A Fast and Efficient Algorithm for Outlier Detection Over Data Streams

Mosab Hassaan[1]
Faculty of Science
Benha University
Egypt

Hend Maher[2], Karam Gouda[3]
Faculty of Computers and Artificial Intelligence
Benha University
Egypt

***Abstract*—Outlier detection over data streams is an important task in data mining. It has various applications such as fraud detection, public health, and computer network security. Many approaches have been proposed for outlier detection over data streams such as distance-,clustering-, density-, and learning-based approaches. In this paper, we are interested in the density-based outlier detection over data streams. Specifically, we propose an improvement of DILOF, a recent density-based algorithm. We observed that the main disadvantage of DILOF is that its summarization method has many drawbacks such as it takes a lot of time and the algorithm accuracy is significant degradation. Our new algorithm is called DILOF^C that utilizing an efficient summarization method. Our performance study shows that DILOF^C outperforms DILOF in terms of total response time and outlier detection accuracy.**

***Keywords*—Data mining; outlier detection; data streams; density-based approach; clustering-based approach**

I. Introduction

Outlier detection (OD) is considered an important data mining task. The objective of this task is to discover elements (points) that show significant diversion from the expected behaviour called outliers. For example, consider the two dimensional data points in Fig. 1. This dataset contains three normal regions namely N_1, N_2, and N_3. We can observe that data points that are significantly far away from the three regions are outliers. In this example o_1, o_2, o_3, and o_4 are outliers. The prominent causes for outliers are malicious activity, change in the environment, instrumentation error, and human error. OD plays a significant role and has been useful for several real-world applications such as intrusion detection systems, interesting sensor events, credit-card fraud, law enforcement, and medical diagnosis.

Outlier Detection raises significant challenges when a stream-based environment is considered[1], [2], [3]. A data stream potentially contains an infinite number of data points. Memory limitations constrain the amount of data points that can be held and processed at a given time. Moreover, no information related to data points appearing in the data stream are available before entering the memory. That is, the state of the current data point as an outlier/inlier must be established before dealing with subsequent data points. For example, in wireless sensor networks, a limited memory is available at each sensor node and outliers must be detected in reasonable time. The communication cost of these networks is also an essential factor. There are many approaches of outlier detection over data stream such as clustering based outlier detection

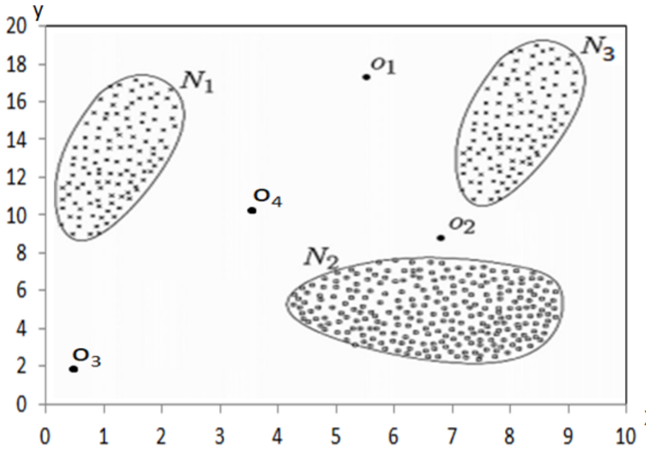

Fig. 1. Running Example (Outliers).

[4],statistical based outlier detection [5], [6], distance based outlier detection [7], [8], [9], [10][11], and density based outlier detection [12], [13], [14] [15], [16]. In this paper,we are interested in the density-based outlier detection over data streams. Specifically, we propose an improvement of DILOF, a recent density-based algorithm. Our new algorithm is called **DILOF**^{C} (Density Incremental LOF using summarization that based on novel m-Center clustering algorithm). We observed that the main problem in DILOF is that the summarization method has drawbacks such as it takes a lot of time and the algorithm accuracy is significant degradation. Note that DILOF is one of the most known algorithm that apply density based outlier detection approach. In density based outlier detection approach, the density of each point is compared with the density of its local neighbors. This approach is based on the assumption that the density of the normal data point is the same as the density of its neighbors and the density of outliers are dissimilar to their local neighbors. For each point, the density is computed by outlier score called LOF (Local Outlier Factor) [17]. We will discuss LOF and DILOF in Sections II-A and II-B respectively in details.

In the remaining sections, we discuss the problem definition and related work in Section II. Section III presents our proposed algorithm. We report the experimental results in Section IV. Finally, Section V concludes the paper.

II. Problem Definition and Related Work

Definition 2.1 A data stream is a possible infinite sequence of data points $P = \{p_1, p_2, p_3, ..., p_n, ...\}$, where data point p_n is arrived at time $p_n.t$

In previous definition, the data points are sorted by the timestamp at which it arrives. Since data stream size is unbounded thus data stream will be processed in a sliding window, i.e. a collection of active data points. Window is small enough to be held in the main memory. Windowing splits the data stream into overlapping finite sets of data points (sliding windows). The splitting can be done by arrival time of the data points, namely, time-based windows or by the count of the data points namely, count-based window. In this paper, we focus on the count-based window.

Problem Definition
Given a data stream $P = \{p_1, p_2, ..., p_n, ...\}$, the objective is to calculate the LOF score for each data point p_i and check p_i outlier or not with respect to the following constraints.

- We store only a small number of data points, $m << |P|$. Note that here m equals to the window size, $|W|$
- The outlier detection of coming data point p_i, must be done when p_i arrives.
- The data distribution is unknown.

A. LOF (Local Outlier Factor), iLOF, and MILOF

LOF [17] is well-known algorithm for outlier detection in static datasets. The objective of LOF is to calculate the LOF score for each data point. Suppose the following:

- We have all data points,
- The count of the nearest neighbors is k,
- $dist(x, y)$ is the Euclidean distance between the two data points x and y,
- $dist_k(x)$ is the Euclidean distance between a data point x and its k nearest neighbor.
- $N_k(x)$ is the set of the k-nearest neighbors of the data point x.

According to the following definitions, we will compute the LOF score for all data points.

Definition 2.2 Given two data points x and y, reachability distance $reach\_dist_k(x, y)$ is defined by

$$reach\_dist_k(x, y) = max\{dist(x, y), dist_k(y)\} \tag{1}$$

Definition 2.3 Local reachability density of data point x, lrd_k (x) is given by

$$lrd_k(x) = \left(\frac{1}{k} \sum_{y \in N_k(x)} reach\_dist_k(x, y)\right)^{-1} \tag{2}$$

Definition 2.4 Local outlier factor of data point x, $LOF_k(x)$ is given by

$$LOF_k(x) = \frac{1}{k} \sum_{y \in N_k(x)} \frac{lrd_k(y)}{lrd_k(x)} \tag{3}$$

To check if data point x is outlier or not, we compare its local outlier factor $LOF_k(x)$ with a given threshold T. If $LOF_k(x) \geq T$ then the data point x is classified as outlier. Note that LOF algorithm is used to compute the LOF scores of all data points only once. Recall LOF algorithm detects outliers in static datasets

iLOF (incremental LOF) [18] was proposed for stream datasets but it stores all data points in memory. Thus iLOF requires a very large memory and is not applicable to stream datasets whose size sharply increasing. Another algorithm called MiLOF [19] was proposed to decrease the space complexity. It stores in memory a small number of data points by using k-means clustering [20] method to summarize old data points. The accuracy of MILOF is inefficient since it uses k-means for summarization which does not preserve the dataset density. To overcome the drawbacks of MILOF, authors of [13] proposed a new algorithm called DILOF (Density summarizing Incremental LOF). Since our proposed algorithm is based on DILOF, we will discuss DILOF algorithm in the next section in details.

B. DILOF Algorithm

DILOF [13] is well-known algorithm for outlier detection over data stream. It is density-based algorithm and applies two steps as follows. The first one is Last Outlier-aware Detection step, LOD which check if the incoming data point x is outlier or not. This done by computing $LOF_k(x)$ on the a variable window of data, W. Then the algorithm updates the information of the old data points (lrd_k and LOF_k) that exist in W and affected by inserting the data point x (i.e. the data points whose neighbor information will be modified when inserting the data point x). Note that the data point x is inserted to W no matter whether it is an outlier or inlier. Also skipping scheme strategy was proposed to detect a long sequence of outliers. in other words, this scheme was proposed to distinguish outliers from the data points in newly emerging classes. This can be done by deleting the new outliers from the window to preserve the low density region where outliers are existed. The following formally outlines skipping scheme strategy in details. First, DILOF computes $dist_1(p)$ for each data point $p \in M$ (Note that M is the set of data points in memory). Then it computes the average of distance $avg\_dist_1 = \sum_{j=1}^{|M|} dist_1(p_j)$. Let $dist(o, p_c)$ be the Euclidean distance between the last detected outlier o and the current data point p_c. If $avg\_dist_1 > dist(o, p_c)$ then set o to p_c and the data point p_c is not inserted to W. In this case, Skipping_Schema parameter will be set to true.

The second step is Nonparametric Density Summarization step, NDS which decrease the memory consumption by summarizing the old data points with respect to the dataset density. In NDS, an estimator called nonparametric Renyi divergence was used to specify the divergence between the original data points and summary candidate of data points. When the count of data points is equals to $|W|$ data points, NDS will summarize the oldest $|W/2|$ data points to $|W/4|$ representative data points such that the density difference between them is minimized. See Fig. 2.

Note that the previous two steps are repeated. LOD executes on every insertion of a data point. NDS is executed

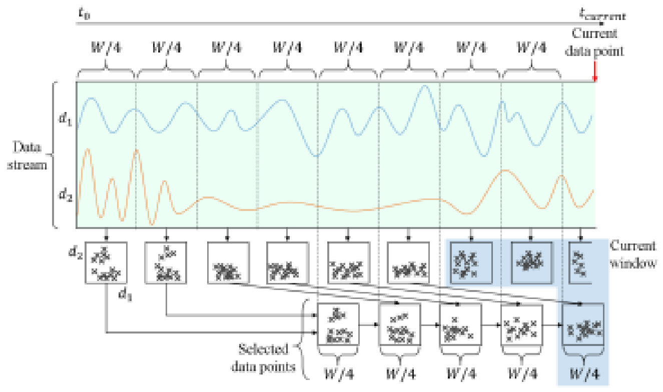

Fig. 2. NDS from Time t_0 to Time $t_{current}$ for a Two Dimensional Data.

when the number of data points in memory equals to $|W|$. In experimental results of DILOF on real-world datasets, DILOF significantly outperforms MiLOF with respect to accuracy and execution time.

III. Proposed Algorithm

Since the summarization method of DILOF algorithm taking many iterations to find its output. Therefore, the summarization method of DILOF algorithm takes a lot of time. Also, by using expensive experiments in many real datasets, we found that the algorithm accuracy is significant degradation. To overcome this issue, in this paper, we will propose a new summarization method called sum_m_center which will be injected in DILOF algorithm instead of its current inefficient summarization method. The proposed summarization method based on a new clustering technique called m-center. First, we propose m-center clustering algorithm then we will discuss the proposed summarization method. The previous clustering algorithms such as k-means require a large number of iterations to compute its output. To address this problem, we propose m-center clustering algorithm that is the partitioning representative, medoid, is sampled from the original data. In this method, we efficiently search for each cluster medoid as follows. In the first iteration we will search for the medoid of the first cluster and in the second iteration we will search for the medoid of the second cluster and so on. So if we set the number of clusters as k then we have only k iterations. In each iteration I we will execute the following steps. For each data point $p \in \mathcal{P}$ (the set of all data points), we calculate its m nearest neighbors set, $mNN(p)$. Then we compute the distance between p and each p_j belongs to $mNN(p)$ (here we will use Euclidean Distance, $dist(p, p_j)$) and compute the summation $sum\_dist(p) = \sum_{j=1}^{m} dist(p, p_j)$. After that we select the point $p_i \in \mathcal{P}$ with minimum $sum\_dist(p)$ as a medoid of the cluster being processing C_I then add all points in $mNN(p)$ to C_I. Finally we remove each point $p \in C_I$ from $\mathcal{P}$. Recall we have k iterations, then we repeat the previous steps $k-1$ times after the initial iteration. Now we have k clusters. If there are remaining data points do not belong to any cluster (i.e. after k iterations we have $|\mathcal{P}| \neq \phi$), then we add each remaining data point p_r to its closest cluster based on the distance between the medoid of each cluster and p_r. Next algorithm (Lines 1 - 13) outlines the m-center algorithm.

Algorithm: m-center($\mathcal{P}, k, m, d_t$)

```
Input: $\mathcal{P}$: the set of data points,
       $k$: the number of clusters,
       $m$: the size of the m nearest neighbors set of a specified
        data point,
       $d_t$: distance threshold.
Output: $\mathcal{C}$: $k$-clusters set;

1.  for $l$ = 1 to $k$ do
2.      for each data point $p_i \in \mathcal{P}$ do
3.          Compute the m nearest neighbors set of $p_i$, $mNN(p_i)$,
            such that $|mNN(p_i)|$ = $m$.
4.          Compute $sum\_dist(p_i)$ = $\sum_{j=1}^{m} dist(p_i, p_j)$,
            where $p_j \in mNN(p_i)$
5.      end for
6.      Select $p_x \in \mathcal{P}$ where $sum\_dist(p_x)$ has the smallest value.
7.      Add the point $p_x$ and the set $mNN(p_i)$ to cluster $C_l$
        where $p_x$ is the cluster medoid.
8.      Remove the point $p_x$ and the set $mNN(p_i)$ from $\mathcal{P}$.
9.  end for
10. for each remaining data point $p_r \in \mathcal{P}$ do
11.     Add $p_r$ to its closest cluster.
12. end for
13. $\mathcal{C}_{temp} = \bigcup_{l=1}^{k} C_l$
14. Combine each nearest clusters set in $\mathcal{C}_{temp}$ into one big
    cluster $CB^h$ with respect to $d_t$        //Optimization
15. $\mathcal{C} = \bigcup_h CB^h$
16. return $\mathcal{C}$
```

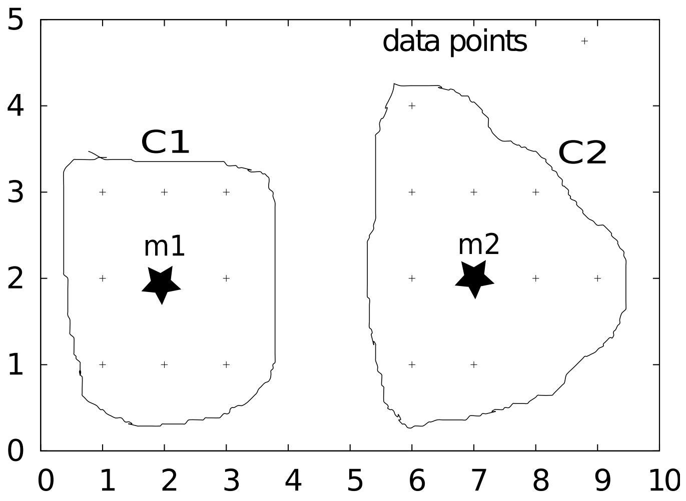

Fig. 3. Running Example (m-center Clustering Algorithm).

Example 3.1 Given two dimensional data point set $\mathcal{P} = \{p_1, p_2, p_3, ..., p_{18}, p_{19}\} = \{(1, 1), (1, 2), (3, 2), (3, 1), (2, 2), (7, 2), (6, 4), (8, 2), (8, 3), (2, 1), (7, 1), (6, 3), (9, 2), (3, 3), (6, 1), (6, 2), (2, 3), (7, 3), (1, 3)\}$. Let k = 2 and m = 4. Since k = 2 then we have two iterations. In the first iteration we found that the point $p_3 = (2, 2)$ is the medoid of the first cluster, C_1, since $mNN(p_3) = \{(1, 2), (2, 1), (2, 3), (3, 2)\}$ has minimum $sum\_dist(p_3)$ that is $sum\_dist(p_3)$ = 4. Then we remove p_3 and $mNN(p_3)$ from $\mathcal{P}$. In the second iteration we found that the point $p_6 = (7, 2)$ is the medoid of the second cluster, C_2, since $mNN(p_6) = \{(6, 2), (7, 1), (7, 3), (8, 2)\}$ has minimum $sum\_dist(p_6)$ that is $sum\_dist(p_6)$ = 4. Then we remove p_6 and $mNN(p_6)$ from $\mathcal{P}$. After the two iterations we check if there exist a remaining data points or not in $\mathcal{P}$. In this example there are a remaining data points. The count of remaining data points is nine since we remove five data points in each iteration. Therefore, we add each remaining data point to its closest cluster. Now we have two clusters as follows $C_1 = \{(1, 1), (1, 2), (3, 2), (3, 1), (2, 2), (2, 1), (3, 3), (2, 3), (1, 3)\}$ and $C_2 = \{(7, 2), (6, 4), (8, 2), (8, 3), (7, 1), (6, 3), (9, 2), (6, 1), (6, 2), (7, 3)\}$. See Fig. 3

A. Optimization

In the previous example we set $k = 2$. If we set $k = 3$ or 4 then we have three or four clusters respectively. But our original data has only two clusters. If we set k large than the number of original clusters in our data then m-center will cluster the data in inefficient way. Therefore, m-center will be optimized as follows. In the case above, some clusters should be merged efficiently. Which clusters will be merged?. The answer is the nearest clusters will be merged. First we define the nearest clusters as follows.

Definition 3.1 Given two clusters C_i and C_j with medoids m_i and m_j respectively and distance threshold d_t. C_i and C_j are called nearest clusters to each other if $dist(m_i, m_j) \leq d_t$.

Definition 3.2 Clusters set, S, are called nearest clusters set if every pair in S contains two nearest clusters.

Let $k = 5$ then we have five clusters C_1, C_2, C_3, C_4, and C_5. Assume after checking the nearest clusters we found two sets of nearest clusters as follows the first nearest clusters set is $NC^1 = \{C_1, C_2, C_3\}$ and the second one is $NC^2 = \{C_4, C_5\}$. We will merge the clusters in the same nearest cluster set into one big cluster. Therefore, we have two big clusters $CB^1 = C_1 \cup C_2 \cup C_3$ and $CB^2 = C_4 \cup C_5$. Lines 14-16 in the m-center algorithm outlines our optimization. Also next example illustrate this optimization.

Example 3.2 Given two dimensional data point set $\mathcal{P}$ with size 32 data points as in Fig. 4. Let $k = 5$, $m = 4$ and $d_t = 5$. After applying m-center clustering method we have five clusters, namely C_1 with medoid m1 = (3, 4), C_2 with medoid m2 = (7, 5), C_3 with medoid m3 = (7, 2), C_4 with medoid m4 = (17, 7), and C_5 with medoid m5 = (16, 4). From above optimization we have two nearest clusters set based on the distance threshold $d_t = 5$. The first nearest clusters set is $NC^1 = \{C_1, C_2, C_3\}$ since every pair in NC^1 contains two nearest clusters, for example C_1 and C_2 are nearest clusters since $dist(m_1, m_2) = 4.12 < 5 = d_t$. The second nearest clusters set is $NC^2 = \{C_4, C_5\}$ since C_4 and C_5 are nearest clusters with $dist(m_4, m_5) = 3.16 < 5 = d_t$. Based on above optimization, we will merge clusters in each nearest clusters set into one big cluster as follows. $CB^1 = \bigcup_{C \in NC^1} C = \{C_1, C_2, C_3\}$ and $CB^2 = \bigcup_{C \in NC^2} C = \{C_4, C_5\}$. Fig. 4 illustrates our optimization.

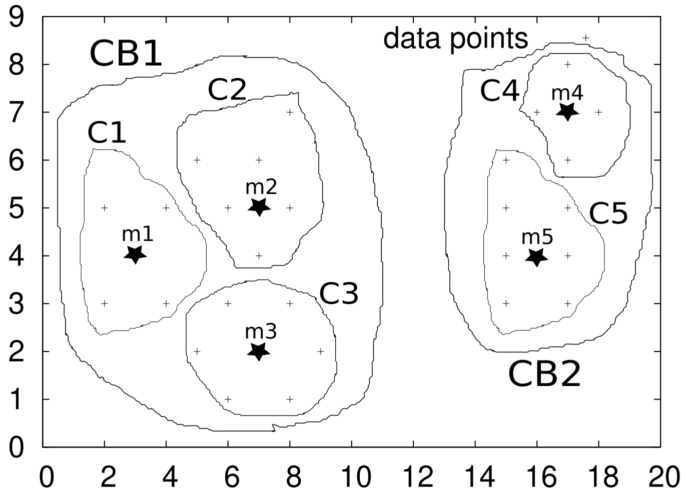

Fig. 4. Running Example (Optimization).

B. Summarization Step

In summarization step, we will delete a half of data points in the window. Which half of data points will be deleted? we will delete a half of data points according to two different deletion methods. In the first deletion method, we keep in each cluster C_I only the half of data points which close to C_I medoid and delete the other half.If we apply the optimization of merge clusters, the medoid of the big cluster CB will be the average of the medoids of the small clusters that contained by CB.

In the second deletion method, the unuseful old data points that do not effect the data density will be deleted that is we keep the half of data points in each cluster that preserve the cluster density and delete the other half. In other words, we delete a half of data points such that these data points are old and its LOF score is high. In experimental evaluation section, we will compare the two deletion methods. Next algorithm outlines the summarization step, sum_m_center.

```
Algorithm: sum_m-center($\mathcal{P}, k, m, d_t$)
Input: $\mathcal{P}$: the set of data points,
       $k$: the number of clusters,
       $m$: the size of the m nearest neighbors set of a specified data point,
       $d_t$: distance threshold.
Output: $\mathcal{S}$: summary of $k$ clusters;

1.  $\mathcal{C}$ = m-center($\mathcal{P}, k, m, d_t$)
2.  if Enable_first_deletion_method
3.     for each cluster $x$ in $\mathcal{C}$
4.        Delete half of data points in $x$ that are far from the
          medoid of $x$
5.     $\mathcal{S} = \mathcal{C}$
6.  if Enable_second_deletion_method
7.     for each cluster $x$ in $\mathcal{C}$
8.        Delete half of data points in $x$ that are old and its
          LOF score is high
9.     $\mathcal{S} = \mathcal{C}$
10. return $\mathcal{S}$
```

C. DILOF^C Pseudocode

Recall, the adaptive algorithm is called DILOF^C. The next algorithm outlines DILOF^C. For each data point p_i coming from stream we do the following. If we enable the skipping scheme and the return value is true then we continue to the next data point (lines 3-5). See section II-B for more details about skipping scheme strategy. Otherwise, we add p_i to the set of data points in memory, M (line 6) . Then we compute LOF score of p_i according to equations 1, 2, and 3 and add p_i to the set of outliers, $\mathcal{O}$, if LOF score of p_i is greater than LOF threshold, T (lines 7-10). At the same time, we update LOF score of each data point p_j in the reverse neighbour set of p_i and if the data point p_j transformed from outlier to inlier, we remove it from $\mathcal{O}$ (lines 11-16). If the size of data points in memory,$|M|$ reach the window size $|W|$, we call the function, sum_m_center, to summarize the oldest $|W|/2$ data points in M. Then we replace the oldest $|W|/2$ data points in M by the outputted summary of sum_m_center function lines (17-22).

Algorithm: DILOF^C

```
Input: Infinite data stream $P = \{p_1, p_2, ...., p_n, ....\}$,
       $T$: LOF threshold,
       $|W|$: Window size,
       $k$: the number of clusters,
       $m$: the size of the m nearest neighbors set of a specified
         data point,
       $d_t$: distance threshold.
Output: The set of outliers in $P$, namely $\mathcal{O}$.

1.  $M = \phi$    //the set of data points in memory
2.  $\mathcal{O} = \phi$
3.  while incoming data point $p_i$ from stream do
4.      if Enable_Skipping_Schema_Strategy and
          Skipping_Schema = TRUE
5.          continue
6.      $M = M \cup \{p_i\}$
7.      Compute the LOF score of $p_i$ according to equations
          1, 2, and 3
8.      if $LOF(p_i) > T$ then
9.          $\mathcal{O} = \mathcal{O} \cup \{p_i\}$
10.     end if
11.     for each data point $p_j$ in the set of reverse mNN($p_i$) do
12.         Update the LOF score of $p_j$
13.         if $p_j$ transfered from outlier to inlier then
14.             $\mathcal{O} = \mathcal{O} - \{p_j\}$
15.         end if
16.     end for
17.     if $|M| = |W|$ then
18.         Let $M'$ be the oldest $|W|/2$ data points in $M$
19.         $S = sum\_m\_center(M', k, m, d_t)$
            //Summarization Step, where $|S| = |M'|/2$
20.         $M = M - M'$
21.         $M = M \cup S$
22.     end if
23. end while
```

D. Time Complexity

Note that the summarization step is one of the main operations in the outlier detection algorithms over data streams. Therefore, in this section, we analyze the time complexity of m-center method for summarization step in the proposed algorithm, DILOF^C. The time complexity of m-center is $O(|W|km)$, where $|W|$ is the window size, k is the number of clusters, and m is the size of the nearest neighbors set of a specified data point. While the time complexity of summarization step in DILOF algorithm is $O((|W|/2)^2)$ [13] and the time complexity of summarization step in MILOF algorithm is $O(IDk|W|/2)$ [19], where I is the maximum count of iterations, D is the dimensionality of dataset, and k is the number of clusters.

The time complexity of our summarization step, m-center, is the best one due to $O(|W|km) << O((|W|/2)^2) << O(IDC|W|/2)$. For instance, if we handle the KDD Cup 99 http dataset where , for DILOF^C, k is 11 and m is 5 and for MILOF, I is 100 (defualt value for MILOF), D is 3, and k is 11 with $|W|$ = 700 for all algorithms, then we have $|W|km = 38500 << 122500 = (|W|/2)^2 << 1155000 = IDk|W|/2$.

IV. Experimental Evaluation

In this section, we evaluate the performance of DILOF^C on four real datasets. we compare the performance of DILOF^C with the DILOF [13]. Here, MiLOF [19] was exculded from this experiment since the experiment results of DILOF showed that DILOF has better performance than MILOF. All experiments were performed on a PC with Intel i5-6700 2.4 GHz, 8G memory running Windows 10 64-bit operating system. DILOF^C was implemented in standard C++ with STL library support. In next section, we discuss the datasets and experiment settings.

TABLE I. Properties of the Four Real-world Datasets

Dataset	# Data Points	Dimension	# Classes
UCI Vowel	1,456	12	11
UCI Pendigit	3,498	16	10
KDD Cup 99 smtp	95,156	3	Unknown
KDD Cup 99 http	567,479	3	Unknown

A. Dataset and Experiment Settings

DILOF^C performance was evaluated by applying it to four real-world datasets. Table I listed the properties of the four datasets. For the two datasets, KDD Cup 99 smtp and KDD Cup 99 http, the number of classes is set to 10 since the number of classes of theses datasets is unknown. The hyper-parameters of DILOF, η and λ are set to 0.3 and 0.001 respectively for all datasets. We set the default values of the parameter t that used in DILOF (t-nearest neighbors in DILOF) as the following. we set t to 19 for UCI Vowel and 8 for the three datasets UCI Pendigit, KDD Cup 99 smtp, and KDD Cup 99 http.

For our algorithm DILOF^C, the two parameters m (number of nearest neighbors) and k (number of cluster) are set to 5 and

11, respectively, for the three datasets UCI Pendigit, KDD Cup 99 smtp, and KDD Cup 99 http. For the dataset UCI Vowel, k and m are set to 10 and 11, respectively. For the parameter d_t (distance threshold for merging clusters) is set to 3.3 for the three datasets UCI Vowel, KDD Cup 99 smtp, and KDD Cup 99 http. For the dataset UCI Pendigit, d_t is set to 2.4.

Recall, DILOF^C and DILOF apply the summarization method when the count of data points equal to window size, $|W|$. Therefore, the performance of outlier detection is measured with different values of $|W|$. Since the two datasets, UCI Vowel and UCI Pendigit, contain a small number of data points, we selected a small window size for them $|W| = \{100, 120, 140, 160, 180, 200\}$. Since the two datasets, KDD Cup 99 smtp and KDD Cup 99 http, contain a large number of data points, we selected a large window size for them $|W| = \{100, 200, 300, 400, 500, 600, 700\}$. The LOF Thresholds were set to {0.1, 1.0, 1.1, 1.15, 1.2, 1.3, 1.4, 1.6, 2.0, 3.0} as in DILOF. For each LOF Threshold, we compute false positive rate and true positive rate then AUC in ROC space was computed for DILOF^C and DILOF.

B. Effects of Optimization and Deletion Methods

In this experiment, we show the effect of optimization (merge clusters) and the two different deletion methods. For this purpose, we implemented four versions of DILOF^C, namely, DILOF^{C_1} that does not use the optimization of merge clusters and use the first deletion method, DILOF^{C_2} that does not use the optimization of merge clusters and use the second deletion method, DILOF^{C_3} that uses the optimization of merge clusters and use the first deletion method, DILOF^{C_4} that uses the optimization of merge clusters and use the second deletion method.

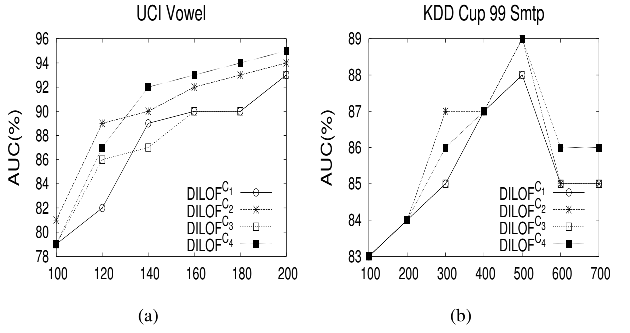

Fig. 5. Outlier Detection Accuracy with Respect to Window Size (Four Versions).

1) Outlier Detection Accuracy: Fig. 5 reports the outlier detection accuracy obtained by running the four versions on two datasets (UCI Vowel: Fig. 5(a) and KDD Cup 99 smtp: Fig. 5(b)). In UCI Vowel dataset, we can see that DILOF^{C_4} always has high accuracy compared with the other versions except for window size 100 and 120, where DILOF^{C_2} shows the best accuracy. In KDD Cup 99 smtp dataset, we can see that DILOF^{C_4} always has high or same accuracy compared with the other versions except for window size 300, where DILOF^{C_2} shows the best accuracy.

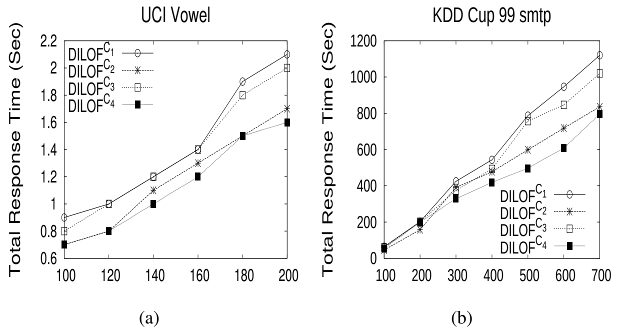

Fig. 6. Total Response Time with Respect to Window Size (Four Versions).

2) Total Response Time: Fig. 6 reports the total response time (sec) obtained by running the four versions on two datasets (UCI Vowel: Fig. 6(a) and KDD Cup 99 smtp: Fig. 6(b)). In UCI Vowel dataset, we can see that DILOF^{C_4} always has less time compared with the other versions. In KDD Cup 99 smtp dataset, we can see that DILOF^{C_4} always has less time compared with the other versions except for window size 100 and 120, where DILOF^{C_2} shows the best execution time.

C. DILOF^C against DILOF

From section IV-B, we showed that DILOF^{C_4} has the best performace among the other versions of the proposed algorithm with respect to outlier detection accuracy and total response time. Therefore, in this experiment, we will use DILOF^{C_4} version and we will call it as DILOF^C for abbreviation. Here, we compare DILOF^C against DILOF on the four datasets, namely, UCI Vowel, UCI Pendigit, KDD Cup 99 smtp, and KDD Cup 99 http with respect to outlier detection accuracy and total response time. See next two sections.

1) Outlier Detection Accuracy: Fig. 7 shows the outlier detection accuracy of DILOF^C and DILOF with respect to the window size using the four datasets (UCI Vowel: Fig. 7(a), UCI Pendigit: Fig. 7(b), KDD Cup 99 smtp: Fig. 7(c), and KDD Cup 99 http: Fig. 7(d)). In UCI Vowel dataset, DILOF^C shows higher accuracy than DILOF at all window sizes. For example, at window size 200, the accuracy of DILOF^C is 95% whereas the accuracy of DILOF is 91%. In UCI Pendigit dataset, DILOF^C and DILOF have approximately the same accuracy. In KDD Cup 99 smtp dataset, DILOF^C shows higher accuracy than DILOF at most cases (for example, at window size 700, the accuracy of DILOF^C is 86% whereas the accuracy of DILOF is 73%) except for window size 600, where accuracy of DILOF^C is 86% whereas the accuracy of DILOF is 88%. In KDD Cup 99 smtp dataset, DILOF^C shows higher accuracy than DILOF at all window sizes (for example, at window size 700, the accuracy of DILOF^C is 83% whereas the accuracy of DILOF is 75%).

2) Total Response Time: Fig. 8 shows the total respose time (sec) of DILOF^C and DILOF with respect to the window size using the four datasets (UCI Vowel: Fig. 8(a), UCI Pendigit: Fig. 8(b), KDD Cup 99 smtp: Fig. 8(c), and KDD Cup 99 http: Fig. 8(d)). DILOF^C shows the best execution time on all datasets. On UCI Vowel dataset, UCI Pendigit dataset, KDD Cup 99 smtp, and KDD Cup 99 http, DILOF^C outperforms

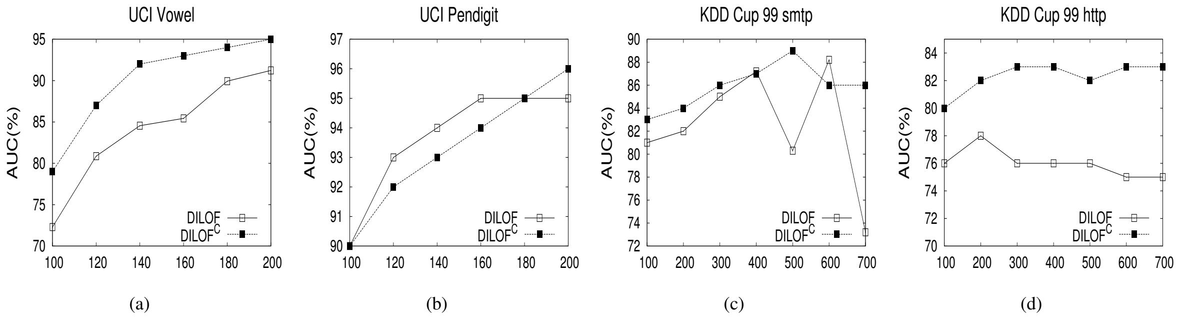

Fig. 7. Outlier Detection Accuracy with Respect to Window Size (DILOF vs. DILOF^C).

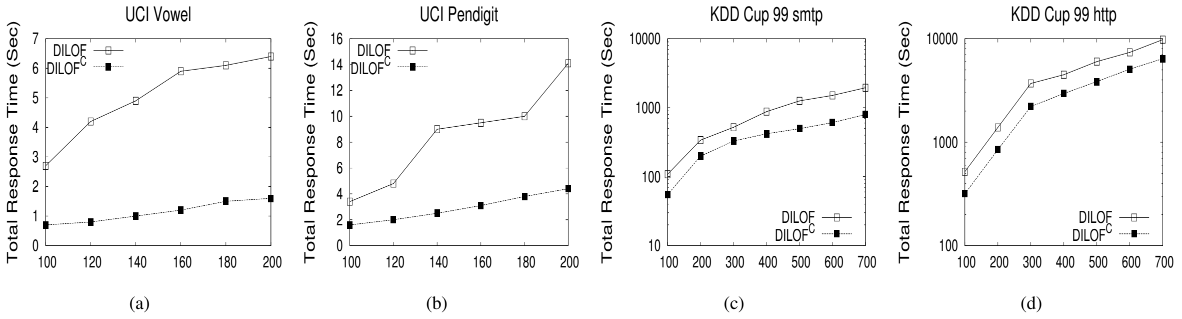

Fig. 8. Total Response Time with Respect to Window Size (DILOF vs. DILOF^C).

DILOF by four factors, three factors, more than two factors, and approximately two factors respectively.

V. Conclusion

Outlier detection over data streams is one important task in data mining. In this paper, we proposed an efficient algorithm called DILOF^C for outlier detection over data streams. Our algorithm used the density based approach. It based on DILOF which is the state-of-the-art density-based algorithm for outlier detection over data streams. Our modification in DILOF as follows. We replace the inefficient summarization method of DILOF by a new efficient summarization method that based on a new clustering technique called m-center. Note that, the time complexity of our summarization method is very small compared to the time complexity of DILOF summarization method. We also optimize m-center clustering technique by merging the nearest clusters. Via an extensive evaluation on real datasets, we show that DILOF^C outperforms the state-of-the-art competitor, DILOF by over two factors. Also DILOF^C achieves very high accuracy for detecting outliers. As future work, we plan to evaluate our method in a real sensor network.

References

[1] R. Domingues, M. Filippone, P. Michiardi, and J. Zouaoui, "A comparative evaluation of outlier detection algorithms," *Experiments and analyses. Pattern Recognit.*, pp. 406–421, 2018.

[2] Y. Yan, L. Cao, and E. Rundensteiner, "Scalable top-n local outlier detection," *In Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Halifax, NS, Canada*, p. 1235–1244, 2017.

[3] S. Cai, R. Huang, J. Chen, C. hang, B. Liu, and S. Y. Y. Geng, "An efficient outlier detection method for data streams based on closed frequent patterns by considering anti-monotonic constraints," *ScienceDirect ELSEVIER*, vol. 555, pp. 125–146, MAY 2021.

[4] A. Senthilkumar and M. Metilda, "A survey on cluster based outlier detection techniques in data stream," *International Journal of Data Mining Techniques and Applications*, vol. 5, 2016.

[5] A. Chandola and V. Kumar, "Article outlier detection," *ACM Computing Surveys*, 2009.

[6] P. Thakkarand, J. Vala, and V. Prajapati, "Survey on outlier detection in data stream," *International Journal of Computer Applications*, vol. 136(2), 2016.

[7] L. Tran, L. Fan, and C. Shahab, "Distance-based outlier detection in data streams," *VLDB Endowment*, vol. 9(12), 2016.

[8] F. Angiulli and F. Fassetti, "Detecting distance-based outliers in streams of data," *in ACM Conference on Information and Knowledge Management*, pp. 811–820, 2007.

[9] A. Gounaris and Y. Manolopoulos, "Continuous monitoring of distance-based outliers over data streams," *International Conference on Data Engineering*, pp. 135–146, 2011.

[10] L. Cao, D. Yangt, Q. Wang, Y. Yu, J. Wang, and E. A. Rundensteiner, "Scalable distance-based outlier detection over high-volume data streams," *International Conference on Data Engineering*, pp. 76–87, 2014.

[11] L. Tran, M. Y. Mun, and C. Shahabi, "Real-time distance-based outlier detection in data streams," *Proceedings of the VLDB Endowment*, vol. 14(2), 2020.

[12] P. Thakkar, J. Vala, and V. Prajapati, "Survey on outlier detection in data stream," *International Journal of Computer Applications*, vol. 136(2), pp. 0975–8887, 2016.

[13] G. S. Na, D. Kim, and H. Yu, "Dilof: Effective and memory efficient local outlier detection in data streams," *In Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '18), London, UK, 19–23 August 2018; Association for Computing Machinery: New York, NY, USA, 2018*, pp. 1993–2002, 2018.

[14] J. Huang, M. Zhong, and B. P. Jaysawal, "Tadilof: Time aware density-based incremental local outlier detection in data streams," *MDPI*, 2020.

[15] T. Kieu, B. Yang, and C. S. Jensen, "Outlier detection for multidimensional time series using deep neural," *In Proceedings of the 2018 19th IEEE International Conference on Mobile Data Management (MDM), Aalborg, Denmark*, pp. 125–134, 2018.

[16] A. Aymen, A. Kachouri, and A. Mahfoudhi, "Outlier detection for wireless sensor networks using density-based clustering approach," *IET Wirel. Sens. Syst*, pp. 83–90, 2017.

[17] M. M. Breunig, H. Kriegel, R. T. Ng, and J. Sander, "Lof: Identifying density-based local outliers," *In Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data (SIGMOD'00), Dallas, TX, USA, 16–18 May 2000; Association for Computing Machinery: New York, NY, USA*, p. 93–104, 2000.

[18] D. Pokrajac, A. Lazarevic, and L. J. Latecki, "Incremental local outlier detection for data streams," *IEEE Symposium on Computational Intelligence and Data Mining*, pp. 504–515, 2007.

[19] M. Salehi, C. Leckie, and C. Leckie, "Fast memory efficient local outlier detection in data streams," *IEEE Trans. Knowl. Data Eng*, vol. 28, pp. 3246–3260, 2016.

[20] S. Chakraborty and N. K. Nagwani, "Analysis and study of incremental k-means clustering algorithm," *In International Conference on High Performance Architecture and Grid Computing; Springer: Berlin/Heidelberg, Germany*, pp. 338–341, 2011.