Hindawi
Complexity
Volume 2021, Article ID 9178461, 20 pages
https://doi.org/10.1155/2021/9178461

Research Article

EMM-CLODS: An Effective Microcluster and Minimal Pruning CLustering-Based Technique for Detecting Outliers in Data Streams

Mohamed Jaward Bah,[1] **Hongzhi Wang**,[2] **Li-Hui Zhao**,[3] **Ji Zhang**,[4] **and Jie Xiao**[5]

[1]*Zhejiang Lab, Hangzhou, China*
[2]*Harbin Institute of Technology, Harbin, China*
[3]*North University of China, Taiyuan, China*
[4]*University of Southern Queensland, Toowoomba, Australia*
[5]*Hangzhou Yugu Technology Co., Ltd., Hangzhou, China*

Correspondence should be addressed to Ji Zhang; zhangji77@gmail.com

Received 7 July 2021; Revised 10 August 2021; Accepted 23 August 2021; Published 13 September 2021

Academic Editor: Fei Xiong

Copyright © 2021 Mohamed Jaward Bah et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Detecting outliers in data streams is a challenging problem since, in a data stream scenario, scanning the data multiple times is unfeasible, and the incoming streaming data keep evolving. Over the years, a common approach to outlier detection is using clustering-based methods, but these methods have inherent challenges and drawbacks. These include to effectively cluster sparse data points which has to do with the quality of clustering methods, dealing with continuous fast-incoming data streams, high memory and time consumption, and lack of high outlier detection accuracy. This paper aims at proposing an effective clustering-based approach to detect outliers in evolving data streams. We propose a new method called Effective Microcluster and Minimal pruning CLustering-based method for Outlier detection in Data Streams (EMM-CLODS). It is a clustering-based outlier detection approach that detects outliers in evolving data streams by first applying microclustering technique to cluster dense data points and effectively handle objects within a sliding window according to the relevance of their status to their respective neighbors or position. The analysis from our experimental studies on both synthetic and real-world datasets shows that the technique performs well with minimal memory and time consumption when compared to the other baseline algorithms, making it a very promising technique in dealing with outlier detection problems in data streams.

1. Introduction

In the current era, the need to detect abnormal behavior to reveal salient facts, observations, and realizing accurate predictions of data is extremely significant. Detecting outliers is one such important data mining task that aims at detecting objects that deviate from the expected pattern of the normal data. The process of detecting outliers is challenging due to the advancement in the digital age. For instance, with the revolution of data from traditional batch data, we have witnessed the advent of a large volume of data that is generated continuously at high speed and dynamically. These kinds of data are known as data streams and are generated by many applications [1–3]. In contrast to traditional datasets, because of the nature of the data, it is not feasible to save in memory the whole data stream or run the data through multiple scans. This is because the data are massive and unbounded, have a varying rate, and continue to evolve.

A significant number of approaches have been proposed to detect outliers in data streams [8–11]. Among the different categories of proposed outlier detection methods, clustering-based approaches have shown to be popular in static data but yet one of the most challenging to adopt for outlier detection tasks in data streams. Although they have shown to be efficient for some outlier detection tasks, they lead to low

computational cost and high scalability in high-dimensional data [5, 12]. However, most of the prevailing data stream clustering approaches suffer from different drawbacks. They can be improved when we consider the spectrum of effectiveness and efficiency, for instance, to deal with the continuous fast-incoming data streams, higher computational demand in terms of its memory and time, the cluster quality, and the outlier detection rate. The process of clustering and detecting outliers in data streams is complicating since the clustering techniques often involve several parameters and operate in low- and high-dimensional spaces, constrained with excessive distance-based computation of object neighbors, noise, and so on. For this reason, clustering-based approach has varying performance for different application domains and data types. It is therefore imperative to design an effective method that will holistically address the issues and produce stable performance in detecting the outliers.

In spite of clustering's occasional challenges and caveats, it is still another good alternative and promising solution for detecting outliers. The advantage of clustering is that it allows for the use of limited amounts of time and memory, which is necessary when processing data streams. This is because clustering is the act of grouping elements using sets that provide the capability of grouping items that are similar to each other that curbs the need of redundant processing and over calculations. Clustering methods offer online and offline process support, which is usually used for data stream applications and is also flexible in adapting to the evolving nature of the data.

In this paper, we propose a new microclustering and minimal pruning clustering-based unsupervised outlier detection scheme to detect outliers in data streams while simultaneously addressing the mentioned challenges. The proposed approach involves different stages to adapt to the dynamic changes of data distribution that aims at eliminating the limitations of previously proposed methods. The newly propose method is called Effective Microcluster and Minimal pruning CLustering-based method for Outlier detection in Data Streams (EMM-CLODS), which is a clustering-based outlier detection approach. We call it CLODS for short and use this abbreviation instead of EMM-CLODS throughout the paper. It detects outliers from evolving data streams by first applying the microclustering technique to cluster dense data points. It then effectively handles objects within a sliding window according to the relevance of their status to their respective neighbors or position through minimal pruning technique.

In our data stream scenario, where the size of the dataset is potentially boundless, we process the data over a fixed period to reduce the complexity of the outlier detection task. When new incoming data points arrive, the microcluster technique is applied, which identifies objects that are more analogous to each other and that meet the fundamental prerequisite of the clustering methods. The methods scan the data once and adapt to the time changes as the streaming data evolve. It constantly and periodically updates incoming data, and the results are obtained. Finally, the CLODS reports key insights from these results to determine whether they are outliers or inliers. The advantages of the technique are that it can effectively save time and memory, thanks to the microclustering technique and minimal pruning. It removes the need to compute every data point in and out of the cluster and store every data point in memory. In summary, the major contributions of this work are as follows:

(i) We propose the CLODS, a new technique based on microclustering and minimal pruning of data points outside the clusters, to solve the problem of detecting outliers in continuous evolving data streams.

(ii) We propose the concept of priority handling of evolving objects outside the clusters to minimize the memory and time consumption during the updating phase according to the relevance of their status to their respective neighbors or position.

(iii) Our propose method can effectively optimize and solve the problems and challenges of time and memory constraints while maintaining its accuracy for detecting outliers in data streams.

(iv) We demonstrate through an extensive experiment on some benchmark datasets the effectiveness of our method against some other methods used for the outlier detection process in data streams.

The rest of the paper is organized as follows: in Sections 2 and 3, we present the related work and problem formulation, respectively. In Section 4, we present in details the method we propose. In Section 5, we present the experimental studies including the results and discussion. Finally, in Section 6, we present the conclusion of the paper.

2. Related Work

Detecting outliers is a well-known domain in the data mining community, and it has been applied in a wide range of application areas [13, 14] and other domains such as community detection [15, 16]. It has been studied extensively [17–19]. In a recent survey [11], we classified outlier detection methods into diverse categories and have proposed effective methods among these categories to detect outliers in data streams [8, 11]. In progress to this study series, the clustering-based category has open research gaps and challenges. Proposing solutions and improving these methods will greatly contribute to the general body of outlier detection methods.

The clustering approach is an unsupervised data mining method that groups similar dense data points. Several methods using clustering techniques and its variant approaches have been proposed for outlier detection tasks. However, some earlier proposed clustering methods suffer from drawbacks such as the buffering of all data points in memory for future handling or, in some cases, not considering data points that often leads to poor clustering. There are a significant number of these methods concentrated on both static data and streaming data types [20, 21]. These methods mostly adopt the two-phase scheme: the online and offline phase. The majority of the earlier proposed method for stream data clustering deals with static

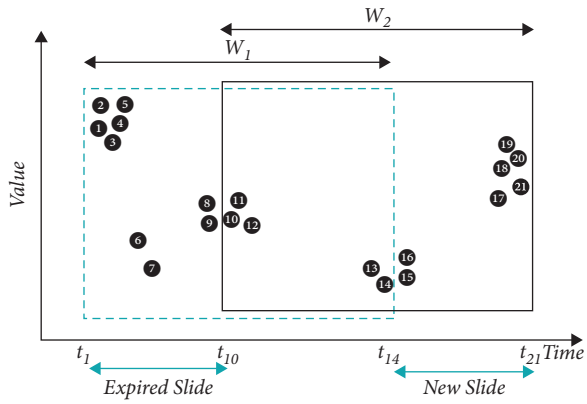

Figure 1: Streaming data.

clustering that is in a continuous form. One shortcoming of this kind of approach is that recent and outdated data are handled the same way. Several moving window models are proposed to solve this issue. For evolving data streams, Toshniwal and Yokita et al. [20] proposed a framework using simple k-means and the attribute weight to detect outliers, while Cao et al. [22] proposed a technique related to density-based clustering for evolving data streams. In their method, the incoming data are selected depending on the distance between their centers to either the outlier or potential core microcluster. In this case, with an increasing number of outliers, the clustering accuracy becomes a problem. Therefore, Liu et al. [23] proposed a new technique to address this drawback. Although they tried to address the issue, it comes at a high computational cost. To salvage the computational cost and improve the clustering and outlier detection accuracy, Kumar and Sharma [24] applied a technique that extracts the boundary points in the overlapped microclusters. Many other clustering techniques have been proposed for outlier detection processes, such as density-based microclustering [22, 25], grid-based clustering [6, 26], and partitioning algorithm for data streams [12, 21]. However, since this is a short paper, Table 1 briefly outlines some of these techniques in comparison to our method in terms of the summarization technique, evolving data model and outlier detection method.

Remarkably, from Table 1, no two methods share the same approach. Our work is the first to use microclustering in the sliding window model using outlier microcluster to handle continuously evolving objects with changing features. For a more comprehensive related work to clustering techniques for outlier detection, we recommend Wang et al. [11] survey paper.

3. Preliminaries and Problem Formulation

3.1. Notations and Definitions. The key symbols used in this paper include but not limited to the following in Table 2.

3.2. Definition of Key Terms

3.2.1. Outliers. For a dataset D of n points, $D = [d_1, d_2, \ldots, d_n]$. Whenever the data point d_i or an entire set of data points $d_1, d_2, \ldots, d_n$ deviates drastically from these other sets, these points are considered outliers.

3.2.2. Neighbor. In the case of two data points d_i and d_n, a data point d_i is considered a neighbor of d_n if the distance between the two does not exceed the distance threshold value $R > 0$. In other words, if d_i is not further than R from d_n, then it is a neighbor of d_n. A data point d cannot be a neighbor of itself.

3.2.3. Sliding Window. In sliding window, the time-based window and the count-based window are two types of window models commonly used for data streams. The former takes into consideration the data points within the time interval of two identify data points, for instance, at point x and y, with t_x and t_y. The latter thus considers the count of the data points within a specified window size.

3.2.4. Microclusters. A microcluster is formed when a data point has a radius of $R/2$ from the center, and in a microcluster, the distance between two data points, let us assume d_1 and d_2, should not exceed R.

The function of the microcluster in our technique is as follows: we applied the microclusters to minimize the range queries and minimize the distance-based computations. The microclusters eliminate the need for excessive range queries by storing the neighbor's data points in the microclusters. This, therefore, improves the underlying evaluation metrics: the memory and time consumption. The microclusters adopted in the proposed methods give the advantage of eliminating the need for range queries and in curbing the distance computations. In addition to only storing crucial inliers in memory, the microclusters also improve the memory constraints, since a single microcluster has the ability to obtain the neighborhood information of each object in the same cluster.

In Figure 1, we can see that $W_1 = t_1 - t_{14}$ and $W_2 = t_{10} - t_{21}$, where W_2 is the current window and W_1 is the expired window. The fast-incoming data points (dp) from 1 to 23 are the data streams. By definition, the data stream $i = S_t | 0 \le t$, with $t =$ time and dp, $S_{i=1,2,n} = S_1, S_2, S_3, \ldots, S_n$. Each dp within its window could have a neighbor or not, but it cannot be a neighbor on its own. The neighbor of any particular data point S_i must not exceed the required distance threshold R, from each other. For instance, in Figure 1, dp $1, 2, 4, 5$ are neighbors of 3, while $17, 18, 20, 21$ are neighbors of 19. The neighbors play a crucial role in the overall outlier detection process; therefore, we pay special attention to them.

In W_2, or when the window slides, determining whether a data point is an outlier or inlier can create additional constraints due to the evolving nature of the data points. Some neighbors will expire, such as dp $8, 9$ among $8 - 12$, and become obsolete when the window slides. In the different window stages, the question of how to perform clustering, how to use minimal pruning to get the most significant data points, how to deal with incoming and expired dp, and what kind of clustering technique to apply comes up, and also, what requirements should the clustering technique meet to ensure that (1) the clusters capture more

Table 1: Some key clustering algorithms.

Method	Summarization technique	Evolving data model	Outlier detection
CluStream [4]	Microcluster	Tilted-time window	—
D-Stream [5]	Grid	Fading window	Sporadic grid
DenStream [6]	Microcluster	Fading window	Outlier microcluster
DENGRIS-Stream [7]	Grid	Sliding window	Sparse grid
Ours-CLODS	Microcluster	Sliding window	Outlier microcluster

Table 2: List of symbols with their interpretations.

Symbols	Interpretations
d_i	i-th data point, $i = 1, \ldots, n$
R	Distance threshold
K	Number of neighbors
W	Window size
S	Window slide size
∞	Data streams
t_i	The specific time
d_{ci}	Data points in the current window
d_{ei}	Expired data points
O_d	Detected outlier/s

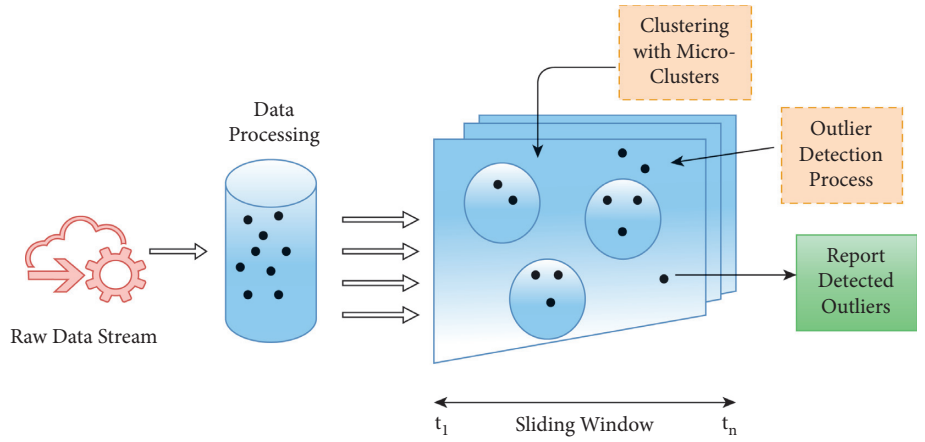

Figure 2: The framework of CLODS

dp and (2) the inliers or outliers are detected correctly and computed with the lowest computational cost possible.

3.3. Problem Formulation. Problem statement: the major goal of this paper is to present an improved solution to address the problem of effectively clustering and detecting outliers in fast-evolving data streams.

For new data streams arriving continuously, $S = \{S_t\}_{t=1,2,\ldots,}$ with dimensionality d at time t, and with evolving feature changes as the data speed increases, we need to design a robust approach that will deal with the evolving data streams by clustering incoming data streams effectively and simultaneously detect all outliers in the shortest conceivable time, with low memory usage, while maintaining high detection accuracy. Also, we handle data points outside the clusters while dealing with the fading of old clusters, new and expired data points, and detecting the outliers. The key challenge is that the actively evolving data point position continues to change due to either the window slides or the arrival and expiration of some data points. This ultimately makes it complicating in addressing the overall problem. It will be a challenging task to process and remove data points one at a time as they arrive over the stream. It will incur a lot of time.

In addition, managing memory space presents another challenge since it is not possible to predict how many data points arrive and expire a priori. It becomes challenging to cluster essential data points and dynamically allocate space for the growing number of unknown data points that arrive and expire.

This brings us to the essential problem statement and question we address in this paper, how do we capture the data points that deviate from the others in streaming data which evolve as time progresses with these additional constraints:

(i) The data point features might change over time.

(ii) Prior unseen data point features might arrive over time.

4. The Proposed Methodology

4.1. Fundamentals of the Proposed Method. As data originate from their source in the form of fast continuous evolving data streams, they become challenging to cluster data points and effectively detect the outliers, as explained in the problem statement. There is a need for special attention on the clustering method and in handling both the inliers and outliers in this scenario. To do this, we propose a new framework, which involves different stages in order to detect the outliers efficiently while maintaining high accuracy. The newly proposed method called Effective Microcluster and Minimal pruning CLustering-based method for Outlier detection in Data Streams (EMM-CLODS) is a kind of clustering-based outlier detection approach that detects outliers in evolving data streams using microcluster and minimal pruning. This is done by first applying a micro-clustering technique to cluster dense data points and effectively handle the data points according to the relevance of their status to their respective neighbors or position in the window. We adopt the sliding window model, and within this model, the microclustering technique helps to cluster dense data points quickly and eliminate the need for a range query search. For the data points outside the clusters, an approximate probing is implemented by excluding a set of inliers whose significance in the computation is trivial in order to reduce the computation demand.

The CLODS makes use of both clustering and approximate probing of data points within the adopted sliding window model and minimal pruning of data points outside the clusters. It simultaneously discovers the outliers and deals with potential outliers outside the clusters, even when they continuously evolve as the data point changes state. In contrast to other conventional clustering-based approaches, it does not limit itself to detecting outliers in static data [2, 11, 27], and for those that support data streams, the clustering procedure is different [12, 20, 28, 29], or they are not clustering-based approaches [4, 8, 30]. Those with similar clustering techniques to ours use a different scheme to deal with data points within the window or adopt different window models [12, 27, 29]. Furthermore, the handling procedure of data points outside the microclusters is different. Unlike some of these methods [12, 20, 27, 28] that deal with every data point outside the microclusters equally, we focus especially on the relevance of data points with respect to its neighbors and position to determine its overall role in the outlier detection process. This is to ensure we identify potential outliers rather than data points that might be falsely labeled as outliers. This consequently saves time and memory constraints without a performance decline.

4.2. The Proposed Framework. Figure 2 shows an illustrative representation of the proposed framework. At the onset, objects in the form of data streams arrive continuously and in an unprecedented manner. We first filter the data through data processing to determine its characteristics. Then, we process the preprocessed data in the sliding window model. During a specified period in the sliding window, we apply probing and clustering process together with pruning the data points outside the clusters and detect the outliers. During this phase, additional processing such as handling of crucial inliers and potential outliers, and handling of both active and expired data points as the window slides is done. In the final stage, the detected outliers are then reported.

Algorithm 1 gives the overall framework of CLODS, with line 3–5 depicting the processes. In Algorithms 2–4, details of algorithmic process are given to understand the whole CLODS algorithm. In Algorithm 5, we extend details of the different steps in Algorithm 1. In the first part, we perform preprocessing. The preprocessed data stream is then computed in the next stage. In processing data points within the window, in line 4 we determine whether they belong to a cluster. If not in a cluster, the relevance of their status with respect to the other members is checked in line 9. The data points outside the clusters and that are not relevant to their respective members can be applied to the function in the last stage and reported as an outlier as can be seen in line 11.

In Algorithm 2, the processing of new data points in the new sliding window is shown. We first discover the cluster and if there is a data point d_p within the cluster, we add the new data point or else initiate a new cluster accordingly (line 2–6), while in Algorithm 3, it shows the processing of the expired data. Similarly, as in 3, we first discover the cluster and if a data point is found in the cluster, we ensure that we check the $d_p's$ relevance status to the other data points before we add it into the cluster (line 4-5). If not, we try to remove it (line 7).

Lastly in Algorithm 4, we process and report the detected outliers. We first initialize the count (line 1), and if d_p is not in any cluster and less number of neighbors to form a cluster, it is returned as an outlier. If it has already expired, it is then removed from data points outside the microclusters.

4.3. The Data Stream Stage. In a data stream model, the input data are not accessible through random disk or memory, such as in the case of static data or batch data in standard databases, but rather arrive in the form of one or more continuous data streams. A data stream is an unlimited number of sequence data points $\infty_i = S_t|0 \le t$, within a specific timestamp or unbounded sequence with data points, $S_i = S_1, S_2, S_3, \ldots, S_n$. They are infinite series of data points, S_{t-2}, S_{t-1}, S_t, observed at a particular time t. The streaming data have the following characteristics:

(i) The data points of streaming data arrive incrementally in real-time. The streaming data are active since all inbound objects/items trigger actions on the data rather than being invited to participate.

(ii) The system has no control over the order or sequence in which the items of the streaming data arrive.

(iii) The streaming data have the possibility of unbounded numbers of data points.

The problem of detecting or mining outliers in such data with the abovementioned characteristics brings a number of significant implications. Firstly, to ensure that the results are continuously up-to-date, it is essential to analyze the incoming data within the shortest time and minimal memory

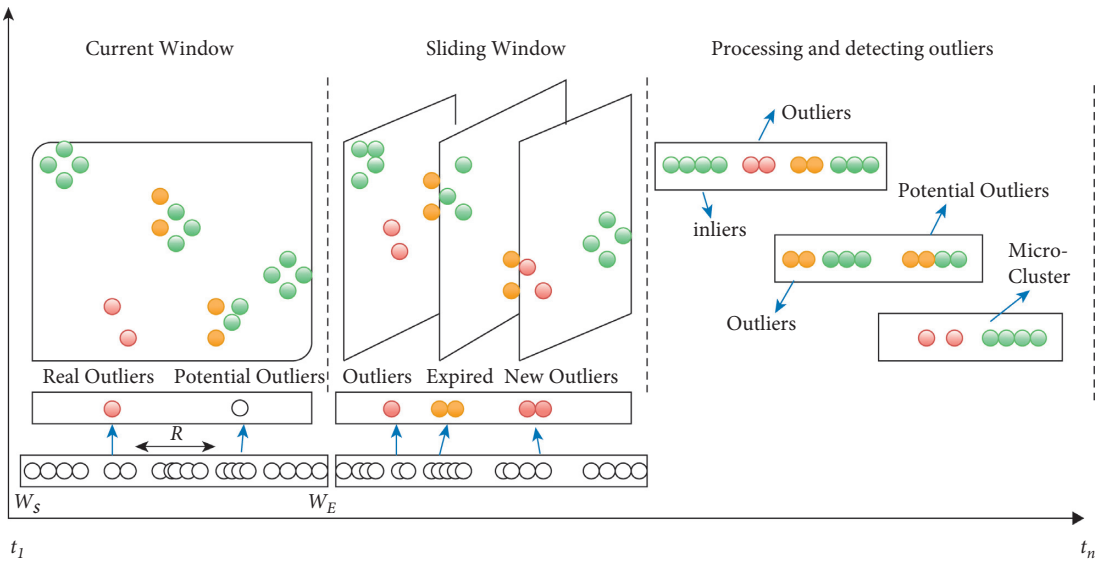

Figure 3: The different phases of processing the outliers in the sliding window.

Input: Preprocess Data Stream ∞, Data point d_p, Parameters: {distance-threshold R, nearest-neighbor count K, sliding size S, Window Size W.}
Output: Outliers in sliding window
(1) **Procedure:**
(2) **While** the window slide or in W_c ▷ between period W_{start} to W_{end} when S arrives
(3) Deal with data within W_c
(4) Deal with new d_p, S and W
(5) Deal with expired d_{ei}, S and W.
(6) Report outliers, O_d
(7) **end**

Algorithm 1: The CLODS algorithm.

usage. In the framework in Figure 3, the continuous infinite series of data points observed at a particular time t_1 is fed to the next stage.

4.4. Data Preprocessing Stage. As the incoming unbounded sequence of data arrives, it is impossible to store the entire data stream. Besides, to apply the clustering technique without taking note of the characteristics of the data makes the overall process more tedious. Therefore, we initially did some preprocessing based on the nature of the data to avoid assumptions about having clean and well-structured data and to tailor the data for our propose model. For instance, real-world datasets are highly susceptible to missing and inconsistent data. Such datasets may give rise to data quality issues, which in turn affects the overall result. During the data preprocessing and wrangling phase, we deal with the missing data and inconsistent data. Although outliers sometimes can influence the quality of the data, in this work we entirely avoid dealing with outliers since our primary goal is to detect outliers. For the missing data, we ensure that we ignore, fill manually, and compute values. For inconsistent data, we normalize the necessary datasets.

4.5. Sliding Window-Based Outlier Detection Stage. In this phase, we manage the evolving data streams; that is, we implement the CLODS and detect data points that deviate from their expected normal behavior when the window slides and expires, also when the data points will expire. We notice that it is not feasible to perform clustering on data streams during the all probable time. We handle the data points at different time windows. The process of exploring the evolving data stream during the different time windows provides the users with additional insights into the evolving nature and performance of the clusters. In terms of processing evolving data streams, different algorithms have adopted different window models. Some existing window models include the damped window model also known as the fading window model, the landmark window model, the tilted-time window model, and the sliding window model. In this paper, we use the sliding window model, in which the data are processed before the end of the streaming data window. This is as opposed to the landmark window model, which is adopted for cases where we want to mine the whole data stream history. It is suitable for static data settings. In the sliding window, the streaming data are considered from the current time to a certain range in its history. The key idea

(1) **for** d_p in new slide, S **do**
(2) c = discoverCluster
(3) **if** d_p in C then
(4) c.add (d_p)
(5) **else**
(6) InitiateNewCluster (d_p)
(7) **else if**
(8) **end for**

Algorithm 2: *Process new data in the new slide window.

(1) **for** d_p in expired slide, S **do**
(2) c = discoverCluster
(3) **if** d_p in C then
(4) CheckRelevance (d_p)
(5) c.add (d_p)
(6) **else**
(7) remove (d_p)
(8) **end If**
(9) **end for**

Algorithm 3: *Process expired data point when slide expires.

(1) Initiate outliers = [ ]
(2) Perform all functions
(3) **for** d_p in W, S **do**
(4) **if** d_p cannot form a new cluster
(5) add.Outlier (d_p)
(6) **else**
(7) Processfunctions
(8) **end if**
(9) **end for**
(10) Return Outliers

Algorithm 4: *Process outlier W.

Input: Data Stream ∞, Data point d_p, Parameters: {distance-threshold R, nearest-neighbor count K, sliding size S, window size W.}
Output: Outliers
(1) **Procedure:** ▷ Preprocessing
(2) **Perform Preprocessing** ▷ Process DataIn W_c
(3) **for** for each d_p of preprocessed data in W **do**
(4) DiscoverInClusters
(5) If $d_p \geq k + 1$ neighbor **then**
(6) InCluster
(7) **elseif**
(8) NotIncluster
(9) CheckRelevance to d_i
(10) **else**
(11) ProcessNewData in S
(12) **end if**
(13) **end for**

Algorithm 5: Overall procedure of the CLODS.

in the sliding window is to do exhaustive analysis of the most up-to-date data items and summarized the outdated items.

As can be seen in Figure 3, in the second phase, we apply the clustering of the data stream in the sliding window model where data points expire as the window slides. Moreover, with an increasing time $t = t + +$, each data points' weight declines as it reaches the expiration point. In setting the window size for a distribution that fluctuates dynamically, we increased and set the window size large enough to minimize the effect caused by the dynamic change of the data. Consequently, this results in increased time usage, which undermines the performance of real-time computation. Eventually, it creates a challenge to find a balance between these two underlying issues.

In Figure 3, as the time increases $t = \{t_1, t_2, t_3, \ldots, t_n\}$ within the time frame, some data points fade out and some data points change state depending on the window slide. Some evolving data points expire, some clusters dissolve, and new ones are created, and some data points might be classified wrongly as an outlier. Therefore, in designing CLODS, we consider the following prerequisite:

(i) Firstly, we consider the status of the data points, i.e., whether they are in a cluster or not and whether data points outside the cluster can be viewed as an inlier or outlier.

(ii) Secondly, we consider the distance between the clusters and data points outside the clusters, whether they are far or close to the clusters, and whether they can be viewed as an outlier or inlier.

(iii) Thirdly, we consider whether the data points share a relationship with few other data points that form a cluster, and also, how to handle both the data points within and out of the clusters to detect the outliers accurately.

(iv) Finally, we consider the characteristics of the summary information, and at what instance we should store or discard the summary information, and what to do with expired data points.

4.6. CLODS Clustering Phase. For a data stream with a set of continuous multidimensional data points S_1, S_n, arriving at different period $t_1, \ldots, t_n$, we considered a set of active data points during the period $t_1, \ldots, t_n$, which are the most recent n data points at the time in the sliding window. During the active period, we employ the microcluster concept, which is a fast-efficient method for clustering objects within the sliding window. We applied the idea of triangular inequality in metric space [30, 31], to guarantee the data points' distance between each other in the microclusters is less than the distance threshold R. Thus, confirming that every data point is labeled as an inlier within the microcluster. Among the labeled inliers, we store in memory only crucial inliers to avoid memory congestion, and it is impossible to store every object in memory. We stored each newly arrived object in a fix size buffer. If the buffer is full, we consider each data point in it as an inlier or outlier, depending on the weight of the objects in relation to its distance to the other objects. The objects that are labeled as outliers are deleted in memory, while all newly incoming labeled inliers are maintained in the updated list. The different actions taken depend on the status of the data points in the different phases.

Figure 3 shows the different stages in the window model, which is divided into three partitions with the x-axis displaying the arrival time of the data points, while the ordinate depicts the number of data points with radius R. In the first partition, during the current window model space (W_{start} to W_{end}), we have a set of evolving data streams $\{s_1, s_3\}$ with fixed radius R, and a neighbor count threshold k from time interval $t_1, \ldots, t_n$. In this partition, for $k = 2$, the microcluster technique is applied to cluster $K + 1$ data points for the objects in the window. These microclusters are data points within the radius of $R/2$ from the center and are not greater than the distance R between the two data points. The window contains four microclusters, c_1 to c_4 with radius $R/2$. The data points that are not within the microclusters are probable outliers depending on their status in relation to the other neighboring data points. To determine whether the probable data points will be labeled as an outlier or not, we consider both its ensuing and prior neighbors and, furthermore, its relative strength to its neighbors. Also, to consider which objects are stored in memory, we used a similar concept as in previous work [8] by storing the data points outside the microcluster in temporary memory while applying the minimal pruning to minimize the computational cost and demand. From Figure 3, the red marked data points show the outliers while the other data points, where $k \geq 2$, are marked in green.

In the next phase, some data points change state due to the sliding of the window, the appearance of new data points, and the expiration of some data points. These new changes create new challenges for detecting the outliers smoothly as compared to the previous phase. In this case, we have three sliding windows. In the first window, we have a single microcluster, outliers, and a full cluster that has some data points that their status will be potentially affected during the next slide. In the next window, at the onset, although two objects have expired, it does not dissolve the microcluster since it has $k + 1$ points. However, in the final window, the microcluster dissolves, which prompts the remaining data points to become outliers. When new data points arrive, they are added to their probable neighboring microclusters, provided it is not greater than the distance threshold R. Otherwise, it is added to the neighboring outlier cluster with more space. If none of the conditions exist, then a new marked outlier cluster is initialized. In the final stage, the figure vividly shows the status of the different data points. The green data points indicate the inliers, yellow expired data points, the orange points are those that have the propensity to change state, and the red are the detected outliers.

In terms of the memory usage, owing to the fast response and limited memory requirements in these kinds of environments, it is not practical to store the majority of the data, and it is impossible to store all the data in memory. Therefore, to salvage the situation, we minimize the memory consumption and stored relevant data points that aid the

overall clustering and outlier detection process. Furthermore, we minimized the number of rearranged micro-clusters as the update in memory is done. As the continuous incoming data arrive, we first determined whether it is in memory or not. If not, it is added to the temporary memory, and then an initialization process is done. The key inliers are temporarily stored in memory, and as the data evolve due to changes in window slides, an update is done with new data points replacing the older ones. We calculated the number of inliers, and all expired data points are deleted from memory to free the memory space. Finally, summary statistical information is obtained, and the outliers are then reported.

4.7. Outlier Detection Stage. The outlier detection process involves various phases. At the onset, we observe the potential outliers through the clusters. By definition, an outlier in an evolving data stream is a data point within the computational time frame that deviates from the clusters and lies beyond the distance threshold R with fewer than k neighbors in the dataset. In every window, data points that do not meet the deviation and threshold criteria are labeled as outliers, while the others are labeled as inliers. All potential outliers are initialized to one and stored in temporary memory. As new potential outliers accumulate, the longstanding vivid outliers stored in the outlier list are deleted from memory to free up space after processing. The detected outliers are reported, and the outlier list is updated.

5. Experiments and Results

In this section, we describe the experimental settings including the datasets, parameter settings, evaluation metrics, and the baseline methods and discuss the performance of *i*GAAL in comparison to the other models.

5.1. Experimental Setup

5.1.1. Environment. We did our experiment using Java to design the source code and ran it on Eclipse Java EE IDE on a PC running Windows 10 Operating System with 3.20 GHz X4 CPU, 8 GB of RAM, and Disk Space of 230 GB. One of the baseline algorithms is from previous work [8], and the other was prepared by Tran et al. [32]. The source code of some baseline methods and all related datasets can be found on the online repository [32].

5.1.2. Datasets. We use similar benchmark datasets that have been adopted in some previous studies [8, 32]. As shown in Table 3, we use three real-world datasets and one synthetic dataset that are openly accessible. The first dataset is the Forest Covertype (FC) [7, 32] which is openly available and can be found from the UCI Machine Learning Repository and has 581,012 records with a high-dimensional range of 1–55 attributes. The dataset comprises tree observations from four zones of the Roosevelt National Forest in Colorado. It has no remote sensing, as the entire observations are cartographic variables from $30\,\mathrm{m} \times 30\,\mathrm{m}$ sections of the forest. The FC dataset includes information on shadow coverage, tree type, distance to nearby landmarks, soil type, and local topography. The data are in raw form (not scaled) and contain binary (0 or 1) columns of data for qualitative independent variables (wilderness areas and soil types).

The second datasets adopted for our experiment are the tropical atmospheric ocean project (TAO) datasets [32, 33], which is a low-dimensional dataset with three attributes and 575, 648 records. The dataset is real-time data extracted from National Oceanic and Atmospheric Administration website [33]. TAO was established to get useful insights and forecast climate variations related to El Nino and the Southern Oscillation (ENSO). The phenomenon, ENSO, signifies the strongest year-to-year climate instability on the planet. Its events undoubtedly interrupt normal patterns of weather variability, thereby disturbing farming, transport, Pacific marine ecosystems, energy produce, and the livelihood of millions of people around the world.

The Stock dataset has only one attribute, and it is available from UPenn Wharton Research Data Services [34] with 1,048,575 records. The dataset shows Stock trading traces of about 1 million transactions throughout the trading hours per day. Since the Wharton Research Data Services is not easily accessible, the available data can be found on the online repository [32] together with the other datasets used in this experiment.

For the Synthetic dataset, we use the Gauss dataset [32]. The dataset is generated to produce streams with measured data distribution types and number of outliers. It is generated by mixing three Gaussian distributions and a random noise distribution, and it contains 1 million records with a single attribute. In each segment of the stream, the Gaussian distributed points and noise are randomly distributed.

5.1.3. Default Parameter Settings. Before performing our experiment, we take into consideration the slide size S, the window size W, the distance threshold R, and the neighboring count threshold K. The window size W is the key parameter which determines the volume of the data streams and number of accommodated clusters, while the slide S affects the speed and the remaining parameters help to determine whether the evolving data points are inliers or outliers or whether they belong to a cluster or not. The default value of W, S, R, and K is shown in Table 3 for the different datasets.

5.1.4. Evaluation Method. We evaluated our method using three evaluation metrics: the running time, memory usage, and the clustering quality. The running time is the time taken to complete the detection of outliers for each window slide. The memory usage is the record of the peak memory used during the outlier detection process, including the storage data for each window. Lastly, the clustering quality defines how accurately our approach clusters the datasets.

Table 3: Datasets with default values.

Dataset	Size (M)	Dim	W	S	R	K	Outlier rate (%)
FC	0.6	55	10,000	500	525	50	1
TAO	0.6	3	10,000	500	1.90	50	0.98
Stock	1.1	1	100,000	5,000	0.45	50	1
Gauss	1.0	1	100,000	5,000	0.028	50	0.96

5.1.5. Baseline Algorithms. We chose three state-of-the-art algorithms, MCOD [4, 35] Thresh_LEAP, MCMP for comparison with the CLODS. MCOD and Thresh_LEAP were the best performing among the existing methods [36] until the hybrid approach called MCMP [8] was proposed, which uses the strength of both techniques to boost the performance in solving outlier detection problems. In MCMP, the key difference when compared to the other baseline methods is in dealing with data points within the current window. MCMP implement uses the concept of strong and trivial inliers of dealing with the objects outside the microclusters. Thresh_LEAP in the majority cases is inferior to both MCOD and MCMP because of their lack of memory-efficient microclusters. It uses an index per slide for its neighbor search. Its minimal probing principle mitigates the expensive range queries and prioritizes the discovery of a minimal number of data points according to their arrival time. It has to continually re-evaluate and manage the data points in the updated list, which consequently increases its computational demand, while MCOD prunes out and minimizes outlier candidates. It uses an index structure called a microcluster that helps to prune out unqualified outlier candidates resourcefully. However, in MCOD, the absence of clearly distinguishing between the points outside the microclusters limits its potential to perform even better. Therefore, MCMP improves this shortcoming by using the strength of Thresh_LEAP minimal probing and the memory-efficient microcluster and introduces the concept of trivial and strong inliers. This consequently improves the overall performance both in terms of reducing time and memory consumption. However, the improved performance comes at a cost, and we noticed that the absence of the extensive distance-based computation of data points outside the microclusters thus would lower the time and memory usage when we focus mainly on the clustering and deal with those points according to the relevance of their respective neighbors. For in-depth understanding of the baseline methods, we request our audience to read the individual references.

5.2. Results and Discussion

5.2.1. CPU Time. In order to observe the CPU time usage, we take into consideration the following: we vary the window size W, the distance threshold R, and the nearest neighbor count K.

Figure 4 shows the outcome of varying the window sizes W, from $10k$–$20k$ for FC and TAO and then $10k$–$200k$ for Stock and Gauss. The results are shown for fixed $K = 50$ and an approximate 1% outlier rate across the datasets. In Figure 4, for all datasets, in most cases as W increases which means more data points to cluster and compute, the CPU time also increases (Figures 4(a) and 4(c)) except for Thresh_LEAP in Figures 4(b) and 4(c), and MCOD in Figure 4(d). The CLODS, similar to MCMP and MCOD in FC and TAO, shows a steady rise in all the datasets. However, in Gauss, when W is above $50K$, we observe a sharp spike for Thresh_LEAP because fewer data points are captured since it does not have microclusters. Both CLODS and MCMP show the lowest CPU time usage when compared to the others since the use of index structures is absent. The CLODS ensures that significant inliers are stored in microclusters, which reduces the computational demand of performing range queries for every data point. Generally, we observe that when W is large enough, there is a tiny effect on the streaming data whose distribution changes dynamically. Nevertheless, if W becomes too large, then it will influence the responding time, and the time will greatly increase, which will, in turn, downgrade its performance.

Figure 5 illustrates the result of changing the neighbor count threshold k, from 1 to 100 across all the datasets. The results are shown for window size, $W = 10K$ for FC and TAO and $W = 100K$ for the remaining two datasets with other default parameters been maintained. In Figure 5, all the methods showed some changes across the different datasets since they depend on the neighbor count threshold k, which affects the outlier rate. From the figures, except for Thresh_LEAP in TAO and Stock (Figures 5(b) and 5(c)), which demands more probing to find k, the other methods showed very good time consumption with CLODS showing superior performance in the majority of the dataset. This is because, in the first three datasets, there are not many data points that fall within the clusters that will require additional computation. For Figures 5(a) and 5(d), an increase in k shows an increase in the time since more probing needs to be done. In Figure 5(d), we see that MCMP slightly outperforms CLODS because few clusters demand additional computation. Overall, our approach performs well for the datasets that have points whose neighbors are close to each other, which makes it easy for clustering and thus makes it easy to differentiate between vivid or false outliers and crucial or insignificant inliers. Consequently, it shows better performance than the others since it can do the least computation possible outside the clusters. The likelihood of getting enough neighbors to ensure the fast clustering process is relatively low for datasets with sparse data points. Therefore, there are fewer clusters in the synthetic dataset, which also results in increased processing time when compared to the real-world datasets.

Figure 6 displays the result and performance of varying the slide size S, from 1% of W to 100% of W. The

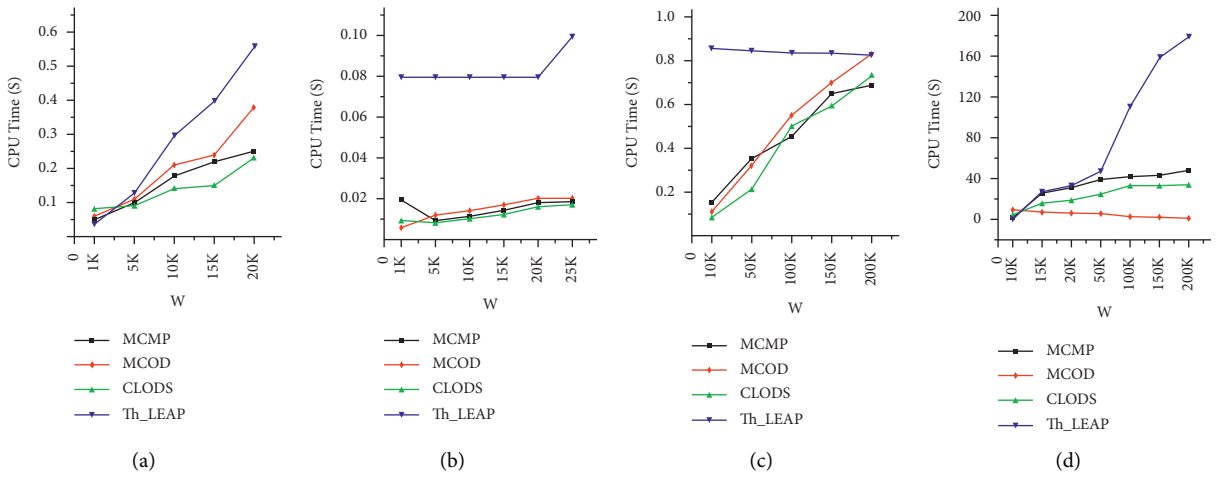

Figure 4: CPU time-varying W. (a) FC. (b) TAO. (c) Stock. (d) Gauss.

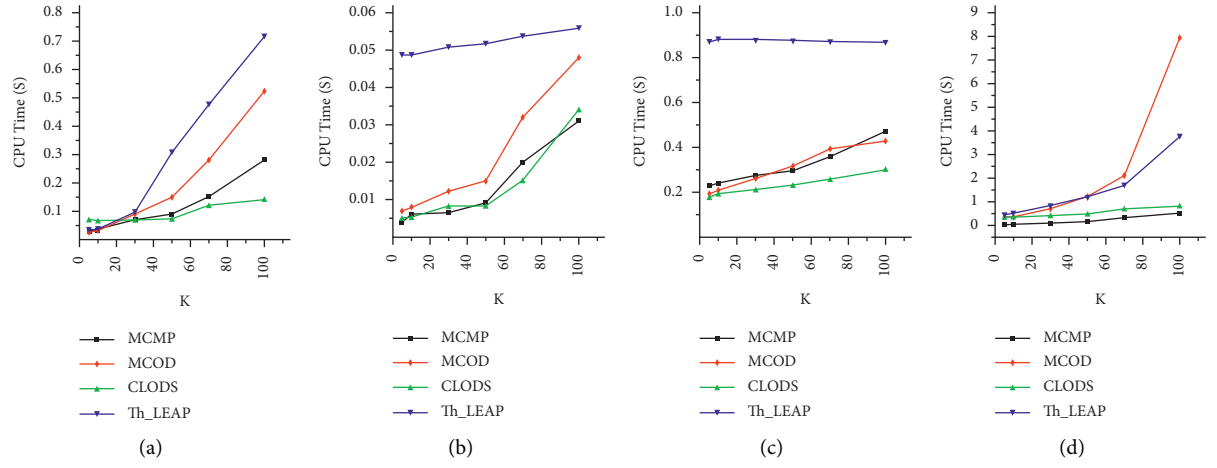

Figure 5: CPU time-varying k. (a) FC. (b) TAO. (c) Stock. (d) Gauss.

slide size depicts the changes in the speed of the data stream. Across all the datasets, the value of k and R is maintained as in Table 3. In Figure 6, we can see that across the datasets the CLODS shows the lowest CPU time usage, while Thresh_LEAP incurs in the majority of the cases the highest CPU usage above that of MCOD and MCMP. In TAO and Stock datasets, we omit the trend of Thresh_LEAP since the CPU time incurs far greater than the others, and for the other two cases, it shows an abnormal trend when compared to the others. The CLODS and the other algorithm show an increase with increase in S/W. It confirms that an increase in S results in arrival and expiration of more data points, thereby consuming additional time. However, the CLODS showed improved performance compared to that of MCMP since it uses less time than MCMP and MCOD that tries to update its neighbors after the detection of strong and trivial inliers and in identifying the outliers. In addition, we can observe that the processing of new arriving data points in CLODS scales well to that of the expired data points when the window size increases. In MCOD, for example, the time taken to process half of the data points outweighs the time for saving in discarding the expired data points. Overall, the slowest CPU time growth is shown across the datasets.

Figure 7 shows the effect of varying the distance threshold R through all the datasets, from 0–1000. The results are shown for slide size, $S = 500$ for the first two datasets and $S = 5K$ for Stock and Gauss. The other parameters are maintained as shown in Table 3. In each dataset, when the value of R is varied, it influences the outlier rate. For Figures 7(c) and 7(d), Thresh_LEAP incurs more time due to its trigger list, which makes it difficult to find neighbors. Overall, the CLODS showed better performance than the others and especially against MCMP since it has less distance computation when compared to MCMP that has to deal with strong and trivial inliers. The CLODS takes into consideration the relevance of K against each other rather than focusing on the influence of R. In Table 4, we notice that the outlier rate of R increases when default value of $R \leq 10\%$.

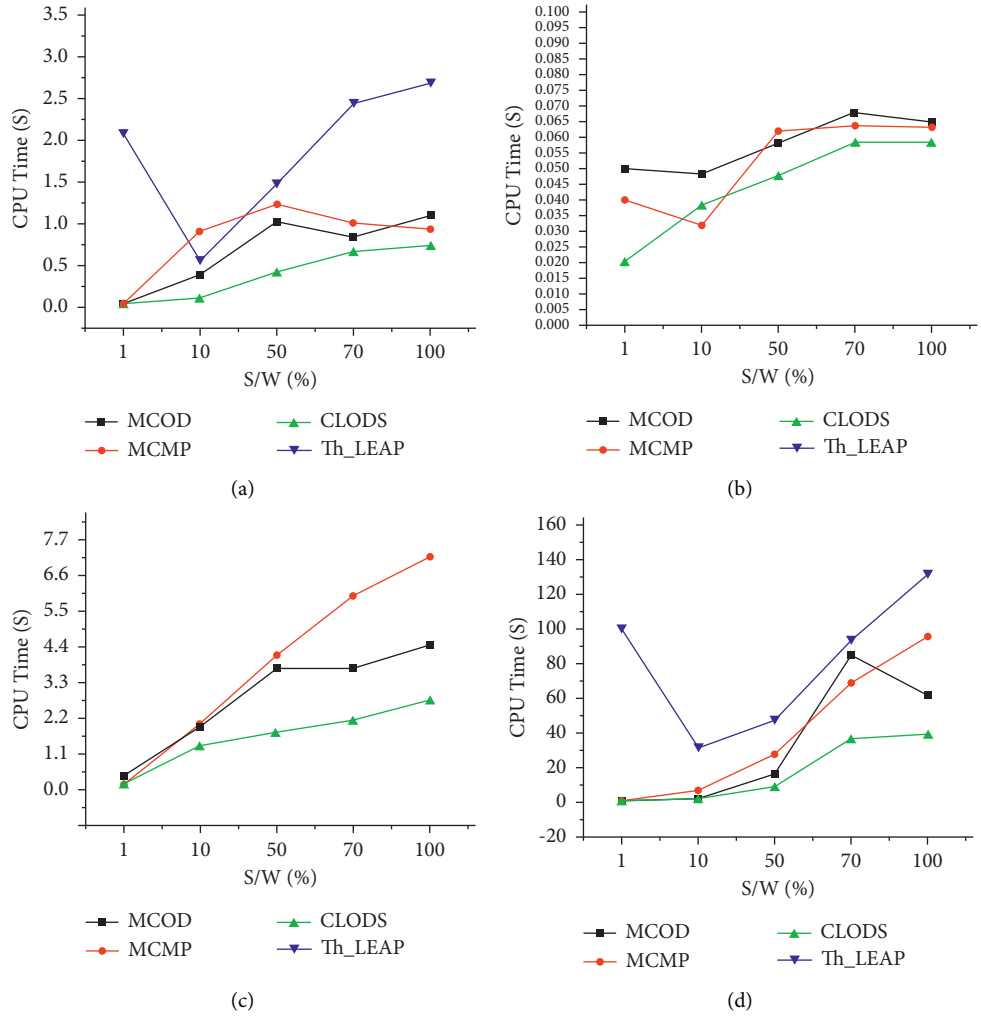

Figure 6: CPU time-varying S. (a) FC. (b) TAO. (c) Stock. (d) Gauss.

5.2.2. Memory Usage. In Figure 8, as the window sizes increase, it shows that more data points need to be processed, which result in an increase in memory usage for the majority of the datasets. More inliers will be in the microclusters, crucial inliers will be stored in temporary memory, and the objects' neighbors information will also be stored. From the figures, all the methods that make use of microclusters showed better performance across the datasets than Thresh_LEAP, which does not have the memory-efficient microcluster. Across all the datasets, it consumes more memory since its trigger list has to be redone every time the slides expire. In Figure 8(d), the Gauss datasets have few neighbors, and it shows an increase in memory usage for the various methods when compared to the other datasets, since finding the neighbors consumes the temporary memory. Our approach shows almost the same performance as MCOD since there is not much computation outside the microclusters like that in MCMP, which incurs slightly more memory. The CLODS, in the majority of cases, showed the least memory consumption due to freeing up space by deleting in memory detected outliers and queuing in temporary memory only significant inliers that are outside the microclusters.

When we vary the neighbor count threshold by increasing the value of k as shown in Figure 9, we expect more memory usage since k impacts the storing of the neighbors. For a few scenarios, it is almost stable, showing a small difference. For instance, in Figure 9(b), Thresh_LEAP difference does not exceed 1MB for $50\,\mathrm{dp} \le K \le 20\,\mathrm{dp}$, likewise for the other datasets in the same figure. The CLODS among the algorithms showed superior performance in most cases due to it is not entirely depending on K, as in the case of MCOD and MCMP. As K increases, more data points are not in microclusters, thereby occupying the temporary memory. For MCMP, the process of differentiating between the inliers utilizes some memory, while the CLODS only keeps a significant inlier in memory temporarily. One notable difference is in Figures 9(a) and 9(d) for Thresh_LEAP, which shows higher memory usage as compared to the others because of the neighbor count list that needs to be processed.

In Figure 10, when we vary the distance threshold R, there is no constant observable trend across the datasets. Overall, the CLODS together with the other algorithms does not make use of range queries; therefore, an increase in R does not result proportionally to an increase in memory usage. Initially, more memory is used for MCOD and

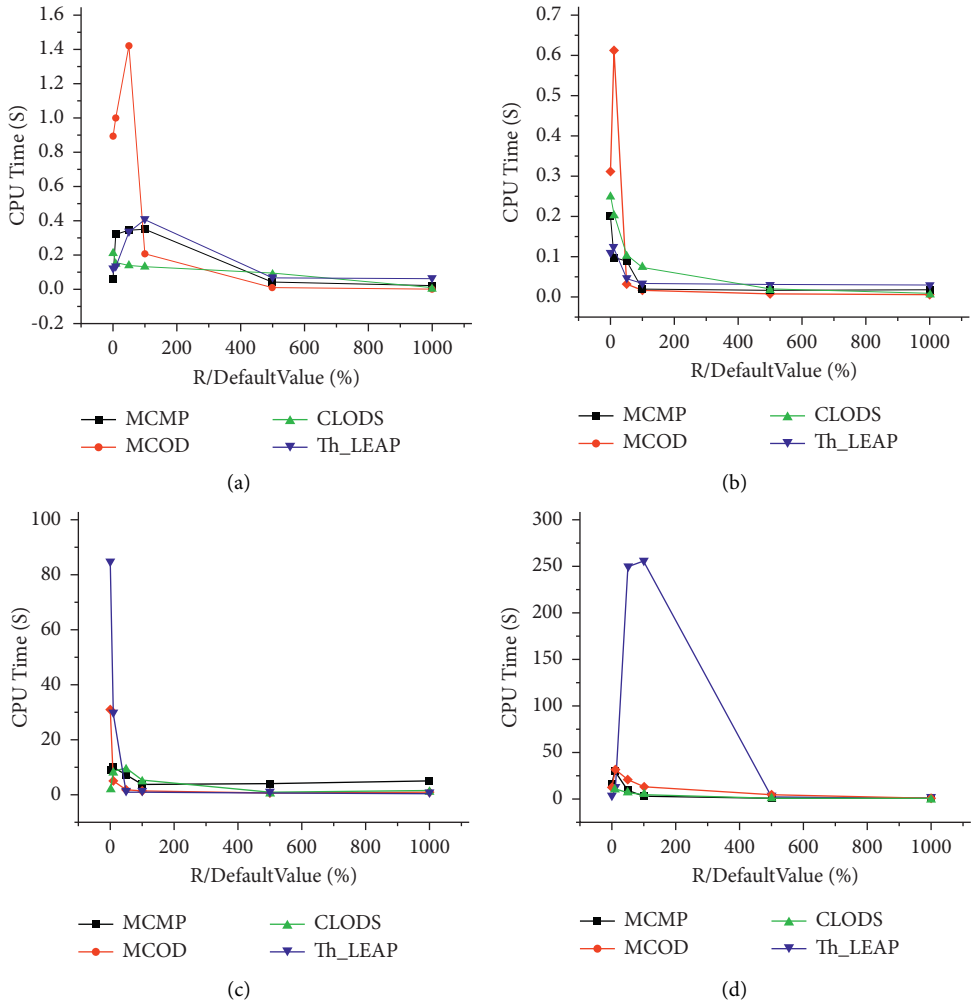

Figure 7: CPU time-varying R. (a) FC. (b) TAO. (c) Stock. (d) Gauss.

Table 4: The outlier rate-varying R

R/default_R (%)	FC (%)	TAO (%)	Stock (%)	Gauss (%)
1	100.0	99.3	44.97	98.9
10	99.8	49.5	6.03	32.3
50	9.90	3.10	2.10	3.00
70	7.80	1.10	2.01	1.60
200	0.93	0.72	0.97	0.85
500	0.00	0.01	0.15	0.20
700	0.00	0.10	0.11	0.20
1000	0.00	0.10	0.07	0.20

MCMP since not many data points can be found in microclusters, and the additional computation to find neighbors occupies the memory. The CLODS showed, in most cases, better performance to some degree since it does not differentiate every outliers or inlier as in the case of MCMP, so it uses less memory at the start. In most cases, the decline in memory usage is because an increase in the value of R translates to more neighbors, which result in more objects within the microclusters and fewer data points outside the microclusters. This thus curbs the memory utilization.

Figure 11 shows the result of the memory usage when S increases. Across the datasets, the CLODS showed a decline in peak memory usage as S increases, likewise the other algorithms. The Thresh_LEAP case shows unique performance, since it differs from the others in how it processes its data points. Thresh_LEAP at the onset has higher peak memory consumption and continues to reduce further. The other memory-efficient microcluster algorithm including CLODS showed less memory consumption since it does not make use of trigger list as in Thresh_LEAP. Thanks to their microclusters, the CLODS is slightly superior to that of

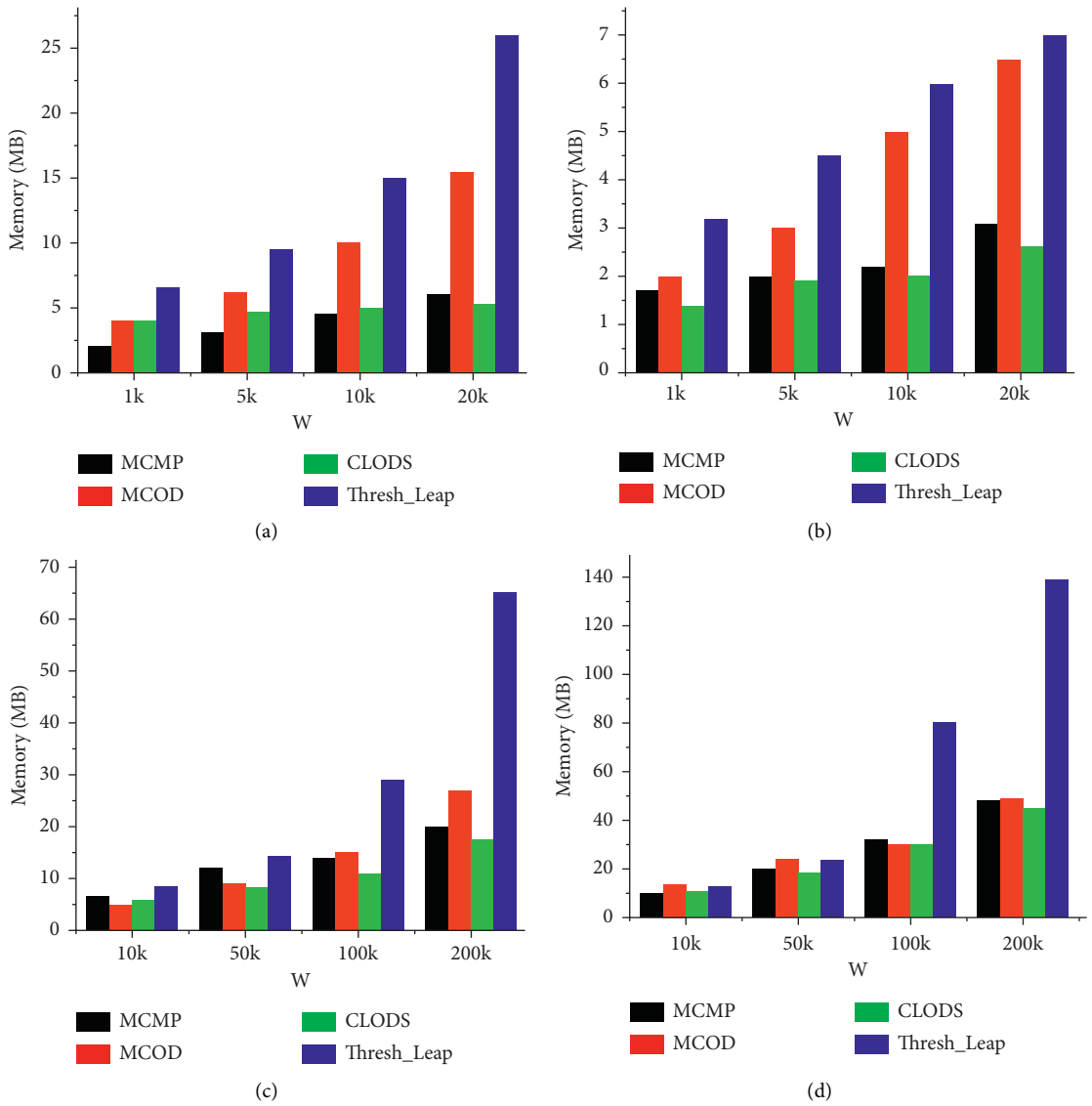

Figure 8: Memory-varying W. (a) FC. (b) TAO. (c) Stock. (d) Gauss.

MCMP in Figures 11(c) and 11(d), since storing the data points occupies the majority of the total memory. The absence of additional computation and to queue in memory trivial inliers gives it an advantage.

5.2.3. Space and Time Complexity. The complexity of the algorithm defines the running time and storage space needed by the algorithm in terms of its input size. The space complexity signifies the amount of memory space required by CLODS in its life cycle. To calculate the worst-case space required by CLODS, we take into consideration the space required to store the data and variables that are independent of the size of the problem. In Tables 5 and 6, we show the time and space complexity of the algorithms.

The time complexity in processing the data points within the current window in the worst-case scenario is the time cost of the function to discover whether the data point is in cluster or not, which is $O(1-c)W$ and in checking the relevance of the d_p with respect to their neighbors in the sliding window. Since we are considering the worst-case scenario, we take into consideration the cost of computing this, which incurs higher cost than the processing of the new data points within the slide. The overall cost in this case is the cost of the data point in the window by the window slide size, that is $O(1-c)W * 1/S$. When the data points expire, in the worst-case, the process of removing expired data points within the slide does not cost as much as when we need to check the relevance of these objects and adding the data point if in cluster. In this case, the overall cost is $O(W/S \log k)$. Therefore, the overall time complexity is $O((1-c)W + (1-c)W/S + W/S \log k)$ which can be approximated to $O(W/S((1-c) + \log k))$. The time complexity of CLODS is better than that of MCMP because in CLODS the cost of checking the relevance of the neighbors to their respective neighbors is less than that of MCMP cost,

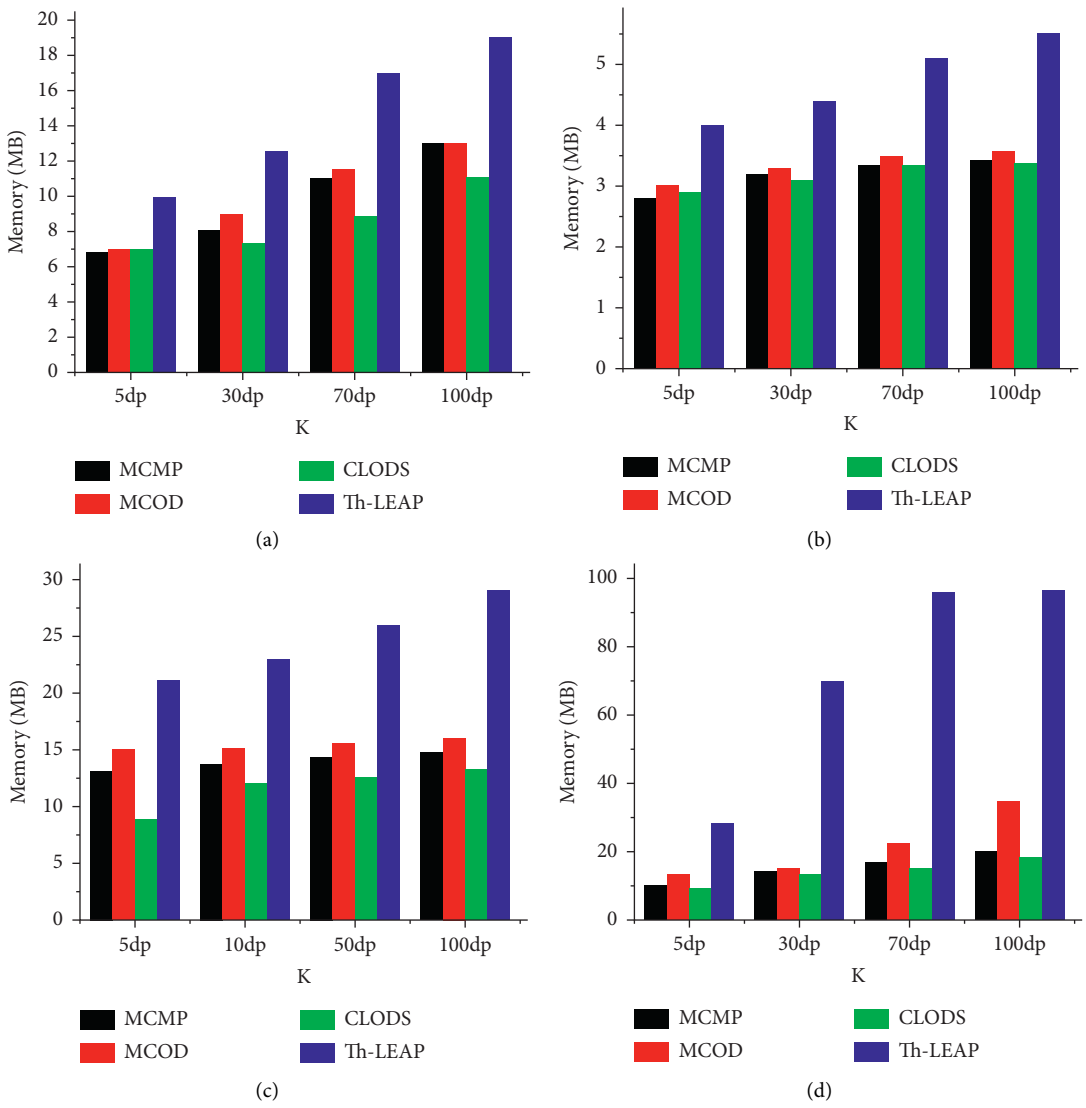

Figure 9: Memory-varying K. (a) FC. (b) TAO. (c) Stock. (d) Gauss.

which incurs additional cost due to the cost of differentiating between the strong and trivial inliers. We can see that the overall time complexity of MCMP is $O(W + W \log k + \log C_w + W \log C_w)$ which is approximately $O(W(\log C_w + \log k))$. This compared to the time complexity of the other two algorithms is almost the same as that of MCOD but superior to that of Thresh_LEAP. The reduction in the time complexity of MCMP confirms that microcluster using the concept of minimal probing by differentiating the strong and trivial inlier reduces the extra time required for computing data points outside the clusters as it minimizes the time complexity of recalculating and evaluating the all the inliers, as in the case of MCOD. Since differentiating between the inliers also incurs some amount of cost, however, this cost is less compared to the other way around.

In terms of the space complexity, a simple answer to the detection of continuous evolving outliers over streaming data in the window model will involve storing neighbors of each data object in the current window. It is apparent such computation in the worst-case will result in a quadratic space requirement $O(n^2)$; therefore, for larger window size w, it will be practically unfeasible. For each data point d_i, instead of keeping all the preceding d_p and succeeding neighbors d_s, we store a number of d_s neighbors and at most k data point will suffice to detect the outliers for specific R and K. The space complexity for managing data points within the current window is $O(kW)$. We first calculate the size of the preceding neighbors d_p that corresponds to the unexpired data points. When the size is less than $k - d_s$, then d_i is labeled as an outlier. When the window slides and expired, the space required to keep the neighbor counts is similar to that of MCMP, that is, $O(W/S)$ since each data point within the window is not stored in for each W/S slide. However, in CLODS with in-depth analysis, we could say that it will slight outperform MCMP since the space complexity needed in PD to store extra trivial inliers is less than that of saving relevant inliers in queue of the memory. The

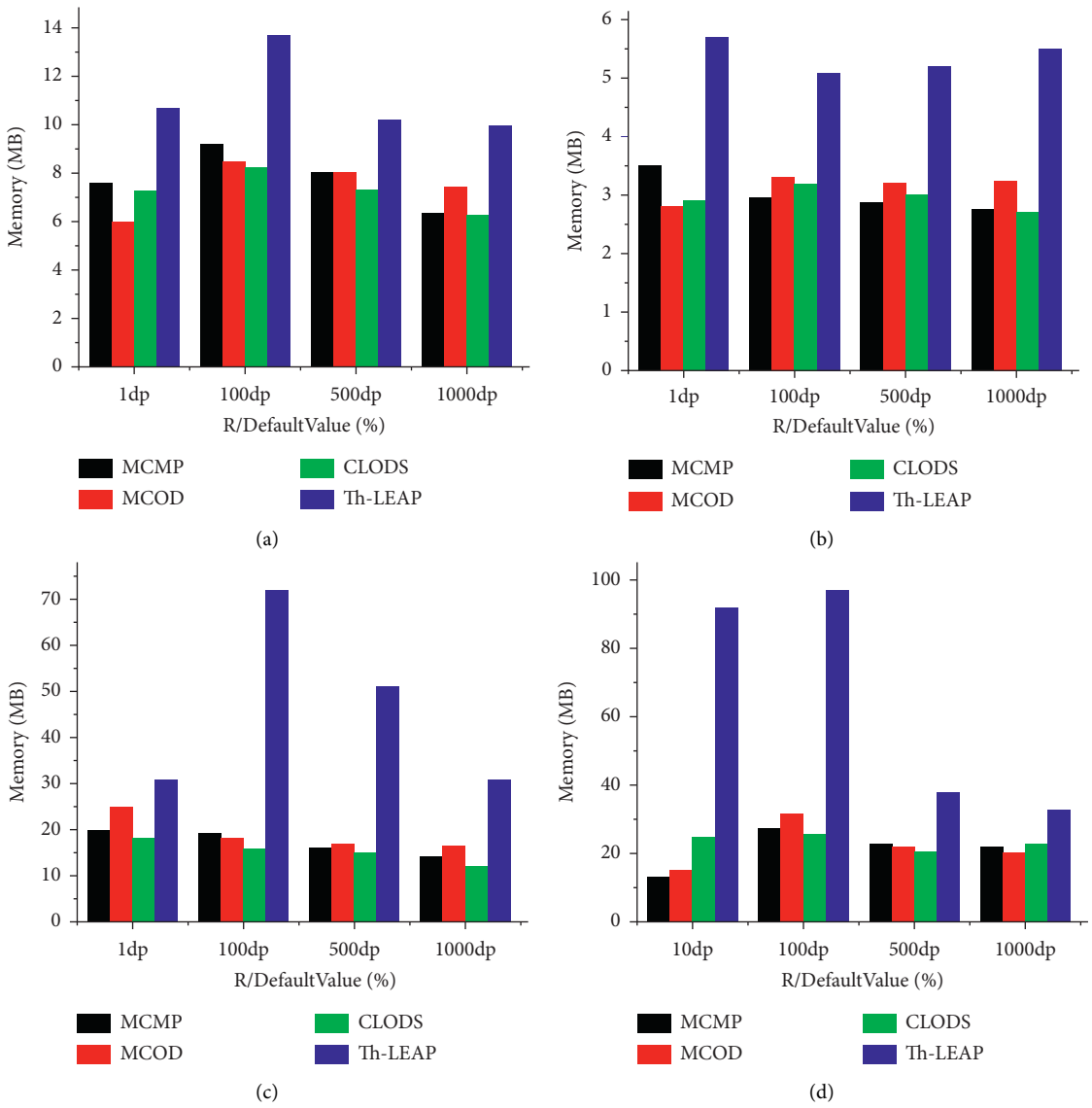

Figure 10: Memory-varying R. (a) FC. (b) TAO. (c) Stock. (d) Gauss.

overall worst space complexity of CLODS is $O(kW + W/S)$ which is almost the same as that of MCMP except that C_w in MCMP implies that during the expired window slides, the trivial inliers are stored in C, with $0 \leq c \leq 1$. Then, the number of data points within the window will be $(1 - c) * \text{Window}, W$. That is, the list of data point in PD will be $(1 - c) * W = C_w$. From Table 6, we can see that MCMP space complexity is also better than that of Thresh_LEAP and MCOD with $O(W^2/S)$ and $O(cW + (1c)kW)$, respectively. It is evident that the space needed for differentiating the inliers is negligible and better off compared to the space needed for data points outside microclusters to save the extra trivial inliers.

5.2.4. The Quality of Data Points in the Clusters. For clustering-based methods, an important metric to consider is the clustering quality, which affects the outlier detection rate in the data streams. Figure 12 shows the effectiveness and clustering quality of CLODS against previous methods that also adopted microclustering technique. For the FC dataset in Figure 12(a), the percentage of clusters is relatively low since the distance between each object is sparse. In another case, for the Gauss dataset, the percentage is almost zero, with little or no data points participating in the microclusters. This is because, in this particular window, the dataset has few neighbors. MCMP shows inferior clustering quality when compared to both MCOD and CLODS because of its extra distance-based computation that involves computing and storing the strong and trivial inliers. In some instances, it influences the neighbor count threshold k's relationship of the points outside the microclusters. The CLODS overall showed better clustering quality in almost all cases due to the absence of the extra computation that is involved in MCMP, and it ensures that clusters are generally formed on the basis of their relevance to their respective

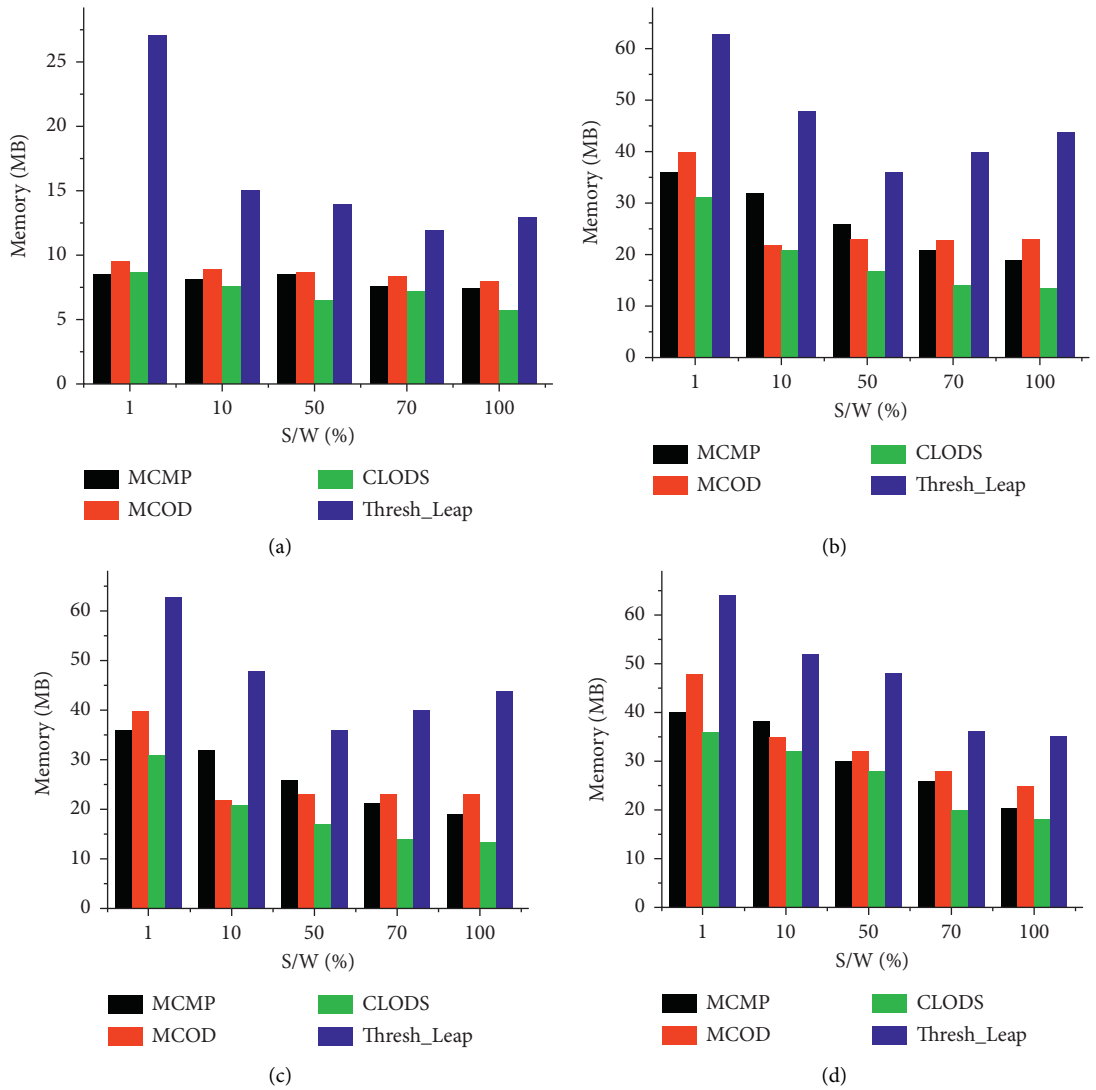

Figure 11: Memory-varying S. (a) FC. (b) TAO. (c) Stock. (d) Gauss.

Table 5: Time complexity analysis results.

Algorithms	Time complexity
Thresh_LEAP	$O(W^2 \log S/S)$
MCOD	$O((1-c)W \log((1-c)W) + kW \log K)$
MCMP	$O(W(\log Cw + \log k))$
CLODS	$O(W/S((1-c) + \log k))$

Table 6: Space complexity analysis results.

Algorithms	Space complexity
Thresh_LEAP	$O(W^2/S)$
MCOD	$O(cW + (1-c)kW)$
MCMP	$O(kC_w + W/S)$
CLODS	$O(kW + W/S)$

neighbors' position. This results in some cases of large percentage of data points discovered in the clusters, as can be seen in Figure 12 across the datasets.

5.2.5. Advantages of CLODS. CLODS through experiments has shown to outperform the existing methods in most cases and succeeded in curbing the computational cost in terms of the time taken and memory usage. It is a general solution used as a clustering-based outlier detection method for clustering evolving data streams based on microclusters and handling of objects within a sliding window according to the relevance of their status to their respective neighbors or position, excluding extended extra distance-based computation. The CLODS dynamically clusters data streams and offers support to meet flexible mining requirements. Furthermore, it has shown

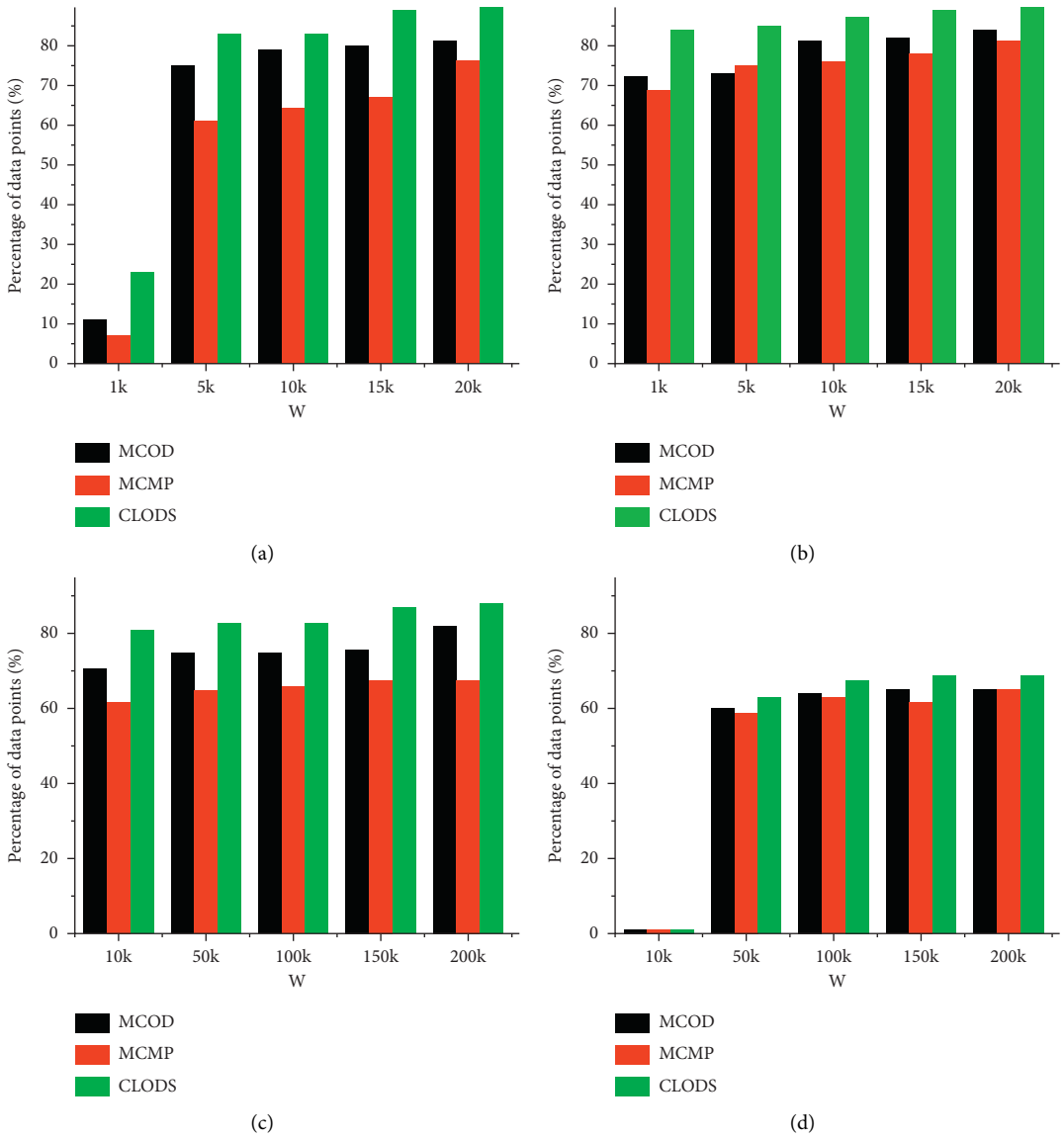

Figure 12: Comparison of the average percentage of data points in microclusters for MCOD, MCMP, and CLODS when we vary W. (a) FC. (b) TAO. (c) Stock. (d) FC.

robustness in the variation of the different performance parameters and its clustering quality with regard to the number of data points in its clusters. Finally, it has shown to be an effective method for detecting outliers.

6. Conclusion

Detecting outliers, which is the process of mining abnormal events from data, is a significant and challenging task. In this paper, we have proposed a clustering-based method called EMM-CLODS to address the problem of detecting outliers in continuous evolving data streams. The proposed method adopts the microcluster technique to group similar data points that are in proximity in the streaming data. It minimized the computational demand and showed an increase in the computational speed while it still maintained its effectiveness to detect outliers in the sliding window through minimal computation of data points outside the microclusters. It terms of its memory usage, not all objects outside the microclusters were stored in memory, and likewise, expired outlier data points were deleted from memory to minimize the memory usage. From the experiments performed on both real and synthetic datasets, our method showed effectiveness in detecting outliers for continuous evolving data streams. In the majority of the cases, it shows superior performance in terms of both CPU and memory utilization when compared to the other baseline algorithms. It has shown to be a good technique for detection outliers in data streams as it is robust to the various parameter variations (W, R, and K).

Data Availability

The data and source code used to support the findings of this study have not been made available. However, all the

datasets except for the source code used have been clearly explained in the experimental section with links of where to directly access these data. Previously reported (FC, TAO, Stock, and Gauss) data were used to support this study and are available at http://infolab.usc.edu/Luan/Outlier/. These prior studies (and datasets) are cited at relevant places within the text.

Disclosure

Mohamed Jaward Bah, Hongzhi Wang, Li-Hui Zhao, and Ji Zhang are co-first authors.

Conflicts of Interest

The authors declare that they have no conflicts of interest regarding this work.

Acknowledgments

The authors would like to thank the support from the Postdoctoral Fund of Hangzhou City (no. 119001-UB2101SJ), PI Research Project of Zhejiang Lab (no. 111007-PI2001), Natural Science Foundation of China (no. 62172372 and no. U1866602), and Zhejiang Provincial Natural Science Foundation (no. LZ21F030001).

References

[1] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Computing Surveys (CSUR)*, vol. 41, pp. 1–58, 2009.

[2] X. Su and C. L. Tsai, "Outlier detection," *WIREs Data Mining and Knowledge Discovery*, vol. 1, no. 3, pp. 261–268, 2011.

[3] Ji Zhang, "Advancements of outlier detection: a survey," *ICST Transactions on Scalable Information Systems*, vol. 13, pp. 1–26, 2013.

[4] L. Cao, Di Yang, Q. Wang, Y. Yu, J. Wang, and E. A. Rundensteiner, "Scalable distance-based outlier detection over high-volume data streams," in *Proceedings of the 2014 IEEE 30th International Conference on Data Engineering*, pp. 76–87, IEEE, Chicago, IL, USA, April 2014.

[5] S. Guha, M. Adam, N. Mishra, R. Motwani, and L. O'Callaghan, "Clustering data streams: theory and practice," *IEEE Transactions on Knowledge and Data Engineering*, vol. 15, pp. 515–528, 2003.

[6] Y. Chen and Li Tu, "Density-based clustering for real-time stream data," in *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 133–142, San Jose, CA, USA, August 2007.

[7] S. Hettich and S. D. Bay, *The UCI KDD Archive Irvine*, Department of Information and Computer Science, University of California, Irvine, CA, USA, 1999, http://kdd.ics.uci.edu.

[8] M. J. Bah, H. Wang, H. Mohamed, F. Zeshan, and H. Aljuaid, "An effective minimal probing approach with micro-cluster for distance-based outlier detection in data streams," *IEEE Access*, vol. 7, pp. 154922–154934, 2019.

[9] L. Cao, J. Wang, and E. A. Rundensteiner, "Sharing-aware outlier analytics over high-volume data streams," in *Proceedings of the 2016 International Conference on Management of Data*, pp. 527–540, San Francisco, CA, USA, July 2016.

[10] J. Tamboli and M. Shukla, "A survey of outlier detection algorithms for data streams," in *Proceedings of the 2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*, pp. 3535–3540, IEEE, New Delhi, India, March 2016.

[11] H. Wang, M. J. Bah, and M. Hammad, "Progress in outlier detection techniques: a survey," *Ieee Access*, vol. 7, pp. 107964–108000, 2019.

[12] C. C. Aggarwal, P. S. Yu, J. Han, and J. Wang, "A framework for clustering evolving data streams," in *Proceedings 2003 VLDB Conference*, pp. 81–92, Berlin, Germany, September 2003.

[13] P. Caroline Cynthia and S. Thomas George, "An outlier detection approach on credit card fraud detection using machine learning: a comparative analysis on supervised and unsupervised learning," in *Intelligence in Big Data Technologies—Beyond the Hype*, J. Dinesh Peter, S. L. Fernandes, and A. H. Alavi, Eds., Springer Singapore, Singapore, pp. 125–135, 2021.

[14] M. E. Villa-Pérez, M. Á. Álvarez-Carmona, O. Loyola-González, M. A. Medina-Pérez, J. C. Velazco-Rossell, and K.-K. R. Choo, "Semi-supervised anomaly detection algorithms: a comparative summary and future research directions," *Knowledge-Based Systems*, vol. 218, Article ID 106878, 2021.

[15] F. Liu, S. Xue, J. Wu et al., "Deep learning for community detection: progress, challenges and opportunities," in *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence*, Yokohama, Japan, July 2020.

[16] X. Su, S. Xue, F. Liu et al., "A comprehensive survey on community detection with deep learning," 2021.

[17] A. Boukerche, L. Zheng, and A. Omar, "Outlier detection: methods, models, and classification," *ACM Computing Surveys*, vol. 53, no. 3, 2020.

[18] X. Ma, J. Wu, S. Xue, J. Yang, Z. S. Quan, and H. Xiong, "A comprehensive survey on graph anomaly detection with deep learning," 2021, http://arxiv.org/abs/2106.07178.

[19] G. Pang, C. Shen, L. Cao, and A. Van Den Hengel, "Deep learning for anomaly detection: a review," *ACM Computing Surveys (CSUR)*, vol. 54, pp. 1–38, 2021.

[20] D. Toshniwal and Yokita, "A framework for outlier detection in evolving data streams by weighting attributes in clustering," *Procedia Technology*, vol. 6, no. 2012, pp. 214–222, 2012.

[21] A. Zhou, F. Cao, W. Qian, and C. Jin, "Tracking clusters in evolving data streams over sliding windows," *Knowledge and Information Systems*, vol. 15, no. 2, pp. 181–214, 2008.

[22] F. Cao, M. Estert, W. Qian, and A. Zhou, "Density-based clustering over an evolving data stream with noise," in *Proceedings of the 2006 SIAM International Conference on Data Mining*, pp. 328–339, SIAM, Bethesda, MD, USA, April 2006.

[23] L.-x. Liu, Y.-f. Guo, J. Kang, and H. Huang, "A three-step clustering algorithm over an evolving data stream," in *Proceedings of the 2009 IEEE International Conference on Intelligent Computing and Intelligent Systems*, pp. 160–164, IEEE, Shanghai, China, November 2009.

[24] M. Kumar and A. Sharma, "Mining of data stream using "DDenStream" clustering algorithm," in *Proceedings of the 2013 IEEE International Conference in MOOC, Innovation and Technology in Education (MITE)*, pp. 315–320, IEEE, Jaipur, India, December 2013.

[25] A. Amini and T. Y. Wah, "A comparative study of density-based clustering algorithms on data streams: micro-clustering approaches," in *Intelligent Control and Innovative Computing*, pp. 275–287, Springer, Berlin, Germany, 2012.

[26] A. Amini, T. Y. Wah, and Y. W. Teh, "DENGRIS-Stream: A density-grid based clustering algorithm for evolving data

streams over sliding window," in *Proceedings of the International Conference on Data Mining and Computer Engineering*, pp. 206–210, Visakhapatnam, India, January 2012.

[27] L. Duan, L. Xu, Y. Liu, and J. Lee, "Cluster-based outlier detection," *Annals of Operations Research*, vol. 168, pp. 151–168, 2009.

[28] M. Elahi, K. Li, W. Nisar, X. Lv, and H. Wang, "Efficient clustering-based outlier detection algorithm for dynamic data stream," in *Proceedings of the 2008 Fifth International Conference on Fuzzy Systems and Knowledge Discovery*, pp. 298–304, IEEE, October 2008, Jinan, China.

[29] A. Forestiero, C. Pizzuti, and G. Spezzano, "A single pass algorithm for clustering evolving data streams based on swarm intelligence," *Data Mining and Knowledge Discovery*, vol. 26, no. 1, pp. 1–26, 2013.

[30] M. S. Sadik and L. Gruenwald, "DBOD-DS: distance based outlier detection for data streams," in *International Conference on Database and Expert Systems Applications*, pp. 122–136, Springer, Berlin, Germany, 2010.

[31] M. B. Al-Zoubi, "An effective clustering-based approach for outlier detection," *European Journal of Scientific Research*, vol. 28, no. 2, pp. 310–316, 2009.

[32] L. Tran, L. Fan, and C. Shahabi, *Distance-Based Outlier Detection in Data Streams Repository*, Information Laboratory University of Southern California, Los Angeles, LA, USA.

[33] Pacific Marine Environmental Laboratory. 2019. Wharton University of Pennsylvania. https://infolab.usc.edu/Luan/Outlier/Datasets/tao.txt.

[34] Wharton Research Data Services, *Distance-Based Outlier Detection in Data Streams Repository*, Wharton Research Data Services, Philadelphia, PA, USA, 2020, https://wrds-web.wharton.upenn.edu/wrds/.

[35] M. Kontaki, A. Gounaris, A. N. Papadopoulos, K. Tsichlas, and Y. Manolopoulos, "Continuous monitoring of distance-based outliers over data streams," in *Proceedings of the 2011 IEEE 27th International Conference on Data Engineering*, pp. 135–146, IEEE, Hannover, Germany, April 2011.

[36] M. Shukla, Y. P. Kosta, and P. Chauhan, "Analysis and evaluation of outlier detection algorithms in data streams," in *Proceedings of the 2015 International Conference on Computer, Communication and Control (IC4)*, pp. 1–8, IEEE, Indore, India, September 2015.