Journal of Theoretical and Applied Information Technology
15th May 2022. Vol.100. No 9

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

# AN ENHANCED MUDI-STREAM ALGORITHM FOR CLUSTERING DATA STREAM

**MAYAS ALJIBAWI[1,2], MOHD ZAKREE AHMAD NAZRI[3] AND NOR SAMSIAH SANI[4]**

[1,3,4]Center for Artificial Intelligence Technology, Faculty of Information Science and Technology,

Universiti Kebangsaan Malaysia, Bangi, Selangor, Malaysia

[2]Department of computer engineering techniques, Al-Mustaqbal University College, Hillah 51001, Iraq

E-mail: [1]mayasaljibawi@gmail.com, [3]zakree@ukm.edu.my, [4]norsamsiahsani@ukm.edu.my

**ABSTRACT**

Streaming data applications are common due to the advancement of technology to continuously capture or produce data, such as sensors for temperature, humidity and precipitation observations, social media or chatbots. These data applications receiving massive data in real-time requires an efficient algorithm and sufficient memory for analytics. Internet-of-Things (IoT) technologies embedded in a system requires a robust algorithm for clustering the streaming data to support decision making by analysing the historical sensor payloads. The MuDi-Stream algorithm, a density-based method, has emerged as one of the important methods for clustering data streams. The main issue with MuDi-Stream is the number of empty grids increased with the dimensional number or the increase of the streaming speed, making it less efficient when handling high-dimensional data. Furthermore, each point that came to a grid in the online phase will be saved, and with time, these points will consume larger memory space. To overcome these issues, we proposed an enhanced version of MuDi-Stream, coded as eMuDiS. Several benchmark datasets have been used in this study, and the performance of eMuDiS is compared to the state-of-the-art methods, including MuDi-Stream. The experimental results show that the proposed eMuDiS has better memory allocation performance than the MuDi-Stream.

**Keywords:** *Clustering, Data Stream, Multi-Dimensional, Density Grid, Stream Speed.*

## 1. INTRODUCTION

In the Fourth Industrial Revolution era, the interconnectivity between cyber-physical systems, the Internet-of-Things (IoT) and mobile devices produced tremendous amounts of data transmitted in a streaming manner. The data stream can be defined as any sequence of data transmitted over a connection-oriented communication. Large corporations and government agencies generate a vast amount of data at a higher speed than ever. For instance, status changes, precipitation observations, and event responses for emergency operations between users and chatbot are considered streaming data applications [1, 2]. Google treats more than 3.5 billion searches daily, whereas NASA satellites generate approximately 4 TeraBytes of images daily [1]. These tremendous amounts of the data stream have been flooding the network between embedded systems solutions and applications that require analysis to inspect or explore for events or hidden knowledge of interest.

However, the streaming data are so expensive to store for analysis because of their enormous size. The value of the data is invaluable to be ignored and thus attracts data scientists to develop approaches for understanding and finding hidden patterns in data streams. One of the critical data mining tasks is clustering. Clustering is the process of dividing the data into similar groups.

Existing conventional clustering algorithms are not fast enough to cluster data streams because techniques have been designed to be used with static data repositories. The drawbacks of the traditional clustering algorithm have attracted researchers to improve the data stream, particularly in reducing execution time and memory consumption. Clustering the data stream has been defined as dividing the continuous sequence of data such as multimedia, telephone records, and financial transactions into similar groups to improve time and memory consumption [2, 3].

There are five main categories of clustering algorithms, hierarchical-based, partitioning-based,

model-based, grid-based and density-based [4]. Hierarchical-based algorithms aim to analyze specific data to build a hierarchy of clusters [5]. In comparison, partitioning-based algorithms aim at dividing data into a number of clusters using seeds or centroids [6]. Model-based algorithms optimize the fit between the data and some mathematical models. Grid-based algorithms try to divide the data space amount of cells grouped to form the clusters [7]. Finally, density-based algorithms aim at accommodating the clustering among the data points based on their distribution or density [8].

The critical advantage behind the density-based compared to other clustering algorithms is that it has the ability to cluster any data with arbitrary shape. Besides, it has the ability to determine the noise points within a data [4]. Such privilege provides a superiority for the Density-based algorithms regarding handling data streams. Data stream clustering problems have attracted researchers such as [2, 9-11] to propose numerous methods. Researchers have shown a great interest in developing robust clustering algorithms, including DenStream [8], FlockStrream [11], D-Stream [12], MR-Stream [13] and MuDI-Stream [9].

However, handling the multi-dimensional density data is still a challenging obstacle that would face the process of data stream clustering concerning the time and memory used to accommodate such a process.

Another issue is the stream speed which can be defined as "*the number of arriving data points in each time unit*" [14]. Thus, the memory allocation will be affected by increasing the streaming speed which means more saved points in the memory.

From all above, we can conclude there is a need for an algorithm that can cluster the data stream despite the amount of dimensionality in the data and the speed of the stream.

This paper proposed a new variant of MuDi-Stream, a multi-density clustering algorithm for evolving data streams that can cluster high dimensional data with high streaming speed. The rest of the paper can be organized as; Section 2 highlights the related work, Section 3 discusses the proposed method, Section 4 illustrates the experimental results, and Section 5 provides a critical discussion.

## 2. RELATED WORK

Density-based methods is a vital clustering technique that is useful in identifying the noise in the database. Among the density-based algorithm that has been proposed in the past few years, some of them focused on stream clustering include MuDi-Stream [9], DCUStream [15], DenStream [8]. These algorithms used density micro-clustering, grid-clustering, or a hybrid between the micro and grid. However, the main challenge is when there is multi-density data which means the cluster has several densities. In general, not all multi-density clustering algorithms are suitable for stream clustering due to the need for two passes of the data to get the clustering results. High memory allocation is one of the common problems in the existing methods, which can be increased by either increasing the speed of the stream or increasing the dimensionality, which means expanding the points that will be saved to the memory [11] and that will lead to reserve a massive space if memory.

Several multi-density algorithms have been developed in the literature. Forestiero et al. [11] overcome the main problem with the single-pass paradigm approach. The number of clusters must be determined as an input parameter in a single-pass approach. Thus, they cannot capture changes in the data stream because the exact weight is given to both outdated and recent data. Forestiero et al. [11] developed a data stream clustering method based on a multi-agent system that uses a decentralized bottom-up self-organizing strategy to group similar data points. Data points are associated with agents that work simultaneously by applying a heuristic strategy based on a bio-inspired model, known as the flocking model.

Cassisi et al., [17] This method proposed cluster density-based data to find the $\epsilon$-neighbourhood by using the influence space (I.S. instead of the conventional ways. However, this method needs two passes for the data, making it not applicable for the streaming data. Moreover, IS-DBSCAN cannot handle the problem of subspace. DBSCAN-DLP [16] is developed with a different strategy where the density variation is used to get statistical information to divide the data into diverse density levels. The algorithm will define $\epsilon$ for each of the density levels and perform the clustering on that level based on its radius value to get the clustering results. This approach suffers from high computational time and I/O consumption, especially when input data is enormously significant.

High dimensional data is another challenge for clustering any data as it faces two main problems: i) The clustering tendency and ii) The curse of dimensionality [19]. The clustering tendency will lose when the dataset contains irrelevant attributes [19]. Searching for clusters will be difficult when

there are no relevant attributes to build clusters. Attribute selection is the best approach to address the problem of selecting irrelevant attributes. The dimensionality curse is another problem in high dimensional data. SUBCLU [17] is a subspace clustering algorithm that uses a greedy strategy to find clusters in subspaces. For spotting dense areas in the subspaces, SUBCLU has two parameters: radius and minPts. However, because the distance between the objects changes with the change of the subspace, the global settings of radius and minPts is impossible. Bohm et al. [18] proposed an algorithm called PreDeCon based on DBSCAN. It uses the subspace preference vector to define a weighted Euclidean distance. PreDeCon depends on the value of the variance of objects in the radius-neighbourhood if it is smaller than the threshold or not, to adopt the subspace for an object as relevant or not. However, for computing the subspace preference, this algorithm requires two more parameters to the ones needed by DBSCAN In [19] to reduce the processing. Not all subspace clusters will be processed. Instead of that, only the identified promising clusters. The idea of mining only the promising clusters is to get enough information to start processing on a higher dimension with more interest without the need to jump into the between subspace. Moreover, this algorithm steers the process by avoiding scanning the database for much redundant subspace projection.

As a conclusion from the literature, one can notice that there is still a severe drawback that lies behind the state of the art of multi-density clustering approaches. Such a drawback can be represented by the information brought within the online phase, which is being stored in the memory that would consume memories.

Amini et al. [9] Proposed a hybrid method between a grid and micro method to propose a new algorithm called MuDi-Stream. The algorithm handles the noise and multi-density by using the grid-based method. However, the number of empty grids increased with the dimensional number, making the algorithm unsuitable for high-dimensional data. Furthermore, each point that came to a grid in the online phase will be saved, and with time, these points will reserve a massive space of memory, and therefore a larger storage memory will be needed.

The problem with the MuDi-stream algorithm is each point that came to a grid in the online phase will be saved. Therefore, these points will reserve a massive memory space over time, and a bigger storage memory will be needed. This needed memory space is a result of two reasons: the first one appears when the stream speed increases, which means more points will be saved in the grids before they are converted into cmcs (core mini clusters). The second reason will appear when the dimensions of issues increase, and these points will also need massive storage.

## 3. PRELIMINARIES

In order to understand the workflow of the proposed eMuDiS, the following pseudocode shows MuDI-Stream [11] with a memory-laden with complete information.

MuDi-Stream
Input: Data Stream $x$
Output: Core Mini-Clusters (CMC)
Steps:
Update GS($n_g$,$t_p$,$w_g$);
if $n_g > 1$ and $w_g \geq \alpha/N(1-2^{-\lambda})$
new $w_{cmc} \leftarrow w_g$;
$c_{cmc} \leftarrow \sum f(t_p-t_i)(p_i)/w_{cmc}$
$r_{cmc} \leftarrow \sum f(t_p-t_i)$ distance($p_{ij}$,$c_{cmc}$)/$w_{cmc}$

As shown in the pseudocode above, the problem lies in these lines of the algorithm where the points that come to grids within the online phase are being saved, then the cmc parameters are being computed. To this end, five significant parameters are illustrated in MuDi-Stream, which are Weight Coefficient, Grid Coefficient, Core Mini Cluster (CMC), Mini-Core Distance (MCD), and Outlier Weight Threshold (OWT).

Weight coefficient $w_x$ is a variable associated with every data point x within the data stream. Such a variable is declining as much as the data point is getting old. In other words, over time, each data point is getting less important, whereas the coefficient is getting decreased. In order to determine the coefficient, another variable λ is being used along with the time t and current time tc where λ is greater than zero. Based on the parameters mentioned above, the following equation describes the computation of the weight coefficient:

$$wx(t_c, t) = 2\lambda(t_c, t)$$

However, the initial value of $w_x$ is assigned to 1. In addition, the Grid Weight $w_g$ is another parameter that should be taken into account. It refers to the

summation of weight coefficients associated with specific data stream points. For grid g at a particular current time $t_c$, the grid weight is calculated using Equation (1). The following discussion shows some definitions that have been the foundation in both, MuDI-Stream and the proposed EMuDiS algorithm:

I. Data point's weight coefficient: For each data point x in the data stream, a weight coefficient ($w_x$) is assigned, which decreases exponentially over time, i.e., the older a point gets, the less important it gets. The parameter λ is used to control the importance of the historical data of the stream. If x arrives at time t, its weight coefficient at tc is ($t_c 4 > t$): $w_x$ (tc,t) = 2λ(tc,t), λ > 0. The initial $w_x$ value of the data point is 1.

II. Grid weight: For a grid g at current time tc, the grid weight is defined based on the sum of the weight coefficients of data points mapped to it.

$$w_g = \sum x \in g^{2-\lambda(t_c - t_x)} \quad (1)$$

III. Update the grid weight: the update of the grid weight in tc using the last updated value tp as follows:

$$w_g(t_p - t_c) = 2^{-\lambda(t_c - t_x)} * w_g(t_p) + 1 \quad (2)$$

IV. Dense grid: grid g will consider as dense at any time *t,* if the following equation is satisfied:

$$w_g(t) = \alpha/1 - 2^{-\lambda} \quad (3)$$

V. Mini-core distance (mcd): mini-core distance is the maximum distance between the mean of all points to all other neighbourhoods

*VI.* Core mini-Cluster (cmc): a group of very closed points of data $p_i ... p_{in}$.

Once the grid and dense are being formed, multiple clusters in which the Core Mini Cluster (CMC) is a group of very similar/closed data points. Hence, the distance between points is formed based on the Mini-Core Distance (MCD), which refers to the maximum distance between the mean average of a group of points and the mean average of all points within other groups. Therefore, to identify the belongingness of any data point, the Outlier Weight Threshold (OWT) is used, which can be calculated as follow:

VII. Outlier weight threshold (OWT): is defined as:

$$OWT(T_P, T_C) = \frac{\alpha(1-2^{-\lambda(t_c - t_p + 1)}}{N(1-2^{-\lambda})} \quad (4)$$

## 4. eMuDiS ALGORITHM

The eMuDiS algorithm is an enhanced version of the MuDi-Stream algorithm [9]. The proposed algorithm can be represented in the pseudocode of the MuDi-Stream, which is stated as follow:

Mudi-Stream: online phase.
Input: a data stream
Output: core mini-clusters
Steps:
1: $t_{pt} = \left\lceil \frac{1}{\lambda} log_2^{\alpha/(\alpha - N(1-2^{-\lambda}} \right\rceil$
2: $t_c \leftarrow 0$;
3: initialize the grid structure using grid granularity;
4: **while** not the end of stream **do**
5: Read data point x from DataStream;
6: $cmc_s \leftarrow$ find the nearest cmc to x in cmc list;
7: if distance $(x,c_{cmc}) \leq mcd_{cmc}$ then
8: $cmc_s \leftarrow cmc_s + x$;
9: else
10: map the new point x to the grid;
11: $n_g \leftarrow n_g + 1$;
12: $w_g \leftarrow 2^{-\lambda(t_c -)} w_g(t_p) + 1$;
13: $t_p \leftarrow t_c$;
14: Update GG.S. $n_g$, $t_p$, $w_g$);
15: if $n_g > 1$ and $w_g > \frac{\alpha}{1-2^{-\lambda}}$ then
16: new $w_{cmc} \leftarrow w_g$;
17: $C_{cmc} \leftarrow \frac{\sum_{i=1}^{n} f(t_p - T_i)(p_i)}{w_{cmc}}$
18: $r_{cmc} \leftarrow \frac{\sum_{i=1}^{n} f(t_p - T_i) distance(p_{ij}, c_{cmc})}{w_{cmc}}$
19: **for** data points $p_i$ in the grid g **do**
20: $mcd_{cmc} \leftarrow$ Maximum {distance($c_{cmc}$,$p_i$)};
21: end for
22: end if
23: end if
24: if $t_c$ mod $t_p$ == 0 then
25: update the weight of all grids in the grid list
$$w_g(t_c) = 2^{-\lambda(t_c - t_p)} * w_g(t_p)$$
26: **for** all grid g **do**
27: $OWT(T_P, T_C) = \frac{\alpha(1-2^{-\lambda(t_c - t_p + 1)}}{N(1-2^{-\lambda})}$
28: if $w_g <$ OWT then
29: remove grid g from the grid list;
30: end if
31: end for
32: for all { cmc } do

33: if $w_{cmc} < \frac{\alpha}{N(1-2^{-\lambda})}$ then
34: remove cmc from { cmc };
35: end if
36: end for

37: end if
38: tc ← tc+1;
39: end while
40: End

The eMuDiS can be considered a new variant of MuDi-Stream. eMuDiS is an online-offline algorithm that depends on the recursive methods, keeping only the important information about the point in the memory instead of the whole point, which will require fewer memory allocations.

The online phase summarizes the received points' information as cmc, detecting and removing the outlier and the pruning process. In the offline phase, the final clusters will be shaped. Note that the Euclidean distance has been used in this algorithm.

Grid Synopsis (G.S.): The grid synopsis of a grid g is a tuple G.S. ($n_g$, $t_p$, $w_g$) where $n_g$ is the number of data points inside the grid. $t_p$ is the last updated timestamp of the grid, and $w_g$ is the grid weight

The points that come to grids in the online phase should not be saved to solve these problems. Instead of saving points coordinates and then calculating the cmc parameters, the cmc parameters will be updated recursively for each new point come to a grid before it is converted into a cmc. If the point becomes a cmc, the calculated parameters will be assigned to that cmc.

The equations which have been used to update the centre of cmc and the radios when a new point has arrived in the following:

$$New\ center = \frac{2^{-\lambda(t_c - t_p)} \times Old\ w_g \times Old\ center + New\ point}{New\ w_g} \quad (5)$$

$$New\ radious = \frac{2^{-\lambda(t_c - t_p)} r_{cmc} old w_{cmc} + dist(p_{new}, c_{cmc})}{new w_{cmc}} \quad (6)$$

The above equations can be applied on the MuDi-stream code, specifically on the update steps (i.e., step 17 and step 18):

14: Update GG.S. $n_g$, $t_p$, $w_g$);

15: if ng > 1 and $w_g > \frac{\alpha}{1-2^{-\lambda}}$then

16: new wcmc ← wg;

17: $C_{cmc} \leftarrow \frac{2^{-\lambda(t_c - t_p)} old w_g C_{cmc} + p_{new}}{new w_g}$

18:

$$r_{cmc} \leftarrow \frac{2^{-\lambda(t_c - t_p)} r_{cmc} old w_{cmc} + dist(p_{new}, c_{cmc})}{new w_{cmc}}$$

## 5. EXPERIMENT AND RESULTS

In this section, results and discussion will be presented. We applied eMuDiS and made the MuDi-Stream as a comparative algorithm. The evaluation will be mainly based on memory usage in which two factors will be considered along with the memory allocation, stream speed and dimensionality.

### 5.1 Dataset and Setup

Real and synthetic datasets were used to evaluate the number of the saved points of the EMuDiS algorithm. Real datasets that were used in this paper are 1) Network Intrusion Detection dataset (KDD Cup'99) [12], which has almost 5 million connection records of training and network-based intrusion sand standard data made by DARPA and where used by [8, 9, 11, 20].

This dataset was converted to be a stream data set by taking the data input order as the order of stream. 2) The land sat satellite data consisting of over 4000 objects, were collected from remote-sensing satellite images. 3*3 regions combined to represent each data object, where the four intensity measures are taken at a different wavelength. Thus, this would lead the objects to be composed of 36 attributes. Moreover, each data object has been given a class label to show the central sub-region type. DS1, DS.2 and DS3 are synthetic data sets used in the experiment. They are shown in Figures 1a, 1b, and 1c, respectively. DS1 has a 12% noise out of 8000 data points with six clusters. DS2 has 2990 data points having nine clusters, and DS3 has 1000 data points with four clusters. The parameters of EMuDiS and MuDi-Stream adopt the following settings for the speed of stream experiment: decay factor λ=0.998, the minimum number of points MinPts=5, stream speed 100-1000 with step size 100. In the second experiment for dimensionality, a new random point for each cluster has been generated depending on the

Gaussian distribution; then, some random points will be generated around each point to make a dataset with a specific number of dimensions. The second experiment used the parameters to adopt the following settings:

i. decay factor λ=0.998,

ii. the minimum number of points MinPts=5,

iii. Stream speed 100

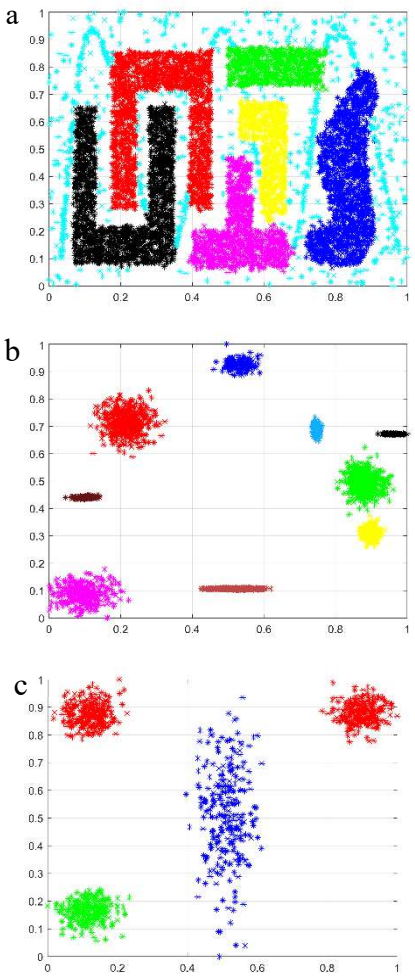


*Figure 1: Synthetic Datasets: (A) Dataset 1 (DS1); (B) Dataset 2 (DS2) And (C) Dataset 3 (DS3)*

### 5.2 Results

The experimental results are divided into two groups, including stream speed and stream dimensionality. In the first group, the proposed clustering method of eMuDiS and the baseline clustering method of MuDi-stream are examined to handle stream data clustering within different speed limits. At the same time, the second group examines both methods in terms of stream data clustering within different dimensions. The results are measured based on the number of the saved points in the memory by the proposed method and the MuDi-Stream with the increase of both streaming speed and the dimensionality. These sub-sections depict the two groups of the experiment.

#### 5.2.1 Stream Speed

As mentioned earlier, the results of applying both the proposed eMuDiS and the baseline MuDi-stream are being highlighted in this section. The stream speed is the number of arriving data points in each time unit [11]. Note that the results are divided upon the four datasets with different speed limits ranging from 100 to 1000.

The first experiment was aimed to explore the effect of changing the stream speed on the memory allocation of both EMuDiS and MuDi-stream. The results have shown that the EMuDiS is superior in memory efficiency to the MuDi-stream.

Figures 2 to 5 show the number of stored points for eMuDiS compared to MuDi-Stream on different stream speed range between 100-1000 point per time unit for the datasets.

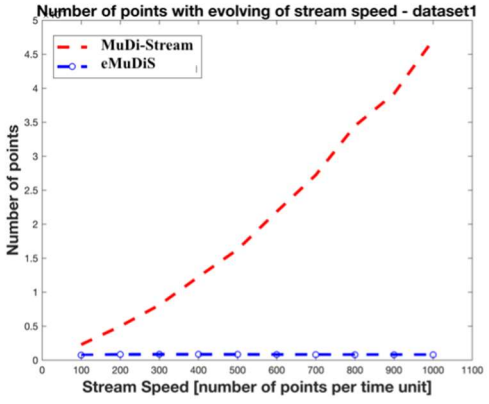


*Figure 2 Emudis Memory Allocation (Dataset 1)*

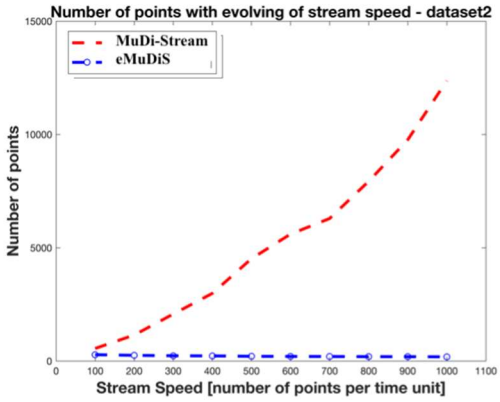


*Figure 3 Emudis Memory Allocation (Dataset 2)*

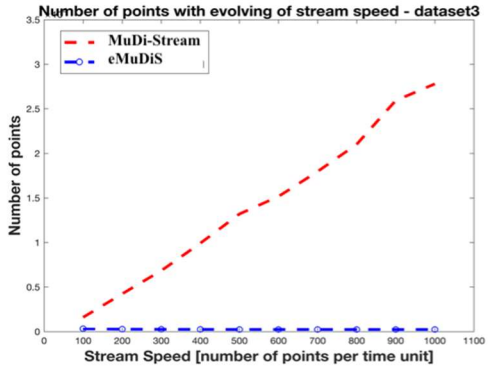


*Figure 4 Emudis Memory Allocation (Dataset 3)*

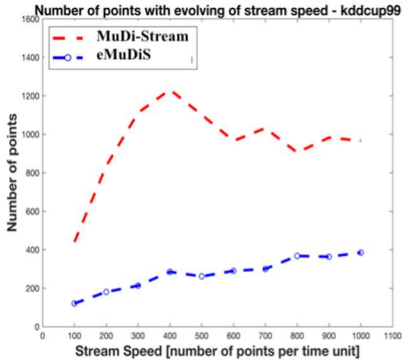


*Figure 5 Emudis Memory Allocation (KDDCUP9 Dataset)*

Comparing the different stream speeds reveals that our approach is less affected by increasing speed than MuDi-stream, which has shown an exponential increase in memory allocation as the speed stream increases. eMuDiS outperforms MuDi-stream over all the time of the experiment.

Figure 6 to 10 shows the memory allocation results through different speed limits, including 100, 200, 300, 400, 500 and 1000. The charts showed the performance of eMuDiS against the MuDi-stream when Dataset 1 was applied.

As shown in Figure 6, the results of memory allocation when the stream speed was 100 indicate that the proposed eMuDiS method required a lower number of data points (i.e., roughly 1000 data points) than the baseline method MuDi-stream where more than 4000 data points have been occupied. Whereas, when the speed of the stream was 200 in Figure 7, the proposed method showed a different small number of data points occupied (i.e., around 1000 data points) compared to the huge number of data points occupied by the baseline method (i.e., more than 9000).

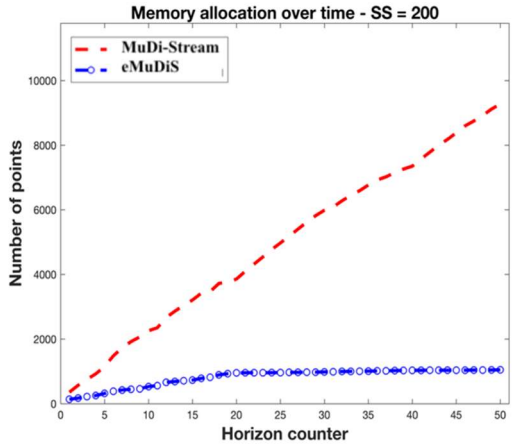


*Figure 6 Memory Allocations When Speed Was 100*

In addition, when the stream speed reaches 300, as shown in Figure 8, the proposed method still obtained approximately the same amount of data points compared to the baseline method that gained a bigger number of data points (i.e., more than 14000). After that, when the stream speed increased to 400, as shown in Figure 9, the proposed method slightly got a higher number of data points (i.e., more than 1200) compared to the baseline method that dramatically witnessed a larger number of data points (i.e., more than 20000).

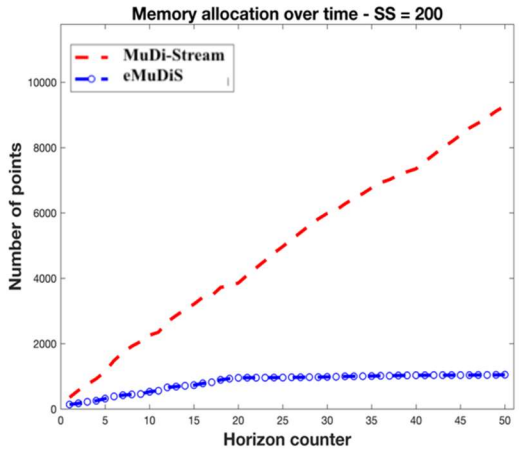


*Figure 7 Memory Allocations When Speed Was 200*

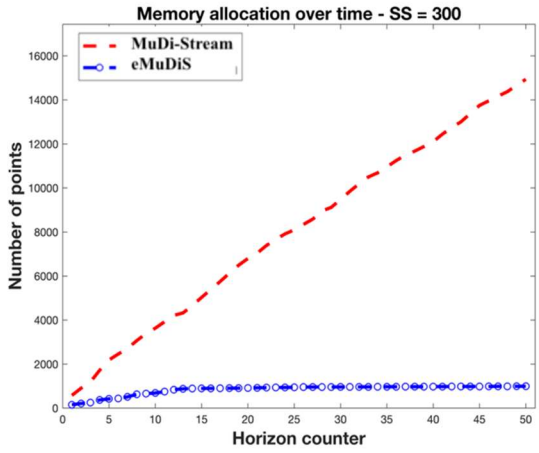


*Figure 8 Memory Allocations When Speed Was 300*

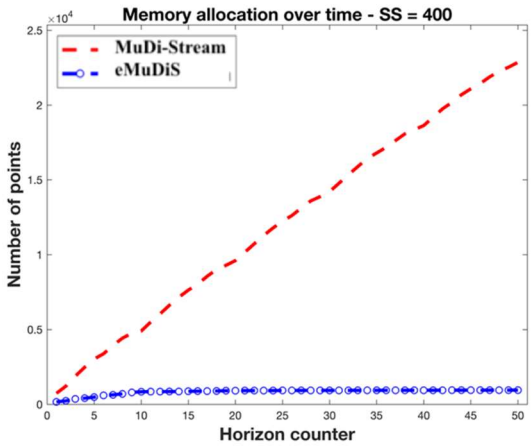


*Figure 9 Memory Allocations When Speed Was 400*

On the other hand, when stream speed reaches 500, as shown in Figure 10, the proposed method still shows similar and steady performance, whereas the baseline method saves more points over time. This has continued until the stream speed reaches 1000 in Figure 11, where the proposed method shows steady performance, and the baseline breaks the number of 90000 data points.

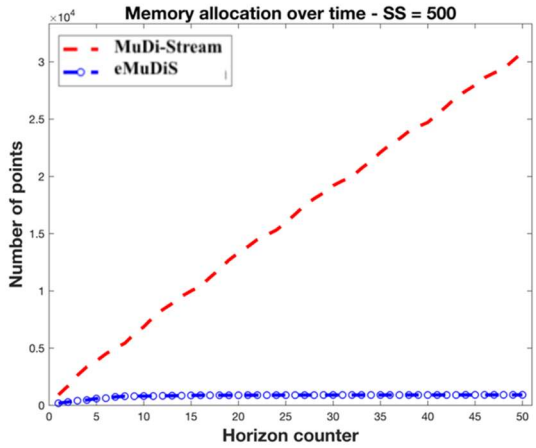


*Figure 10 Memory Allocations When Speed Was 500*

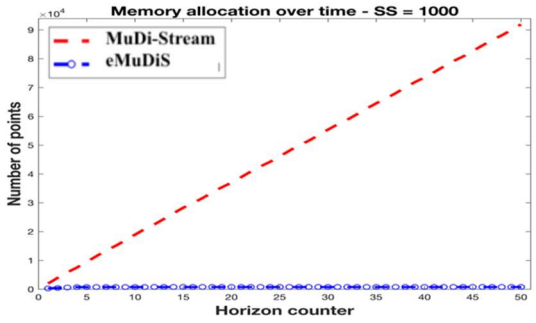


*Figure 11 Memory Allocations When Speed Was 1000*

### 5.2.2 Stream Dimension

In terms of dimensionality, this section examines the capabilities of both the proposed eMuDiS and the baseline of MuDi-Stream regarding stream data clustering within different dimensions. Figure 4.6 shows such results.

As shown in Figure 12, when the dimension was 2, the proposed method showed a smaller number of data points (i.e., around 50) than the baseline methods, which started with 450 as the number of

data points. When the dimension adjusted as 3 in Figure 13, the proposed method showed an increase in the number of data points (i.e., 250), as well as the baseline (i.e., 3400).

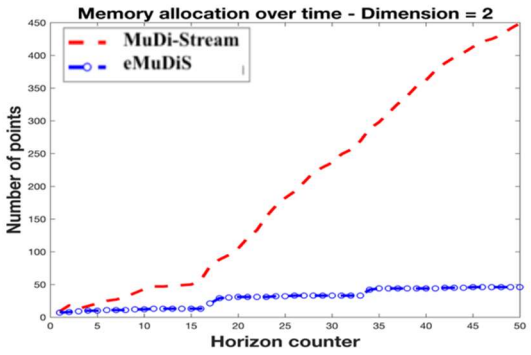


*Figure 12 Stream Dimension = 2*

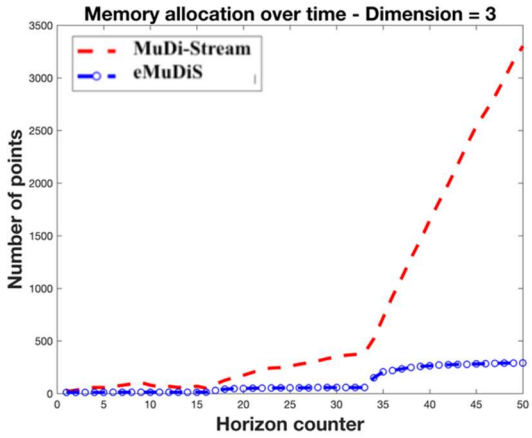


*Figure 13 Stream Dimension = 3*

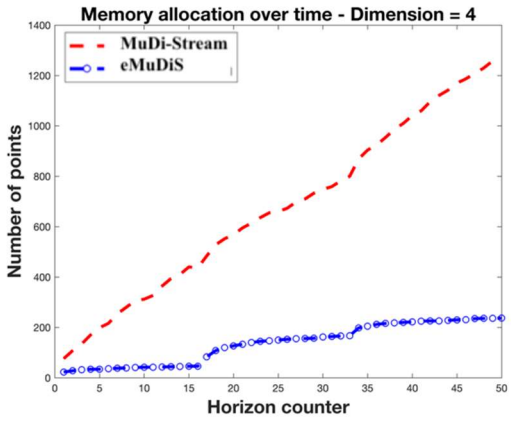


*Figure 14 Stream Dimension = 4*

After that, when the dimension was set to 4, as shown in Figure 14, both the proposed and baseline methods witnessed a drop in the number of data points. The eMuDiS occupied around 200 data points and the baseline settled around 1200 data points. This has been followed by a gradual increase in the number of data points for both the proposed and the baseline method until the dimension reaches 10, displayed in Figure 15. The proposed method showed around 5000 data points, and the baseline showed around 6500 data points.

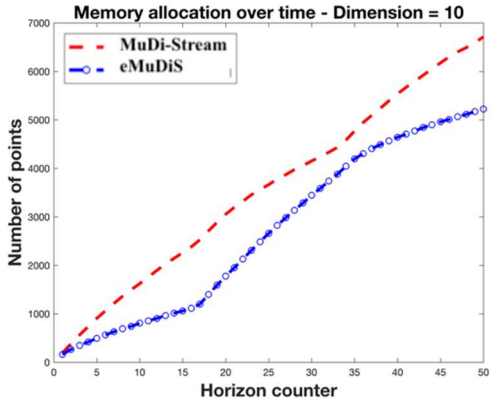


*Figure 15 Stream Dimension 10*

Note that there are some variations in terms of the gap between the proposed and baseline among the datasets. The gaps between the proposed eMuDiS and MuDi-Stream on synthetic datasets were significantly higher than the real datasets (i.e. e KDD-CUP99's gaps. The KDD-CUP99 contain many data records (i.e., ~ millions). Besides, plenty of noisy and duplicated data are located in such a dataset. Hence, the superiority of the proposed eMuDiS over the MuDi-Stream was comparable to the performance in the syntectic datasets.

Yet, the general results show that the proposed work has proven that saving only necessary information to the memory by the proposed MDSS-stream would have better and more efficient performance compared to the mechanism of saving all the information to the memory.

## 6. CONCLUSION

In this paper, a new algorithm to cluster data stream of multi-density high dimensional data with fast streaming speed EMuDiS by hybridizing micro and grid clustering on the online-offline, respectively is proposed. A recursive method was used to overcome the memory allocation problem. the algorithm recursively updates the cmc parameter for each new point instead of saving the point's coordinate that comes to the grid in the online phase.

This procedure will be repeated until this grid became a cmc. The proposed method has been examined in terms of two significant criteria (stream speed and stream dimension). The results on different real and synthetic data sets show that the performance of our methods outperforms the MuDi-Stream. As shown in the above figures, it is obvious that the changes and enhancements made on the original MuDi-Stream framework have better results than the original MuDi-Stream in terms of allocating a lower number of memory data points. This has occurred in both circumstances of speed change and dimension change, as well as for all the five datasets. As future work, we will work on the offline phase to increase the clustering quality while reducing the memory allocation and compare the results with more existing algorithms such as D-Stream and DenStream.

**Acknowledgement:**

This research is supported by the Universiti Kebangsaan Malaysia Research University Grant GUP-2020-091 and FTSM Community Transformation Grant TT-2020-016.

**REFERENCES:**

[1] Ma, W.H. *Survey on data streams clustering techniques*. in *Advanced Materials Research*. 2014. Trans Tech Publ.

[2] Jung, J.J.J.E.S.w.A., *Semantic preprocessing for mining sensor streams from heterogeneous environments.* 2011. **38**(5): p. 6107-6111.

[3] Aggarwal, C.C., N. Ashish, and A. Sheth, *The internet of things: A survey from the data-centric perspective*, in *Managing and mining sensor data*. 2013, Springer. p. 383-428.

[4] Han, J., J. Pei, and M. Kamber, *Data mining: concepts and techniques*. 2011: Elsevier.

[5] Aljibawi, M., M.Z.A. Nazri, and Z.J.I.J.E.T. Othman, *A survey on clustering density based data stream algorithms.* 2018. **7**(36): p. 147-153.

[6] Wei, C.-P., Y.-H. Lee, and C.-M.J.E.S.w.a. Hsu, *Empirical comparison of fast partitioning-based clustering algorithms for large data sets.* 2003. **24**(4): p. 351-363.

[7] Wu, B. and B.M.J.I.T.o.I.I. Wilamowski, *A fast density and grid based clustering method for data with arbitrary shapes and noise.* 2016. **13**(4): p. 1620-1628.

[8] Cao, F., et al. *Density-based clustering over an evolving data stream with noise*. in *Proceedings of the 2006 SIAM international conference on data mining*. 2006. SIAM.

[9] Amini, A., et al., *MuDi-Stream: A multi density clustering algorithm for evolving data stream.* Journal of Network and Computer Applications, 2016. **59**: p. 370-385.

[10] Aggarwal, C.C. and C.K. Reddy, *Data clustering: algorithms and applications. Chap man and Hall*. 2013, CRC Press, Boca Raton, Florida.

[11] Forestiero, A., et al., *A single pass algorithm for clustering evolving data streams based on swarm intelligence.* Data Mining and Knowledge Discovery 2013. **26**(1): p. 1-26.

[12] Tu, L. and Y.J.A.T.o.K.D.f.D. Chen, *Stream data clustering based on grid density and attraction.* 2009. **3**(3): p. 1-27.

[13] Wan, L., et al., *Density-based clustering of data streams at multiple resolutions.* ACM Transactions on Knowledge discovery from Data (TKDD) 2009. **3**(3): p. 14.

[14] Amini, A., T.Y. Wah, and H. Saboohi, *On density-based data streams clustering algorithms: A survey.* Journal of Computer Science and Technology, 2014. **29**(1): p. 116-141.

[15] Yang, Y., et al. *Dynamic density-based clustering algorithm over uncertain data streams*. in *2012 9th international conference on fuzzy systems and knowledge discovery*. 2012. IEEE.

[16] Xiong, Z., et al., *Multi-density DBSCAN algorithm based on density levels partitioning.* 2012. **9**(10): p. 2739-2749.

[17] Kailing, K., H.-P. Kriegel, and P. Kröger. *Density-connected subspace clustering for high-dimensional data*. in *Proceedings of the 2004 SIAM international conference on data mining*. 2004. SIAM.

[18] Bohm, C., et al. *Density connected clustering with local subspace preferences*. in *Fourth IEEE International Conference on Data Mining (ICDM'04)*. 2004. IEEE.

[19] Jahirabadkar, S. and P.J.I.J.o.C.A. Kulkarni, *Clustering for high dimensional data: density based subspace clustering algorithms.* 2013. **63**(20).

[20] Li, X., et al., *On cluster tree for nested and multi-density data clustering.* 2010. **43**(9): p. 3130-3143.