Fast Distance-based Outlier Detection in Data Streams based on Micro-clusters

Luan Tran
luantran@usc.edu
University of Southern California
Los Angeles, CA, USA

Liyue Fan
liyuefann@albany.edu
University at Albany, SUNY
SUNY, New York, USA

Cyrus Shahabi
shahabi@usc.edu
University of Southern California
Los Angeles, CA, USA

ABSTRACT

Continuous outlier detection in data streams is one important topic in data mining. It has many applications in public health, network intrusion detection, and fraud detection. Over the last two decades of research, many studies have been conducted on distance-based outlier detection algorithms which are viable, scalable, and parameter-free approaches. Because streaming data points arrive and expire over time, the challenge is to monitor the outlier status of data points with time and space efficiency. In this study, we propose three algorithms: O-MCOD, U-MCOD, and M-MCOD. These algorithms improve upon the state-of-the-art algorithm in distance-based outlier detection in data streams, i.e., MCOD, by relaxing the constraints of micro-clusters and using the minimal probing principal. With extensive experiments on synthetic and real-world datasets, we show that the proposed algorithms are superior in time and space efficiency. Specially, our proposed algorithms are 1.5 to 95 times faster than MCOD, require as low as 25% peak memory compared to MCOD, and outperform the most recent algorithm NETS.

CCS CONCEPTS

• **Information systems** → **Data streams**; **Data mining**.

KEYWORDS

distance-based, outlier detection

ACM Reference Format:
Luan Tran, Liyue Fan, and Cyrus Shahabi. 2019. Fast Distance-based Outlier Detection in Data Streams based on Micro-clusters. In *The Tenth International Symposium on Information and Communication Technology (SoICT 2019), December 4–6, 2019, Hanoi - Ha Long Bay, Viet Nam.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3368926.3369667

1 INTRODUCTION

Outlier detection in data streams [1] is an important task in data mining. It has many applications in several domains such as fraud detection, computer network security, and medical and public health anomaly detection. An outlier is a data object that does not conform to the expected behavior. In this study, we focus on detecting *distance-based* outliers. A data object o in a generic metric space is a distance-based outlier if it has less than k objects located within distance R from o.

SoICT 2019, December 4–6, 2019, Hanoi - Ha Long Bay, Viet Nam

ACM ISBN 978-1-4503-7245-9/19/12…$15.00
https://doi.org/10.1145/3368926.3369667

In data streams, the size of the dataset is potentially unbounded. Therefore, outlier detection is conducted in every *sliding window*, i.e., a number of active data objects, to ensure computation efficiency and outlier detection in a local context. A number of studies [2–4, 7–9, 11] have been performed for Distance-based Outlier Detection in Data Streams (DODDS). In our previous work [10], five recent algorithms for distance-based outlier detection in data streams are compared and MCOD [7] is shown to be the best algorithm in space and time efficiency. MCOD utilizes a data structure called micro-cluster to efficiently reduce the number of pair-wise distance computations and neighbor information storage. A micro-cluster contains at least $k + 1$ data points located within the distance $R/2$ from the center data point. However, when the number of micro-clusters is small, for example, R is too small or k is too large, MCOD has to perform linear searches for finding neighbors and it is time consuming. In addition, a micro-cluster can be dispersed when some of its members expire and the remaining data points have to be re-processed as new data points.

Motivated by those observations, we propose three novel algorithms that overcome those shortcomings: O-MCOD which allows one data points belong to more than one micro-cluster, U-MCOD which relaxes the constraints of a micro-cluster, and M-MCOD which reduces the number of dispersed micro-cluster. Furthermore, we also improve the memory consumption by storing the counts of neighbors in every slide instead of a neighbor list for each data point which is not in any micro-clusters. With extensive experiments with synthetic and real-world datasets, we show that our proposed algorithms outperform MCOD and NETS, i.e., the state-of-the-art algorithms, in time and space efficiency.

This paper is structured as follows. In Section 2, we review the related work. In Section 3, we present definitions and notions that are used in DODDS. In Section 4, we introduce our proposed algorithms. In Section 5, we provide our detailed evaluation results. Finally, we conclude the paper with discussions and future research directions in Section 6.

2 RELATED WORK

Distance-based outlier detection was first studied for static datasets [5]. Several approaches for detecting distance-based outliers in data streams have been proposed in [2, 3, 7, 11]. For each data point, Exact-Storm in [2] stores up to k preceding neighbors and the number of succeeding neighbors. Abstract-C in [11] keeps the neighbor count of each data point in every window it participates in. DUE in [7] employs a priority queue called *event queue* stores all the data points whose outlier status can be affected by expiring slides. Exact-Storm, Abstract-C and DUE use an index structure to support range queries for neighbor search. Thresh_LEAP in [3] mitigates the expensive range queries by employing a separate index for each slide. It also follows *minimal probing principal* which searches for neighbors in most recent slide first, then older slides.

Micro-cluster Based Algorithm (MCOD) in [7] employs micro-clusters for eliminating the need for range queries. A more detailed explanation of MCOD is presented in Section 3.2. In our previous work [10], we showed that MCOD is the fastest algorithm and consumes the least memory in the various settings of window size W, slide size S, distance radius R, neighbor threshold k, and the number of data dimensions.

NETS in [12] groups the data points at similar locations into cells and the outlier detection is conducted at cell level first then point level. It takes advantage of the similarity between expired and new data points for reducing the number of updates.

3 PRELIMINARIES

3.1 Problem Definition

In this section, the formal definitions that are used in DODDS are presented. We first define the concepts in data streams and then outlier detection definitions.

Definition 3.1 (Data Stream). [10] A data stream is a possible infinite series of data points $..., o_{n-2}, o_{n-1}, o_n, ...$, where data point o_n is received at time $o_n.t$.

In this definition, the data points are ordered by the timestamp $o.t$ at which it arrives. Since the size of data streams is potentially unbounded, data streams are typically processed in a *sliding window*, i.e., a set of active data points. In this study, we adopt *count-based window* model in data streams similarly to the previous works [2, 3, 7, 10] which is defined as follows.

Definition 3.2 (Count-based Window). [10]
Given data point o_n and a fixed window size W, the count-based window D_n is the set of W data points: $o_{n-W+1}, o_{n-W+2}, ..., o_n$.

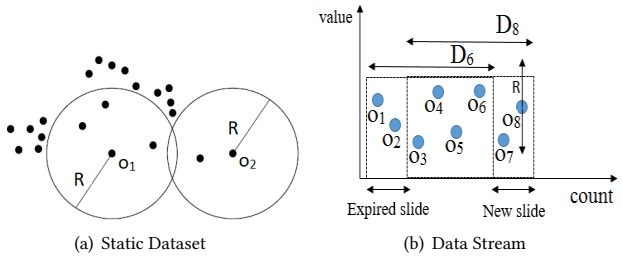
Given the window size W, all count-based windows have the same number of data points. Therefore, we can control the volume of data streams and evaluate fairly the *scalability* of algorithms. In the rest of the paper, we use the term *window* to refer to the *count-based window*. Every time the window slides, the oldest S data points are discarded and new S data points are incorporated to the window. The slide size S characterizes the speed of the data streams. Figure 1(b) in [10] shows an example of two consecutive windows with $W = 8$ and $S = 2$. The x-axis reports the arrival time of data points and the y-axis reports the data values. When two data points $\{o_7, o_8\}$ arrive in a new slide, the window D_6 slides, two data points $\{o_1, o_2\}$ expire and will be discarded.

Definition 3.3 (Neighbor). [10] Given a distance threshold R ($R > 0$), two data points o and o' are neighbors of each other if the distance between them is not greater than R. A data point is not considered a neighbor of itself.

We assume that the distance function between two data points is defined in the metric space.

Definition 3.4 (Distance-based Outlier). [10] Given a dataset D, a count threshold k ($k > 0$) and a distance threshold R ($R > 0$), a distance-based outlier in D is a data point that has less than k neighbors in D.

A data point that has at least k neighbors is called an inlier. Figure 1(a) shows an example of a dataset from [7, 10] that has two outliers with $k = 4$. Two data points o_1, o_2 are outliers since they have 3 and 1 neighbors, respectively.



(a) Static Dataset (b) Data Stream

Figure 1: Static and Streaming Outlier Detection [10]

As the data points in a stream are ordered by the arrival time, it is important to distinguish between the following two concepts: *preceding neighbor* and *succeeding neighbor*. A data point o is a *preceding neighbor* of a data point o' if o is a neighbor of o' and expires before o' does. On the other hand, a data point o is a *succeeding neighbor* of a data point o' if o is a neighbor of o' and o expires in the same slide with or after o'. For example, in Figure 1(b) from [10], o_8 has one succeeding neighbor o_7 and four preceding neighbors, i.e., o_3, o_4, o_5, o_6. Note that an inlier which has at least k succeeding neighbors will never become an outlier in the future. Those inliers are thus called *safe inliers*. On the other hand, the inliers which have less than k succeeding neighbors are *unsafe inliers*, as they may become outliers when the preceding neighbors expire.

The Distance-based Outlier Detection in Data Streams (DODDS) is defined as follows.

PROBLEM 1 (DODDS). *[10] Given the window size W, the slide size S, the count threshold k, and the distance threshold R, detect the distance-based outliers in every sliding window $..., D_n, D_{n+S}, ...$*

The challenge of DODDS is that the outlier status of one data point can change when the window slides. Expired neighbors from the expired slide can cause one data point from an inlier to be an outlier. New neighbors from the new slide can cause one data point from an outlier to be an inlier. Figure 2 from [2] illustrates how the sliding window affects the outlierness of the data points. The two diagrams represent the evolution of a data stream of 1-dimensional data points. The x-axis reports the time of arrival of the data points and the y-axis reports the value of each data point. With $k = 3, W = 7$ and $S = 5$, two consecutive windows D_7 and D_{12} are depicted by dash rectangles. In D_7, o_7 is an inlier as it has 4 neighbors, i.e., o_2, o_3, o_4, o_5. In D_{12}, o_7 becomes an outlier because o_2, o_3, o_4, o_5 expired.

A simple solution is to store the neighbors of every data point and recompute the neighbors when the window slides, which is computationally expensive. Another approach is to store the neighbors of each data point that can prove it an inlier or outlier. When the window slides, the neighbor information of the data points which have at least one expired neighbor is updated. The neighbor information of a data point can be stored either as a list of neighbors or by the number of neighbors. If only the number of neighbors is stored, an other data structure is needed to find data points that have their neighbors expired. The incremental exact-algorithms differs from each other one based on the data structure for storing and the mechanism for updating neighbor information.

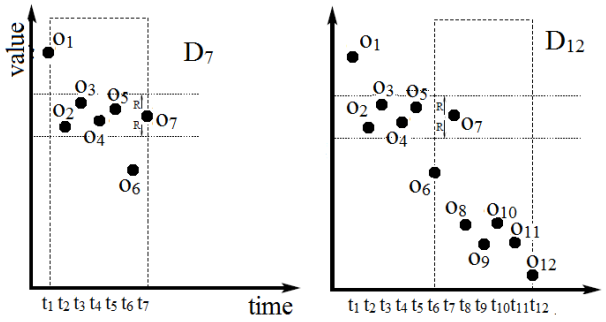


Figure 2: Example of DODDS from [2] with $k = 3$, $W = 7$, and $S = 5$

3.2 Micro-Cluster Based Outlier Detection - MCOD

Finding neighbors for every data point in a large data set can be expensive. MCOD [7] stores the neighboring data points in micro-clusters to eliminate the need for range queries. A micro-cluster contains at least $k + 1$ data points. In every micro-cluster, one data point is its center and all other data points are located within the distance of $R/2$ from the center. According to the triangular inequality in the metric space, the distance between every pair of data points in a micro-cluster is no greater than R. Therefore, every data point in a micro-cluster is an inlier. Figure 3 shows an example of three micro-clusters, and data points in each micro-cluster are represented by different symbols. The data points that are not in any micro-clusters can be either outliers or inliers, e.g., having neighbors from different micro-clusters. Such data points are stored in a list called *PD*.

Expired slide processing. When the current window slides, the expired data points are removed from micro-clusters and *PD*. The *PD* list is polled to update the unsafe inliers which are data points having less than k succeeding neighbors. If a micro-cluster has less than $k + 1$ data points, it is dispersed and the non-expired members are processed as new data points.

New slide processing. For each data point o, it can be added to an existing micro-cluster, become the center of its own micro-cluster, or added to *PD*. If o is within distance $R/2$ to the center of a micro-cluster, o is added to the closest micro-cluster. Otherwise, MCOD searches in *PD* for $o's$ neighbors within distance $R/2$. If at least k neighbors are found in *PD*, these neighbors and o form a new micro-cluster with o as the micro-cluster center. Otherwise, o is added to *PD*.

Outlier reporting. After the new slide and expired slide are processed, the data points in *PD* that have less than k neighbors are reported as outliers.

One advantage of MCOD is that it effectively prunes the pairwise distance computations for each data point's neighbor search, utilizing the micro-clusters centers. The memory requirement is also lowered as one micro-cluster can efficiently capture the neighborhood information for each data point in the same micro-cluster.

3.3 Evaluation Metrics

In this study, we adopt CPU running time and peak memory requirement which are two important utility metrics for streaming algorithms for performance comparison, as in [2, 3, 7, 10]. The CPU running time includes the processing time for the new slide, expired slide and outlier detection for each window. The peak memory requirement measures the highest memory used by a DODDS algorithm including the data storage as well as the algorithm-specific structures to incrementally update neighborhood information.

4 DODDS ALGORITHMS

In this section, we introduce our three proposed algorithms. The general procedure for outlier detection in data streams consists of three main tasks, i.e., processing expired slides, processing new slides and reporting outliers, as in Algorithm 1.

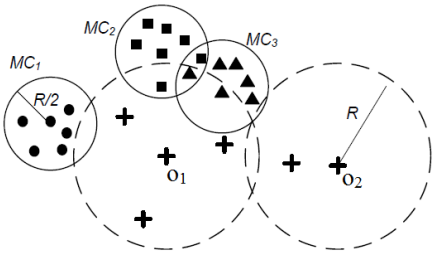


Figure 3: Example micro-clusters with $k = 4$ [7]

4.1 Overlapping Micro-cluster Based Algorithm - O-MCOD

In MCOD, each data point belongs to at most 1 micro-cluster. To improve the pruning ability of micro-clusters, we propose O-MCOD to allow that one data point can belong to more than one micro-clusters, specifically at most *max_cluster* micro-clusters, *max_cluster* > 1. The pseudo code of O-MCOD is presented in Algorithm 2. For each data point, we use a list to store the references of the micro-clusters it participates in. For each new data point, the process of finding micro-clusters only stops when it finds enough *max_cluster* micro-clusters. One advantage of O-MCOD is that more micro-clusters can be formed due to the overlapping property; as a result, more pruning can be achieved in the search for neighbors of new data points. Figure 4 illustrates an example of 4 overlapping micro-clusters. There are 3 data points represented by triangle, square, and diamond in MC4 belong to 2 micro-clusters simultaneously. The data points which are not in micro-clusters are stored in *PD* list. A data point in *PD* stores a hash map of slide indexes and the corresponding number of neighbors in those slides instead of a neighbor list.

Expired slide processing. O-MCOD processes expired data points in *PD* similarly to MCOD. For each data point in micro-clusters, because it can participate in more than one micro-clusters, only when all of its micro-clusters disperse, O-MCOD re-processes it as a new data point.

New slide processing. For each data point o from the new slide, O-MCOD finds at most *max_cluster* micro-clusters to add o. If there are more than *max_cluster* micro-clusters are found, the closest *max_cluster* micro-clusters are chosen to add o. Otherwise, o is added to the *PD* similarly to MCOD.

Outlier reporting. After the new slide and expired slide are processed, the data points in *PD* that have less than k neighbors are reported as outliers.

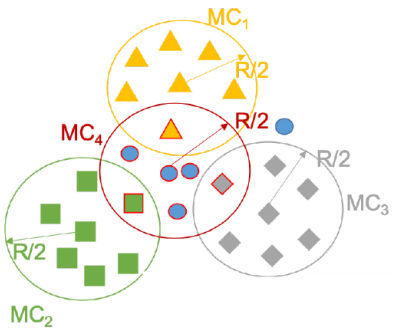


Figure 4: Example overlapping micro-clusters with $k = 6, max\_cluster = 2$

The advantage of O-MCOD compared to MCOD is that it allows overlapping micro-clusters, hence there are more data points in micro-clusters. Therefore O-MCOD can utilizes better the advantage of micro-clusters in neighbor finding.

Algorithm 1 General Outlier Detection Algorithm

Input: Stream $\{o\}$, Window Size W, Slide Size S, Distance threshold R, Neighbor threshold k
Output: Outliers in every sliding window
Procedure:

```
1: while S_new do                                   ▷ A new slide arrives
2:     ProcessExpiredData(S_expired)
3:     ProcessNewData(S_new)
4:     ReportOutlier()
```

4.2 Unsafe Micro-cluster Based Algorithm - U-MCOD

Micro-clusters are used for quickly finding neighbors. However, if the condition of micro-cluster is too strict, e.g., k can be too high and R can be too small, the number of micro-clusters is small. In the worst case, if there is no micro-clusters, for every data point, MCOD has to run a linear search for neighbors. This is very inefficient. Motivated by this, we propose U-MCOD with new concepts of *unsafe micro-cluster* and *safe micro-cluster*. The pseudo codes of U-MCOD for processing expired and new slides are presented in Algorithm 3. A *safe micro-cluster* is a micro-cluster that has at least $k + 1$ data points and an *unsafe micro-cluster* is a micro-cluster with fewer than $k + 1$ data points. All data points in a *safe micro-cluster* are inliers. The data points in *unsafe micro-clusters* can be either outliers or inliers. Therefore, for each data point in an *unsafe micro-cluster*, we create a hash map of slide indexes and the corresponding number of neighbors in those slides. Note that there is no *PD* list in U-MCOD. Figure 5 shows an example of two *unsafe micro-clusters* with 4 and 5 data points and one *safe micro-cluster*. Both *safe micro-clusters* and *unsafe micro-clusters* are used in finding neighbors for each data point. Specifically, with a data point o and a micro-cluster centering at C. If the distance from o to C is not larger than $R/2$, all the data points in the micro-cluster C are neighbors of o. If the distance from o to C is larger than $3R/2$, all the data points in the micro-cluster are not neighbors of o. In both cases, we do not have to compute explicitly the pair wise distances between o and the data points in the micro-cluster. When the distance between o and C is between $R/2$ and $3R/2$, we must compute the pair wise distance between o and each data point in C to find neighbors for o. Intuitively, the number of pair wise distance computations is linear to the size of each micro-cluster (*safe* or *unsafe*).

Algorithm 2 Overlapping Micro-Cluster Based Algorithm

Global variables: Micro-cluster list *microClusters*, PD list *PD*, Maximum number of micro-clusters for each data point *max_cluster*

```
1: function ProcessExpiredData(S_expired)
2:     re_prob = set()
3:     for o in S_expired do
4:         if o.cluster is not None then          ▷ o is in a micro-cluster
5:             cs = o.cluster_list
6:             for c in cs do
7:                 c.remove(o)                    ▷ Remove o from micro-cluster
8:                 if c.num_neighbors < k + 1 then
9:                     re_prob.add(c.non_expired_members())
10:        else
11:            for s in o.succeeding_neighbors do
12:                s.remove_neighbors(o)
13:    ProcessNewData(re_prob)
```

```
1: function ProcessNewData(S_new)
2:     for o in S_new do
3:         cs =FindClusters(o, max_cluster)
4:         if cs is not None then
5:             for c in cs do
6:                 c.add(o)
7:         else
8:             neighbors = FindNeighbor(o)
9:             if CanFormCluster(o, neighbors) then
10:                FormCluster(o, neighbors)
11:            else AddToPD(o, PD)
```

```
1: function ReportOutlier(PD)
2:     outliers = []
3:     for o in PD do
4:         if o.num_neighbors < k then
5:             outliers.add(o)
       return outliers
```

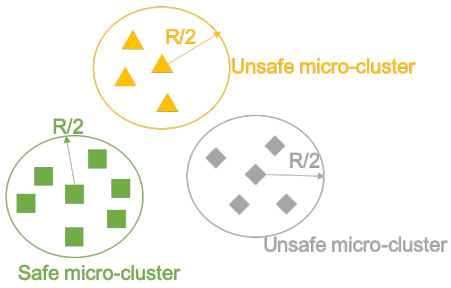


Figure 5: Example unsafe micro-clusters with $k = 6$

Expired slide processing. The data points in the expired slide are removed from micro-clusters. If a *safe micro-cluster* becomes an *unsafe micro-cluster*, its non-expired data points are re-probed for neighbor information as new data points.
New slide processing. For each data point o from the new slide, U-MCOD first finds a *safe micro-cluster* to add o. If o cannot be added to any *safe micro-cluster*, U-MCOD searches for any *unsafe micro-cluster* for o. Otherwise, a new *unsafe micro-cluster* is formed with o is the center.
Outlier reporting. The data points in *unsafe micro-cluster* are evaluated for reporting outliers.

The advantage of U-MCOD compared to MCOD is that it utilizes *unsafe-clusters* for quicker neighbor finding compared to *PD* in MCOD.

Algorithm 3 Unsafe Micro-Cluster Based Algorithm

```
1:  function ProcessExpiredData(S_expired)
2:      re_prob = set()
3:      for o in S_expired do
4:          c = o.cluster
5:          if c is safe then
6:              c.remove(o)
7:              if c.num_members == k then
    re_prob.add(c.non_expired_members())
8:          else
9:              for s in o.succeeding_neighbors do
10:                 s.remove_neighbors(o)
11:     ProcessNewData(re_prob)
```

```
1:  function ProcessNewData(S_new)
2:      for o in S_new do
3:          c =FindSafeClusters(o)
4:          if c is not None then
5:              c.add(o)
6:          else
7:              c =FindUnSafeClusters(o)
8:              if c is not None then
9:                  c.add(o)
10:             else
11:                 FormCluster(o)
```

```
1:  function ReportOutlier(unsafeClusters)
2:      outliers = []
3:      for c in unsafeClusters do
4:          for o in c.members do
5:              if o.num_neighbors < k then
6:                  outliers.add(o)
        return outliers
```

4.3 Minimal Probing Micro-cluster Based Algorithm - M-MCOD

In MCOD, when some data points in a micro-cluster expire and the micro-cluster is dispersed, the remaining data points are processed as new data points. This is inefficient when the number of dispersed micro-clusters is large. Motivated by this, we propose M-MCOD. To reduce the number of dispersed micro-clusters when the window

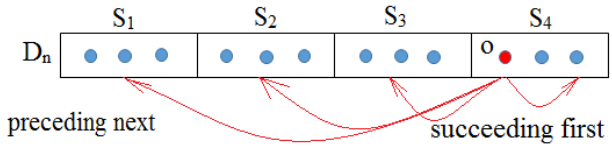


Figure 6: Probing for new data point o.

slides, in M-MCOD, when forming a micro-cluster, the most recent data points are probed first, then the older data points. This ensures the micro-clusters containing the most recent data points. Similarly, when a micro-cluster is dispersed, the remaining data points are processed as new data points. Each slide has a *trigger list* to store data points and micro-clusters whose outlier status can be affected by the slide's expiration. Each micro-cluster stores a hash map of slide indexes and the number of its member in those slides. Each data point stores the micro-cluster center if it is in a micro-cluster or stores a hash map of slide indexes and the number of its neighbors in those slides.
New slide processing. For each data point o in a new slide, it utilizes the *minimal probing principal* by finding the micro-clusters in the same slide first then older slides. If it cannot find any eligible micro-clusters, M-MCOD finds the neighbors for o by checking the data points in most recent slides first, then older slides. Figure 6 shows an example of probing for a data point o when it comes in the new slide. In order to find micro-clusters or neighbors for o, the order of slides that will be probed is S_4, S_3, S_2, S_1. In this approach,

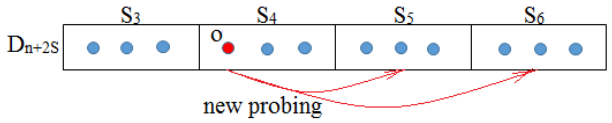


Figure 7: Re-probing as preceding neighbors expire or the micro-cluster is dispersed.

two hash maps of slide index and the data points, micro-cluster centers in that slide are used to efficiently retrieve the data points to probe.
Expired slide processing. When a slide S expires, the data points and micro-clusters in the trigger list are re-evaluated. The slide index of the expired slide is removed from hash maps of data points and micro-clusters in the *trigger list*. Those micro-clusters and data points are re-evaluated. If a micro-cluster do not have at least $k+1$ members, its members are re-processed as new data points. The pseudo code of neighbor finding procedure is presented Algorithm 4. Figure 7 shows an example of re-probing operation for o when slides S_1 and S_2 expire. The order of slides that will be probed as S_5, S_6.
Outlier reporting. All the data points in *PD* are evaluated for outlier reporting.

The advantage of M-MCOD compared to MCOD is that it forms micro-clusters of most recent data points hence reduces the number of dispersed micro-clusters.

Algorithm 4 Minimal Probing Micro-Cluster Based Algorithm

```
1: function FindNeighbors(o, Reversed_Sorted_Slides)
2:     num_neighbors = 0
3:     o.neighbor_map = {}
4:     neighbors = []
5:     for Slide in Reversed_Sorted_Slides do
6:         neighbors.add(FindNeighbor(o, Slide))
7:         o.neighbor_map[S] = neighbors.size()
8:         return o, neighbors
```

5 EXPERIMENTS

5.1 Experimental Methodology

We compare our proposed algorithms with MCOD which was shown to be the most efficient algorithm in memory and time [10] and the most recent algorithm NETS [12]. For fair evaluation, all the algorithms are implemented in Java. Our experiments were conducted on a Linux machine with a 3.47 GHz processor and 15 GB Java heap space.

Datasets. We chose the following real-world and synthetic datasets for our evaluation similarly to [10]. **Forest Cover** (FC) contains 581, 012 records with 55 attributes. It is available at the UCI KDD Archive[1] and was also used in [6]. **TAO** contains 575, 648 records with 3 attributes. It is available at Tropical Atmosphere Ocean project[2]. A smaller TAO dataset was used in [2, 7]. **Stock** contains 1, 048, 575 records with 1 attribute. It is available at UPenn Wharton Research Data Services[3]. A similar stock trading dataset was used in [3]. **Gauss** contains 1 million records with 1 attribute. It is synthetically generated by mixing three Gaussian distributions. A similar dataset was used in [2, 11].

Table 1: Default Parameter Setting

Dataset	Size	Dim.	W	S	Outlier Rate
FC	581,012	55	10,000	500	1.00%
TAO	575,648	3	10,000	500	0.98%
Stock	1,048,575	1	100,000	5,000	1.02%
Gauss	1,000,000	1	100,000	5,000	0.96%

Default Parameter Settings. There are four parameters: the window size W, the slide size S, the distance threshold R, and the neighbor count threshold k. W and S determine the volume and the speed of data streams. They are the major factors that affect the performance of the algorithms. The default values of W and S are set accordingly for two smaller datasets and two larger datasets, provided in Table 1. The values of k and R determine the outlier rate, which also affect the algorithm performance. For example, memory consumption is related to k as all the algorithms store information regarding k neighbors of each data point. For fairness of measurement, the default value of k is set to 50 for all datasets. To derive comparable outlier rate across datasets as suggested in [2, 7], the default value of R is set to 525 for *FC*, 1.9 for *TAO*, 0.45 for *Stock*, and 0.028 for *Gauss*. Unless specified otherwise, all the parameters take on their default values in our experiments. For O-MCOD, the maximum number of micro-clusters one data can participate in is set to 3. For NETS, we set the number of sub-dimensions parameter as suggested in the paper, which is 3 for *FC*, *TAO* and 1 for *Stock*, *Gauss*.

Performance Measurement. We measured the CPU time of all the algorithms for processing each window with ThreadMXBean in Java and created a separate thread to monitor the Java Virtual Machine memory. Measurements averaged over all windows were reported in the results.

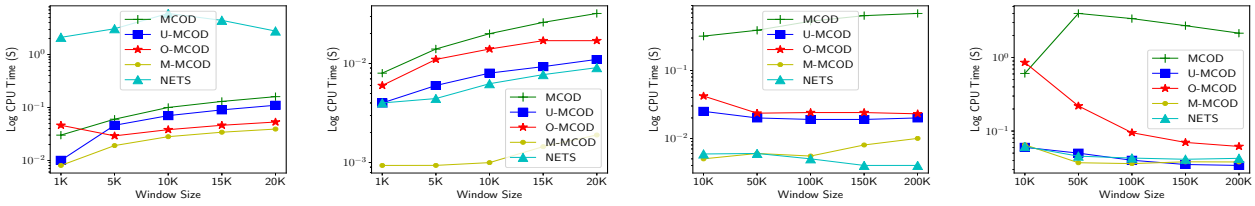
5.2 Varying Window Size

We first evaluate the performance of all the algorithms by varying the window size W. Figures 8 and 9 depict the resulting CPU and peak memory requirement, respectively. The window size is varied from 1000 to 20000 for FC and TAO datasets, and from 10000 to 200000 for Stock and Gaussian datasets.

CPU Time. As can be seen in Figure 8, when W increases, the CPU time for all methods increases in most cases due to a larger number of data points to process in every window, with exceptions of O-MCOD for FC and Gauss data and MCOD for Gauss. For FC, the CPU time of NETS is much higher than the CPU time of the other algorithms. NETS utilizes grid cells instead of micro-clusters to quickly determine outliers. However, with FC which has a high number of dimensions, most cells are not determined to contain all inliers or outliers due to the curse of dimensionality. M-MCOD incurs the lowest CPU running time for all the datasets. For FC and Gauss datasets, with small window size $W = 1K$, there are not many micro-clusters, in O-MCOD and MCOD, most data points are in *PD* therefore they incur the highest CPU running times. When the window size increases, e.g., $W = 5K$ with FC dataset and $W = 50K$ with Gauss dataset, O-MCOD utilizing overlapping micro-clusters can gather more data points in micro-clusters, hence the CPU running time decreases. When the window size $W \geq 50K$ with Gauss dataset, MCOD has more data points in micro-clusters, and the CPU running time decreases. For Stock and Gauss datasets, when the window size $W \geq 100K$, M-MCOD, O-MCOD and U-MCOD have comparable CPU running times because most data points are in *safe micro-clusters*. Their CPU times are also comparable to NETS.

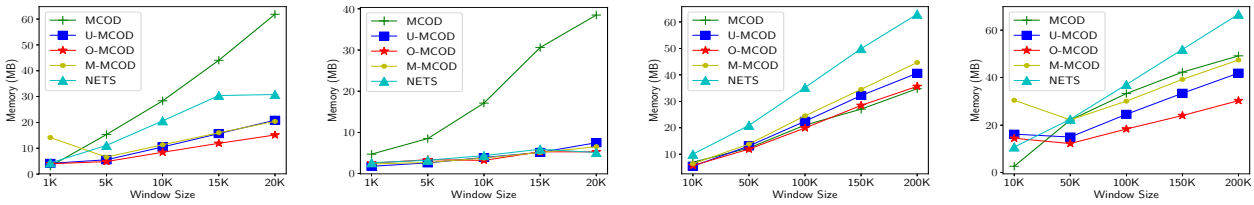
Peak Memory. Figure 9 reports the peak memory requirement of all the algorithms when varying the window size. As reported in Figure 9, in most cases the peak memory of all algorithms increases when W increases as there are more data points in each sliding window. In NETS, there is extra space needed for the map between grid cells and data points. In M-MCOD, there are extra space needed for *trigger lists* and hash maps of slide indexes and corresponding micro-clusters. In U-MCOD, there are extra space needed for *unsafe micro-clusters*. In O-MCOD, there are extra space needed for overlapping micro-clusters. Therefore, with small window size $W = 1K$, if there are not many data points in micro-clusters, the peak memory of those methods is higher than MCOD. When the window size increases, there are more data points in micro-clusters, U-MCOD, O-MCOD, and M-MCOD offer lower peak memory requirements than MCOD and NETS in most cases. For Stock and Gauss datasets with large window sizes, since the memory for storing all data points in a sliding window consumes the major part of the total memory, the difference between the methods is marginal.

[1] http://kdd.ics.uci.edu
[2] http://www.pmel.noaa.gov
[3] https://wrds-web.wharton.upenn.edu/wrds/



(a) FC (b) TAO (c) Stock (d) Gauss

Figure 8: CPU Time - Varying Window Size W



(a) FC (b) TAO (c) Stock (d) Gauss

Figure 9: Peak Memory - Varying Window Size W

5.3 Varying Slide Size

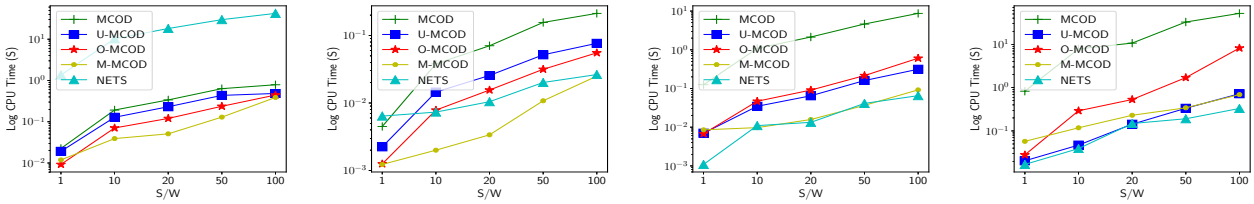
We further examine the algorithms' performances when varying the slide size S to change the speed of the data streams, e.g., from 1% to 100% of the window size W as in [10]. When the slide size increases, there are more data points expire and arrive when the window slides. $S = W$ is the extreme case, in which every data point exists in only one window. Figures 11 and 10 depict the results of peak memory requirement and CPU running time of all the algorithms, respectively.

CPU Time. As shown in Figure 10, M-MCOD incurs the lowest CPU time in all the cases. For FC, the CPU time of NETS is much higher than the other algorithms due to the high dimensinality of FC. With small slide size, e.g., when $S = 1\%W$, all the methods incur comparable CPU running time because the number of processed data points is small and the effect of overlapping micro-clusters in O-MCOD, *unsafe micro-clusters* in U-MCOD and *minimal probing principle* in M-MCOD is small. When the slide size increases, all the algorithms incur more CPU time as expected and the difference is larger. Because of more micro-clusters in U-MCOD, O-MCOD, and less dispersed micro-clusters in M-MCOD by the *minimal probing principal*, they incur less CPU running time than MCOD. When $S/W \geq 20\%$, M-MCOD is 4 to 95 times faster than MCOD.

Peak Memory. Figure 11 depicts the resulting peak memory of all the algorithms. We observe that when S increases, the memory cost of all the algorithms (except NETS) decreases. When S increases, NETS always uses more memory to derive the similarity between the expired slide and the new slide. When $S = W$, every data point does not have any preceding neighbors and participates in only one window. In this case, all the algorithms show similar memory consumption. O-MCOD continues to be superior to other algorithms in memory efficiency thanks to overlapping micro-clusters. For Stock and Gauss datasets, since the memory for storing the data points with a large window size is the major part in the total memory, our proposed methods have marginal advantage compare to MCOD. When slide size is small, there is less succeeding neighbors and more preceding neighbors for each data point. Therefore, every data point stores a longer preceding neighbor list in MCOD and neighboring hash map in M-MCOD, O-MCOD and U-MCOD. When $S/W = 1\%$, with Gauss dataset, M-MCOD shows higher memory consumption. The reason is the *trigger list* for each slide is long when S is small and the neighbor map for slides is proportional to W/S.

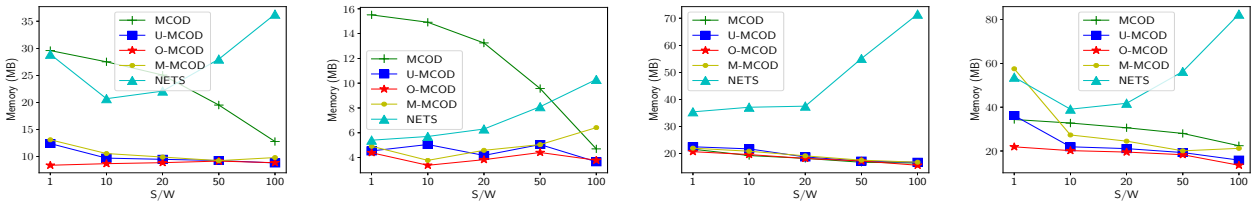
5.4 Varying *max_cluster* in O-MCOD

In O-MCOD, the parameter *max_cluster* specifies the maximum number of micro-clusters that one data point can participate in. When *max_cluster* increases, there can be more micro-clusters, but for each data point a longer list of its micro-clusters must be stored. Table 2 depicts the CPU running time, the peak memory requirement and the average number of micro-clusters in each sliding window in O-MCOD with the TAO dataset. As can be seen in Table 2, when *max_cluster* increases, the average number of micro-clusters increases and the CPU running time decreases because the time for finding neighbors decreases. The memory requirement first decreases because there are more data points in micro-clusters which are more memory efficient than neighbor lists. When *max_cluster* further increases, each data point has a longer list of clusters, therefore, the memory requirement increases. When *max_cluster* = 3, the CPU running time, memory requirement and average number of clusters achieve nearly optimal values.



(a) FC (b) TAO (c) Stock (d) Gauss

Figure 10: CPU Time - Varying slide size S



(a) FC (b) TAO (c) Stock (d) Gauss

Figure 11: Peak Memory - Varying Slide Size S

Table 2: Varying *max_cluster* in O-MCOD

max_cluster	1	3	5	7	10
CPU Time (s)	0.021	**0.015**	**0.015**	**0.015**	**0.015**
Memory (MB)	3.822	**3.767**	3.768	3.771	3.790
Average No. Micro-clusters	21.957	23.802	**23.803**	**23.803**	**23.803**

6 CONCLUSION AND DISCUSSIONS

In this study, we propose three algorithms for distance-based outlier detection in data streams. All proposed algorithms utilize micro-cluster for efficient neighbor finding. Specifically, O-MCOD allows one data point to participate in more than one micro-clusters. U-MCOD relaxes the constraints of micro-cluster to allow *unsafe micro-cluster* can have less than $k+1$ data points. M-MCOD adopts *minimal probing principle* to reduce the number of dispersed micro-clusters. We observe that M-MCOD incurs the lowest CPU running time, O-MCOD requires the least memory requirement in most cases. The proposed methods outperform MCOD and NETS in most cases. And each proposed method has its own advantage. The combination of all the proposed methods can be explored in the future. We can combine O-MCOD and M-MCOD for utilizing the advantages of overlapping micro-clusters and the *minimal probing principal*.

7 ACKNOWLEDGMENTS

This work has been supported in part by NSF CNS-1915828, the USC Integrated Media Systems Center, and unrestricted cash gifts from Oracle and Google. The opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the sponsors.

REFERENCES

[1] Charu Aggarwal (Ed.). 2007. *Data Streams – Models and Algorithms.* Springer.
[2] Fabrizio Angiulli and Fabio Fassetti. 2007. Detecting distance-based outliers in streams of data. In *Proceedings of the sixteenth ACM conference on Conference on information and knowledge management.* ACM, 811–820.
[3] Lei Cao, Di Yang, Qingyang Wang, Yanwei Yu, Jiayuan Wang, and Elke A Rundensteiner. 2014. Scalable distance-based outlier detection over high-volume data streams. In *2014 IEEE 30th International Conference on Data Engineering.* IEEE, 76–87.
[4] Dimitrios Georgiadis, Maria Kontaki, Anastasios Gounaris, Apostolos N. Papadopoulos, Kostas Tsichlas, and Yannis Manolopoulos. 2013. Continuous Outlier Detection in Data Streams: An Extensible Framework and State-of-the-art Algorithms. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data (SIGMOD '13).* ACM, New York, NY, USA, 1061–1064.
[5] Edwin M. Knorr and Raymond T. Ng. 1998. Algorithms for Mining Distance-Based Outliers in Large Datasets. In *Proceedings of the 24rd International Conference on Very Large Data Bases (VLDB '98).* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 392–403.
[6] M. Kontaki, A. Gounaris, A.N. Papadopoulos, K. Tsichlas, and Y. Manolopoulos. 2011. Continuous monitoring of distance-based outliers over data streams. In *Data Engineering (ICDE), 2011 IEEE 27th International Conference on.* 135–146. https://doi.org/10.1109/ICDE.2011.5767923
[7] Maria Kontaki, Anastasios Gounaris, Apostolos N Papadopoulos, Kostas Tsichlas, and Yannis Manolopoulos. 2011. Continuous monitoring of distance-based outliers over data streams. In *2011 IEEE 27th International Conference on Data Engineering.* IEEE, 135–146.
[8] Md. Shiblee Sadik and Le Gruenwald. 2010. *Database and Expert Systems Applications: 21st International Conference, DEXA 2010, Bilbao, Spain, August 30 - September 3, 2010, Proceedings, Part I.* Springer Berlin Heidelberg, Berlin, Heidelberg, Chapter DBOD-DS: Distance Based Outlier Detection for Data Streams, 122–136. https://doi.org/10.1007/978-3-642-15364-8_9
[9] S. Subramaniam, T. Palpanas, D. Papadopoulos, V. Kalogeraki, and D. Gunopulos. 2006. Online Outlier Detection in Sensor Data Using Non-parametric Models. In *Proceedings of the 32Nd International Conference on Very Large Data Bases (VLDB '06).* VLDB Endowment, 187–198.
[10] Luan Tran, Liyue Fan, and Cyrus Shahabi. 2016. Distance-based outlier detection in data streams. *Proceedings of the VLDB Endowment* 9, 12 (2016), 1089–1100.
[11] Di Yang, Elke A. Rundensteiner, and Matthew O. Ward. 2009. Neighbor-based Pattern Detection for Windows over Streaming Data. In *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology (EDBT '09).* ACM, New York, NY, USA, 529–540.
[12] Susik Yoon, Jae-Gil Lee, and Byung Suk Lee. 2019. NETS: extremely fast outlier detection from a data stream via set-based processing. *Proceedings of the VLDB Endowment* 12, 11 (2019), 1303–1315.