FlockStream: a Bio-inspired Algorithm for Clustering Evolving Data Streams

Agostino Forestiero, Clara Pizzuti, Giandomenico Spezzano
Institute for High Performance Computing and Networking, ICAR-CNR
Via Pietro Bucci, 41C
87036 Rende (CS), Italy
{folino,pizzuti,spezzano}@icar.cnr.it

Abstract

Existing density-based data stream clustering algorithms use a two-phase scheme approach consisting of an online phase, in which raw data is processed to gather summary statistics, and an offline phase that generates the clusters by using the summary data. In this paper we propose 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 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. This strategy allows to merge the two phases of density-based approaches and thus to avoid the offline cluster computation, since a swarm represents a cluster. Experimental results show the capability of the bio-inspired approach to obtain very good results on real and synthetic data sets.

1. Introduction

In recent years, many organizations are collecting tremendous amount of data, often generated continuously as a sequence of events and coming from different locations. Credit card transactional flows, telephone records, sensor network data, network event logs are just some examples of data streams. The design and development of fast, efficient, and accurate techniques, able to extract knowledge from these huge data sets too large to fit into the main memory of computers, pose significant challenges. First of all, data can be examined only once, as it arrives. Second, using an entire data stream of a long period could mislead the analysis results due to the weight of outdated data considered in the same way as more recent data. Since data streams are continuos sequences of information, the underlying clusters could change with time, thus giving different results with respect to the time horizon over which they are computed. Traditional clustering approaches are not sufficiently flexible to deal with data that continuously evolves with time, thus, in the last few years, many proposals to data stream clustering have been presented [2, 3, 4, 5, 11, 12, 10, 1, 14]. Aggarwal et al. [1] have been the first to address the problem of the impossibility to revise a data stream during the computation. They suggested that a stream clustering algorithm should be separated in an online micro-clustering component, that gathers appropriate summary statistics on the data stream, and an offline macro-clustering component that makes use of the information stored to provide the clustering results. One of the main drawback of their $CluStream$ algorithm is that it is unable to discover clusters of arbitrary shapes, furthermore the number k of clusters must be fixed a-priori. A more amenable approach to clustering data streams is that based on the concept of density introduced by DBSCAN [9]. Density-based approach [4, 5] apply the two-phase scheme [1] described above in a density-based framework. In particular, the former, *DenStream*, extensively described in the next section, extends the concept of *core point* introduced in $DBSCAN$ and employs the notion of *micro-cluster* to store an approximate representation of the data points in a damped window model. The latter applies analogous concepts by associating a decay factor to the density of each data point and by partitioning the data space into discretized grids where new data points are mapped. The authors study the relation between time horizon, decay factor, and data density to guarantee the generation of high quality clusters.

The main drawback of these methods is the execution time required when dealing with large data sets of high dimensionality. In fact, both the methods, when a new data point arrives, in order to determine whether it should be merged into an existing cluster or consider it as the seed for a new group, need to do a comparison with all the clusters generated so far.

In this paper a density-based data stream clustering method, named *FlockStream*, that employs a multi-agent system using a decentralized bottom-up self-organizing strategy to group similar data points, is proposed. Each data point is associated with an agent. Agents are deployed onto a 2D space, called the *virtual space*, and work simultaneously by applying a heuristic strategy based on a bio-inspired model known as *flocking model* [8]. Agents move onto the space for a fixed time and when they encounter other agents into a predefined visibility radius they can decide to form a flock (i.e a micro-cluster) if they are similar. However, the participation of an agent to a group is not definitive since, if during the space exploration a more similar agent is encountered, the current flock can be dropped and the agent can join to the nearest agents' flock. The movement of the agents in the 2D space is not random, but it is guided by the similarity function that aggregates the agents to their closer neighbors. As different similar micro-clusters can be created, by applying the flocking rules, they are aggregated to form swarms of close micro-clusters.

FlockStream introduces two main novelties. First, it replaces the exhaustive search of the nearest neighbor of a point, necessary to assign it to the appropriate micro-cluster, with a local stochastic multi-agent search that works in parallel. The method is thus completely decentralized as each agent acts independently from each other and communicates only with its immediate neighbors in an asynchronous way. Locality and asynchronism implies that the algorithm is scalable to very large data sets.

Second, since flocks of agents can join together into swarms of similar groups, the two-phase scheme of data stream clustering methods mentioned above is replaced by a unique online phase, in which 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 paper is organized as follows. The next section describes the $DenStream$ algorithm. Section 3 introduces the Flocking model. Section 4 describes our approach. In section 5, finally, the results of the method on synthetic and real life data sets are presented.

2. DenStream Algorithm

$DenStream$ [4] is a density-based clustering algorithm for evolving data streams that uses summary statistics to capture synopsis information about the nature of the data stream. These statistics are exploited to generate clusters with arbitrary shape. The algorithm assumes the *damped window model* 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.

The authors extend the concept of *core point* introduced in $DBSCAN$ [9] and employ the notion of *micro-cluster* to store an approximate representation of the data points. 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. Three definitions of micro-clusters are then introduced: the *core-micro-cluster*, the *potential core-micro-cluster*, and the *outlier micro-cluster*.

A *core-micro-cluster* (abbreviated *c-micro-cluster*) at time t for a group of close points $p_{i_1}, \ldots, p_{i_n}$ with time stamps $T_{i_1}, \ldots, T_{i_n}$ is defined as $CMC(w, c, r)$, where w is the weight, c is the center, and r is the radius of the c-micro-cluster. The weight $w = \sum_{j=1}^{n} f(t - Ti_j)$ must be such that $w \geq \mu$. The center is defined as

$$c = \frac{\sum_{j=1}^{n} f(t - Ti_j) p_{i_j}}{w}$$

and the radius

$$r = \frac{\sum_{j=1}^{n} f(t - Ti_j) dist(p_{i_j}, c)}{w}$$

is such that $r \leq \epsilon$. $dist(p_{i_j}, c)$ is the Euclidean distance between the point p_{i_j} and the center c. Note that the weight of a micro-cluster must be above a predefines threshold μ in order to be considered core. The authors assume that clusters with arbitrary shape in a data stream can be described by a set of c-micro-clusters. However, since as data flows it can change, structures apt to incremental computation, similar to those proposed by [1] are introduced.

A *potential c-micro-cluster*, abbreviated *p-micro-cluster*, at time t for a group of close points $p_{i_1}, \ldots, p_{i_n}$ with time stamps $T_{i_1}, \ldots, T_{i_n}$ is defined as $\{\overline{CF^1}, \overline{CF^2}, w\}$, where the weight w, as defined above, must be such that $w \geq \beta\mu$. β, $0 < \beta \leq 1$ is a parameter defining the outlierness threshold relative to c-micro-clusters. $\overline{CF^1} = \sum_{j=1}^{n} f(t - Ti_j) p_{i_j}$ is the weighted linear sum of the points, $\overline{CF^2} = \sum_{j=1}^{n} f(t - Ti_j) p_{i_j}^2$ is the weighed squared sum of the points. The center of a p-micro-cluster is $c = \frac{\overline{CF^1}}{w}$ and the radius $r \leq \epsilon$ is $r = \sqrt{\frac{\overline{CF^2}}{w} - (\frac{\overline{CF^1}}{w})^2}$

A p-micro-cluster is a set of points that could become a micro-cluster.

An *outlier micro-cluster*, abbreviated *o-micro-cluster*, at time t for a group of close points $p_{i_1}, \ldots, p_{i_n}$ with time stamps $T_{i_1}, \ldots, T_{i_n}$ is defined as $\{\overline{CF^1}, \overline{CF^2}, w, t_0\}$. The definition of w, CF^1, CF^2, center and radius are the same of the p-micro-cluster. $t_0 = T_{i_1}$ denotes the creation of the o-micro-cluster. In an outlier micro-cluster the weight w must be below the fixed threshold, thus $w < \beta\mu$. However it could grow into a potential micro-cluster when, adding new points, its weight exceeds the threshold.

The algorithms consists of two phases: the online phase in which the micro-clusters are maintained and updated as new points arrive online; the off-line phase in which the final clusters are generated, on demand, by the user. During the online phase, when a new point p arrives, $DenStream$ tries to merge p into its nearest p-micro-cluster c_p. This is done only if the the radius r_p of c_p does not augment above ϵ, i.e. $r_p \leq \epsilon$. If this constraint is not satisfied, the algorithm tries to merge p into its nearest o-micro-cluster c_o, provided that the new radius $r_o \leq \epsilon$. The weight w is then checked if $w \geq \beta\mu$. In such a case c_o is promoted to p-micro-cluster. Otherwise a new o-micro-cluster is generated by p. Note that for an existing p-micro-cluster c_p, if no new points are added to it, its weight will decay gradually. When it is below $\beta\mu$, c_p becomes an outlier.

The off-line part of the algorithm uses a variant of the DBSCAN algorithm in which the potential micro-clusters are considered as virtual points. The concepts of density-connectivity and density reachable, adopted in DBSCAN, are used by DenStream to generate the final result. DenStream cannot be used to handle huge amounts of data available in large-scale networks of autonomous data sources since it needs to find the closest micro-cluster for each newly arrived data point and it assumes that all data is located at the same site where it is processed. In the next section a computational model that overcome these disadvantages is described.

3 The Flocking model

The *flocking model* [8] 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 [13], 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.

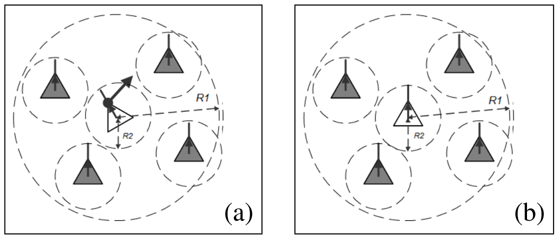

Figure 1. Alignment rule.

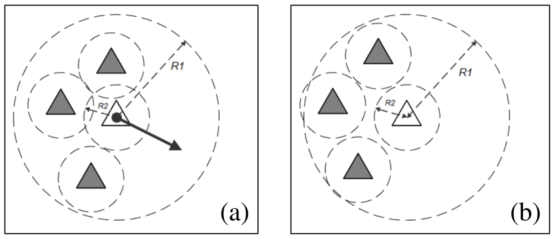

Figure 2. Separation rule.

A *Multiple Species Flocking* (MSF) model [7] 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. The basic behavior rules [6] of a single entity of the MSF model are illustrated in figures 1, 2, 3. Let R_1 and R_2, with $R_1 > R_2$, be the radius indicating the visibility range of the boids and the minimum distance that must be maintained among them respectively, and d(F_i, A_c) the distance between the current agent A_c and a flockmate F_i. The alignment rule means that a boid tends to move in the same direction of the nearby boids, i.e. it tries to align its velocity vector with the average velocity vector of the flocks in its local neighborhood. This rule is depicted in figure 1 and it is formally described as

$$if\ d(F_i, A_c) \leq R_1 \wedge d(F_i, A_c) \geq R_2 \Rightarrow \vec{v}_{ar} = \frac{1}{n}\sum_{i}^{n} \vec{v}_i$$

where $\vec{v}_{ar}$ is the velocity driven by the alignment rule, $d(F_i, A_c)$ is the distance between the boid A_c and its neighbor flockmate F_i, $\vec{v}_i$ is the velocity of the boid F_i, and n is the number of neighbors.

The separation rule avoids that a boid be too close to another boid, thus

$$if\ d(F_i, A_c) \leq 2R_2 \Rightarrow \vec{v}_{sr} = \sum_{i}^{n} \frac{\overrightarrow{\vec{v}_i + \vec{v}_c}}{d(F_i, A_c)}$$

where $\vec{v}_{sr}$ is the separation velocity, $\vec{v}_c$ and $\vec{v}_i$ are the velocities of the current boid and of the i-th flockmate, respectively. This rule is shown in figure 2.

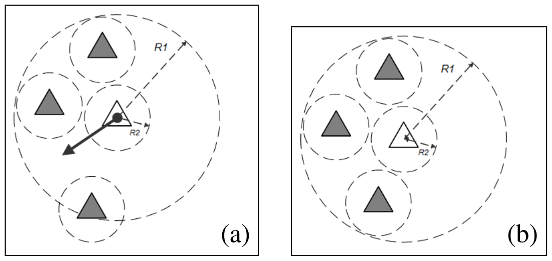

Figure 3. Cohesion rule.

The cohesion rule moves a boid towards other nearby boids (unless another boid is too close) orienting the velocity vector of the boid in the direction of the centroid of the local flock.

$$d(F_i, A_c) \leq R_1 \wedge d(F_i, A_c) \geq R_2 \Rightarrow \vec{v}_{cr} = \sum_i^n (P_i - P_c)$$

where $\vec{V}_{cr}$ is the cohesion velocity and $(P_i - P_c)$ calculates a directional vector point, being P_i and P_c the positions of the current boid A_c and a neighbor boid F_i. Figure 3 depicts the cohesion rule.

When two boids are too close, the separation rule overrides the other two, which are deactivated until the minimum separation is achieved. Similar boids try to stay close using the strength of the attracting force, that is proportional to the distance between the boids and the similarity between the boids' feature values. This strength is computed as

$$v_{sim} = \sum_i^n (Sim(F_i, A_c) * d(P_i, P_c))$$

where v_{sim} is the velocity driven by feature similarity, $Sim(F_i, A_c)$ is the similarity value between the features of boids A_c and F_i, and $d(P_i, P_c)$ is the distance between their positions in the 2D space.

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.

$$v_{dsim} = \sum_i^n \frac{1}{Sim(F_i, A_c) * d(P_i, P_c)}$$

where v_{dsim} is the velocity driven by feature dissimilarity.

The overall flocking behavior can be expressed by a weighted linear combination of the velocities calculated by all the rules that represent the net velocity vector $\vec{v}$ of the boid in the virtual space.

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 The *FlockStream* algorithm

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 time unit), *p-representative* and *o-representative* (corresponding to *p-* or *o-* micro-clusters). *FlockStream* 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. The basic agents work in parallel for a predefined number of iterations and move according to the MSF heuristic. 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 $d(A, B) \leq \epsilon$. 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 other 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. Analogously to DenStream, different cases can occur (see figure 4) :

- a *p-representative* c_p or an *o-representative* c_o encounters another representative agent. If the distance between them is below ϵ then they compute the velocity vector by applying the Reynolds' and similarity rules (step 5), and join to form a $swarm$ (i.e. a cluster) of similar representatives (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 $\beta\mu$, 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 worth to note that, the swarms of representative agents avoid the off-line phase of $DenStream$ to apply a clustering algorithm to get the final result. In fact a swarm represents a cluster, thus the clustering generation on demand by the user can be satisfied at any time by simply showing all the swarms computed so far. Figure 5 illustrates the virtual space at a generic iteration. The figure points out the presence of swarms of agents, as well as the presence of isolated agents not yet aggregated with other similar agents. In the next section we show that our approach successfully detects clusters in evolving data streams.

```
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($\vec{v}$);}
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($\vec{v}$);}}
13. end foreach
14. end for
```

Figure 4. The pseudo-code of the FlockStream algorithm.

5. Experimental Results

In this section we study the effectiveness of *FlockStream* on real and synthetic datasets. The 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 6(a). Each of them contains 10,000 points and they are similar to those employed by Cao et al. in [4] 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

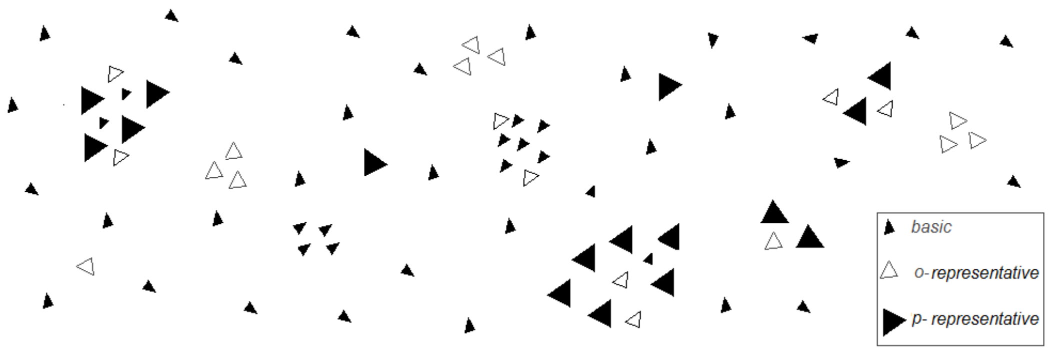

Figure 5. Visualization of swarms of agents and isolated agents on the virtual space at a generic iteration.

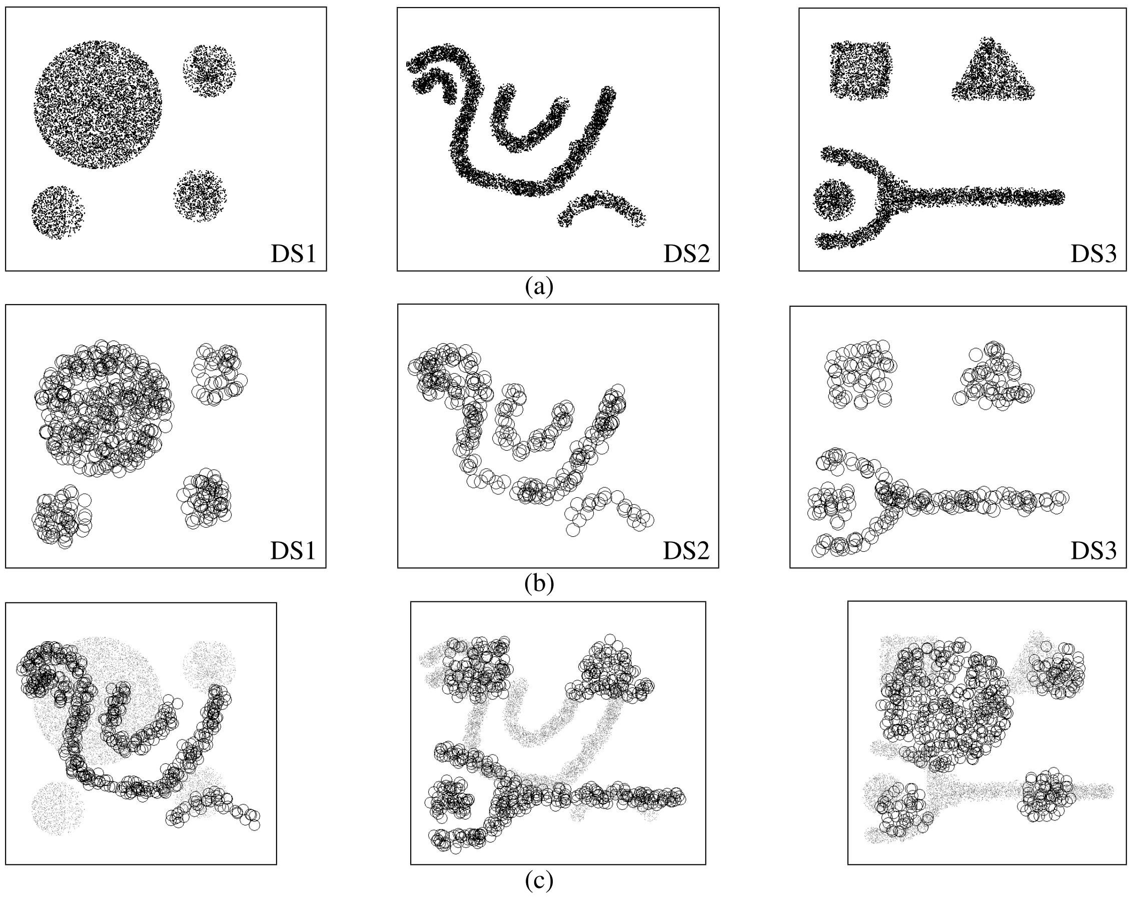

Figure 6. (a) Synthetic data sets.(b) Clustering performed by FlockStream on the synthetic datasets. (c) Clustering performed by FlockStream on the evolving data stream EDS.

number of points with the dominant class label in cluster i, and $|C_i|$ denotes the number of 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-

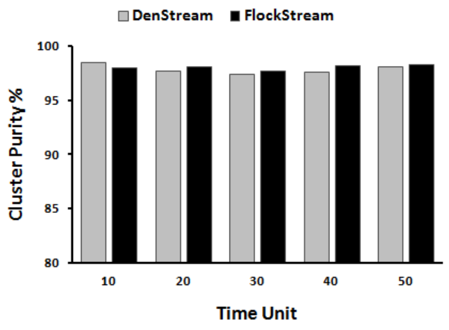

Figure 7. Clustering quality for evolving data stream EDS with horizon=2 and stream speed=2000.

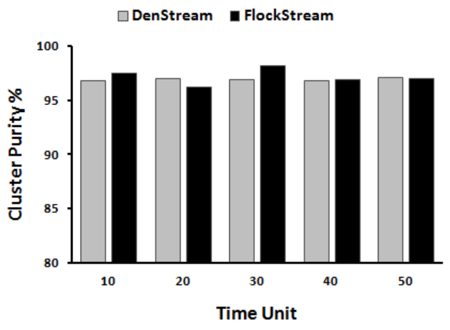

Figure 8. Clustering quality for evolving data stream EDS with horizon=10 and stream speed=1000.

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 6(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 6(c). In the figure, points indicate the raw data while circles denote the micro-clusters. It can be seen that *FlockStream* captures the shape of each cluster as the data streams evolve.

The purity results of *FlockStream* compared to $DenStream$ on the EDS data stream are shown in figure 7. Here, the 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 8 shows the results of *FlockStream* when the stream speed is set to 1000 points per time unit and the 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 9 and 10. 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 achieved 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 11 and 12 respectively. The results demonstrate that *FlockStream* achieves high clustering quality, comparable to $DenStream$, also when noise is present.

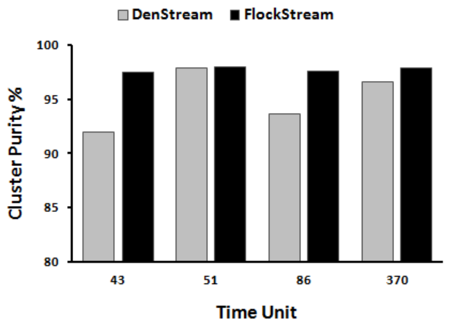

Figure 9. Clustering quality for Network Intrusion dataset with horizon=1 and stream speed=1000.

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. Another main novelty of the approach is that the two-phase scheme of density-based data stream clustering methods is replaced by a unique online phase, in which 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. Experimental results on real and synthetic data sets confirm the validity of the approach proposed. Future work aims at extending the method to a distributed framework, more apt to real life applications.

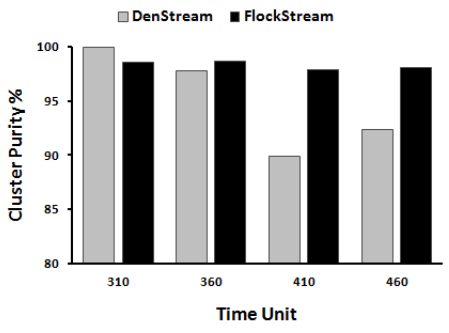

Figure 10. Clustering quality for Network Intrusion dataset with horizon=5 and stream speed=1000.

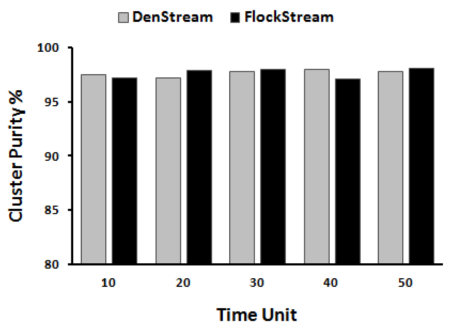

Figure 11. Clustering quality for evolving data stream EDS with horizon=2, stream speed=2000 and with 1% noise.

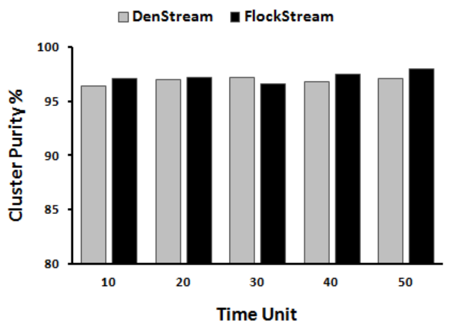

Figure 12. Clustering quality for evolving data stream EDS with horizon=10, stream speed=1000 and with 5% noise.

References

[1] C. C. Aggarwal, J. Han, J. Wang, and P. Yu. On clustering massive data streams: a summarization paradigm. In *in Data Streams- Models and Algorithms, Charu C. Aggarwal eds.*, pages 11–38. Springer, 2006.

[2] B. Babock, M. Datar, R. Motwani, and L. O'Callaghan. Maintaining varaince and k-medians over data stream windows. In *Proceedings of the 22nd ACM Symposium on Principles of Data base Systems (PODS 2003)*, pages 234–243, 2003.

[3] D. Barbará. Requirements for clustering data streams. *SIGKDD Explorations Newsletter*, 3(2):23–27, 2002.

[4] F. Cao, M. Ester, W. Qian, and A. 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.

[5] Y. Chen and L. Tu. Density-based clustering for real-time stream data. In *Proceedings of the 13th ACM SIGKDD International conference on Knowledge discovery and data mining (KDD'07)*, pages 133 – 142. ACM, 2007.

[6] X. Cui, J. Gao, and T. E. Potok. A flocking based algorithm for document clustering analysis. *Journal of Systems Architecture*, 52(8-9):505–515, 2006.

[7] X. Cui and T. E. Potok. A distributed agent implementation of multiple species flocking model for document partitioning clustering. In *Cooperative Information Agents*, pages 124–137, 2006.

[8] R. C. Eberhart, Y. Shi, and J. Kennedy. *Swarm Intelligence (The Morgan Kaufmann Series in Artificial Intelligence)*. Morgan Kaufmann, March 2001.

[9] M. Ester, H.-P. Kriegel, J. Sander, and X. 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.

[10] S. Gua, A. Meyerson, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data strams: Theory and practise. *IEEE Transactions on Knowledge and Data Engineering*, 15(3):515–528, 2003.

[11] S. Gua, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams. In *Proceedings of the Annual IEEE Symposium on Foundations of Computer Science*, pages 359–366, 2000.

[12] L. O'Callaghan, N. Mishra, N. Mishra, and S. Gua. Streaming-data algorithms for high quality clustering. In *Proceedings of the 18th International Conference on Data Engineering (ICDE'01)*, pages 685–694, 2002.

[13] C. 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.

[14] A. Zhou, F. Cao, W. Qian, and C. Jin. Tracking clusters in evolving data streams over sliding windows. *Knowledge and Information Systems*, 15(2):181–214, 2007.