Nature-inspired Methods for Monitoring Applications with Time-Evolving Data

Clara Pizzuti, Giandomenico Spezzano

CNR – National Research Council of Italy
Institute for High Performance Computing and Networking (ICAR)
Via P. Bucci 41C - 87036 Rende (CS), Italy
{pizzuti, spezzano}@icar.cnr.it

Abstract. Supporting real-time monitoring of large amount of information from diverse information sources is receiving an increasing demand because of the advances in data gathering and communication technologies. In this paper we propose to use a density-based clustering algorithm, built on the Multiple Species Flocking model, for the monitoring of large volume of streaming data, generated from numerous applications such as machine monitoring, health monitoring, sensor networks, speech recognition. The approach has two main characteristics that make it particularly apt for real life monitoring applications. The first is that the clustering results are available on demand at any time. The second one is that clusters can be visually tracked during their generation. This enables a user to visually detect changes in cluster distribution and gives him insights about the evolving nature of clusters, and thus when to eventually take actions and decisions in real time because of the changed conditions.

1 Introduction

Data-stream analysis and mining require novel algorithms that are able to produce models of the data in an online way, looking at each data record only once, and within a limited amount of time. Although standard data-analysis and data mining algorithms are a useful starting point, they must typically be adapted to operate in the stream setting. Stream clustering is a technique that performs cluster analysis of data streams able to produce results in real time.

Although researchers in the data-stream mining field have successfully tackled many of the issues that are of major concern of data-streams, the area is still new and it has many open problems. Actually, what has been addressed so far is related to having stream-mining systems that can handle the endless flow of data by being incremental, fast, and clever enough to approximate answers with a certain level of accuracy, based on the stream samples that have been seen so far.

In this paper, we propose the use of a data stream clustering method FlockStream [5], for monitoring applications. FlockStream is 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 and deployed onto

a 2D space, to work simultaneously by applying a heuristic strategy based on a bio-inspired model, known as flocking model. Agents move onto the space for a fixed time and, when they encounter other agents into a predefined visibility range, they can decide to form a flock, if they are similar. Flocks can join to form swarms of similar groups. FlockStream is particularly apt for those applications that must monitor enormous amount of data generated continuously as a sequence of events and coming from different locations, since it allows to track the evolving nature of clusters by displaying the movement of agents onto the virtual space. This enables a user to visually detect changes in cluster distribution and gives him insights when to eventually take actions and decisions in real time because of the changed conditions. Another main characteristics of FlockStream is that it allows the user to obtain an approximate, but faster result, by reducing the time agents can move onto the virtual space.

The paper is organized as follows. In the next section we give an introduction to the problem of monitoring applications. Section 3 describes the flocking model, the algorithm is reported in section 4, section 5 shows the application of the approach on a synthetic and real life data set. Section 6, finally concludes the paper.

2 Stream clustering for monitoring applications

With the advances of data gathering and communication technologies, it becomes increasingly possible to support real-time monitoring of large amount of information from diverse information sources. Large volumes of streaming data are generated from numerous applications such as machine monitoring, health monitoring, sensor networks, speech recognition, and so forth. In many cases, it is inevitable to analyze streaming data, as they are an important source of knowledge that enable us to take extremely important decisions in real time. Monitoring of applications plays an increasingly important role in many domains since it allows to detect events in monitored systems and to take actions, such as invoke a program or notify an administrator. The following applications have been identified as significant [7]:

1. Network monitoring and traffic engineering, sensor monitoring & surveillance (e.g., air quality monitoring, car traffic monitoring), Web logs and Web page click streams.
2. Discovering the evolution of workload in an "e-commerce' server, which can help in dynamically fine tune the server to obtain better performance.
3. Discovering meteorological data, such as temperatures registered throughout a region, by observing how clusters of spatial-meteorological points evolve in time.
4. Discovering the evolution of the spread of illnesses. As new cases are reported, finding out how clusters evolve can prove crucial in identifying sources responsible for the spread of illness.

Data stream clustering has evolved as a new form of online data analysis where real-life concepts tend to change over time. Stream clustering is a technique that performs cluster analysis of data streams able to monitor the results in real time. A data stream is a continuously generated sequence of data whose characteristics can evolve over time. A good stream clustering algorithm should recognize such evolution and yields a cluster model that dynamically adapts to the current data.

The main problem in applying clustering to data streams is that systems can examine data only once, as it arrives, but, at the same time, they must take into account data evolution. Thus a mechanism to remember old data, such as, for example, the compression of old information, is necessary to adapt to new concepts. Because of the dynamic nature of evolving data streams, in fact, new clusters often emerge while old clusters fade out. By exploring their temporal property, the influence of older data in the current representation can be decreased by applying an exponential decay function.

A promising research direction in stream clustering is the development of anytime algorithms that provide a result at any time of interruption and improve their quality as time elapses. Furthermore, the ability to process data in a single pass and summarize it, while using limited memory, is crucial to stream clustering.

Recent achievements in some of these areas are discussed in the following subsections.

2.1 Clustering data streams from sensors

Wireless sensor networks are a distributed autonomous network constituted by micro nodes with sensor, data processing unit, and wireless communication component. Wireless sensor networks provide reliable operations in various application areas, including environmental monitoring, health monitoring, vehicle tracking system, military surveillance and earthquake observation. Sensors networks facilitate the process of monitoring the physical environment and make real-time decision about events in the environment. In such monitoring applications, automatic event detection is an essential task which aims at identifying emergent physical phenomena of particular concern to the users. In particular, the change of clustering patterns often indicates something important is happening [6]. For example, clustering network event streams can help us to understand the normal patterns, and attack alarms can be raised if the clustering pattern changes.

In a pervasive computing environment, sensors are often distributed and in many cases embedded in several devices. Data, possibly in the form of streams, may be emanated from multiple sensors and these streams have to be aggregated, fused, stored, managed, and analyzed for various applications. So it is important to mine the data in real time to extract useful information for determining potential problems.

Clustering streaming sensors is the task of clustering different sources of data streams, based on the data series similarity. This process tries to extract knowledge about the similarity between data produced by different sensors through

time. The basic requirements usually presented for clustering data streams are that the system must possess a compact representation of clusters, must process data in a fast and incremental way and should clearly identify changes in the clustering structure. Algorithms aim to find groups of sensors that have a similar behavior through time.

2.2 Self-monitoring of distributed systems

Monitoring and online data analysis in distributed systems is important for characterizing *usage patterns* and *load distribution*. Such an understanding would help track the relative importance of components and services, delineate possible over/under or malicious utilization of the network, and manage user QoS (Quality of Service).

The control and timely management of large-scale distributed systems, such as device networks, data centers, and compute clusters, are tasks that are rapidly exceeding human ability, given their complexity, dynamics, and large amounts of data involved. Thus, the automated and online management of these systems is essential to ensure their continued performance and robust operations.

Fortunately, systems available in-network resources can be harnessed to perform self-monitoring and data analysis tasks, which are crucial for effective management.

A self-monitoring system is able to observe and analyze system state and behavior, to discover anomalies or violations, and to notify autonomic or human administrators in a timely manner so that appropriate management actions can be effectively applied. Furthermore, implementing the analysis technique in a decentralized and in-network fashion (using network resources and minimal extraneous information) ensures computational tractability and acceptable response times. However, because self-monitoring mechanisms are subject to the same failures that occur in the network they are helping to manage, the robustness of these mechanisms is of great importance to ensure overall system reliability. Therefore, it is very important to ensure the robustness of the proposed solution at different levels.

The main contribution, toward achieving the goals outlined above, is given by the work [8] that concerns the formulation and validation of a robust decentralized data analysis mechanism that applies density-based clustering techniques to identify anomalies and clusters of arbitrary size and shape in monitoring data.

Clustering data is given in the form of *periodic behavior* and operational status updates events from system components, defined in terms of known attributes. The event attributes are used to construct a multidimensional coordinate space, which is then used to measure the similarity of events. Components that behave in a similar fashion can then be identified by the clusters formed by their status events in this space, while devices with abnormal behavior will produce isolated events. The clustering algorithm requires minimal computation at processing nodes, which makes it suitable for online execution and to support event investigation by automatically clustering events on streams.

In the next sections, after an introduction to the flocking model, an approach for clustering data streams, based on a bio-inspired model, is described.

3 The Flocking model

The *flocking model* [3] is a biologically inspired computational model for simulating the animation of a flock of entities. In this model each individual (also called bird) makes movement decisions without any communication with others. Instead, it acts according to a small number of simple rules, depending only upon neighboring members in the flock and environmental obstacles. These simple rules generate a complex global behavior of the entire flock.

The basic flocking model was first proposed by Craig Reynolds [9], in which he referred to each individual as a "boid". This model consists of three simple steering rules that a boid needs to execute at each instance over time: *separation* (steering to avoid collision with neighbors); *alignment* (steering toward the average heading and matching the velocity of neighbors); *cohesion* (steering toward the average position of neighbors). These rules describe how a boid reacts to other boids' movement in its local neighborhood. The degree of locality is determined by the visibility range of the boid's sensor. The boid does not react to the flockmates outside its sensor range. A minimal distance must also be maintained among them to avoid collision.

A *Multiple Species Flocking* (MSF) model [2] has been developed to more accurately simulate flocking behavior among an heterogeneous population of entities. MSF includes a feature similarity rule that allows each boid to discriminate among its neighbors and to group only with those similar to itself. The addition of this rule enables the flock to organize groups of heterogeneous multi-species into homogeneous subgroups consisting only of individuals of the same species.

Dissimilar boids try to stay away from other boids that have dissimilar features by a repulsive force that is inversely proportional to the distance between the boids and the similarity between the boids.

The advantage of the flocking algorithm is the heuristic principle of the flock's searching mechanism. The heuristic searching mechanism helps boids to quickly form a flock. Since the boids continuously fly in the virtual space and join the flock constituted by boids more similar to them, new results can be quickly re-generate when adding entities boids or deleting part of boids at run time. This feature allows the flocking algorithm to be applied to clustering to analyze dynamically changing information stream.

4 Algorithm description

FlockStream is a heuristic density-based data stream clustering algorithm built on the Multiple Species Flocking model. The algorithm uses agents with distinct simple functionalities to mimic the flocking behavior. Each multi-dimensional data item is associated with an agent. In our approach, in addition to the standard action rules of the flocking model, we introduce an extension to the flocking

model by considering the *type* of an agent. The agents can be of three types: *basic* (representing a new point arriving in one unit time), *p-representative* and *o-representative* .

The introduction of the latter two types of agents follows the analogous concepts introduced by Cao et al. in [1] in their algorithm density-based clustering algorithm *DenStream* for data streams, of *micro-cluster*. In the following we briefly review the main concepts employed in this method. A detailed description can be found in [1].

A micro-cluster is an extension of the concept of *core point* defined in the clustering method $DBSCAN$ [4], to store a compressed representation of the data points examined so far, and corresponds to the notions of *potential core-micro cluster (p-micro-cluster)* and *outlier micro-cluster (o-micro-cluster)* of these authors.

A core point is an object in whose ϵ neighborhood the overall weight of the points is at least an integer μ. A clustering is a set of core objects having cluster labels. The definitions of micro-clusters are based on the concepts of summary statistics and weight. The former allows to maintain approximate representation of data points assigned to a micro-cluster, and thus to capture synopsis information about the nature of the data stream. The latter gives decreasing importance to data as time flows. The weight w of a micro-cluster must be such that $w \geq \mu$, i.e. it must be above a predefined threshold μ, in order to be considered a core.

As already observed, the volume of data in streaming applications is very huge and possibly infinite, too large to fit into the main memory of computers, thus a mechanism to store summary of data seen so far is necessary.

These statistics are exploited to generate clusters with arbitrary shape. The algorithm assumes the *damped window model* [1] to cluster data streams. In this model the weight of each data point decreases exponentially with time t via a fading function $f(t) = 2^{-\lambda t}$, where $\lambda > 0$. The weight of the data stream is a constant $W = \frac{v}{1-2^{-\lambda}}$, where v is the *speed of the stream*, i.e. the number of points arrived in one unit time. Historical data diminishes its importance when λ assumes higher values.

FlockStream adapted the above described concepts in the flocking model, and distinguishes between the initialization phase, in which the virtual space is populated of only basic agents, and the micro-cluster maintenance and clustering, in which all the three types of agents are present.

Initialization. At the beginning a set of basic agents, i.e. a set of points, is deployed randomly onto the virtual space and works in parallel for a predefined number of iterations. Basic agents move according to the MSF heuristic and, analogously to birds in the real world, agents that share similar object vector features will group together and become a flock, while dissimilar birds will be moved away from the flock. Agents uses a proximity function to identify objects that are similar. In our algorithm we use the Euclidean distance to measure the dissimilarity between data points A and B and assume that A and B are similar if their Euclidean distance $d(A, B) \leq \epsilon$. Note that the agents are clustered in the data space instead of their attribute space. While iterating, the behavior

(*velocity*) of each agent A with position P_a is influenced by all the agents X with position P_x in its neighborhood. The agent's velocity is computed by applying the local rules of Reynolds and the similarity rule. The similarity rule induces an adaptive behavior to the algorithm since the agents can leave the group they participate for another group containing agents with higher similarity. Thus, during this predefined number of iterations, the points join and leave the groups forming different flocks. At the end of the iterations, for each created group, summary statistics are computed and the stream of data is discarded. As result of this initial phase we have the two types of agents: *p*-representative and *o*-representative agents.

Representative Maintenance and Clustering. When a new data stream bulk of agents is inserted into the virtual space, at a fixed stream speed, the maintenance of the *p*- and *o*- representative agents and online clustering are performed for a fixed number of iterations. Different cases can occur (see figure 1) :

- a *p-representative* c_p or an *o-representative* c_o encounter another representative agent. If the distance between them is below ϵ then they compute the velocity vector by applying the Reynolds' and similarity rule (step 5), and join to form a *swarm* (i.e. a cluster) of similar representative (step 6).
- A basic agent A meets either a *p-representative* c_p or an *o-representative* c_o in its visibility range. The similarity between A and the representative is computed and, if the new radius of c_p (c_o respectively) is below or equal to ϵ, A is absorbed by c_p (c_o) (step 9). Note that at this stage FlockStream does not update the summary statistics because the aggregation of the basic agent A to the micro-cluster could be dropped if A, during its movement on the virtual space, encounters another agent more similar to it.
- A basic agent A meets another basic agent B. The similarity between the two agents is calculated and, if $d(A, B) \leq \epsilon$, then the velocity vector is computed (step 11) and A is joined with B to form an *o-representative* (step 12).

At the end of the maximum number of iterations allowed, for each swarm, the summary statistics of the representative agents it contains are updated and, if the weight w of a *p-representative* diminishes below $\beta\mu$, where β is a fixed outlierness threshold, it is degraded to become an *o-representative*. On the contrary, if the weight w of an *o-representative* becomes above $\beta\mu$, a new *p-representative* is created.

It is wort to note that, swarms of agents represent clusters, thus the clustering generation on demand by the user can be satisfied at any time by simply showing all the swarms computed so far.

5 Experimental Results

In this section we study the effectiveness of FlockStream on real and synthetic datasets and compare it with the *DenStream* algorithm of Cao et al. in [1] . The

```
1. for i=1 ... MaxIterations
2.  foreach agent (all)
3.  if (typeAgent is (p-representative ∨ o-representative))
4.  then{
5.        computeVelocityVector(flockmates, MSF rules);
6.        moveAgentAndFormSwarm(v);}
7.  else{
8.     if (typeAgent is (basic ∧ in neighborhood of a similar swarm) )
9.     then agent temporary absorbed in the swarm;
10.    else{
11.       computeVelocityVector(flockmates, MSF rules);
12.       moveAgentAndFormSwarm(v);}}
13. end foreach
14. end for
```

Fig. 1. The pseudo-code of the FlockStream algorithm.

algorithm has been implemented in Java and all the experiments have been performed on an Intel(R) Core(TM)2 6600 having 2 Gb of memory. The synthetic datasets used, named DS1, DS2 and DS3, are showed in Figure 2(a). Each of them contains 10,000 points and they are similar to those employed by Cao et al. in [1] to evaluate *DenStream*. For a fair comparison, the evolving data stream, denoted EDS, has been created by adopting the same strategy of Cao et al. Each dataset has been randomly chosen 10 times, thus generating an evolving data stream of total length 100,000 having a 10,000 points block for each time unit. The real dataset used to test the performances of FlockStream is the *KDD Cup 1999 Data set* (http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html). This data set comes from the 1998 DARPA Intrusion Detection Evaluation Data and contains training data consisting of 7 weeks of network-based intrusions inserted in the normal data, and 2 weeks of network-based intrusions and normal data for a total of 4,999,000 connection records described by 41 characteristics. The main categories of intrusions are four: Dos (Denial Of Service), R2L (unauthorized access from a remote machine), U2R (unauthorized access to a local super-user privileges by a local un-privileged user), PROBING (surveillance and probing). The data set has been transformed into a data stream by taking the input order as the streaming order. For all the datasets the true class label is known, thus the clustering quality can be evaluated by the average purity of the clusters.

The average purity of a clustering is defined as:

$$purity = \frac{\sum_{i=1}^{K} \frac{|C_i^d|}{|C_i|}}{K} * 100\%;$$

where K indicates the number of clusters, $|C_i^d|$ denotes the number of points with the dominant class label in cluster i, and $|C_i|$ denotes the number of

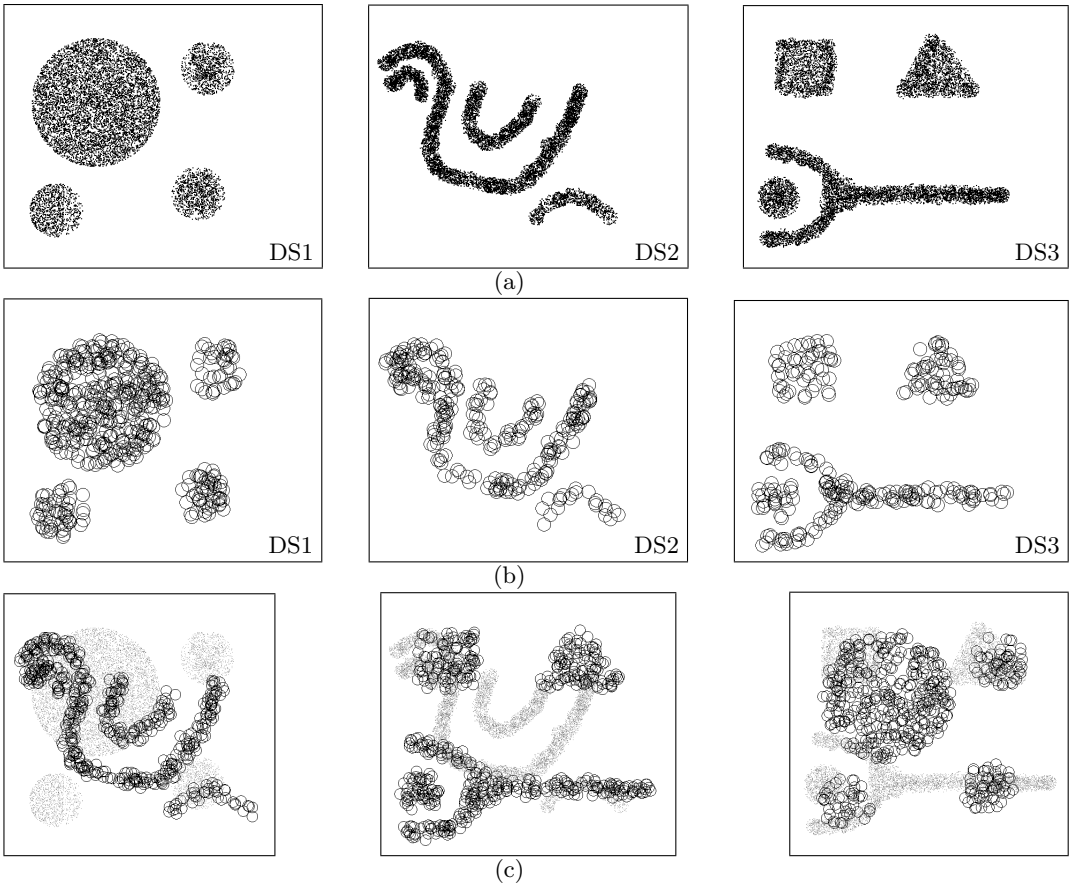
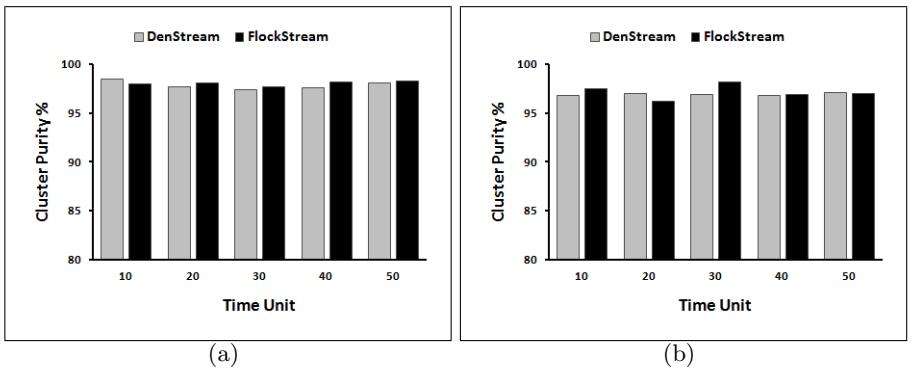


Fig. 2. (a) Synthetic data sets.(b) Clustering performed by FlockStream on the synthetic datasets. (c) Clustering performed by FlockStream on the evolving data stream EDS.

points in cluster i. The purity is calculated only for the points arriving in a predefined window (horizon), since the weight of points diminishes continuously. The parameters used by FlockStream in the experiments are analogous to those adopted by Cao et al., that is initial points/agents $Na = 1000$, stream speed $v = 1000$, decay factor $\lambda = 0.25$, $\epsilon = 16$, $\mu = 10$, outlier threshold $\beta = 0.2$ and $MaxIterations = 1000$. Initially we evaluated the FlockStream algorithm on the non-evolving datasets DS1, DS2 and DS3, to check the ability of the method to get the shape of each cluster. The results are reported in Figure 2(b). In this figure the circles indicate the micro-cluster detected by the algorithm. We can see that FlockStream exactly recovers the cluster shape.

The results obtained by FlockStream on the evolving data stream EDS, at different times, are shown in figure 2(c). In the figure, points indicate the raw

data while circles denote the micro-clusters. It can been seen that FlockStream captures the shape of each cluster as the data streams evolve.



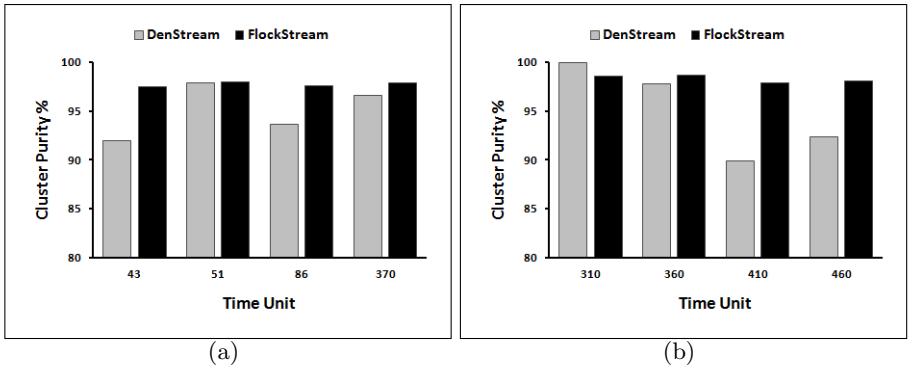
(a) (b)

Fig. 3. (a) Clustering quality for evolving data stream EDS with horizon=2 and stream speed=2000. (b) Clustering quality for evolving data stream EDS with horizon=10 and stream speed=1000.

The purity results of FlockStream compared to *DenStream* on the EDS data stream are shown in figure 3. In figure 3(a), horizon is set to 2 and the stream speed is set to 2000 points per time unit. We can see the very good clustering quality of FlockStream, in fact it is always higher than 95% and comparable to *DenStream*. Figure 3(b) shows the results of FlockStream when the stream speed is set to 1000 points per time unit and horizon is set to 10 on the EDS data stream. The results show that FlockStream, similarly to *DenStream*, is insensitive to the horizon.

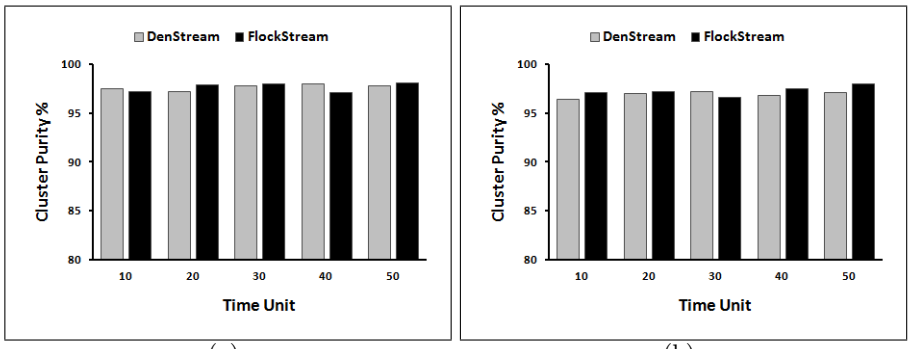
The comparison between FlockStream and *DenStream* on the Network Intrusion data set is shown in Figures 4(a) and 4(b). We selected the same time points, when some particular attacks happen, chosen by *DenStream*, and we report the results obtained at these moments. In the former figure the horizon is set to 1, whereas in the latter the horizon is set to 5; the stream speed is set to 1000 for both. We can see how FlockStream clearly ouperforms *DenStream* in almost all the time units chosen and the very high clustering quality achieves also on this dataset.

The purity of FlockStream compared to DenStream in a dataset with noise is calculated and the clustering purity results of EDS with 1% and 5% noise are shown in figures 5(a) and 5(b) respectively. The results demonstrate that FlockStream achieves high clustering quality, comparable to *DenStream*, also when noise is present.



(a) (b)

Fig. 4. (a) Clustering quality for Network Intrusion dataset with horizon=1 and stream speed=1000. (b) Clustering quality for Network Intrusion dataset with horizon=5 and stream speed=1000.



(a) (b)

Fig. 5. (a) Clustering quality for evolving data stream EDS with horizon=2, stream speed=2000 and with 1% noise. (b) Clustering quality for evolving data stream EDS with horizon=10, stream speed=1000 and with 5% noise.

6 Conclusions

A heuristic density-based data stream clustering algorithm built on the Multiple Species Flocking model has been presented. The method employs a local stochastic multi-agent search strategy that allows agents to act independently from each other and to communicate only with immediate neighbors in an asynchronous way. Decentralization and asynchronism makes the algorithm scalable to very large data sets. The approach has two main characteristics that make it to be particularly apt for real life monitoring applications. The first is that the clustering results are always available. This means that the clustering generation on demand by the user can be satisfied at any time by simply showing all the

swarms computed so far. The second one is that clusters can be visually tracked since the movement of agents onto the virtual space is displayed. This enables a user to visually detect changes in cluster distribution and gives him insights about the evolving nature of clusters , and thus when to eventually take actions and decisions in real time because of the changed conditions. Future work aims at applying the method to real life monitoring applications.

References

1. Feng Cao, Martin Ester, Weining Qian, and Aoying Zhou. Density-based clustering over evolving data stream with noise. In *Proceedings of the Sixth SIAM International Conference on Data Mining (SIAM 2006)*, pages 326–337, 2006.
2. Xiaohui Cui and Thomas E. Potok. A distributed agent implementation of multiple species flocking model for document partitioning clustering. In *Cooperative Information Agents*, pages 124–137, 2006.
3. Russell C. Eberhart, Yuhui Shi, and James Kennedy. *Swarm Intelligence (The Morgan Kaufmann Series in Artificial Intelligence)*. Morgan Kaufmann, March 2001.
4. Martin Ester, Hans-Peter Kriegel, J?rg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second ACM SIGKDD International conference on Knowledge discovery and data mining (KDD'96)*, pages –, 1996.
5. Agostino Forestiero, Clara Pizzuti, and Giandomenico Spezzano. A single pass algorithm for clustering evolving data streams based on swarm intelligence. *Data Mining and Knowledge Discovery Journal, to appear*. Online Nov. 2011.
6. João Gama, Pedro Pereira Rodrigues, and Luís M. B. Lopes. Clustering distributed sensor data streams using local processing and reduced communication. *Intell. Data Anal.*, 15(1):3–28, 2011.
7. Madjid Khalilian and Norwati Mustapha. Data stream clustering: Challenges and issues. *CoRR*, abs/1006.5261, 2010.
8. Andres Quiroz, Nathan Gnanasambandam, Manish Parashar, and Naveen Sharma. Robust clustering analysis for the management of self-monitoring distributed systems. *Cluster Computing*, 12(1):73–85, 2009.
9. Craig W. Reynolds. Flocks, herds and schools: A distributed behavioral model. In *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 25–34, New York, NY, USA, 1987. ACM.