The first level of ZHB-Tree is a B-Tree, used for maintaining the ID of these boxes. The second level is a group of boxes. One advantage is we need not to update the structure of ZHB-Tree in most cases when objects flow into or expire from the window. In addition, it is insensitive to the distribution of streaming data. Last of all, we maintain the summary information of objects' distribution in each box, leading that we can use, as small as possible, cost for evaluating whether an object having high probability to become an outlier.
- We propose a novel outlier candidate maintain algorithm. It uses M-Tree for maintaining candidates. When an object flows into or expires from the window, we can efficiently find which candidates are impacted by them, and update candidate set accordingly. In addition, we could remove some objects from candidate set if they cannot become outliers in a high probability.

The rest of the paper is organized as follows. Section II reviews the previous work and basic concepts related to ours. Then, Section III discusses the framework overview. Section IV explains the index and the candidate maintenance algorithm. Section V reports the results of our experimental evaluation. Finally, Section VI concludes the paper with a summary of our findings.

II. PRELIMINARIES AND DEFINITIONS

In this section, we first review the algorithms about the problem of continuous outlier detection over streaming data. Thereafter, we define the approximate outlier detection over streaming data. Table1 summarizes the mathematical notations used in the paper.

A. RELATED WORK

Exact Algorithms: With the development of information science and technology [16]–[23], outlier detection [24] over sliding window has been deeply studied over ten years. Many scholars have studied the distance-based continuous outlier detection algorithms. This definition is first introduced by Knorr and Ng. Given the parameters k and r, according to the definition of outliers, an object is a distance-based outlier if it has less than k neighbours.

Yang at al. [13] propose a distance-based algorithm that maintains all neighbors information for each object. Obviously, both the space cost and computing cost are all high. Therefore, they improve it via using an important feature of sliding window, that is, the ''predictability'' of the expiration of each existing object.

Yamanishi at al. [12] introduce the distance-based outlier detection algorithm named SmartSifter. It uses the *online discount learning* algorithm to incrementally learn the probabilistic mixture model. Considering the factor of drift, outliers are calculated according to the probabilistic fitting value of the learning mixture model. The online discount algorithm can also be used to maintain the frequency of a set of attribute values.

STORM [11] is another efficiently algorithm. Its key idea is maintaining objects in the window via an index. In this way, when a newly arrived object o flows into the window, STORM submits a range query to find neighbours for o. In addition, STORM updates neighbour list for each object. In particularly, if an object has more than k succeeding neighbors, we call it as a safe object. From then on, STORM needs not to maintain neighbour list for it.

M.Kontaki at al. [10] propose a micro-clustering based algorithm named MCOD. Its advantage is effectively reducing range query cost via using micro clustering. However, it still cannot avoid the costly range query when newly arrived objects flow into the window. In addition, range query cost is increasing with data dimension d.

In [25], an algorithm named LOF (short for Local Outlier Factor) is proposed. Its key idea is measuring the local deviation of input data points relative to their neighbors. Cao at al. [9]propose a novel algorithm named Thresh-LEAP via using another parameter s. Compared with other methods, it first uses the *lightweight probing operation* to gather minimal yet sufficient evidence for outlier detection. Secondly, it uses the temporal relationships among stream data points to prioritize the processing order among them during the probing process. Based on these two principles, they design an outlier detection strategy which is proven to be optimal in CPU cost.

Approximate Algorithms: Angiulli et al [11]. introduce an approximate algorithm Approx-storm. The Approx-storm algorithm is derived from Exact-storm algorithm, which makes a tradeoff between space requirement and detection accuracy. The algorithm effectively reduces the memory requirement via two measures, and returns approximate results based on the estimation of statistical guarantee. However, the algorithm still needs to maintain each newly arrived data object. In most application scenarios, object that deviates from the expected behavior is only a small part of the entire data set, and most data objects are inliers. Therefore, when

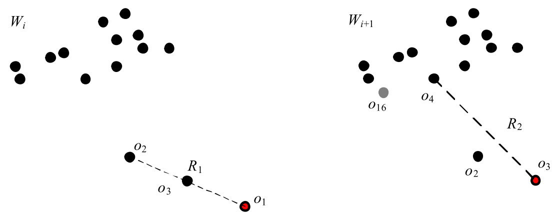


FIGURE 1. Outlier Detection over Sliding Window(k = 2).

the amount of data is large, it may fail to meet the real-time requirements of users.

B. PROBLEM DEFINITION

At the beginning of this section, we first introduce the concept of *neighbour*, *S-Neighbour* and *distance*-based outlier detection. For simplicity, the distance between o_1 and o_2, denoted as $\mathsf{D}(o_1, o_2)$, is computed based on their Euclidean distance.

Definition 1 (Neighbour): Let o_1 and o_2 be two objects in the data set D, r be a threshold. If the distance between o_1 and o_2 is smaller than r, we call o_1 and o_2 are neighbours of each other.

Definition 2 (S-Neighbour): Let o_1 and o_2 be two objects in the window W. If o_1 arrives later than o_2, and $\mathbf{D}(o_1, o_2) \leq r$, we call o_1 as a S-Neighbour of o_2.

Definition 3 (Continuous Distance-Based Outlier Detection): Let W be the sliding window with size n, $q\langle k, r\rangle$ be an outlier query. q monitors the window. When the window W slides, q returns all outliers to the system.

Obviously, if an object o in the window has more than k S-Neighbours, it cannot become an outlier before it expires from the window. Otherwise, if an object o in the window has less than k S-Neighbours, when some of its neighbours expire from the window, it may become an outlier. Thus, we should monitor, and report it as an outlier at the moment its neighbour amount turns to less than k. In order to achieve this goal, an efficient method is maintaining the last arrived k neighbours for o. When one of these objects expires from the window, we report o as an outlier. Take an example in Figure 1. Let the parameter k be 2, and the distance threshold be R_1. When the window slides from W_i to W_{i+1}, o_1 expires from the window. At that moment, the distance between o_3 and its 2-nearest neighbour, i.e., o_4, is larger than R_1. Therefore, o_3 turns to an outlier.

We find that when an object arrives in the window, we have to construct an inverted-list for it. In the worst cases, the total space cost is $\mathcal{O}(nk)$. Obviously, the cost is so high that cannot efficient work under edge computation environment. We also find that, comparing with the whole object set, the number of outliers is small, most objects can not become outliers before they turn to meaningless. This obversion opens the door of studying approximate distance-based outlier detection with probability guarantee. In the following, we formally discuss the problem definition.

Definition 4 (ρ-ACDOD): Let W be the sliding window with size n, $q\langle \rho, k, r\rangle$ be an $\rho-$ approximate outlier detection. q monitors the window. When the window W slides, for each outlier in the window, we report it as an outlier with probability no less than ρ.

ρ-ACDOD allows algorithm identifies outliers as inliers occasionally. Let o be a newly arrived object. If there are many neighbours around o, we regard the region bounding o as a high-density region. If the distribution of streaming data is not changed a lot, objects located at such regions all cannot become outliers in most cases. In other words, we can predict which objects may become outliers via evaluating the local density of these objects. If the density is high, we should use the method discussed before to maintain it. Otherwise, we avoid monitoring it. For simplicity, in the following, if an object cannot become an outlier before it expires from the window, we call it as a *safe object*. If an object cannot become an outlier before it expires from the window with probability no less than ρ, we call it as *$\rho-$safe object*. Otherwise, we call it as an unsafe object. For this kind of objects, we should associate them with neighbour-lists.

III. THE PPVBF FRAMEWORK OVERVIEW

In this section, we propose the framework PTAOD(short for Probability Threshold based Approximate Outlier Detection) for supporting ρ-ACDOD over sidling window. Let D be a set of streaming data in the window. We first use Z-order based method to construct an index named ZHB-Tree for maintaining streaming data in the window. Based on the index, we design the algorithm to summarize the distribution of steaming data. In addition, we evaluate the probability of objects becoming outliers in the future, construct neighbour-lists for unsafe objects.

ZHB-Tree is a two level index. The first level index is a B-Tree. The second level index is a group of non-empty cubes. One advantage of ZHB-Tree is if a cube contains more than two objects, we need not to delete it when an object in it expires from the window. In addition, if a newly arrived object is contained in this cube, we also need not to construct a new cube for it. We will discuss the index details in the later section.

When a newly arrived object o flows into the window, we first access ZHB-Tree I, evaluate whether the probability of o becoming an outlier is high. If the answer is yes, we use the algorithm discussed in [9] to construct neighbour-list for it. After o arrives in the window, if its corresponding cube turns to a density region, we check whether existing objects in this cube are $\rho-$safe object. If so, we delete these corresponding neighbour-lists. Last of all, for each non-safe object, we should check whether it is impacted by o. If the answer is yes, we should update its neighbour-list. We will discuss the searching and maintenance of the impacted objects later.

When an object o' expires from the window, we first remove it from the ZHB-Tree I. Next, we check whether its corresponding cube turns to a non-density region. If the answer is yes, we label it as a non-density cube. Thirdly, for each non-safe object, we should check whether it is impacted by o'. If so, we should update its impacted list.

Algorithm 1 The Framework Overview

```
Input: Window $W$, $\rho$ outlier detection $q$, Newly object
       $o$, expired object $o'$, ZHB-Tree $I$
Output: Outlier Set $O$
1  $W \leftarrow W \cup o$, $W \leftarrow W - o'$;
2  insert($I, o$);
3  delete($I, o'$);
4  $Im(O) \leftarrow$ I-impact($I, o$);
5  for $i$ from 1 to $|IM(O)|$ do
6      update($IM(O)[i]$);
7      if $IM(O)[i] = safe \vee IM(O)[i] = \rho - safe$ then
8          $Im(O) \leftarrow Im(O) - IM(O)[i]$;
9  $Im(O) \leftarrow$ D-impact($I, o'$);
10 for $i$ from 1 to $|IM(O)|$ do
11     update($IM(O)[i]$);
12     if $IM(O)[i] = outlier$ then
13         $O \leftarrow O \cup IM(O)[i]$;
14 return;
```

IV. THE INDEX ZHB-TREE

In this section, we first discuss the ZHB-Tree structure. Next, we discuss how to maintain ZHB-Tree when objects arrive in the window or expire from the window. Lastly, we discuss the algorithm about searching on ZHB-Tree.

A. THE ZHB-TREE DATA STRUCTURE

As is depicted in Figure 2, ZHB-Tree is a Z-curve [26], [27] based two-level index. The first level of ZHB-Tree is a B-Tree, which is used to maintain location relationship among streaming data. The second level of ZHB-Tree is a group of cubes. They are used for maintaining streaming data. Since the first level is constructed based on the second-level index, in the following, we first discuss the second level index.

Second-Level Index Construction: Let G be a grid file that partitions the whole space into a group of cells with size $\frac{r}{\sqrt{d}}$. The benefit is, for every two objects in a cube, the distance between them must be smaller than r. Next, we map all streaming data in the window into G according to their position. For each non-empty cell c, we compute its Z-address, and record how many objects are contained in it. In particularly, if the number of objects in c is higher than $2k$, we label it as a high-density cell. Otherwise, we label it as a low-density cell. For this kind of cells, we use queues to maintain objects in them.

First-Level Index Construction: Based on the computing result, we are going to construct the first-level of ZHB-Tree. In this step, we use a B-Tree to maintain the ID of these cubes. After constructing, we use the final result to support continuous outlier detection.

Example 1 (The Index ZHB-Tree): Take an example in Figure 2. We use a grid file to partition the whole space into a group of cells with size $\frac{r}{\sqrt{d}}$. The benefit is, for every

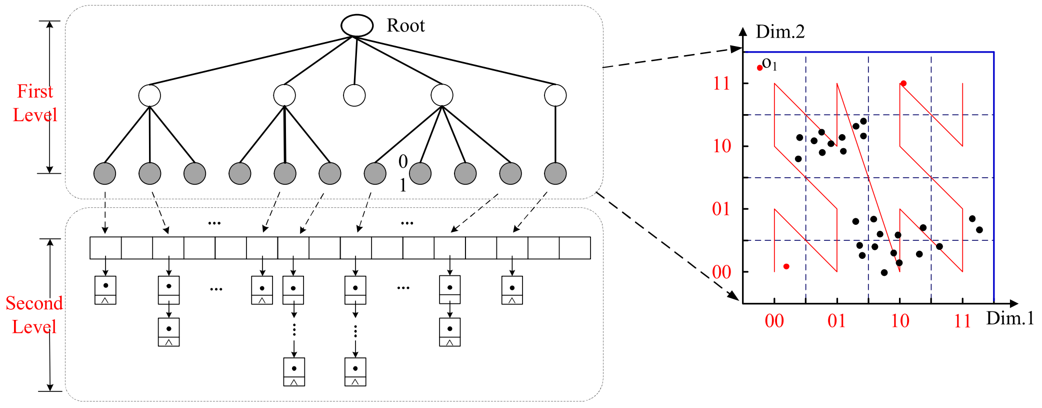


FIGURE 2. The Index ZHB-Tree Overview.

two objects in the same cell, we can make sure that the distance between them is smaller than a threshold r. In order to maintain these cells, we associate them with keys, and use a B-Tree to maintain these cells.

B. THE ZHB-TREE MAINTENANCE

Once the index ZHB-Tree I is constructed, we use it for processing newly arrived objects. Let o_{in} be a newly arrived object. We first compute $z(o_{in})$, i.e., the Z-address of o_{in}. Next, we search on ZHB-Tree I for finding whether existing a non-empty cell c contained in the ZHB-Tree I. If the answer is yes, we insert it into its corresponding cell. In particularly, after insertion, if the number of objects in c is more than the threshold $2k$, we label it as a high-density cell. Otherwise, we create a new cell, and insert it into the first level of ZHB-Tree I.

Let o_{exp} be an expired object. We first find which cube c contains o_{exp}, and then delete it from the ZHB-Tree I. After deleting, if $|c| \leq k$, we label it as a low-density cell. Here, $|c|$ refers to the number of objects contained in c. In particularly, if c turns to an empty cell, we do not delete it at once. By contrast, we use an integer, i.e., $\mathsf{E}(I)$, to record how many empty cells are contained in the ZHB-Tree I. If $\mathsf{E}(I) \geq \frac{n}{B}$, we remove all empty cells from the ZHB-Tree I. One straightforward benefit is we could avoid frequently updating ZHB-Tree.

As is depicted in algorithm 2, we first find the cell containing o_{in}. If the searching result is null, we create a new cell, and update the index ZHB-Tree I accordingly. Otherwise, we insert o_{in} into c directly. When handling expired object o_{exp}, we only focus on whether its corresponding cell turns to empty. If the answer is yes, we update the ZHB-Tree I if the number of empty cells achieve to a threshold.

Discussion: We want to highlight two points. First of all, ZHB-Tree is not sensitive to the distribution of streaming data. Even many objects are located in a small region, we need not to construct many nodes for them. Secondly, ZHB-Tree has a powerful ability to handle newly arrived or expired object. If an object is inserted into a non-empty cell, we do not need to update the structure of ZHB-Tree at all.

Algorithm 2 The ZHB-Tree Maintenance Algorithm

```
Input: ZHB-Tree $I$, newly arrived object $o_{in}$, expired
       object $o_{exp}$
Output: ZHB-Tree $I$
1  Cell $c \leftarrow$ search$(I, o_{in})$;
2  if $c \neq \emptyset$ then
3      $c \leftarrow c \cup o_{in}$;
4      if $|c| = 0$ then
5          E$(I) \leftarrow$ E$(I)$-1;
6  if $c = \emptyset$ then
7      create$(c)$,$c \leftarrow c \cup o_{in}$;
8      insert$(I,c)$, update$(I)$;
9  Cell $c' \leftarrow$ search$(I, o_{exp})$;
10 $c' \leftarrow c' - o$;
11 if $|c| = 0$ then
12     E$(I) \leftarrow$ E$(I)$-1;
13     if E$(I) \geq \frac{n}{B}$ then
14         batch-deletion$(I)$;
15 return;
```

C. SEARCHING ON ZHB-TREE

In this section, we discuss the searching algorithm. Let o_{in} be a newly arrived object. The intuition behind is if o_{in} has many neighbours, it implies the region o_{in} located is a density region. Even the window slides many times, if the distribution of streaming data is not changed a lot, the density of the region that o_{in} locates is still high. In this case, o_{in} cannot become an outlier within a relatively high probability. Before discussing the algorithm, we first explain Theorem 1. For simplicity, $\mathsf{NL}(o)$ refers to the neighbour amount of o.

Theorem 1: Let o be an object in the window. If its neighbour amount is larger than $\rho(k)$, it cannot become an outlier before it expires from the window with probability ρ. Here, m refers to the solution of $\frac{k-mp}{\sqrt{mp(1-p)}} \geq \rho$.

Proof: Let o be an object in the window, $|\mathsf{NL}(o)|$ be m. If both k and m are big enough, we could use normal distribution $N(np, \sqrt{np}(1-p))$ to approximate $\Pr(NL(o) \leq k)$

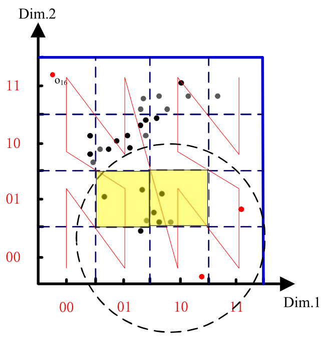


FIGURE 3. Searching On ZHB-Tree.

according to *demovire-laplace theorem*. Since $\Pr(NL(o) \leq k) \approx \Phi(\frac{k-mp}{\sqrt{mp(1-p)}})$, if $\frac{k-mp}{\sqrt{mp(1-p)}} \geq \rho$, $\Pr(NL(o) \leq k) \geq \rho$. ■

Theorem 1 implies if an object's neighbour amount is larger than a threshold m, it cannot become an outlier in most cases. Therefore, when an object flows into the window, we evaluate the probability of o becoming an outlier via counting its neighbour amount. Take an example in Figure 3. Since there are many objects contained in the cell "0110", we need not to check objects contained in this cell.

Specially, let c be the cell containing o. We first compute how many objects are contained in c. If the amount is larger than $\rho(k)$, the algorithm is terminated. Otherwise, we access the neighbour cells of o. Here, we call a cell c' as a neighbour cell of c if the minimal distance between c and c' is 0. In order to access these cells, we first compute their IDs. Next, we search on the ZHB-Tree for finding these cells. Thirdly, we access these cells, find neighbours for o. The searching algorithm is terminated in the following two cases: (i) we can find m neighbours for o; (ii) we cannot find m neighbours for o after searching all neighbour cells of o.

In case (i), we do not construct neighbour-list for o. In case (ii), we should construct neighbour-list for o. Specially, we select the k last arrived neighbours of o as elements of o's neighbour-list. Next, we insert o into the candidate set C. In figure 3, since we can find enough neighbours for o, we can terminate the searching after accessing two cubes.

D. THE CANDIDATE MAINTENANCE ALGORITHM

In this section, we will discuss how to maintain candidate set. As is discussed before, we use a queue to maintain neighbours for each candidate. When an object arrives in the window or expires the window, we check which candidates are impacted by it, and then update their associated neighbour-list.

In this paper, we use M-Tree [28], [29] to maintain candidates. When an object o_{in} arrives in the window, we search on the M-Tree I' for finding which candidates are impacted by o_{in}. Here, an object o is impacted by o_{in} if $\mathsf{D}(o, o_{in})$ is smaller than r. Let IO be a set of objects impacted by o. For each of them, we remove the element located at the top of neighbour-list. In particular, for each object $o \in IO$, if its neighbour-list turns to null, we remove o from the candidate set.

We want to highlight that since we allow neglect few outliers based on problem definition, it opens the door of removing candidate outliers from candidate set in advance. Theorem 2 implies if there are enough neighbours whose arrived moment are near to o, o also cannot become an outlier with high probability. For simplicity, let o be a candidate, $n_1(o)$ and $n_k(o)$ be the first arrived and last arrived neighbour contained in $NL(o)$. $SW_{T(n_1(o))}^{T(n_k(o))}$ refers to a sub-window, which satisfies that the first arrived object is $n_1(o)$, and the last arrived object is $n_k(o)$.

Theorem 2: If $n_k(o)$ arrives later than o, and $T(n_k(o)) - T(n_1(o)) < m$, o also cannot become an outlier before it expires from the window with probability no less than ρ.

Proof: Let R be a circle with radius r and center $p(o)$. We could use normal distribution $N(mp, \sqrt{mp(1-p)})$ to approximate $\Pr(NL(o) \leq k)$ according to *demovire-laplace theorem*, where $p = \frac{k}{m}$ after enough objects in the window expire the window. Here, m refers to the number of objects in the sub-window. Since $\Pr(NL(o) \leq k) \approx \Phi(\frac{k-mp}{\sqrt{mp(1-p)}})$, if $\frac{k-mp}{\sqrt{mp(1-p)}} \geq \rho$, we can find a suitable m via computing $\frac{k-mp}{\sqrt{mp(1-p)}} = \rho$. ■

When an object expires from the window, we first search on the M-Tree I' for finding which objects are contained in the searching region. Let IO' be the search result set. For each $o \in IO'$, if the object locates at the top of $NL(o)$ is o, we remove o from the candidate set C, and then insert it into the outlier set. Note, we could use the similar method to maintain outliers in the outlier set. For the limitation of space, we skip the details.

V. EXPERIMENTAL EVALUATION

In this section, we conduct extensive experiments to demonstrate the efficiency of PTAOD. The experiments are based on three real datasets. In the following, we first explain the settings of our experiments, and then report our findings.

A. EXPERIMENTAL SETTING

Data Set: In total, three real datasets are used in our experiments, including Stock, Tao and HPC. They contain 1048575 1-dimensional records, 575648 3-dimensional records, and 1289534 7-dimensional records respectively.

Experiment Method: In the implementation, we insert all records into a buffer. After insertion, we use two pointers for simulating the window sliding. Specially, when the window slides, s objects flow into the window, and the other s objects expired from the window. We simulates it via moving these two pointers. After processing all objects in the buffer, we compute the average CPU time and memory we consume.

Parameters Setting: In our experiments, we measure the following metrics by varying different parameters of the system, which are *response time,space cost*. Here, *response time* refers to the total time we consume after we process 1KB objects. *Space cost* refers to the memory we consume. In this paper, we consider five parameters, i.e., the window size n, the number s of new objects that slide into the window whenever the window slides. In addition, we should

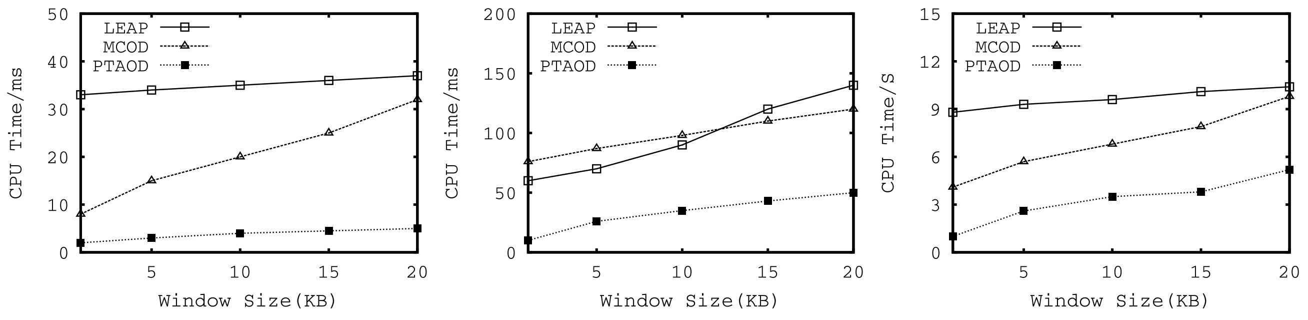


FIGURE 4. Running time comparison under different window size.

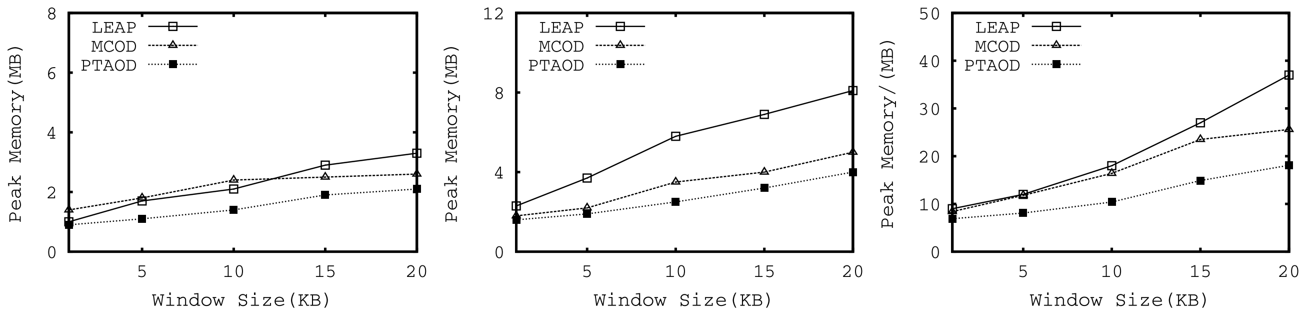


FIGURE 5. Space cost comparison under different window size.

TABLE 2. Parameter settings.

Parameter	value
n	1,5,10,15,20 ($\times KB$)
k	10, 20, **50**, 80, 100
s	1, $\frac{1}{10}$, $\frac{1}{20}$, $\frac{1}{50}$, $\frac{1}{100}$ ($\times n$)
r	0.01%, 0.05% ,**2%**, 5%, 10% ($\times R$)
ρ	0.99, 0.98 ,**0.97**, 0.95, 0.9

evaluate the impact of parameters k and r to the algorithm performance. Last of all, we should evaluate the impact of parameters ρ to the algorithm performance. The parameter settings are listed in Table 2 with the default values bolded. All the algorithms are implemented with C++, and all the experiments are conducted on an CPU i7 with 16GB memory, running Microsoft Windows 7.

B. ALGORITHM PERFORMANCE

In this section, we compare our proposed framework PTAOD with that of MCOD and LEAP. First of all, we evaluate their performance under different window size. Other parameters are default values.

As is depicted in Figure 4, PTAOD performs best of all. The reason behind is, for one thing, we use ZHB-Tree to maintain streaming data in the window. When we insert an object o into the window, we need not to update the structure of ZHB-Tree. By contrast, we only need to find the cell containing o. For another, compared with other methods, we only need to maintain a small number of candidates. In this way, when the window slides, we only need to update fewer candidates' neighbour list.

For the space cost, as is depicted in Figure 5, PTAOD also performs best of all. The reason behind is we only maintain a small number of objects' neighbour list. In addition, when updating candidates' neighbour list, if we find that the probability of an object becoming outlier is small than a threshold, we can remove this candidate from candidate set directly.

As is depicted in Figure 6, PTAOD performs best of all. We also find that with the increasing of s, their performance all turn to better. The reason behind is the larger the s is, the more the objects having the same arrived order. It leads that we need to construct neighbour list for fewer candidates. Thus, when the parameter s is large enough, the performance of these three algorithms are roughly the same. However, it implies our proposed framework is unsensitive to the parameter s.

The space cost of PTAOD is the smallest of all. Similar with the reason discussed before, we only maintain neighbour list for a small number of candidates, leading that the space cost overall could be reduced a lot(See Figure 7).

As is depicted in Figure 8, PTAOD performs best of all. We also find that, the running time of these algorithms are all increasing with the parameter k. Another observation is that, as k increases, the running time of both MCOD and MESI goes up rapidly. This is because when the k becomes large, we should maintain more candidates. Because our proposed PTAOD only maintains a small number of candidates. Therefore, PTAOD is not sensitive to the parameter k, and can efficiently work under different parameter k.

As is depicted in Figure 9, PTAOD also consumes the smallest space cost of all. Similar with the reason discussed before, although the space cost of these three algorithms are all increasing with the parameter k, PTAOD increases slowest of all. Besides the reason discussed before, another reason is we can delete a candidate from candidate set before it turns to a safe object in many cases.

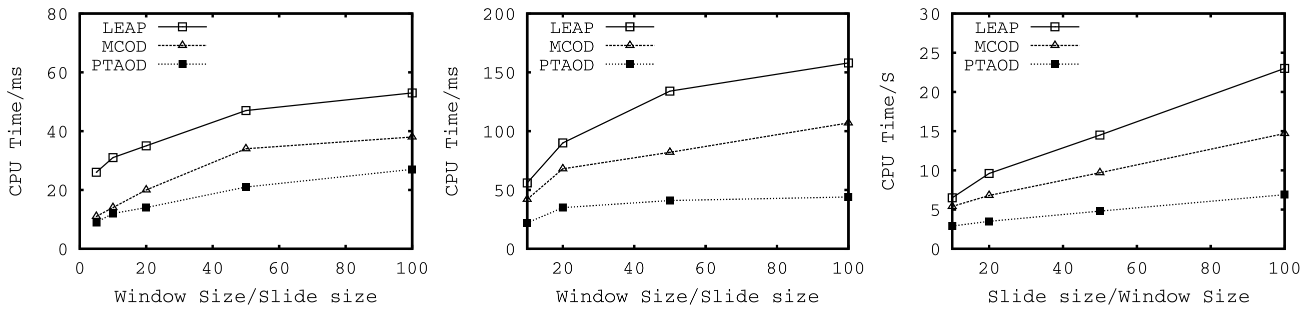


FIGURE 6. Running time comparison under different *s*.

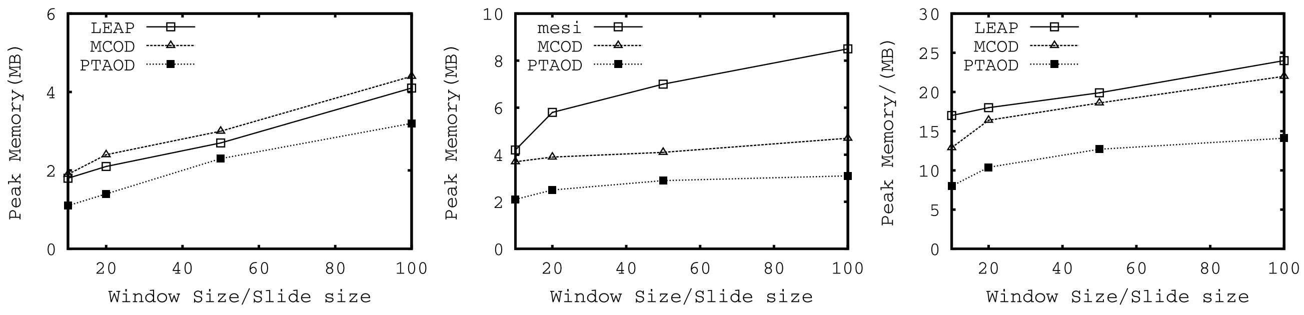


FIGURE 7. Space cost comparison under different *s*.

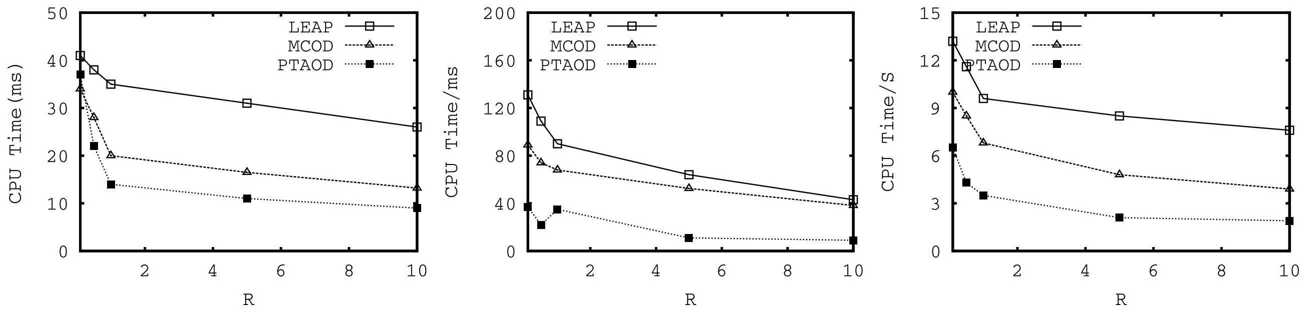


FIGURE 8. Running time comparison under different *R*.

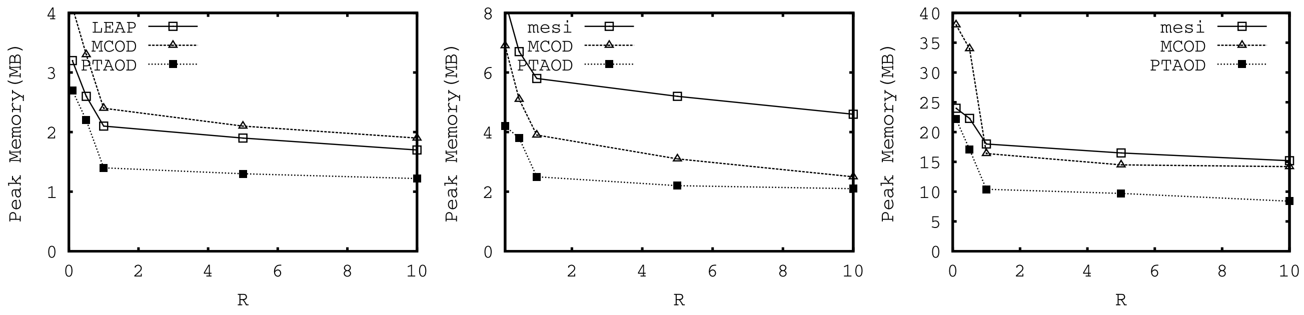


FIGURE 9. Space cost comparison under different *R*.

As is depicted in Figure 10, with the increasing of the parameter r, PTAOD performs best of all. The reason behind is ZHB-Tree could self-adaptively adjust the cell size according to the parameter r. In other words, the higher the parameter r is, the larger the size of the cell size. Let o be a newly arrived object. In many cases, we can terminate the searching via counting the number of objects in the cell that o locates. Since this cost is only $O(1)$, in many cases, we can use $O(1)$ cost for evaluating whether o could not become an outlier with high probability.

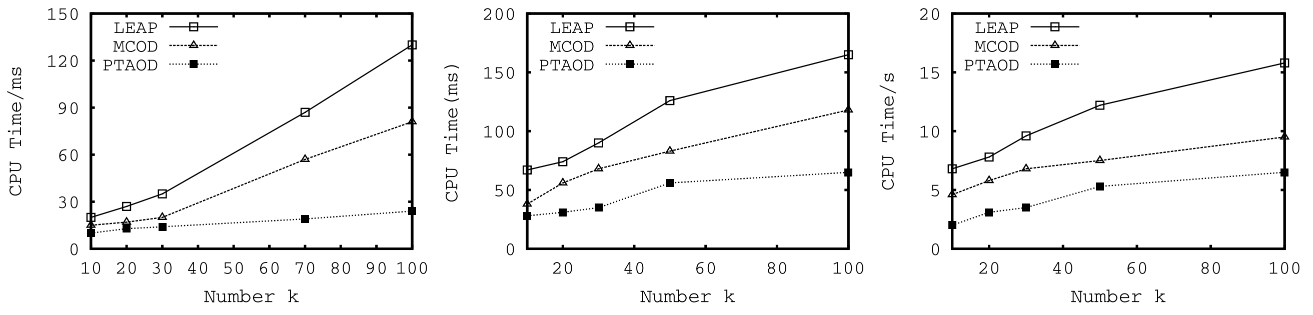


FIGURE 10. Running time comparison under different *k*.

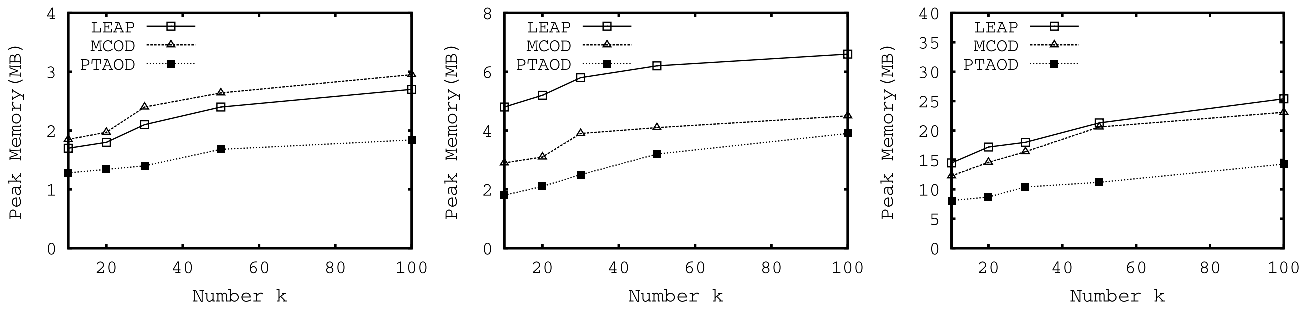


FIGURE 11. Space cost comparison under different *k*.

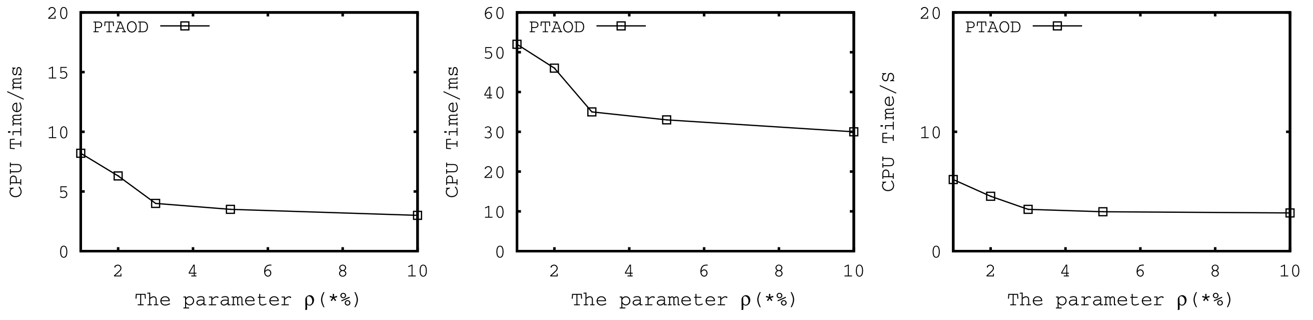


FIGURE 12. Running time comparison under different ρ.

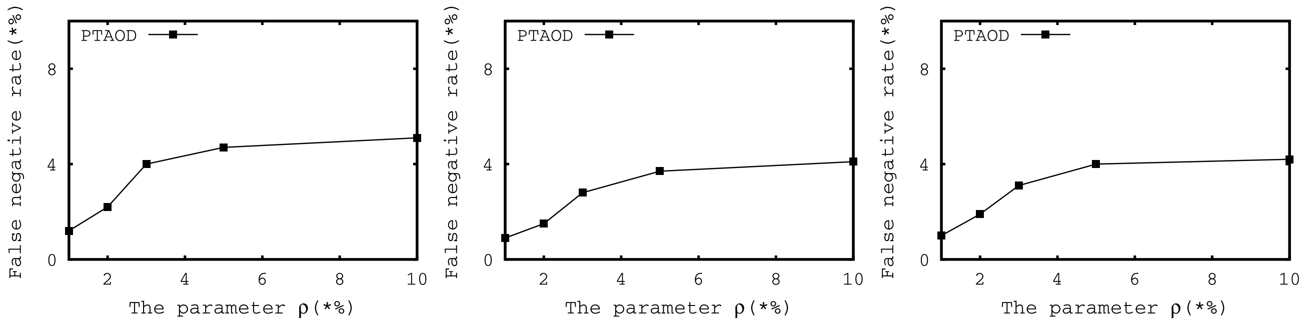


FIGURE 13. False negative rate comparison under different ρ.

For the space cost, as is depicted in Figure 11, PTAOD also performs best of all. Similar with the reason discussed before, we only need to maintain the neighbour-lists for parts of non-safe objects.

As is depicted in Figure 12, with the decreasing of the parameter ρ, the CPU time of PTAOD is reduced. The reason behind it is the smaller the ρ is, the more outliers we could ignore. However, we also find that when ρ is small enough, the running time of PTAOD turns to reduced slowly. The reason is we should spend parts of time to maintain index.

As is depicted in Figure 13, with the decreasing of the parameter ρ, the false negative rate of PTAOD is increased.

The reason behind it is the smaller the ρ is, the more objects we ignored. Some of them may become outliers sometimes. However, we also find that when ρ is small enough, the false negative rate of PTAOD turns to increased slowly. The reason is there are only a small part of objects that may become outliers. In addition, if the distribution of streaming data is not changed a lot, an object located at a high-density region can not become outlier in most cases.

VI. CONCLUSION

In this paper, we propose a novel framework named PTAOD. It uses the fact that if objects density in a region is high, objects in this region cannot become an outlier with a high probability. Firstly, we propose a novel index named ZHB-Tree to maintain streaming data, and summarize the distribution of streaming data in the window. Based on ZHB-Tree, we propose a novel algorithm to support approximate outlier detection. Theoretical analysis and extensive experimental results demonstrate the effectiveness of the proposed algorithms.

REFERENCES

[1] L. Tran, L. Fan, and C. Shahabi, "Distance-based outlier detection in data streams," *Proc. VLDB Endowment*, vol. 9, no. 12, pp. 1089–1100, Aug. 2016.

[2] D. Georgiadis, M. Kontaki, A. Gounaris, A. N. Papadopoulos, K. Tsichlas, and Y. Manolopoulos, "Continuous outlier detection in data streams: An extensible framework and state-of-the-art algorithms," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, New York, NY, USA, Jun. 2013, pp. 1061–1064.

[3] D. M. Hawkins, *Identification of Outliers* (Monographs on Applied Probability and Statistics). Springer, 1980, doi: 10.1007/978-94-015-3994-4.

[4] E. M. Knorr and R. T. Ng, "Algorithms for mining distance-based outliers in large datasets," in *Proc. 24th Int. Conf. Very Large Data Bases (VLDB)*, New York, NY, USA, Aug. 1998, pp. 392–403.

[5] J. Wu, L. Zou, L. Zhao, A. Ai-Dubai, L. Mackenzie, and G. Min, "A multi-UAV clustering strategy for reducing insecure communication range," *Comput. Netw.*, vol. 158, pp. 132–142, Jul. 2019.

[6] R. Zhu, B. Wang, X. Yang, B. Zheng, and G. Wang, "SAP: Improving continuous top-k queries over streaming data," *IEEE Trans. Knowl. Data Eng.*, vol. 29, no. 6, pp. 1310–1328, Feb. 2017.

[7] R. Zhu, B. Wang, S. Luo, X. Yang, and G. Wang, "Approximate continuous top-k query over sliding window," *J. Comput. Sci. Technol.*, vol. 32, no. 1, pp. 93–109, 2017.

[8] B. Wang, R. Zhu, X. Yang, and G. Wang, "Top-K representative documents query over geo-textual data stream," *World Wide Web*, vol. 21, no. 2, pp. 537–555, 2018.

[9] 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 30th Int. Conf. Data Eng. (ICDE)*, Chicago, IL, USA, Mar./Apr. 2014, pp. 76–87.

[10] M. Kontaki, A. Gounaris, A. N. Papadopoulos, K. Tsichlas, and Y. Manolopoulos, "Continuous monitoring of distance-based outliers over data streams," in *Proc. 27th Int. Conf. Data Eng. (ICDE)*, Hannover, Germany, Apr. 2011, pp. 135–146.

[11] F. Angiulli and F. Fassetti, "Detecting distance-based outliers in streams of data," in *Proc. 16th ACM Conf. Inf. Knowl. Manage. (CIKM)*, Lisbon, Portugal, Nov. 2007, pp. 811–820.

[12] K. Yamanishi, J.-I. Takeuchi, G. J. Williams, and P. Milne, "On-line unsupervised outlier detection using finite mixtures with discounting learning algorithms," *Data Mining Knowl. Discovery*, vol. 8, no. 3, pp. 275–300, 2004.

[13] D. Yang, E. A. Rundensteiner, and M. O. Ward, "Neighbor-based pattern detection for windows over streaming data," in *Proc. 12th Int. Conf. Extending Database Technol. (EDBT)*, Saint Petersburg, Russia, Mar. 2009, pp. 529–540.

[14] L. Zhao, X. Li, B. Gu, Z. Zhou, S. Mumtaz, V. Frascolla, H. Gacanin, M. I. Ashraf, J. Rodriguez, M. Yang, and S. Al-Rubaye, "Vehicular communications: Standardization and open issues," *IEEE Commun. Standards Mag.*, vol. 2, no. 4, pp. 74–80, Dec. 2018.

[15] L. Zhao, A. Al-Dubai, A. Y. Zomaya, G. Min, A. Hawbani, and J. Li, "Routing schemes in software-defined vehicular networks: Design, open issues, and challenges," *IEEE Intell. Transp. Syst. Mag.*, to be published.

[16] L. Qi, R. Wang, C. Hu, S. Li, Q. He, and X. Xu, "Time-aware distributed service recommendation with privacy-preservation," *Inf. Sci.*, vol. 480, pp. 354–364, Apr. 2018.

[17] L. Qi, Y. Chen, Y. Yuan, S. Fu, X. Zhang, and X. Xu, "A QoS-aware virtual machine scheduling method for energy conservation in cloud-based cyber-physical systems," *World Wide Web*, 2019.

[18] J. Yu, Z. Kuang, B. Zhang, W. Zhang, D. Lin, and J. Fan, "Leveraging content sensitiveness and user trustworthiness to recommend fine-grained privacy settings for social image sharing," *IEEE Trans. Inf. Forensics Security*, vol. 13, no. 5, pp. 1317–1332, May 2018.

[19] J. Yu, C. Zhu, J. Zhang, Q. Huang, and D. Tao, "Spatial pyramid-enhanced NetVLAD with weighted triplet loss for place recognition," *IEEE Trans. Neural Netw. Learn. Syst.*, to be published.

[20] J. Yu, M. Tan, H. Zhang, D. Tao, and Y. Rui, "Hierarchical deep click feature prediction for fine-grained image recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, to be published.

[21] J. Yu, J. Li, Z. Yu, and Q. Huang, "Multimodal transformer with multi-view visual representation for image captioning," 2019, *arXiv:1905.07841*. [Online]. Available: https://arxiv.org/abs/1905.07841

[22] H. Gao, W. Huang, and X. Yang, "Applying probabilistic model checking to path planning in an intelligent transportation system using mobility trajectories and their statistical data," *Intell. Automat. Soft Comput.*, vol. 25, no. 3, pp. 547–559, 2019.

[23] H. Gao, W. Huang, Y. Duan, X. Yang, and Q. Zou, "Research on cost-driven services composition in an uncertain environment," *J. Internet Technol.*, vol. 20, no. 3, pp. 755–769, 2019.

[24] J. Pu, Y. Wang, X. Liu, and X. Zhang, "STLP-OD: Spatial and temporal label propagation for traffic outlier detection," *IEEE Access*, vol. 7, pp. 63036–63044, 2019.

[25] J. Zhu, Y. Wang, D. Zhou, and F. Gao, "Batch process modeling and monitoring with local outlier factor," *IEEE Trans. Control Syst. Technol.*, vol. 27, no. 4, pp. 1552–1565, Mar. 2019.

[26] K. C. K. Lee, W. C. Lee, B. Zheng, H. Li, and Y. Tian, "Z-SKY: An efficient skyline query processing framework based on Z-order," *VLDB J.*, vol. 19, no. 3, pp. 333–362, 2010.

[27] K. C. K. Lee, B. Zheng, H. Li, and W. C. Lee, "Approaching the skyline in Z order," in *Proc. 33rd Int. Conf. Very Large Data Bases*, Sep. 2007, pp. 279–290.

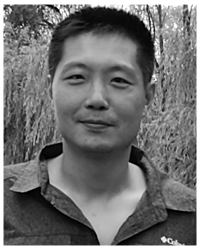
[28] P. Ciaccia, M. Patella, and P. Zezula, "M-tree: An efficient access method for similarity search in metric spaces," in *Proc. 23rd Int. Conf. Very Large Data Bases (VLDB)*, Athens, Greece, Aug. 1997, pp. 426–435.

[29] P. Zezula, P. Savino, G. Amato, and F. Rabitti, "Approximate similarity retrieval with M-trees," *VLDB J.*, vol. 7, no. 4, pp. 275–293, 1998.

RUI ZHU received the M.Sc. degree in computer science from the Department of Computer Science, Northeastern University, China, in 2008, the Ph.D. degree in computer science from Northeastern University, in 2017. He is currently an Associate Professor with Shenyang Aerospace University. His research interests include design and analysis of algorithms, databases, data quality, and distributed systems.

TIANTIAN YU received the bachelor's degree in network engineering from the Department of Computer Science, Shenyang Aerospace University, in 2017. She is currently pursuing the master's degree with Shenyang Aerospace University. Her research interests include algorithms, databases, and the planning and distribution of spatio-temporal data.

ZHIYUAN TAN received the B.Eng. degree in computer science and technology from Northeastern University, China, the M.Eng. degree in software engineering from the Beijing University of Technology, China, and the Ph.D. degree in computer systems from the University of Technology Sydney, Ultimo, NSW, Australia. He is an Associate Editor of IEEE ACCESS and an Organizer of Special Issues for the *Ad Hoc and Sensor Wireless Networks Journal*, the *International Journal of Distributed Sensor Networks*, *Computers and Electrical Engineering*, and IEEE ACCESS.



WEI DU received the M.Sc. degree in mechanical engineering from HIT, in 2008. He is currently an Engineer with Chinese People's Liberation Army Force. His research interests include wireless communications technology, edge computation, ad hoc and sensor wireless networks.

LIANG ZHAO received the Ph.D. degree in computing from the School of Computing, Edinburgh Napier University, in 2011. Before joining Shenyang Aerospace University, he worked as an Associate Senior Researcher with Hitachi (China) Research and Development Corporation from 2012 to 2014. He is currently an Associate Professor with Shenyang Aerospace University, China. His research interests include VANETs, SDN, and WMNs.

JIAJIA LI received the M.S. and Ph.D. degrees in computer science from Northeastern University, in 2010 and 2014, respectively. She is currently an Associate Professor with Shenyang Aerospace University. She undertakes one National Natural Science Foundation of China and one Natural Science Foundation of Liaoning Province of China. Her research interests include spatial-temporal database, uncertain data management, and machine learning.

XIUFENG XIA received the M.Sc. degree in computer science from the Department of Computer Science, Northeastern University, China, in 1991, the Ph.D. degree in computer science from Northeastern University, in 2005. He is currently a Professor with Shenyang Aerospace University. His research interests include design and analysis of algorithms, databases, data quality, and distributed systems.