DynamicEvoStream: An EvoStream based Algorithm for Dynamically Determining The Number of Clusters in Data Streams

Z. Amighi[1], M. Yousef Sanati[2], M. Dezfulian[3]

Department of Computer Engineering, Faculty of Engineering, Bu-Ali Sina University, Hamedan, Iran.
e-mail: [1]amighi.zahra.ac@gmail.com, [2]mysanati@basu.ac.ir, [3]dezfoulian@basu.ac.ir
[*]Corresponding author: mysanati@basu.ac.ir

Abstract
EvoStream is a stream clustering algorithm which gradually clusters data in the idle times of the stream. In comparison with other algorithms in this field, EvoStream has a lower computation overload in the offline phase and has better accuracy. Also, in this algorithm, the number of clusters is taken as constant whereas in an authentic stream this number varies with the complexity of input data. In this work, we present DynamicEvoStream as an improved version of the original EvoStream. In this algorithm, we detect and exploit variations in the distribution and speed of the stream. Also, we modified the cleanup function to merge overlapping clusters. Therefore, in contrast to the basic EvoStream, DynamicEvoStream identifies the number of clusters in a dynamic manner. Also, the speed of evolutionary steps is increased while improving the quality of the clusters. Finally, experiments using DynamicEvoStream on different streams showed that it can cluster the stream up to four times faster than the original EvoStream with fewer computation and memory resources. In the worst case, the quality of clusters is competitive to the original EvoStream, however improves the quality of clusters up to 30% in the majority of cases.

Keywords
Data stream, Evostream Algorithm, Data stream clustering, Dynamic clustering, Online Clustering.

1. Introduction

In today's world, the emergence of Internet of Things (IoT)[1], advancements in hardware technologies, the large number of websites and large companies such as Amazon have led to the production of huge amounts of various data at an increasing pace. These data can be processed in two ways: batch processing or stream processing. In batch processing, the data are assumed to be static and the results of the process will be available as soon as the computation on all data has been finished. In contrast, in some applications, all the required data for processing are not available and are produced over time. These data are given to a data stream management system (DSMS) through one or more ongoing streams in order to be processed in a real-time and almost sequential manner. In data stream processing, the results should be prepared with minimal delay and the result of each process should be reflected in the last output in a few seconds. From a commercial point of view, data stream processing provides a competitive advantage that helps companies and organizations to achieve new insights and customize their services. This can result in better and more rapid realization of organizational objectives. The reason is that, in response to the changing circumstances of business environments, organizations are now compelled to process large amounts of data as quickly as possible and adapt themselves to the changes based on the results of these processes. Thus, demands for stream processing are increasing [2-4].

A data stream can be defined as an incessant sequence of data which are produced in large amounts at a high rate. In other words, it is a sequence of data objects in time intervals. Given this definition, processing the information within a stream as a single entity is extremely difficult and sometimes impossible. Therefore, a number of methods have so far been proposed to facilitate the processing of such data. One of the most common methods is clustering whereby similar information items are placed into a group [5-7].

Researchers have developed numerous methods for stream clustering. The majority of the proposed algorithms make use of a two-phase approach consisting of online and offline phases. Each of these phases is conducted using online and offline components, respectively. The online component summarizes the stream in a real-time manner. As a result, small primary clusters are created in the stream called “micro-clusters” which indicate the dense areas in the data space of the stream. As soon as clustering is requested, an offline component converts these micro-clusters into macro-clusters which indicate the finally identified clusters in the stream. In cases where data generation in the stream is slow, the offline component becomes idle and waits for the next observation to come. This will waste the resources of the system. EvoStream is an algorithm proposed to overcome this problem. This algorithm uses idle times for building, updating, and modifying clusters in order to efficiently reduce the computation overload of the offline phase [6].

In EvoStream, the number of final clusters is given as an input parameter, which may lead to incorrect results in non-uniform streams. In addition, the offline component in EvoStream uses genetic algorithm. If the clustering request is received before the completion of evolutionary

steps, the algorithm will need an auxiliary component to modify the results of the offline component. The other input parameter is a constant value which specifies the threshold of initializing final solutions or macro-clusters. Even if the radius is determined with relatively high precision, the constancy of this value in streams with non-uniform distribution will break a single cluster into several ones and decrease the quality of clusters.

To overcome these issues, the present paper proposes an algorithm that dynamically determines the number of clusters using the notion of shared radius among clusters and based on the DBSCAN algorithm [8]. Upon clustering request, a new offline component will be used if the execution of evolutionary steps has not been completed in previous idle intervals or a change in the stream distribution has invalidated the results of the previous evolutionary step. To put it more exactly, by using this algorithm, the clustering request will never remain unhandled and the quality of clustering will always be acceptable. In this component, some concepts existing in DBSCAN algorithm are used for finding the number of clusters and clustering the data. To improve the quality of clusters, micro-clusters with the highest fitness are selected as primary clusters.

To evaluate our proposed algorithm, it was tested using real-world data streams and the results were compared with the execution of EvoStream. As EvoStream has already been compared in [6] with many of the existing algorithms, comparing our algorithm with EvoStream can be a good measure of desirability and efficiency.

In summary, the paper contributions are

1) Determinig the number of clusters dynamically
2) Increasing the quality of clusters
3) Increasing the clustering speed
4) Decreasing memory consumption and caluculation requirements

The paper is organized as follows. Section 2 explains the fundamentals of stream clustering and reviews the literature. Section 3 discusses the concepts of evolutionary optimization and EvoStream algorithm. Section 4 introduces the new algorithm for stream clustering which is called DynamicEvoStream and Section 5 evaluates the performance of this algorithm. The final section draws conclusions and describes future directions of research in this area.

2. Related works

In general, clustering is aimed at identifying patterns and groups of similar objects in the data [۹]. Traditional clustering algorithms require several passes of data as well as random access to the data to be able to cluster the data. If a new piece of data is added, the entire model must be reconstructed, which is ineffective in terms of time and costs. On the other hand, when there is a large amount of data, multi-pass clustering algorithms are not efficient; therefore, single-pass algorithms have been proposed as a solution. During the creation of the clustering model, these algorithms process each data item only once. However, these methods are still inefficient for many applications where the data are produced as a continuous stream of observations over time. It should be noted that, in these applications, there is not a fixed set of data [6].

To satisfy the conditions of data stream processing such as unlimited volume of data and the impossibility of storing all the observations as well as high rates of data production and the necessity of real-time processing, the first proposed approaches to stream clustering revolved around incremental learning. In this kind of algorithms, a statistical model for the data is generated based on stream distribution and evolves over time. Also, to overcome memory limitations, only one or more representatives are stored instead of all the observations. Examples of this

Table 1 - Stream clustering algorithms[14-24].

Algorithm	Year	Pros	Cons
DenStream	2006	- Handling arbitrary shapes clusters - Detection of outliners	- Time complexity
CluDistream	2007	- Handling missing and noise data	- Based on the landmark window scenario - Parameter sensitivity
D-Stream	2009	- Detecting arbitrary shapes clusters	- Time complexity when handling high dimensional data
SWEM	2009	- Handling missing data - Handling memory limitation	- Parameter sensitivity
SNCStream	2015	- Quality clusters accordingly to the CMM - Scale-free model	- Parameter sensitivity
DCSTREAM	2016	- Memory efficiency - Handling concept drift	- Outlier detection
CEDAS	2017	- Handling noise - Drift and anomaly detection	- Handling high dimensional data - Memory requirement
SODA	2018	- High quality of clusters - Computation efficiency	- Initial seed dataset
FStream	2018	- Fewer parameter to define	- Training set requirement
DGStream	2019	- Handling outliers - Handeling noise - Detecting arbitrary shapes clusters	- Need to detect dense grids
ACSC	2019	- Few parameters required - Handeling noise	- Parameter sensitivity - Not able to detect arbitrary shapes clusters - Not able to work well in multi density data

approach can be found in [10-13]. Since in incremental learning approaches a model should be built, many initial parameters must be provided and the computation complexity is high. Thus, implementation of such algorithms is difficult. In these algorithms, when the model is updated after new observations arrive, some clusters are merged or removed. In the course of observing the stream, some of the previous instances of merging may turn out to have been incorrect. But as the representatives of previous clusters have been removed, the merged clusters cannot be returned to the original clusters [25-27].

A proposed solution to this problem was two-phase online-offline learning. In the online phase, the received data are summarized in real time [28] and micro-clusters are formed. The number of micro-clusters is greater than the main clusters in the stream. In the offline phase, the final clustering is performed with the help of micro-clusters and the response is sent to the user. The two-phase method is able to easily handle large streams. The major issue in this method is that users must wait for the data required for their request to be received, summarized, and stored by the system [25-27]. The majority of stream clustering algorithms make use of the two-phase method. More than 60 important algorithms for stream clustering have so far been developed. Table 1 lists some of the important algorithms by the year of publication.

3. EvoStream: Evolutionary stream Clustering utilizing idle times

As mentioned earlier, the majority of stream clustering algorithms make use of the two-phase learning method. In these algorithms, the online and offline phases are performed by the online and offline components, respectively. The online component is responsible for analyzing the stream and creating the necessary elements of processing for the offline component. For this reason, in cases where the speed of stream is slow, this component becomes idle and waits for the next observation to arrive. This will waste the resources of the system. EvoStream is a stream clustering algorithm that has been proposed as a solution. This algorithm utilizing idle times for building, refining, and modifying clusters in order to efficiently reduce the computation overload of the offline phase.

The online component processes the arriving data as quickly as possible and uses DBSTREAM algorithm strategy for summarization of the data. Researchers in [28] compared the most popular stream clustering algorithms and showed that the clusters produced by DBSTREAM had the highest quality and the lowest computation time. In EvoStream, every micro-cluster is described as a tuple (c, t, ω) in which c is the cluster's centre, t is the last time the micro-cluster was updated, and ω is the weight of the micro-cluster. If the new observation x falls within the radius R from the centre c, the online component assigns the observation to this micro-cluster. Following the idea of competitive learning, the cluster or clusters which have absorbed x will move toward x by a magnitude determined by the Gaussian function (Equation 1):

$$h(x\,.\,c) = \exp\left(-\frac{(|x-c|^2/3r)^2}{2}\right) \quad (1)$$

If x is absorbed by multiple clusters, all of them will move toward x. After absorbing a new observation, the t and ω of any micro-cluster that has absorbed that observation will be updated. If the observation is not absorbed by any cluster, a new cluster will be created in its place. In addition, the online component of EvoStream executes a cleanup function in regular intervals. This function merges the micro-clusters which overlap by more than half of their radii. The reason for doing this is that the observations absorbed by these micro-clusters may probably belong to an identical micro-cluster. Also, micro-clusters whose weight is less than an acceptable threshold will be recognized as outdated and eliminated. After creating an γ for the micro-cluster (a specified threshold), the online component initializes macro-clusters C by random use of micro-clusters. In the experiments, it was assumed that $\gamma = 2k$.

The offline component of the algorithm is executed when either there is no new observation or the stream is running slowly. Determining whether a stream is active or inactive is difficult and usually requires expert knowledge. In its offline component, EvoStream makes use of an evolutionary algorithm similar to genetic algorithms [29] in order to gradually improve the macro-clusters which are initialized in the online phase.

In a genetic algorithm, first, a population of possible solutions to the problem is selected or generated and, then, the solutions evolve based on the rules of selection and other operators such as recombination and mutation. The quality of each individual from the population is assessed by the fitness function. Each individual is coded using a string representation called a chromosome. Each chromosome consists of several genes. Hereafter, we shall use the term chromosome instead of individual for the sake of facilitating our discussion. To create a new population, chromosomes with higher fitness are selected as parents and begin to exchange genes by using recombination and mutation operators. This process results in the generation of new chromosomes called offspring. The offspring chromosomes will replace the chromosomes in the existing population if their fitness is higher. This repetitive process will continue until the condition(s) of the termination of the algorithm is satisfied [6, 30].

One of the most common evolutionary algorithms for data clustering is GA-clustering [31]. In EvoStream, the offline component uses a method similar to GA-clustering which has been adapted to the two-phase learning approach to stream clustering. The offline component makes use of fast clustering by utilizing cluster variance to evaluate the solutions. The number of clusters is considered as fixed and the solution of clustering the data space is stored as a string which contains the centres of the discovered clusters. This string represents the concept of chromosome in the genetic algorithm. For instance, in a d-dimensional space with two clusters, the first d genes of the chromosome indicate the dimensions of the centre of the first cluster while the

next d genes show the centre of the second cluster. Also, if the number of clusters is k and each cluster centre has d attributes, each chromosome is coded in form of a k*d string as a solution to data clustering. To create each chromosome in the starting population of the genetic algorithm, the cluster centres for each solution are randomly selected. Next, the remaining points on the data which have not been selected as centres are assigned to the closest cluster centre as in the k-means algorithm. It should be noted that the appropriate number of individuals in the starting population should be determined beforehand by the expert of the system. The original chromosomes are not necessarily optimum solutions to clustering and are likely to evolve during the execution of the genetic algorithm [6].

After generating the initial population, the fitness of the chromosomes is evaluated and two chromosomes with the highest fitness are selected as parents using a simple Roulette wheel. The shorter the distance inside clusters, the higher the fitness of the chromosome. Then the selected parents will be used for recombination and generation of new offspring. The recombination uses single-point crossover whereby a point is randomly selected on the chromosome and the parents' genes are exchanged as in Figure 1, which results in two offspring.

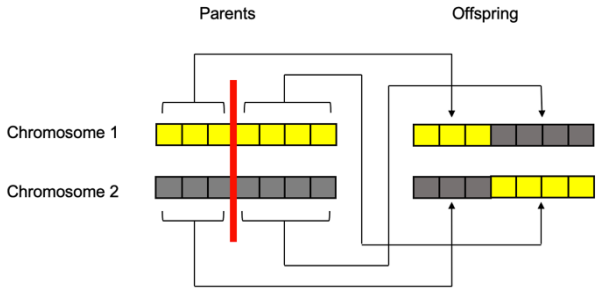

Fig. 1. An instance of single-point crossover in gene 3 (the red line)

Next, all the genes of the offspring (g_i) mutate with a specific probability (Pm) which amounts to the probability of mutation in the GA-clustering algorithm. Therefore, the value of $2\delta \cdot g_i$ (where δ is a random variable in [0, 1]) either increases or decreases with equal probability. If $g_i=0$, the value of the i-th gene mutates to 2δ. Finally, the offspring are evaluated by the fitness function and they will replace previous chromosomes if their fitness is greater than that of the current population. This process will continue until the condition for the termination of the genetic algorithm (i.e. achieving a certain number of generations) is satisfied. Thus, each generation is better than the previous generation.

In EvoStream, the fitness function is calculated using the sum of squares (SSQ). Equation 2 calculates the sum of squares.

$$SSQ = \sum_{i=1}^{K} \sum_{mc \in C_i} (mc[c] - C_i)^2 \qquad (2)$$

The fitness function in EvoStream is defined as f=1/SSQ. Whenever the final clusters are requested, the chromosome with the highest fitness will be presented as the best clustering. In this algorithm, the number of clusters is given as an input parameter. This value is considered fixed and should be provided by the expert who is expected to have sufficient knowledge of the stream. Importantly, the number of clusters in real-world streams cannot be regarded as fixed in all moments due to the complexity of input data. On the other hand, even if the number of clusters is assumed to be fixed in the stream, the expert's error in recognizing the correct number of clusters may result in faulty results.

The offline component in EvoStream uses genetic algorithm. As the idle time of a stream does not follow a periodic pattern with a specific periodicity, if the duration of the execution of the evolutionary steps of the genetic algorithm is longer than the idle time, it will be stopped and we have to wait for the next idle time to complete the steps. If the clustering request is received before the completion of evolutionary steps, the algorithm will need an auxiliary component to modify the results of the offline component. Otherwise, the quality of the clusters will severely deteriorate. In addition, EvoStream considers the changes of micro-clusters between two idle times by updating the fitness function of the micro-clusters at the beginning of each idle time. If the number of clusters in the stream is variable, it is no longer possible to use the results of the previous idle time and complete the evolutionary steps by merely updating their fitness function. Instead, the genetic algorithm should be executed once again.

Another constant parameter is the initialization threshold of final solutions or macro-clusters. In streams with non-uniform distribution, if the radius is determined with relatively high precision, assuming the constant parameter may break a cluster into several clusters and decrease the quality of clusters, which could waste memory and increase the computational load. In the worst case, the number of micro-clusters never reaches the threshold; therefore, the evolutionary algorithm will never be executed. As a result, DynamicEvoStream algorithm was is proposed here to overcome the mentioned problems.

4. DynamicEvoStream algorithm

This algorithm dynamically recognizes the number of clusters using the notion of the common radius of clusters and based on DBSCAN algorithm. As long as the stream is active, the new observations are read and processed similarly to what happens in EvoStream. The pseudocode of the online component is summarized in Algorithm 1.

In the online component, a cleanup function is executed in fixed periods as in EvoStream. This function merges overlapping micro-clusters by calculating their overlap radius. To remove outdated micro-clusters or noises, the cleanup function uses an exponential fading function. Using this function, the weight of each micro-cluster in each time unit is reduced by $2^{-\lambda}$. When the weight of a micro-cluster reaches below a fading threshold, it is removed by the cleanup function. As removing or merging clusters may indicate changes in the distribution of the stream, the proposed algorithm defines a flag called Change_stream_distribution for representing the probability of change in stream distribution which is activated in the event of removing or merging clusters.

The pseudocode of the cleanup function is summarized in Algorithm 2.

In the online component, the two flags slowstream and Change_stream_distribution are checked when responding to clustering requests. Slowstream flag is used to recognize the speed of the stream. This flag is activated when the stream is slow and the algorithm can achieve clustering with acceptable quality by completing the evolutionary steps of the genetic algorithm in idle times (Algorithm 1, line 2). When responding to clustering requests, a disabled speed flag means that the speed of the stream is high and the algorithm has not succeeded in completing the evolutionary steps. As a result, DynamicEvoStream uses a new offline component to present the result in high-speed streams. After the stream speed flag is checked, the stream distribution change flag will be checked if the stream is slow. If Change_stream_distribution is enabled, it indicates that the clustering from the evolutionary algorithm in the previous idle time is no longer valid because the cleanup function was executed after the idle time and removed or merged several micro-clusters. In this case, DynamicEvoStream uses the new offline component (Algorithm 3) for clustering. If the stream distribution change flag is disabled, the online component will return the results of the genetic algorithm which were obtained in the previous idle time.

The new offline component makes use of the concepts of DBSCAN algorithm to discover the number of clusters. First, the micro-clusters are sorted by their fitness in descending order. Next, for each micro-cluster, the closest micro-cluster which falls within the radius of overlap is identified. From among the near micro-clusters which fulfill the criterion of radius overlap, two micro-clusters with the smallest distance are merged. If a micro-cluster is located within the radius of overlap with several other micro-clusters, it will be merged with the micro-cluster with the highest fitness. This process will continue until no two micro-clusters fall within the radius overlap of each other. In the end, the number of micro-clusters is equal to the number of main clusters in the stream and the micro-clusters indeed represent the clusters in the stream.

When there is no new observation in the stream to be processed, Algorithm 3 is executed to calculate the number of clusters in the stream. Next, as in EvoStream, the starting population of chromosomes is generated and the genetic algorithm performs the evolutionary steps to gradually improve the starting initial population. The pseudocode of the evolutionary function is summarized in Algorithm 4. As soon as a new observation arrives, the evolutionary steps are stopped and the new observation is processed by the online component. This guarantees that only idle times are used for evolutionary clustering. If a new observation arrives during an evolutionary step, the algorithm stops this step before processing the new observation. If the number of steps of the genetic algorithm is not yet finished, the results will be stored.

Algorithm 1 – Online Component of DynamicEvostream

Require: radius *r*, decay rate λ, cleanup interval *tgap*, Population size *P*
Initialize: t=0, MicroClusters=∅, C=∅, slowstream = 0
1: while stream is active do
2: read x from stream
3: t←t+1 , new ← (x, t, 1)
4: for each microcluster ∈ MicroClusters do // microcluster := (c ,t ,ω)
5: if dist (microcluster, new) < r then
6: microcluster [c] ← microcluster [c] +h (new[c], microcluster [c]) ·(new[c] − microcluster [c])
7: microcluster [t] ←t
8: microcluster [ω] ← microcluster [ω] ·2 $^{-\lambda (t - \text{microcluster [t]})}$ + 1
9: if new has not been absorbed by any microcluster ∈ MicroClusters then
10: MicroClusters ← MicroClusters ∪ new
11: if t mod tgap = 0 then cleanup (·)
12: if has a request for clustering
13: if slowstream = 0 or (slowstream = 1 and Change_stream_distribution = 1)
14: predict_k_and_result (results)
15: else return Evolutionary algorithm results
16: while idle do
17: K ← predict_k_and_result (numberofcluster)
18: for i ←1, ..., P do
19: Ci ← K randomly chosen micro-cluster
20: evolution (·)
21: slowstream ← 1
22: Change_stream_distribution ← 0

Algorithm 2 – Cleanup Function

1: **function cleanup (·)**
2: for each microcluster ∈ MicroClusters do
3: microcluster [ω] ← microcluster [ω] ·$2^{-\lambda (t - \text{microcluster [t]})}$
4: if microcluster [ω] ≤$2^{-\lambda t_{gap}}$ then
5: Remove microcluster from MicroClusters
6: Change_stream_distribution ← 1

```
7:      Merge all microcluster_i microcluster_j where dist. (microcluster_i, microcluster_j) ≤ r
8:      if merge microclusters
9:              Change_stream_distribution ← 1
```

Algorithm 3 – Offline Component of Dynamic Evostream

```
1:      function predict_k_and_result (in)
2:      Arrange the microclusters in descending order based on fitness
3:      for each C ∈ MicroClusters do
4:              Find the nearest microcluster to C
5:                      if dist (nearest_microcluster, C) < = 2r/3   then
6:                              temp_microclusters ← new (nearest_microcluster, C, dis)
8:      find two microclusters with the shortest distance in temp_microclusters and Merge them
7:      Repeat lines 2 to 8 as long as there are microclusters closer than the threshold for integration
8:      K ← size of microclusters
9:      if in = results then   macroclusters← microclusters
```

Algorithm 4 – Evolutionary Function

```
1:      function evolution (·)
2:      P1, P2←Select two solutions proportionally to their fitness from population
3:      O1, O2 ←Create offsprings of P1, P2 using binary crossover
4:      for each g_i in O1, O2 do_For each child-gene
5:              if random (0,1) < Pm then
6:                      if g_i = 0 then g_i ← 2δ
7:                      else g_i ←2δ· g_i
8:      Add O1, O2 to population and discard the two least fitest solutions
```

In the next idle time, if the stream distribution has not changed, the weights of the micro-clusters are updated and then the remaining evolutionary steps are executed. However, if the distribution of the stream has changed, the previous results are regarded as invalid and the genetic algorithm is executed anew with the new micro-clusters.

In the development of EvoStream, the other existing approaches to mutation and recombination were examined. However, no significant difference was observed between these approaches in the output results of the algorithm. Therefore, EvoStream utilizes the same operators as those used by GA-clustering. These operators have also been used in DynamicEvoStream without any modification.

5. Evaluation

To evaluate DynamicEvoStrean and EvoStream, the algorithms were implemented in C++ and tested on a desktop computer with Intel Core i7 4510U 2GHz and 8GB RAM DDR3.

Initial tests were done with two laboratory samples datasets and the final analysis was conducted with the real datasets KDDCup'99.4, Covertype, Abalone, and Avila (Table 2). For precise evaluation of the accuracy of the proposed algorithm, a set of classified data was used in which the classes of all data were labeled. Thus, in addition to evaluating the criteria for examining the validity of the results such as SSQ, we can also check the accuracy of the algorithm in assigning the data to the correct class.

The laboratory samples datasets consist of a large number of numerical data items which have been labeled by researchers.

The first set is composed of simulated data from the sensors of temperature, humidity, and oxygen. The values of each sensor were randomly produced in the real range of the performance of the sensor. The data were normalized and labeled in nine geographical classes according to the values. The second set is the simulation of a child's development consisting of the attributes of age, height, weight, and head circumference. The values of each attribute were randomly produced in the real-world ranges and classified into four groups, i.e. normal, requiring further monitoring and measurement, warning, and requiring medical examination.

KDDCup'99.4 is a relatively old but still popular data stream for testing stream clustering algorithms. It consists of 4M records of network traffic data. The data have 42 features which describe the connection features and the connection status. 31 attributes are numerical. The present authors have used the first 2,500,000 items for evaluation. The dataset is available online[1].

The static dataset Covertype consists of 581,012 items of cartographic information used to predict the vegetation of forests in the United States. Each item of this dataset has 10 features including information about the physical features of vegetation in forests such as slope, altitude, and soil type. The label of each class represents one of seven types of forest vegetation. The dataset is available online[2].

Abalone is a dataset to estimate the age of an abalone's shell based on physical measurements. To find the age of an abalone, one must cut through the cone-shaped shell, stain it, and count the number of rings using a microscope, which is a tedious and time-consuming task. In this dataset, physical attributes such as length, diameter, and height which are easier to measure are suggested for estimating the age of the shell. The dataset consists of 4177 items of information about abalones. The data have

[1] https://archive.ics.uci.edu/ml/datasets/kdd+cup+1999+data

[2] https://archive.ics.uci.edu/ml/datasets/Covertype

8 features, with 7 of them being numerical and only one of them being non-numerical which refers to the sex of the snail. As the sex of a snail is equally important in determining the age of the shell, this column cannot be ignored. Thus, to convert the type of sex to a format which could be readable for the algorithm, the researchers added one column to the data for each sex and then removed the sex column. In other words, one column was added to all items for each existing sex (male, female, newborn). For each item, the corresponding sex column was set to 1 and the other two columns were set to 0. The dataset is available online[3].

Avila dataset consists of 20,867 data items. This set consists of the data related to 800 images of the Avila Bible which is a Latin copy of the Bible dating back to 12th century Italy and Spain. Graphical analysis of this manuscript is indicative of the existence of 12 different manuscripts of the book. Each data item has 10 features. The aim is to assign each item to one of the 12 manuscripts. The initial data were normalized using the Z-normalization method. The dataset is available online[4]. Unfortunately, none of the four mentioned datasets include information about time so that we could perform a precise simulation of the speed of data stream and input. Therefore, by sending and processing the data items one by one, we considered them as a single data stream and, just as in EvoStream, modeled the input time of the data as a Poisson process with an average input rate of 1 observation per second.

Table 2 - Specification of Datasets

Dataset Name	Data Set Characteristics	Attribute Characteristics	Missing Values	Number of instances	Dimensions (number of attributes)
lab_data_1	Multivariate, Time-Series	Laboratory samples	no	500,000	3
lab_data_2	Multivariate, Time-Series	Laboratory samples	no	500,000	4
KDDCup '99.4	Multivariate	Real	N/A	2,500,000	31
Covertype	Multivariate	Real	No	581,012	10
Abalone	Multivariate	Real	No	4,177	10
Avila	Multivariate	Real	N/A	20,867	10

As significant differences in the variation ranges of the data would negatively affect the algorithm, all attributes were standardized by subtraction from the mean and division by the standard deviation in order to reduce differences in the scales of the data. This is referred to as data normalization [32]. It should be noted that we simulated the real-world speed of streams in evaluating EvoStream and DynamicEvoStream on these datasets. The evaluations, therefore, took days or even weeks to finish. It should be borne in mind that the speed of the input stream in other algorithms can be artificially increased. In other words, the idle times can be safely eliminated without any disruption in the work of these algorithms because they do not make use of idle times.

As in [6], the automatic algorithm configuration package called irace [33] was used to find the optimum values of radius and t_{gap} in both algorithms. irace examines new parameters and removes those which are statistically more inefficient. The sampling process acts iteratively towards better configurations. The fading rate parameter is assumed as $\lambda = 0.001$. As in GA-clustering, the parameters of the evolutionary algorithm are set as population size P=100, mutation rate Pm=0.001, and crossover rate Pc=0.8. Researchers in [6] tested several scenarios of different values and concluded that the values of GA-clustering parameters are the best selections. The number of clusters in EvoStream is equal to the total number of classes in each stream. It should be noted that parameter settings are not self-evident and it is difficult to determine these settings for new, unknown data streams.

5.1. Evaluation of the algorithm in terms of recognizing the number of clusters

Real-world data do not have clear-cut boundaries that could show where exactly stream distribution is changed. With the aim of evaluation, therefore, both algorithms were separately executed four times on each dataset and, in each execution, responded to 100 different clustering requests at random times. The average results are listed in Table **3**.

As can be seen, the results of DynamicEvoStream in recognizing the number of clusters are significantly better than those of EvoStream. It should be noted that determining the value of t_{gap} is of great importance in DynamicEvoStream. This parameter specifies the time in which the cleanup function, which is responsible for merging overlapping clusters and removing outdated clusters, is executed. The value of this parameter is set according to changes in the stream. If this value is too small, the cleanup function runs many times without merging or removing any cluster, which unnecessarily increases the computational overload while having no effect on the results of clustering. If the value is too large, the cleanup function is executed over longer time periods. As a result, outdated clusters will remain for a longer time and may negatively affect clustering in the event of clustering requests. At the same time, overlapping clusters which must be merged into a single cluster will appear as two or more separate clusters in the clustering results.

To initialize the macro-clusters, EvoStream sets a threshold value for the number of micro-clusters. This means that if the number of micro-clusters is twice the number of clusters in the stream, EvoStream initializes the macro-clusters and the macro-clusters will be optimized by the genetic algorithm in the following idle times.

If the radius of the clusters is set as close to real and the stream is noiseless enough, the number of micro-clusters never reaches the threshold and the algorithm cannot perform the clustering. Therefore, the radius is set as smaller than the real value so as to ensure that the number

[3] https://archive.ics.uci.edu/ml/datasets/Abalone

[4] https://archive.ics.uci.edu/ml/datasets/Avila

of micro-clusters could reach the threshold. This wastes some memory because each real cluster in the stream is represented by multiple micro-clusters. However, if the radius gets close to the real value and the threshold value for the number of micro-clusters is removed, micro-clusters will be reduced and the final macro-clusters will be of higher quality.

In addition, in heterogeneous streams, execution of EvoStream inevitably requires that the number of clusters be equal to the maximum number of clusters in the stream. For example, in a stream with at most 10 visible clusters, the number is set as K=10 and the threshold for the initialization of micro-clusters is 2K=20 which will remain constant for the entire stream. As a result, in intervals in which the clusters are less than 10, each cluster is represented by more micro-clusters, thereby wasting memory without any significant effect on the quality of micro-clusters.

As increasing micro-clusters will increase computations and decrease the speed of the algorithm, DynamicEvoStream can increase speed and optimize memory usage by eliminating the threshold as well as dynamic identification of the number of clusters in the stream. By eliminating the threshold value for micro-clusters in this algorithm, the number of micro-clusters is remarkably reduced, which optimizes memory usage while maintaining the quality of final macro-clusters.

Also, dynamic identification of the number of clusters causes the algorithm to retain its flexibility and adaptability to the stream in times when the number of clusters in the stream decreases as well as to produce the necessary macro-clusters if needed. As an example, the results of the laboratory samples dataset 1 are shown in Figure 2.

As can be seen in Figure 2, EvoStream always detects K=9 clusters in the stream which is equal to the number of macro-clusters in the memory. As the threshold of the number of micro-clusters in the stream is 18 (= 2K), in the best case, there are 27 clusters in the memory consisting of both micro-clusters and macro-clusters. Existence of noise causes the number of micro-clusters in parts of the stream to become more than 18. As a result, the number of clusters in the memory is usually more than 27. In DynamicEvoStream, the number of clusters is recognized dynamically. In the best case, the numbers of micro-clusters and macro-clusters are equal when there is no noise and the boundary of clusters is clear. But if there is some noise in the stream, the number of micro-clusters in the memory increases and, this increase is always less than twice the number of macro clusters due to the implementation of the cleanup function. In both cases, memory consumption in DynamicEvoStream is smaller than in EvoStream.

To evaluate memory consumption with the real datasets, the number of micro-clusters and macro-clusters was recorded over periods of 2000 seconds. The results show that the closer the number of clusters in the stream is to the number of clusters set by EvoStream, the closer the memory consumption of DynamicEvoStream and EvoStream. As the number of recognized clusters in DynamicEvoStream falls below the number set in EvoStream with a greater difference, the memory consumption of DynamicEvoStream becomes lower than that of EvoStream.

5.2. Time complexity of DynamicEvoStream

In this section, we shall examine the time complexity of each component of DynamicEvoStream.

5.2.1. Time complexity of the cleanup function

This function first reduces the weight of each micro-cluster by use of the fading function. If the weight falls below the threshold, it will remove the micro-cluster and enable the stream distribution change flag. This operation is of the order O(n), where n is the number of micro-clusters. Next, for each micro-cluster, the function examines the other micro-clusters and merges micro-clusters with more than 50% overlap. In the worst case, no two micro-clusters are merged, which is an act of the order $O(n_2)$. In the best case, all micro-clusters are merged, which is an act of the order O(n).

If the clusters are merged, the stream distribution change flag is enabled. In the worst case, the cleanup function is of the order $O(n_2)$. Given that the number of micro-clusters in DynamicEvoStream is always less than or (in the worst case) equal to the number of micro-clusters in EvoStream, the time complexity of the cleanup function in DynamicEvoStream is always less than or (in the worst case) equal to the time complexity of the cleanup function in EvoStream.

5-2-2- Time complexity of the function of recognizing the number of clusters

In this function, the micro-clusters are first sorted by the value of their fitness function. Using merge sort, this act is of the order O(n.log(n)), where n is the number of micro-clusters. Next, for each micro-cluster, the closest micro-cluster is found, and they are merged if they fulfill the criterion of overlap. In the worst case, no two micro-clusters are merged, which is an act of the order $O(n_2)$. In the best case, all micro-clusters are merged, which is an act of the order O(n). In the end, the final number of micro-clusters is recorded as the number of clusters, which is an act of the order of O(1). In the worst case, this function is of the order $O(n_2)$.

5-2-3- Time complexity of the evolutionary algorithm

EvoStream first sorts the micro-clusters by the values of the fitness function and using merge sort, which is an act of the order O(n.log(n)); but DynamicEvoStream has already sorted the micro-clusters when recognizing the number of clusters where n denotes the number of micro-clusters. As the number of clusters in DynamicEvoStream is always less than or (in the worst case) equal to the number of micro-clusters in EvoStream, the time complexity of sorting in DynamicEvoStream is always less than or (in the worst case) equal to EvoStream.

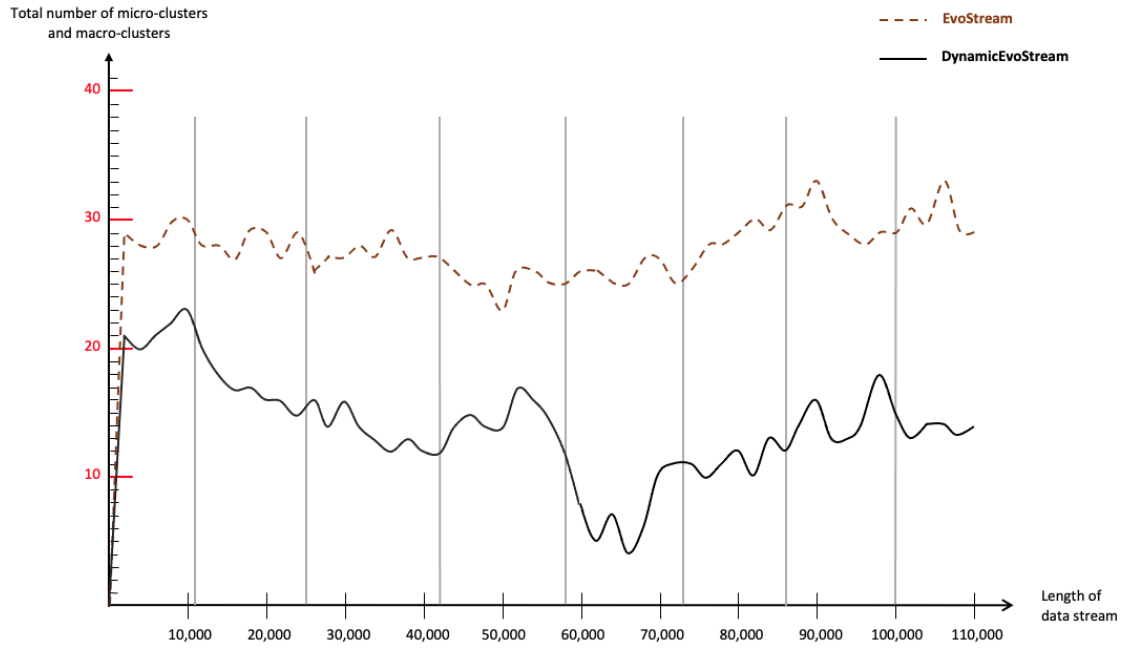

Fig. 2. Average memory consumption during 4 executions on the experimental dataset 1(gray vertical lines show the approximate range of stream distribution changes)

Next, the parents are selected according to their fitness and their genes are combined. The number of the genes of each chromosome is equal to the number of clusters. Therefore, the generation of each offspring is of the order O(K), where K denotes the number of clusters. The number of clusters in DynamicEvoStream is always less than or equal to the number of clusters in EvoStream; therefore, the time complexity of generating offspring in DynamicEvoStream is always less than or (in the worst case) equal to EvoStream.

The time complexity of executing each evolutionary step in the genetic algorithm is of the order O(n.log(n)+K) in EvoStream and of the order O(K) in DynamicEvoStream. It should be noted that the values of n and K in DynamicEvoStream are less than or equal to those in EvoStream.

5.3. Evaluation of the quality of clusters using purity index

One of the indexes of external evaluation in clustering is purity index which refers to the percentage of agreement between clustering and actual labels. For this purpose, the label assigned to each cluster is compared with the actual label of the class which has the most in common with the cluster, and the number of points from the cluster which are within the correct class is counted. The ratio of this number to the total number of points is called the purity index. Equation 3 calculates the purity index.

$$Purity(S.C) = \frac{\sum_m \max_n |S_m \cap C_n|}{N} \quad (3)$$

$|S_m \cap Cn|$ denotes the points in common between the cluster Cn and the class S_m, and N denotes the total number of points. The maximum value of purity index is 1, which is reached when the labels resulting from clustering for all the possible points in that cluster completely conform to the actual labels. Conversely, if none of the cluster labels conform to the actual labels, the index becomes 0.

Table 3- The average results of clustering the test

Dataset	The number of executions	The number of requests in each execution	The number of correct responses in terms of recognizing the number of clusters in DynamicEvoStream	The number of correct responses in terms of recognizing the number of clusters in EvoStream
Test_data_1	4	100	376	224
Test _data_2	4	100	391	269
KDDCup '99.4	4	100	400	400
Covertype	4	100	396	317
Abalone	4	100	389	244
Avila	4	100	394	271

Figure 3 illustrates the average results from the calculation of purity index in both algorithms on the laboratory samples dataset 1.

During the changes in stream distribution, the clustering quality of DynamicEvoStream deteriorates before the execution of the cleanup function (starred points) while it returns to an acceptable level after the function is executed. In parts of the stream where the number of recognized clusters is equal to the number of clusters set for EvoStream, the clustering quality of the two algorithms is quite close to each other. In those parts where the number

of clusters is less than the number set for EvoStream, the results of EvoStream severely deteriorate and a great percentage of the observations are wrongly clustered. It should be noted that the quality of clustering real datasets confirmed the results of our experiments. In general, the results indicate that DynamicEvoStream has a higher quality of clustering than EvoStream.

5.4. Limitations

Notwithstanding the acceptable performance of DynamicEvoStream, there still remain several important issues:

1. DynamicEvoStream is not able to recognize clusters of arbitrary form.
2. Radius and gap time are two parameters in the cleanup function that directly affect the results of DynamicEvoStream. Any error in the setting of these parameters can severely affect the performance of the algorithm.
3. If the number of observations belonging to a cluster is low and these observations arrive at relatively long intervals, DynamicEvoStream may spot the cluster as outdated and remove it.
4. In some cases, observations with noise are likely to be considered as a single cluster. The cleanup function removes this cluster, but if clustering is requested before the execution of the function, the number of clusters will be wrongly recognized.

6. Conclusion and future works

This paper introduced DynamicEvoStream which is an efficient algorithm of stream clustering using the idle times of the stream. It is the optimized version of EvoStream and can recognize the number of clusters dynamically. The results of our experiments indicate that DynamicEvoStream outperforms EvoStream. The major advantages of DynamicEvoStream over EvoStream are as following:

Recognizing the number of clusters: Recognizing the number of clusters is an expert task which requires a deep understanding of the stream. Moreover, this is impossible in streams with non-uniform distribution. In addition to eliminating the risk of expert's errors in recognizing the number of clusters, dynamic recognition of the number of clusters allows the algorithm to have a good performance for both uniform and non-uniform streams.

- Increased quality of clustering: DynamicEvoStream correctly recognizes the number of clusters. In the worst case where the boundaries of clusters are too close, the number of clusters recognized by DynamicEvoStream is more similar to the real number of clusters in the stream than is the number recognized by EvoStream. It is obvious that this will increase the quality of clusters.
- Reduced computation and memory requirements: While maintaining the quality of clusters, DynamicEvoStream has lower requirements and performs less computation than EvoStream. In the worst case, depending on the dimensions of the data and the speed of streams with non-uniform distribution, the quality of cluster recognition is close to EvoStream while in the average case it shows more than 30 percent of improvement. This is more obvious in those parts of the stream where the input rate of the data is high.
- Increased speed: This algorithm has a higher speed than EvoStream due to reduced computation and memory requirements, elimination of outdated clusters, and merging clusters with large amounts of overlap. In those parts of the stream where the number of real clusters in the stream has the greatest difference from the number of clusters set in EvoStream, DynamicEvoStream is approximately four times as fast as EvoStream.

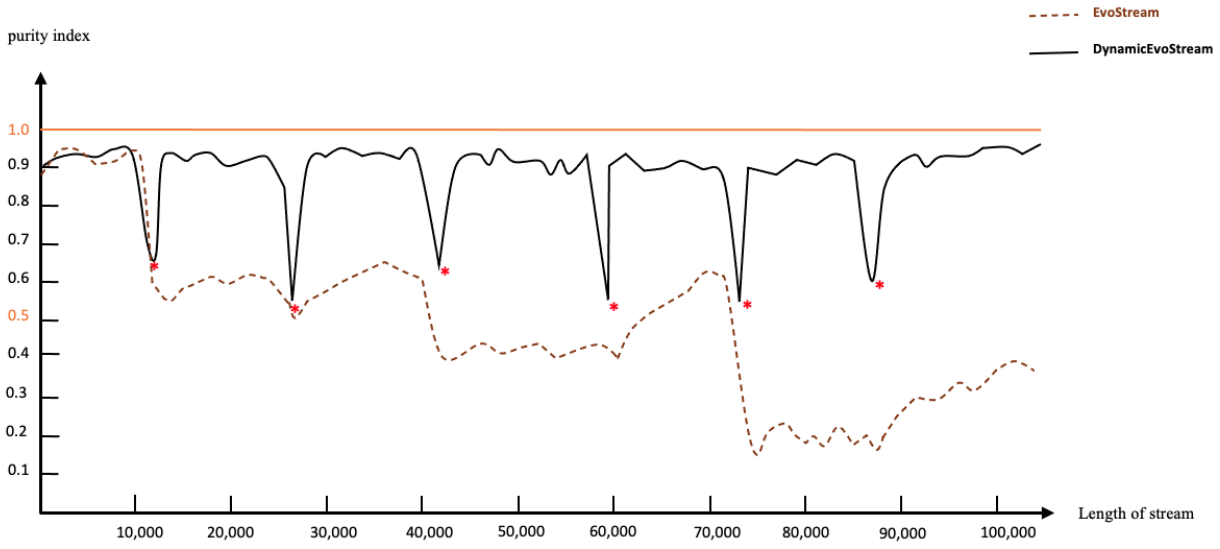

Fig. 3. The average purity index for the experimental dataset 1

If instead of merging overlapping clusters, we hold their centres in a data structure as the representatives of the clusters, clusters of an arbitrary form can also be recognized. In the next step, our aim is to improve the algorithm in a way that it could recognize any cluster of an arbitrary form. A more general research objective in the future is to use techniques for identifying irrelevant data or noise in order to make the algorithm immune to wrong cluster recognition. This can protect the results of the algorithm against the excessive influence of the gap time parameter (t_{gap}). In the end, we hope to find a method for dynamic modification of the radius of clusters to improve the performance of the algorithm.

7. References

[1] B. Shayesteh, V. Hakami, S.A. Mostafavi, A. Akbari Azirani, "A Novel Trust Computation Scheme for Internet of Things Applications", *Tabriz Journal of Electrical Engineering*, vol 50, no. 2, pp. 743-755, 2020 (in persian).

[2] M. Chen, S. Mao, Y. Liu, "Big Data: A Survey", *Mobile Networks and Applications*, vol. 19, pp. 171-209, 2014.

[3] T. Kolajo, O. Daramola, A. Adebiyi, "Big data stream analysis: a systematic literature review", *Journal of Big Data*, vol.47, no. 6, 2019.

[4] V. Gurusamy, S. Kannan, K. Nandhini, "The Real Time Big Data Processing Framework: Advantages and Limitations", *International Journal of Computer Sciences and Engineering*, vol. 5, no. 12, pp. 305-312, 2017.

[5] D. Namiot, "On Big Data Stream Processing", *International Journal of Open Information Technologies*, vol. 3, no. 8, pp. 48-51, 2015.

[6] M. Carnein, H. Trautmann, "EvoStream Evolutionary Stream Clustering Utilizing Idle Times", *Big Data Research*, vol. 14, pp. 101–111, 2018.

[7] A.K. Jain, M.N. Murty, P.J. Flynn, "Data clustering: a review", *ACM Computing Surveys*, vol.31, no. 3, pp. 264-323, 1999.

[8] M. Ester, H.P. Kriegel, J. Sander, X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise", *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, pp. 226–231, 1996.

[9] I. Behravan, S.H. Zahiri, S.M. Razavi, R. Trasarti, "Using Gray Wolf Optimization Algorithm in Big Data Clustering", *Tabriz Journal of Electrical Engineering*, vol. 50, no. 1, 2020 (in persian).

[10] Leite DF, Costa P, Gomide F (2009) Evolving granular classification neural networks, IJCNN 2009, pp 1736–1743

[11] T. Smith, D. Alahakoon, "Growing self-organizing map for online continuous clustering", *Studies in computational intelligence*, vol. 204, pp. 49–83, 2009.

[12] X. Dang, V. Lee, W. Ng, A. Ciptadi, K. Ong, "An EM-based algorithm for clustering data streams in sliding windows", *Lecture notes in computer science*, vol. 5463, pp 230–235, 2009.

[13] S. Guha, N. Mishra, R. Motwani, L. O'Callaghan, "Clustering data streams", *Proceding 41st Annual Symposium Foundations Computer Science*, 2000.

[14] F. Cao, M. Ester, W. Qian, and A. Zhou, "Density-based clustering over an evolving data stream with noise", *Proceeding of the Sixth SIAM Conference on Data Mining*, 2006.

[15] A. Zhou, F. Cao, Y. Yan, C. Sha, X. He, "Distributed Data Stream Clustering: A Fast EM-Based Approach", *IEEE 23rd International Conference on Data Engineering*, pp. 736-745, 2007.

[16] L. Tu, Y. Chen, "Stream data clustering based on grid density and Attraction", *ACM Transactions on Knowledge Discovery from Data*, vol. 3, no. 3, pp. 1–27, 2009.

[17] X.H. Dang, V.C. Lee, W.K. Ng, K. Ong, "Incremental and Adaptive Clustering Stream Data Over Sliding Window", *20th International Conference on Database and Expert Systems Applications*. pp. 660-674, 2009.

[18] J.P. Barddal, H.M. Gomes, F. Enembreck, "SNCStream: A Social Network-Based Data Stream Clustering Algorithm", *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, pp. 935–940, 2015.

[19] M. Khalilian, N. Mustapha, N. Sulaiman, "Data Stream Clustering by Divide and Conquer Approach Based on Vector Model", *Journal of Big Data*, vol. 3, no. 1, 2016.

[20] R. Hyde, P. Angelov, A. MacKenzie, "Fully Online Clustering of Evolving Data Streams Into Arbitrarily Shaped Clusters", *Information Sciences*, vol. 382, pp. 96–114, 2017.

[21] X. Gu, P. Angelov, D. Kangin, J. Principe, "Self-Organised Direction Aware Data Partitioning Algorithm", *Information Sciences*, vol. 423, pp. 80–95, 2017.

[22] J. Su, Y. Li, X. Zhao, "Data Stream Clustering by fast Density-Peak-Search", *Statistics and its Interface*, vol. 11, no. 1, pp. 183–189, 2018.

[23] R. Ahmed, G. Dalkılıç, Y. Erten, "DGStream: High quality and efficiency stream clustering algorithm", *Expert Systems with Applications*, vol. 141, 2020,

[24] C. Fahy, S. Yang and M. Gongora, "Ant Colony Stream Clustering: A Fast Density Clustering Algorithm for Dynamic Data Streams", *IEEE Transactions on Cybernetics*, vol. 49, no. 6, pp. 2215-2228, 2019.

[25] H. Longnguyen, Y. Kwongwoon,W. Keongng, "A Survey On Data Stream Clustering And Classification", *Knowledge And Information Systems*, vol. 45, pp. 535–569, 2014.

[26] S.Mansalis, E. Ntoutsi, N. Pelekis, Y. Theodoridis, "An Evaluation of Data Stream Clustering Algorithms", *Statistical Analysis and Data Mining: The ASA Data Science Journal*, vol. 11, no. 4, 2018.

[27] M. Carnein, H. Trautmann, "Optimizing Data Stream Representation: An Extensive Survey on Stream Clustering Algorithms", *Business &*

Information Systems Engineering, vol. 61, no. 3, pp. 277–297, 2019.

[28] A. Mohiuddin, "Data Summarization: A Survey", *Knowledge and Information Systems*, vol. 58, pp 249-273, 2019.

[29] D. Whitley, Darrell. "A genetic algorithm tutorial", *Statistics and Computing*, vol.4, no.2, pp. 65–85, 1994.

[30] W.M. Spears, K.A. De Jong, T. Bäck, D.B. Fogel, H. de Garis, "An overview of evolutionary computation", *Lecture Notes in Computer Science*, vol. 667, pp. 442-459, 1993.

[31] U. Maulik, S. Bandyopadhyay, "Genetic algorithm-based clustering technique", *Pattern Recognition*, vol. 33, no. 9, pp. 1455–1465, 2000.

[32] C. C. Aggarwal, J. Han, J. Wang, P.S. Yu, "A framework for projected clustering of high dimensional data streams", *Proceedings of the thirtieth international conference on very large data bases*, vol. 30, pp. 852–863, 2004.

[33] M. López-Ibáñez, J. Dubois-Lacoste, L. Pérez Cáceres, T. Stützle, M. Birattari, "The irace package: iterated racing for automatic algorithm configuration", *Operation Research Perspectives*, vol. 3, pp. 43–58, 2016.