Data Min Knowl Disc
DOI 10.1007/s10618-011-0242-x

A single pass algorithm for clustering evolving data streams based on swarm intelligence

Agostino Forestiero · Clara Pizzuti · Giandomenico Spezzano

Received: 24 May 2010 / Accepted: 19 October 2011

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 article 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 computing demanding offline cluster computation, since a swarm represents a cluster. Experimental results show that the bio-inspired approach can obtain very good results on real and synthetic data sets.

Keywords Data streams · Density-based clustering · Bio-inspired flocking model

Responsible editor: Charu Aggarwal.

A. Forestiero · C. Pizzuti (✉) · G. Spezzano
National Research Council of Italy–CNR, Via P. Bucci 41C, Rende (CS), 87036, Italy
e-mail: pizzuti@icar.cnr.it

A. Forestiero
e-mail: forestiero@icar.cnr.it

G. Spezzano
e-mail: spezzano@icar.cnr.it

Published online: 02 November 2011

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 Second, using an entire data stream of a long period could mislead the analysis results due to the weight of outdated data, considered in the same way as more recent data.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 continuous 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 (Aggarwal et al. 2003, 2006; Babock et al. 2003; Barbará 2002; Cao et al. 2006; Chen and Li 2007; Guha et al. 2000; O'Callaghan et al. 2002; Guha et al. 2003; Nasraoui et al. 2003; Nasraoui and Coronel 2006; Zhou et al. 2007). Aggarwal et al. (2003) have been the first to address the problem of the impossibility to revise a data stream during the computation, and the evolving characteristics of stream data. 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. However, their algorithm, $CluStream$, being based on the k-means approach, finds only spherical clusters and needs to know in advance the number of clusters. Since density-based clustering (Ester et al. 1996) overcomes these limitations, the density-based framework has been recently extended to the two-phase scheme described above (Cao et al. 2006; Chen and Li 2007; Li and Chen 2009; Li et al. 2009). All these methods use the fading data model to deal with the evolving cluster distribution and discover changes in stream data.

As pointed out in Li et al. (2009), the offline component of the two-phase scheme is a computing demanding step because it actually has to apply a clustering algorithm on the synopsis to produce the result. Another problem is that to discover changes in cluster distribution, the offline phase must be executed frequently. A proposal to detect cluster evolution in constant time has been suggested by Li et al. (2009).

In this article we present a bio-inspired and density-based clustering method for data streams that merges the online and offline phases of the above described scheme, by making always available the current clustering result on demand without any further computation. The algorithm, named *FlockStream*, analogously to the mentioned density-based methods, uses the damped window model to deal with cluster evolution, and transposes the concepts of potential and outlier micro-cluster introduced in Cao et al. (2006) into the bio-inspired framework. *FlockStream*, in fact, employs a multi-agent system that uses a decentralized bottom-up self-organizing strategy to group similar data points. 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* (Eberhart et al. 2001). 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.

The movement of the agents in the 2D space, thus, is not random, but it is guided by the similarity function that aggregates the agents to their closer neighbors. As several similar micro-clusters can be created, by applying the flocking rules, they are aggregated to form swarms of close micro-clusters.

The main contributions of *FlockStream* can be summarized as follows.

- *FlockStream* 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 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.
- Since each agent interacts only with the other agents present in its visibility range, it does not compare itself with all the other agents. Thus a reduced number of distance computations is performed. Such a number depends on how many agents it encounters during its movement in the virtual space. This implies that the algorithm is very efficient and achieves a linear speed-up with respect to the input size.
- Flocks of agents can join together into swarms of similar groups. The two-phase scheme of data stream clustering methods mentioned above is thus 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 delivering all the swarms computed so far.
- The evolving nature of clusters can be tracked 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 ask for clustering results.
- *FlockStream* allows the user to obtain an approximate but faster result by reducing the time agents can move onto the virtual space.
- The method is robust to noise. In fact, in the experimental results we show that *FlockStream* obtains clusters of high quality also when noise is present.

This article is organized as follows. The next Section reviews the main proposals on clustering data streams. Section 3 describes the algorithm of Cao et al. (2006), that inspired our method. Section 4 introduces the Flocking model. Section 5 describes our approach. In Sect. 6 the results of the method on synthetic and real life data sets are presented and compared with those obtained by Cao et al. (2006). Section 7, finally, discusses the advantages of the approach and concludes this article.

2 Related work

In the last few years special attentions has been paid towards searching efficient and efficacious methods for clustering data streams (Aggarwal 2007). The first approaches

adopted the single pass paradigm (Charikar et al. 2003; Guha et al. 2000; O'Callaghan et al. 2002; Guha et al. 2003) to deal with the data stream requirement that data can be examined only once. According to this paradigm, as data is scanned, summaries of past data are stored to leave enough memory for processing new incoming data. These algorithms apply a divide-and-conquer technique that partitions the data stream in disjoint pieces and clusters each piece by extending the k-Median algorithm. A theoretical study of the approximation error obtained in using the extended schema is also provided in Guha et al. (2003). The main drawbacks of these approaches are that the number of clusters must be given as input parameter, and they are not able to capture changes in the data stream since outdated and recent data have the same weight.

In order to take into account the evolution of data streams, Aggarwal et al. (2003, 2006) proposed a new model based on a two-phase schema: 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. The summary information is defined by two structures: the micro-clusters and the pyramidal time frame. The micro-clusters maintain statistical information about the data locality. Furthermore, the micro-clusters are stored at snapshots in time which follow a pyramidal pattern. This pattern provides a trade-off between the storage requirements and the ability to recall summary statistics from different time horizons. The proposed algorithm $CluStream$ uses the micro-cluster structure to cluster streaming data. The micro-clusters are built online by exploiting ideas from the k-means and nearest neighbor algorithms. At any moment, q micro-clusters are assumed to be present. Generally q is a significantly larger value of the natural number of clusters. At the beginning q micro-clusters are built by using the k-means algorithm. Whenever a new point X is arrived, the distances from X to the micro-clusters centers are computed, and X is assigned to the nearest micro-cluster if it lies within its maximum boundary. Otherwise a new micro-cluster is created containing the data point X. In this case, since the number of micro-clusters must maintain constant, either a micro-cluster must be deleted or two micro-clusters must be joined. The $CluStream$ algorithm has two main limitations. It is unable to discover clusters of arbitrary shapes, and the number k of clusters must be fixed a-priori.

The two-phase paradigm of $CluStream$ has been inspiring many data stream clustering algorithms both to improve it (Wang et al. 2004) and to allow the clustering of multiple data streams (Dai et al. 2006), parallel (Beringher and Hullermeier 2006) and distributed data streams (Sanghamitra et al. 2006).

More recently, extensions of density-based clustering (Ester et al. 1996) to deal with streams of data have been proposed. Density-based methods overcome the drawbacks of $CluStream$ since they are able to find clusters of any shape and do not require prior knowledge about the number of clusters. The main proposals in this context are the *DenStream* algorithm of Cao et al. (2006), the *D-Stream* method of Li and Chen (2009), and the *MR-Stream* algorithm of Li et al. (2009).

DenStream, described in details in the next section, extends the concept of *core point* introduced in $DBSCAN$ (Ester et al. 1996) and employs the notion of *micro-cluster* to store an approximate representation of the data points in a damped window model.

D-Stream uses a different approach based on partitioning the data space into discretized grids where new data points are mapped. A decay factor is associated with the density of each data point to give less importance to historical information and more weight to recent data. The relation between time horizon, decay factor, and data density is studied to guarantee the generation of high quality clusters. High dimensional data could rapidly increase the number of grids. The authors found that, in practice, many grids are sparse or empty, thus they developed an efficient technique to manage them. The experiments reported have a maximum dimensionality of 40. However, as the number of dimensions augments, the number of grid cells increases exponentially. Thus, even though the empty grid cells do not need to be explicitly stored, there could be many cells containing only one point. This means that *D-Stream* could not be able to deal with very high dimensional data, analogously to grid-based clustering approaches, that have poor performance on high-dimensional data (Tan et al. 2006). The $DenStream$ algorithm, on the contrary, does not suffer of this problem. A comparison with $CluStream$ shows the better performance of *D-Stream* with respect to $CluStream$. At present no comparison exists between $DenStream$ and *D-Stream*.

MR-Stream, analogously to *D-Stream*, partitions the search space in cells. Each time a dimension is divided in two, a cell can be further partitioned in 2^d subcells, where d is the data set dimensionality. A user-defined parameter, however, limits the maximum of divisions that can be done and a quadtree structure is used to store the space partitioning. The tree structure allows the data clustering at different resolution levels. During the online phase, at each time stamp, *MR-Stream* assigns new incoming data to the appropriate cell and updates the synopsis information. The offline component obtains a portion of the tree at a fixed hight h and performs clustering at the resolution level determined by h. In order to discover changes in the data, the authors provide a time gap value, denoted t_p, for checking the change of a cell from dense to sparse. They prove that evolving cluster information can be obtained by sampling the memory cost, and thus the number of nodes in the tree, every t_p intervals. This gives an insight when to execute the offline phase. A comparison of *MR-Stream* with *D-Stream* shows the better performance of the former method.

Many approaches to clustering based on the bio-inspired paradigm have been proposed (Azzag et al. 2003; Folino et al. 2009; Hamdi et al. 2008; Handl and Meyer 2007; Liu et al. 2004). None of them has been designed to cope with data streams. There is only one proposal from Cui and Potok (2006b) that uses the flocking model to cluster streams of documents. Analogously to our approach, each agent contains the feature vector of data point it represents. In this case a data point is a document. The stream of documents is simulated by periodically changing the feature vector of each agent. Thus they neither summarize nor take into account the past information since, after a prefixed time unit, old documents are just discarded, and the new documents are re-elaborated from scratch to generate new clusters.

In the next section a detailed description of $DenStream$ is given. Our approach, in fact, adopts the concepts introduced by $DenStream$ adapting them for the flocking model.

3 The *DenStream* algorithm

DenStream (Cao et al. 2006) 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 time unit. Historical data diminishes its importance when λ assumes higher values.

The authors extend the concept of *core point* introduced in $DBSCAN$ (Ester et al. 1996) 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 $w = \sum_{j=1}^{n} f(t - T_{i_j})$ must be such that $w \geq \mu$. The center is defined as

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

and the radius

$$r = \frac{\sum_{j=1}^{n} f(t - T_{i_j}) dist(p_{i_j}, c)}{w} \tag{2}$$

is such that $r \leq \varepsilon$. $dist(p_{i_j}, c)$ 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 predefined 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 Aggarwal et al. (2006) 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 $w \geq \beta\mu$. $\beta, 0 < \beta \leq 1$ is a parameter defining the outlierness threshold relative to c-micro-clusters.

$$\overline{CF^1} = \sum_{j=1}^{n} f(t - T_{i_j}) p_{i_j} \tag{3}$$

is the weighted linear sum of the points,

$$\overline{CF^2} = \sum_{j=1}^{n} f(t - T_{i_j}) p_{i_j}^2 \tag{4}$$

is the weighed square sum of the points. The center of a p-micro-cluster is

$$c = \frac{\overline{CF^1}}{w} \tag{5}$$

and the radius $r \leq \varepsilon$ is

$$r = \sqrt{\frac{\overline{CF^2}}{w} - \left(\frac{\overline{CF^1}}{w}\right)^2} \tag{6}$$

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 radius r_p of c_p does not augment above ε, i.e. $r_p \leq \varepsilon$. 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 $r_o \leq \varepsilon$. 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 overcomes these disadvantages is described.

4 The flocking model

The *flocking model* (Eberhart et al. 2001) is a biologically inspired computational model for simulating the animation of a flock of entities. In this model each individual 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.

Flocking in biology is an example of self-organizing complex system that reflects the idea at the base of distributed and self-organized systems asserting that a population of autonomous and independent agents interacting only locally may generate *intelligent behavior*. A main characteristic of this intelligent behavior is the emergence of new patterns, structures, and properties observable at a macro-level, though generated at a micro-level, due to the agents interactions.

The basic flocking model was first proposed by Reynolds (1987), 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 collisions.

A *Multiple Species Flocking* (MSF) model has been developed by Cui and Potok (2006a) to more accurately simulate flocking behavior among an heterogeneous population of entities. This model 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. Cui and Potok use the concept of *velocity vector* of the flock to describe the alignment, cohesion, separation, and feature (dis)similarity rules, and show how the nearby boids bias the boid's velocity. In our approach we adopt the Multiple Species Flocking model, but we do not introduce the similarity and dissimilarity rules proposed by Cui and Potok. Our MFS model modifies the basic flocking rules to take into account the (dis)similarity of an agent with its neighbors.

Let $\mathscr{R}^d$ denote the d-dimensional feature space of the data stream points and $\mathscr{R}_v^2$ the two dimensional Cartesian space representing the virtual space where agents are deployed and move according to the flocking rules. The virtual space is assumed to be discrete and not continuous, and it is implemented as a two dimensional toroidal grid of fixed size, where each cell of the grid can contain only one agent and determines the position $P = (x, y)$ of the agent. Each data point $p = \{x_1, \ldots, x_d\}$ in $\mathscr{R}^d$ is associated with a boid A in the virtual space $\mathscr{R}_v^2$. An agent $A = (P, \overrightarrow{v})$ is represented through its position $P = (x, y)$ in the virtual space and a *velocity vector* $\overrightarrow{v} = (m, \theta)$ with magnitude m and direction determined by the angle θ formed between $\overrightarrow{v}$ and the positive x axis. We assume that the magnitude is constant for all the boids and it is equal to 1. This means that in the virtual space a boid moves one cell at a time.

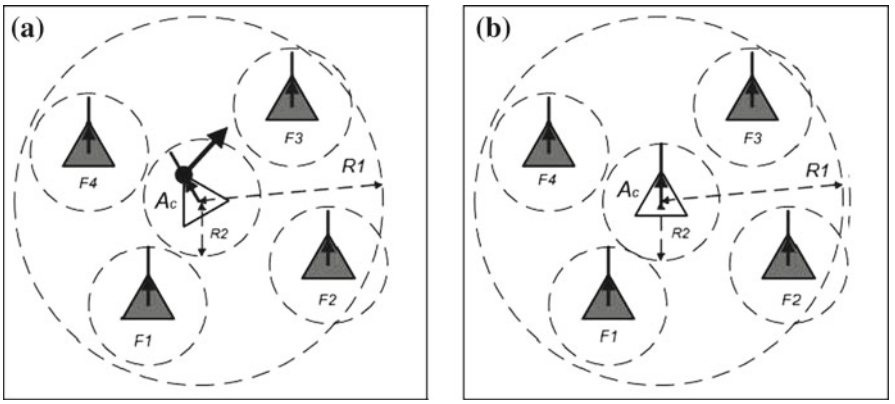

Fig. 1 Alignment rule: **a** initial direction of the current agent A_c, **b** direction after the alignment with the nearby boids

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. Let p_c be a data point in the feature space, and A_c the associated agent in the virtual space. Suppose that A_c has $F_1, \ldots, F_n$ boids in its visibility range R_1, i.e. $d_v(F_i, A_c) \leq R_1, i = 1, \ldots, n$, where d_v denotes the Euclidean distance, computed in $\mathscr{R}_v^2$, between the points $P_{F_i} = (x_{F_i}, y_{F_i})$ and $P_{A_c} = (x_{A_c}, y_{A_c})$ representing the positions of the boids F_i and A_c respectively, and that $p_1, \ldots, p_n$ are the data points corresponding to the agents $F_1, \ldots, F_n$. $dist(p_i, p_c), i = 1, \ldots, n$, denotes the Euclidean distance in $\mathscr{R}^d$ between p_c and the p_i points.

We say that two agents A_c and F_i are *similar* if $dist(p_i, p_c) \leq \varepsilon$, where ε is the maximum threshold value that the radius of a micro-cluster can assume, as described in Sect. 3, formulas (1) and (2). Thus ε determines the neighborhood of p_c in the feature space.

The basic behavior rules of a single entity of our MSF model are illustrated in Figs. 1, 2, and 3. The alignment rule means that a boid tends to move in the same direction of the nearby and similar 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 Fig. 1 and it is formally described as

$$for\ i\ \in \{i, \ldots, n\},\ if\ dist(p_i, p_c) \leq \varepsilon \wedge d_v(F_i, A_c) \leq R_1\ \wedge\ d_v(F_i, A_c) \geq R_2 \Rightarrow$$

$$\overrightarrow{v}_{ar} = \frac{1}{n} \sum_{i=1}^{n} \overrightarrow{v}_i$$

where ε is the neighborhood of p_c. Thus the alignment rule computes the velocity vector $\overrightarrow{v}_{ar}$ of the current boid A_c as the mean velocity of all those boids contained in its visibility range such that the corresponding data points in the feature space are contained in the ε neighborhood of p_c.

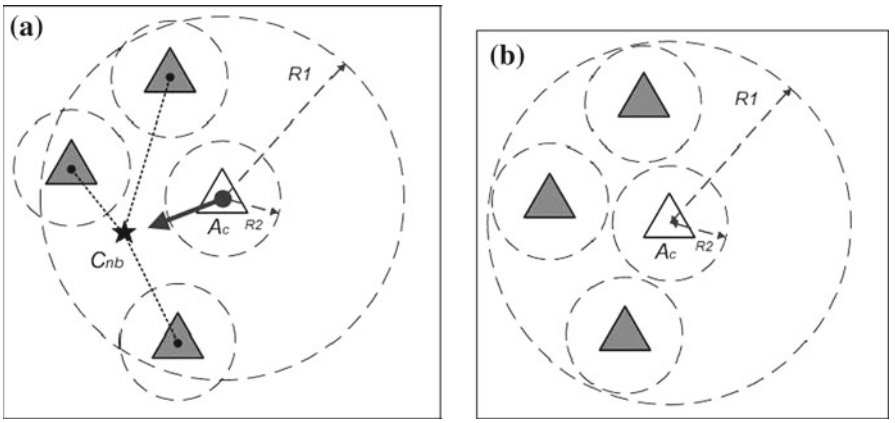

Fig. 2 Cohesion rule: **a** a boid distant from the flockmates, **b** its position after the cohesion

The cohesion rule moves a boid towards other nearby and similar boids (unless another boid is too close) by orienting the velocity vector of the boid in the direction of the centroid of the local flock. Let P_c and $P_i, i = 1, \ldots, n$, be the positions of the current boid A_c and its neighbor boids F_i, respectively. The centroid $C = \frac{1}{n} \sum_{i=1}^{n} P_i$ is the average position of $P_1, \ldots, P_n$. The cohesion velocity $\overrightarrow{v}_{cr}$ is computed as follows:

$$for\ i\ \in \{i, \ldots, n\},\ if\ dist(p_i, p_c) \leq \varepsilon\ \wedge d_v(F_i, A_c) \leq R_1 \wedge d_v(F_i, A_c) \geq R_2 \Rightarrow$$

$$\overrightarrow{v}_{cr} = \overrightarrow{C_{nb} P_c}$$

where C_{nb} is the centroid of those boids present in the visibility range of A_c, such that the corresponding data points in the feature space *are contained* in the ε neighborhood of p_c, and $\overrightarrow{C_{nb} P_c}$ denotes the direction of the line joining the position P_c of the current agent with the position of C_{nb}. Figure 2 depicts the cohesion rule.

The separation rule avoids that a boid moves towards its neighboring boids if either it is too close to another boid, or it is not similar to them. The separation velocity $\overrightarrow{v}_{sr}$ is computed as follows:

$$for\ i\ \in \{i, \ldots, n\}\ if\ dist(p_i, p_c) > \varepsilon\ \vee d_v(F_i, A_c) \leq R_2 \Rightarrow$$

$$\overrightarrow{v}_{sr} = \overrightarrow{C_{db} P_c}$$

where C_{db} is the centroid of those boids contained in the visibility range of A_c, such that the corresponding data points in the feature space *are not contained* in the ε neighborhood of p_c, and $\overrightarrow{C_{db} P_c}$ denotes the direction of the line joining the position P_c of the current agent with the position of C_{db}. The separation velocity moves the current

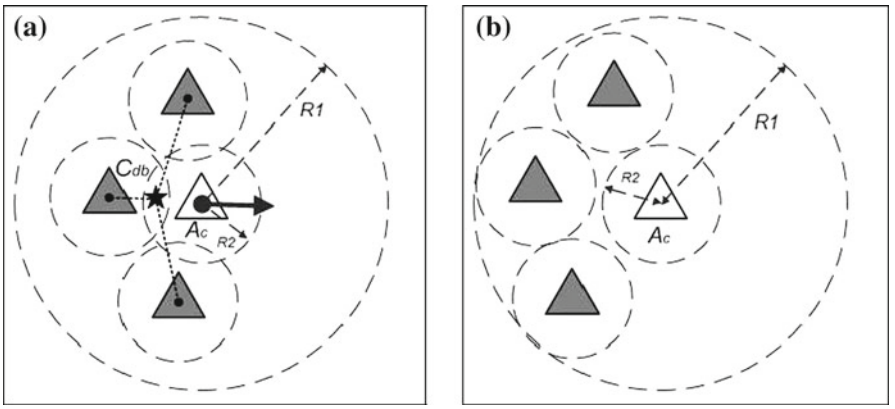

Fig. 3 Separation rule: **a** position of the agent A_c with respect to its flockmates, **b** position of A_c after the separation

boid A_c along the line joining P_c and the centroid C_{db} but in the opposite sense. This rule is shown in Fig. 3.

When two boids are too close, the separation rule overrides the other two, which are deactivated until the minimum separation is achieved. The flocking behavior of each agent A_c can be expressed by linearly combining the velocities calculated by all the rules described above, i.e. $\overrightarrow{v}_A = \overrightarrow{v}_{ar} + \overrightarrow{v}_{cr} + \overrightarrow{v}_{sr}$.

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 similar to them, new results can be quickly regenerated when adding entities boids or deleting part of boids at run time. This characteristic allows the flocking algorithm to be applied to clustering to dynamically analyze changing stream information.

5 The *FlockStream* algorithm

FlockStream is a heuristic density-based data stream clustering algorithm built on the Multiple Species Flocking model described in the previous section. The algorithm uses agents (i.e. boids) with distinct simple functionalities to mimic the flocking behavior. As said above, each multi-dimensional data item is associated with an agent. Agents are arranged on the virtual space constituted by a two-dimensional grid of fixed size. Every cell in the grid contains an agent. In our approach, in addition to the standard action rules of the flocking model, an extension to the flocking model is introduced 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, corresponding to the set of initial data points, is deployed onto the virtual space. The position P of each agent $A = (P, \overrightarrow{v})$ on the grid is generated at random. Its velocity vector $\overrightarrow{v} = (m, \theta)$ is initialized such that $m = 1$ and the angle θ assumes a random value in the interval [0, 360]. 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 in the feature space $\mathscr{R}^d$, will group together and become a flock, while dissimilar agents will be moved away from the flock. As already described in the previous section, two agents are considered similar if the distance of the corresponding data points in the feature space is less than ε, the maximum threshold value that the radius of a micro-cluster can assume.

It is worth to note that, though the agents are clustered in the virtual space, their similarity and the summary statistics described in Sect. 3 are computed in the feature space. While iterating, the behavior (*velocity*) of each agent A with position P_A is influenced by all the agents F with position P_F in its visibility radius. The agent's velocity is computed by applying the rules described in the previous section. These rules induce an adaptive behavior to the algorithm since an agent can leave the group it participates for another group on the basis of the agents F it encounters during its motion. 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, the summary statistics described in Sect. 3 are computed and the stream of data is discarded. As result of this initial phase we have two new types of agents: a p-representative agent, corresponding to a *p-micro-cluster* $c_p = \{\overline{CF^1}, \overline{CF^2}, w\}$ and an o-representative agent that corresponds to an *o-micro-cluster* $c_o = \{\overline{CF^1}, \overline{CF^2}, w, t_o\}$, as defined in Sect. 3.

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 maximum number of iterations. The agent maintenance algorithm of *FlockStream* is summarized in Fig. 4. Analogously to *DenStream*, different cases can occur:

- A basic agent A, corresponding to a data point $p_A \in \mathscr{R}^d$, meets another basic agent B corresponding to a data point $p_B \in \mathscr{R}^d$. The similarity between the two agents is calculated and, if $dist(p_A, p_B) \leq \varepsilon$, A is joined with B to form an *o-representative*.
- A basic agent A meets either a *p-representative* B, corresponding to a p-micro-cluster c_p^B, or an *o-representative* B, corresponding to an o-micro-cluster c_o^B, in its visibility range. A is merged with B if the new radius r_p of c_p^B (r_o of c_o^B respectively) is below or equal to ε. 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 similar to it.
- a *p-representative* A, corresponding to a p-micro-cluster c_p^A, or an *o-representative* A, corresponding to an o-micro-cluster c_o^A, encounters another representative agent

```
while a new bulk of data arrives
for i=1 ... MaxIterations
    for each agent A
        if A is a basic agent, ∀ agents B contained in its visibility range
            if B is another basic agent ∧ dist(p_A, p_B) ≤ ε
                join A and B to form an o-representative
            else ( B is a p-representative or an o-representative)
                compute the new radius r_p (r_o resp.) of c_p^B (c_o^B resp.)
                if r_p ≤ ε (r_o ≤ ε resp.)
                    A is merged with B
                end if
        else
            the agent A is a p-representative (an o-representative ),
            ∀ agents B contained in its visibility range
                if B is another p-representative agent (o-representative)
                    ∧ dist(c_p^A, c_p^B) ≤ ε ( dist(c_o^A, c_o^B) ≤ ε resp.)
                    join A and B to form a swarm of similar representatives
                else (B is a basic agent)
                    B is joined with A, provided that dist(c_p^A, p_B) ≤ ε (dist(c_o^A, p_B) ≤ ε resp.)
                end if
        end if
        compute the velocity vector of the agent A by applying the MSF rules
        move the agent
    end for each
end for
    update summary statistics
    compute the weight of each agent by applying the fading function
end while
```

Fig. 4 The pseudo-code of the FlockStream algorithm

B in its visibility range. If the distance between the corresponding micro-clusters is below ε then they join to form a $swarm$ (i.e. a cluster) of similar representatives.

– a *p-representative* or an *o-representative* A encounters a basic agent B in its visibility range. In such a case B is joined with A, provided that the corresponding data point p_B is similar to the micro-cluster associated with A.

Note that, when an agent in the virtual space is a *p-representative* or an *o-representative*, it corresponds to a *p-micro-cluster* or an *o-micro-cluster* in the feature space. In such a case the distance between a data point and a micro-cluster can be computed by considering the summary $\overline{CF^1}$ as the feature vector of the micro-cluster.

At this point the agent computes its new velocity vector by applying the MSF rules described in the previous section and moves onto the virtual space according to it. At the end of the maximum number of iterations allowed, for each swarm, the summary statistics (formulas (3), (4), (5), (6) of Sect. 3) 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.

Note that the swarms of representative agents avoid the offline phase of $DenStream$ that applies 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 different steps of the algorithm execution. The initialization step starts by deploying a bulk of agents randomly on the virtual space, as showed in Fig. 5a. After a

number of iterations, agents begin to aggregate with other similar agents (Fig. 5b). As the agents continue to move, Fig. 5c depicts groups of boids, enclosed in ellipses, that will become *p-representatives* (black triangles) or *o-representatives* (white triangles) and single agents (small black triangles), not yet joined to any group. Figure 5d shows the virtual space after the max number of iterations has been executed and a new bulk of boids is deployed onto the virtual space. The figure points out the presence of swarms of agents, as well as the presence of isolated agents. These agents include some agents remained from the previous step not aggregated with other similar agents, and the new ones. In the next section we show that our approach successfully detects clusters in evolving data streams.

6 Experimental results

In this section we study the effectiveness of *FlockStream* on both synthetic and real-life data sets, and compare it with $DenStream$. Both *FlockStream* and $DenStream$[1] algorithms have been implemented in Java and all the experiments have been performed on an Intel(R) Core(TM)2 6600 having 2GB of memory. The parameters used for *FlockStream* and $DenStream$ are the same, that is initial number of points/agents $Na = 1000$, stream speed $v = 1000$, decay factor $\lambda = 0.25$, maximum micro-cluster radius $\varepsilon = 16$, minimum number of points/agents necessary to create a p-micro-cluster $\mu = 10$, outlier threshold $\beta = 0.2$. The number of iterations $MaxIterations$ for *FlockStream* has been fixed to 1000. In the following by *horizon* (or window) we mean how many time steps from the current time we consider when running the clustering algorithms. Thus, for example, if the horizon is 10, it means that the last 10 blocks of data are clustered.

The synthetic datasets used, named DS1, DS2 and DS3, are showed in Fig. 6a. Each of them contains 10,000 points and they are similar to those employed by Cao et al. (2006) 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.

The two real data sets are the *KDD Cup 1999 Data set*[2] and the *Forest Covertype data set*.[3] The former 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 *Forest Covertype* data set is the prevision forest cover type from cartographic variables only (no remotely sensed data), made available from the Remote Sensing

[1] We implemented the $DenStream$ algorithm since its code is not available from the authors.

[2] http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html.

[3] http://archive.ics.uci.edu/ml/machine-learning-databases/covtype/.

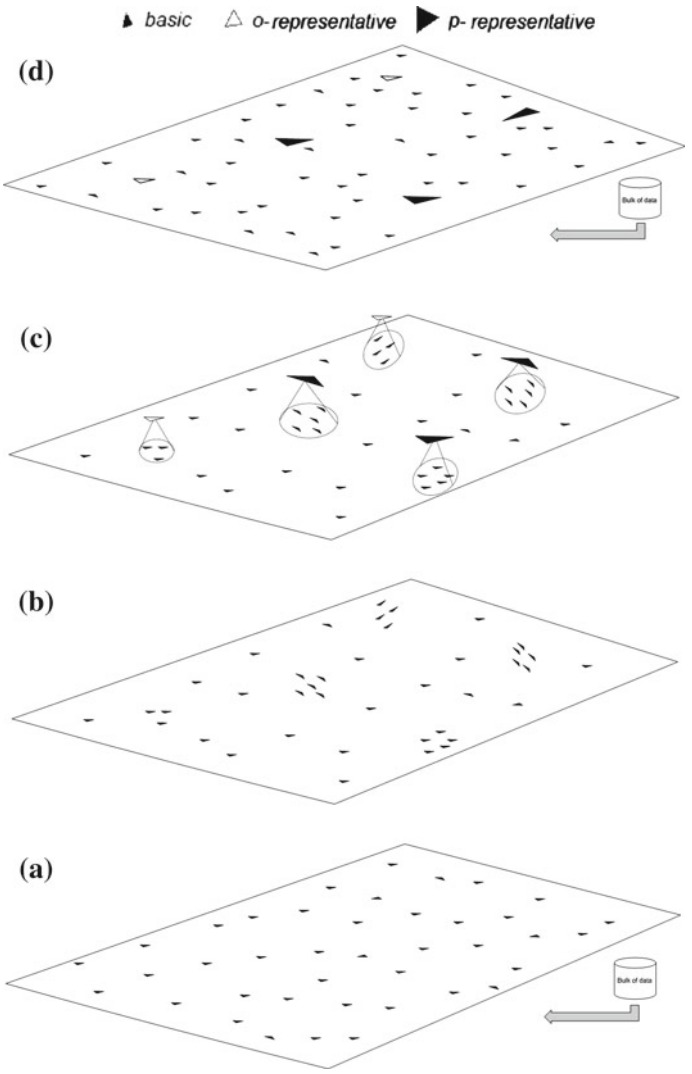

Fig. 5 Visualization of swarms of agents and isolated agents on the virtual space at different iterations. **a** First step of initialization: a bulk of data deployed onto the virtual space, **b** agents begin to aggregate, **c** groups of boids, enclosed in *ellipses*, that will become *p-representatives* (*black triangles*) or *o-representatives* (*white triangles*) and single agents (*small black triangles*), **d** virtual space after the max number of iterations executed and the deploying of a new bulk of boids

and GIS Program Department of Forest Sciences College of Natural Resources, Colorado State University Fort Collins. The area includes four wilderness areas located in the Roosevelt National Forest of northern Colorado. The data set is composed by 581,012 instances (observations) represented by 54 geological and geographical attributes describing the environment in which trees were observed. Both data sets have been transformed into data streams by taking the input order as the streaming order.

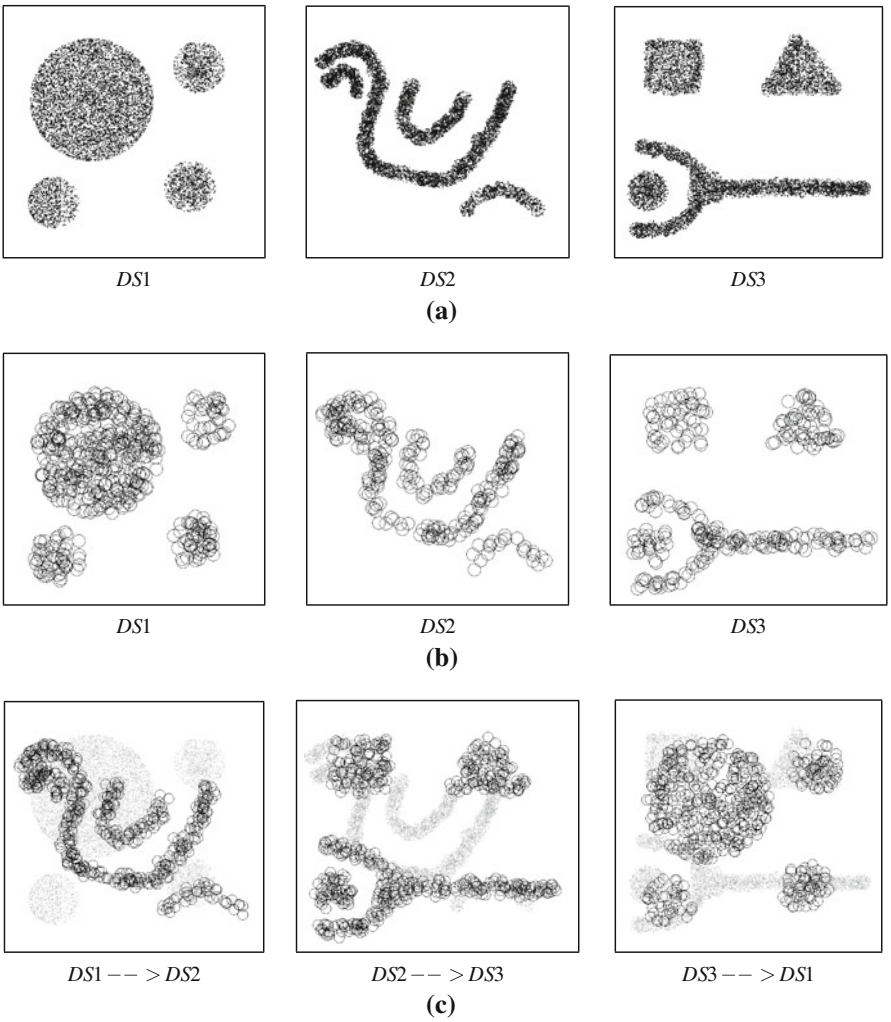

Fig. 6 **a** Synthetic data sets. **b** Clustering performed by *FlockStream* on the synthetic data sets. **c** Clustering performed by *FlockStream* on the evolving data stream EDS

6.1 Evaluation metrics

Since for all the data sets used in the experiments the true class labels are known, the quality of the clustering obtained has been evaluated by considering two well known external criteria: the *average purity* of the clusters and the *normalized mutual information*. Cluster purity measures the purity of the clusters obtained with respect to the true clusters by assigning each cluster to the most frequent class appearing in the cluster. More formally, the average purity of a clustering is defined as:

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

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 points in cluster i. The purity is calculated only for the points arriving in a predefined window, since the weight of points diminishes continuously.

The *normalized mutual information* (NMI) is a well known information theoretic measure that assesses how similar two clusterings are. Given the true clustering $A = \{A_1, \ldots, A_k\}$ and the grouping $B = \{B_1, \ldots, B_h\}$ obtained by a clustering method, let C be the confusion matrix whose element C_{ij} is the number of records of cluster i of A that are also in the cluster j of B. The normalized mutual information $NMI(A, B)$ is defined as:

$$NMI(A, B) = \frac{-2 \sum_{i=1}^{c_A} \sum_{j=1}^{c_B} C_{ij} log(C_{ij} N / C_{i.} C_{.j})}{\sum_{i=1}^{c_A} C_{i.} log(C_{i.}/N) + \sum_{j=1}^{c_B} C_{.j} log(C_{.j}/N)}$$

where c_A (c_B) is the number of groups in the partition A (B), $C_{i.}$ ($C_{.j}$) is the sum of the elements of C in row i (column j), and N is the number of points. If $A = B$, $NMI(A, B) = 1$. If A and B are completely different, $NMI(A, B) = 0$.

Another evaluation criterion we report in the experiments is the number of distance computations executed by *FlockStream* and *DenStream*. In fact, one of the main differences between *FlockStream* and *DenStream* is the computation of the nearest micro-cluster of a point. When a new data point is elaborated, in order to determine whether it should be merged into an existing micro-cluster or consider it as the seed for a new group, *DenStream* needs to do a comparison with all the micro-clusters generated so far. Thus *DenStream* finds the true nearest micro-cluster of a data point. *FlockStream*, instead, computes the distance between the current agent and the other agents encountered during its movement on the virtual space. This means that *FlockStream* returns an approximate nearest neighbor distance. In the following we show that the number of distance computations executed by *FlockStream* is much lower than that done by *DenStream*, and, despite the approximation returned, the cluster quality is higher.

6.2 Synthetic data sets

We first evaluate the quality of the clusters obtained by the *FlockStream* algorithm to check the ability of the method to get the shape of each cluster. The results on the non-evolving datasets DS1, DS2 and DS3 are reported in Fig. 6b. In this figure the circles indicate the micro-clusters 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 Fig. 6c. In the figure, points indicate the raw data while circles denote the micro-clusters. It can been seen that also in this case *FlockStream* captures the shape of each cluster as the data streams evolve.

Table 1 Cluster purity of *FlockStream* and *DenStream* on the EDS data set without noise for horizon = 2, stream speed = 2000 and horizon = 10, stream speed = 1000

tu	EDS data set			
	Speed = 2000, horizon = 2		Speed = 1000, horizon = 10	
	FlockStream (%)	DenStream (%)	FlockStream (%)	DenStream (%)
10	98.15	98.17	97.66	97.75
20	97.41	98.19	97.28	97.96
30	98.31	98.22	97.78	97.35
40	97.55	97.94	97.85	98.07
50	98.15	97.63	97.55	97.94
60	97.21	96.84	98.38	97.37
70	98.12	97.05	97.89	97.35
80	98.19	98.01	97.49	97.09
90	98.57	97.61	97.41	97.25
100	97.74	97.66	98.19	97.33

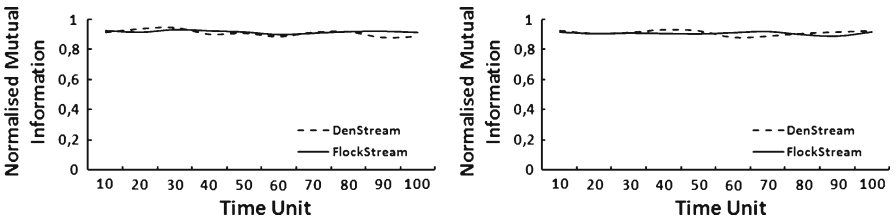

Fig. 7 Normalised mutual information for evolving data stream EDS with horizon = 2 and stream speed = 2000 (*left*), horizon = 10 and stream speed = 1000 (*right*)

The purity results of *FlockStream* compared to *DenStream* on the EDS data stream without noise are showed in Table 1. The results are computed at time units (tu) 10, 20, . . . , 100, for 10 times, with (i) horizon set to 2 and stream speed 2000 points per time unit, and (ii) horizon 10 with stream speed 1000. We can note the very good clustering quality of both *FlockStream* and *DenStream* when no noise is present in the data set, in fact purity values are always higher than 97% and comparable. The table points out that both methods are insensitive to the horizon length.

Figure 7 shows the normalized mutual information of both methods for the same settings. Also in this experiment the results obtained by the two methods are comparable. However, if we consider Fig. 8, we can observe that the number of distances computed by *FlockStream* and *DenStream* is noteworthy. In fact, while *DenStream* at each step must find the nearest neighbor micro-cluster of each data point, *FlockStream* compares each agent (point) only with the other agents encountered during its movement and falling in its visibility radius. This drastic reduction of distance computations allows a fast convergence of the method.

For the same horizon and speed values, the experiments have been repeated with 1% and 5% noise. Table 2 reports the purity values obtained by both the methods. When

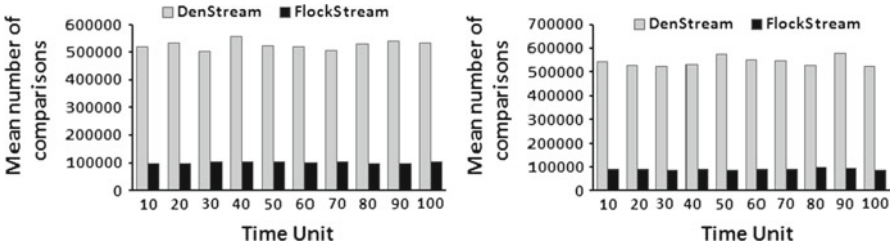

Fig. 8 Mean number of comparisons for evolving data stream EDS with horizon = 2 and stream speed = 2000 (*left*), horizon = 10 and stream speed = 1000 (*right*)

Table 2 Cluster purity of *FlockStream* and *DenStream* on the EDS data set with noise set to 1 and 5%, for horizon = 2, stream speed = 2000 and horizon = 10, stream speed = 1000

tu	EDS data set with noise			
	Speed = 2000, horizon = 2, noise = 1%		Speed = 1000, horizon = 10, noise = 5%	
	FlockStream (%)	DenStream (%)	FlockStream (%)	DenStream (%)
10	98.77	97.78	97.72	94.23
20	98.49	98.11	97.19	95.25
30	98.94	97.40	97.88	95.06
40	98.00	97.06	97.12	95.24
50	98.59	97.55	97.78	94.46
60	98.07	97.51	97.02	93.94
70	98.42	97.45	97.61	93.55
80	98.02	97.16	96.62	94.17
90	98.33	98.02	97.03	93.61
100	97.82	97.79	97.10	93.81

the data set with 1% noise is considered, *FlockStream* still obtains high purity values, which are better than those obtained by *DenStream*. For all the time units, except the last, purity increases to 98%. The difference between *FlockStream* and *DenStream* is more remarkable when data streams with 5% of noise are considered. In such a case *DenStream* does not exceeds 94% of purity, while *FlockStream* achieves values above 97% for all the time units, except time unit = 80. The high quality of the results obtained by *FlockStream* show its capability of dealing with noisy environments.

The same behavior can be observed as regards the normalized mutual information. In fact, Fig. 9 on the left shows that the NMI values obtained by the two methods are comparable when the noise percentage is low (1%). However, as noise increases, *FlockStream* still obtains very high values of normalized mutual information (above 90%), showing its ability to unveil the cluster shape, even when noise is present (Fig.9 on the right). Figure 10 shows that, also for this experiment, the mean number of distance computation performed by *FlockStream* is sensibly lower than that calculated by *DenStream*.

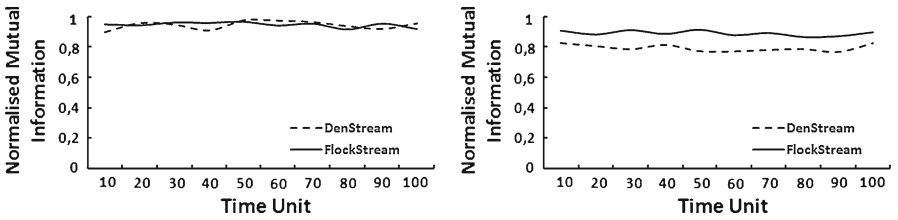

Fig. 9 Normalised mutual information for evolving data stream EDS with horizon = 2 and stream speed = 2000 and with 1% noise (*left*), horizon = 10, stream speed = 1000 and with 5% noise (*right*)

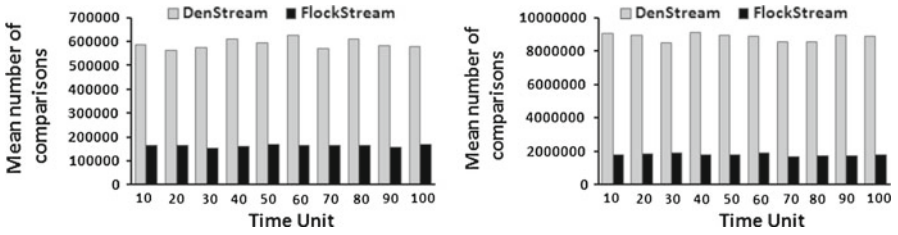

Fig. 10 Mean number of comparisons for evolving data stream EDS with horizon = 2, stream speed = 2000 and with 1% noise (*left*), horizon = 10, stream speed = 1000 and with 5% noise (*right*)

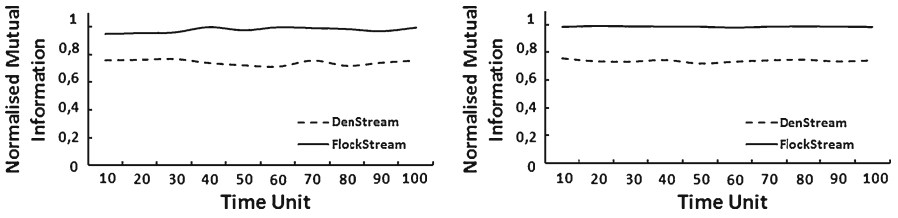

Fig. 11 Normalised mutual information for Network Intrusion data set with horizon = 1 and stream speed = 1000 (*left*), horizon = 5 and stream speed = 1000 (*right*)

6.3 Real-life data sets

The comparison between *FlockStream* and *DenStream* on the Network Intrusion data set are shown in Table 3 and Figs. 11 and 12. The results have been computed by setting the horizon to 1 and 5, whereas the stream speed is 1000. We can clearly see the very high clustering quality achieved by *FlockStream* on this data set. For all the time units cluster purity is above 98%, and reaches 100% at time units 40, 60, 100 when the horizon is set to 1. Analogous results are obtained when an horizon equal to 5 is used. In this case purity above 99% is attained for all the horizons. On this data set *FlockStream* always ouperforms *DenStream*, which obtains a maximum value of purity of 91% for both the horizons.

The normalized mutual information reported in Fig. 11 shows that *FlockStream* overcomes *DenStream* and obtains almost always the true class division of the data set. In fact its value approaches 1 for both the horizons. Figure 12 reveals that the number of distance computations performed by *FlockStream* is almost the half of that

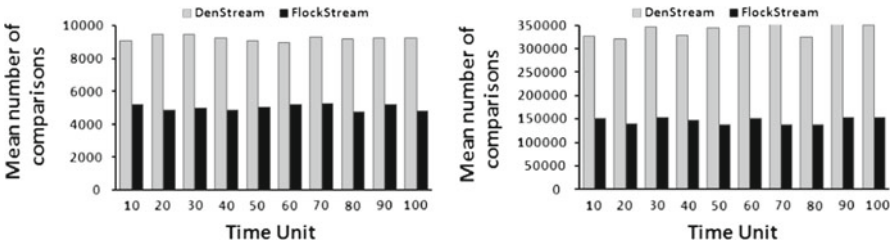

Fig. 12 Mean number of comparisons for Network Intrusion data set with horizon = 1 and stream speed = 1000 (*left*), horizon = 5 and stream speed = 1000 (*right*)

Table 3 Cluster purity of *FlockStream* and *DenStream* on the Network Intrusion data set for stream speed = 1000 and horizon = 1 and 5

tu	Network Intrusion data set			
	Speed = 1000, horizon = 1		Speed = 1000, horizon = 5	
	FlockStream (%)	DenStream (%)	FlockStream (%)	DenStream (%)
10	98.14	90.25	99.89	91.30
20	98.32	91.38	99.98	88.32
30	98.74	90.01	99.93	89.21
40	100.00	90.17	99.93	90.28
50	99.21	88.11	99.92	89.73
60	100.00	89.10	99.71	89.06
70	99.85	89.01	99.96	90.48
80	99.31	89.50	99.94	91.18
90	99.16	88.66	99.94	88.94
100	100.00	89.01	99.73	89.21

needed by *DenStream* to find the clusters. It is worth to note that, despite the lower number of comparisons needed by *FlockStream* to assign a point to the right group (meaning that an approximate nearest neighbor of a point could have been computed), the accuracy obtained is much higher than that obtained by *DenStream*.

Table 4 and Figs. 13 and 14 reports the same experiments executed on the Covertype data set. Also for this data set *FlockStream* outperforms *DenStream* with respect to the cluster purity, the normalized mutual information, and the number of distance computations performed.

6.4 Scalability results

The execution time of *FlockStream* is influenced by two main factors. The first is the number of data points processed at each time unit, i.e. the stream speed. The second is the number of iterations allowed to each agent to move onto the virtual space, every time unit. The first factor determines the dimension of the virtual space. In fact, the higher the number of agents, the higher the dimension of the 2-D grid, since the agents

Table 4 Cluster purity of *FlockStream* and *DenStream* on the Forest Covertype data set for stream speed = 1000 and horizon = 1 and 5

tu	Forest Covertype data set			
	Speed = 1000, horizon = 1		Speed = 1000, horizon = 5	
	FlockStream (%)	DenStream (%)	FlockStream (%)	DenStream (%)
10	99.96	98.81	99.88	98.71
20	99.93	99.86	99.67	97.35
30	99.85	98.80	99.79	96.73
40	99.85	99.61	99.93	98.85
50	99.93	99.03	99.63	96.69
60	99.93	98.30	99.64	97.78
70	99.88	99.51	99.70	96.88
80	99.76	98.81	99.50	96.47
90	99.89	99.18	99.84	97.85
100	99.82	99.83	99.71	96.60

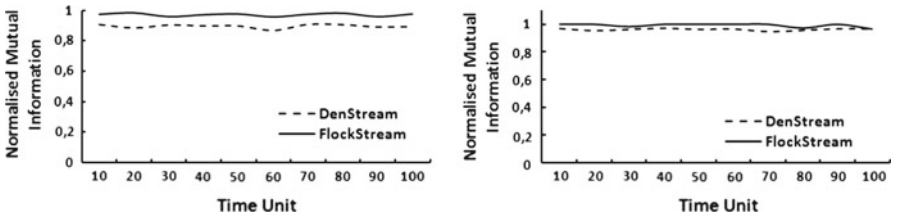

Fig. 13 Normalised mutual information for Forest Covertype data set with horizon = 1 and stream speed = 1000 (*left*), horizon = 5 and stream speed = 1000 (*right*)

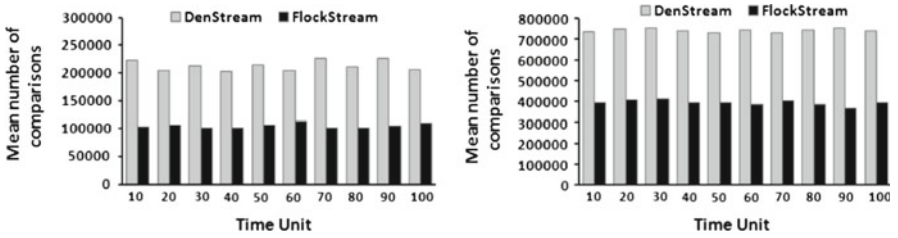

Fig. 14 Mean number of comparisons for Forest Covertype data set with horizon = 1 and stream speed = 1000 (*left*), horizon = 5 and stream speed = 1000 (*right*)

must have sufficient free cells to perform their movement. We experimented that, if the stream speed value is v, then the number of cells must be at least $4 \times v$, and thus the virtual space must be a $d \times d$ grid such that $d \times d \geq 4 \times v$. Figure 15 shows the execution time in seconds on the DS1 data set for both *FlockStream* and *DenStream*, when the stream speed augments from 2000 to 1,6000 data items and the virtual space is a 2000×2000 grid. The figure points out that the execution time of both the methods increases linearly with respect to the stream speed. However, as pointed out, the

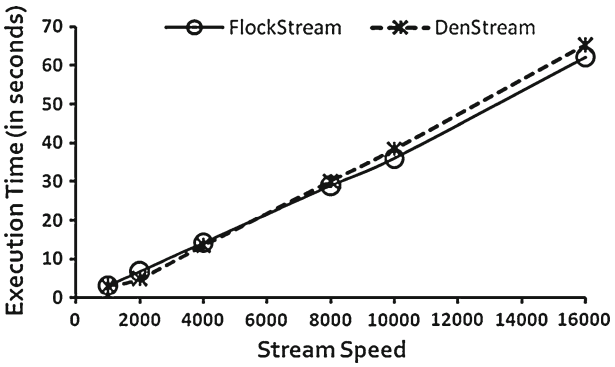

Fig. 15 Execution time for increasing stream length on the DS1 data set

execution time of *FlockStream* depends on the above mentioned factors, thus it could need more or less computation time on the base of the fixed iteration number.

Regarding the other aspect related to the maximum number of iterations an agent can perform, it is worth to note that every time a new bulk of data is deployed onto the virtual space, the agents must have an adequate time interval necessary to meet each other and to decide to join together. A low number of iterations, in fact, forbids cluster formation. Figure 16 shows this behavior for the DS1 data set. *FlockStream* has been executed on all the 10,000 data items for 1000 iterations. The figure reports the clusters obtained after 100, 300, 500, 700, 900, and 1000 iterations. The points depicted with different gray scale denote the swarms formed at the current iteration. In each of the four clusters the points of prevalent color are the approximate clusters constituted until the fixed iteration. The other points are swarms of agents not yet aggregated with the true clusters. Notice that, as the number of iterations augments, the swarms join together and the clustering result improves. This behavior is clearly discernible also from Table 5, where the number of clusters found is reported each 100 iterations. Note the decrease of such a number as time evolves. Thus, if we reduce the number of iterations the execution time diminishes, but the cluster quality is lower. Asking for a faster answer generates an approximate result that, in some application contexts, could be satisfying for the final user.

7 Discussion and 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.

It is worth noting that designing a multi-agent system to efficiently perform a dynamic simulation using the flocking model is a complex task. Many models of flocking behavior describe the behavior of each individual in relation to the position and velocity of the neighboring animals. However, only when certain parameters in

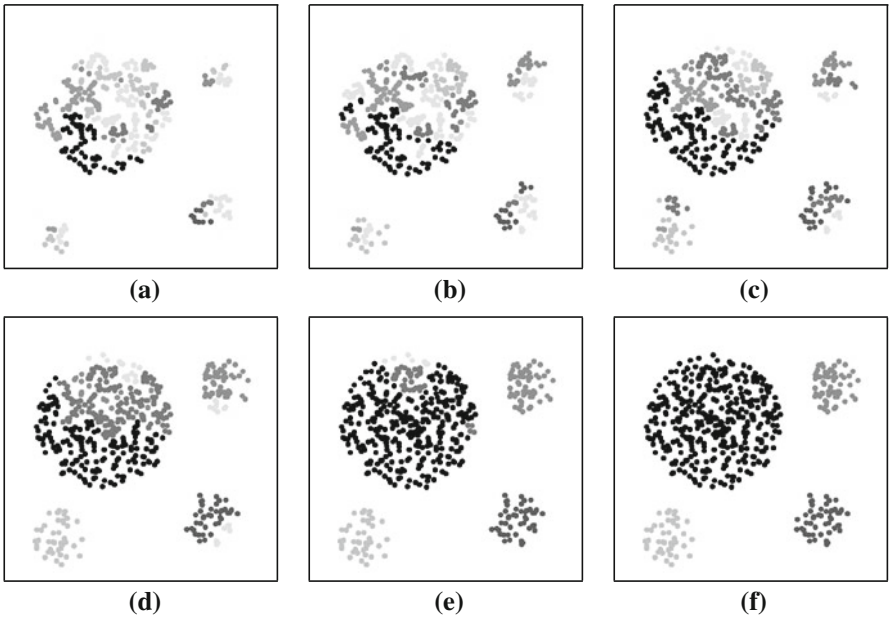

Fig. 16 Clustering result after 100 (**a**), 300 (**b**), 500 (**c**), 700 (**d**), 900 (**e**), and 1000 (**f**) iterations of *FlockStream* on the 10,000 point synthetic data set DS1

Table 5 Cluster number at different iterations

Iterations	Number of clusters
100	102
200	98
300	91
400	89
500	56
600	28
700	12
800	7
900	6
1000	4

the model are properly tuned, collective motion emerges spontaneously. In order to simplify the tuning phase, we adopt a model of flocking behavior that uses agents moving at a constant velocity in a two-dimensional, discrete and toroidal world. The world is composed of cells, and one cell can be occupied by only one agent at a time. Each agent is identified by its coordinates and direction at time t. At each timestep the agent looks at all the other agents within a fixed distance and updates its direction of motion to coincide with that of the other agents. The motion of a group of mobile agents is obtained by using local control laws. The strategy adopted is inspired by the

early flocking model proposed by Reynolds. The control laws presented ensure that all agent headings and speeds converge asymptotically to the same value and collisions between the agents are avoided. The model directs the agents to adjust their velocities repeatedly by averaging them with their neighbors within a fixed radius. The model is deterministic, but it can tolerate a reasonable amount of noise due to the mistakes made by an agent when evaluating each neighbors position and direction.

A naive implementation of the Multiple Species Flocking model would require $O(n^2)$ time, where n is the number of agents deployed onto the virtual space. In fact, each boid should compare with all the other boids in order to compute the similarity and decide to join or leave a group. However, each boid has a spatial position and a visibility range, thus it has a quick access to the flockmates by visiting the nearby cells of the toroidal grid. Though it is not possible a priori to evaluate the number of flockmates encountered, since it depends on the motion of each agent, which is dynamic and unpredictable, experimental results showed that the number of comparisons is low for both synthetic and real life data sets.

A main novelty of our 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.

References

Aggarwal CC (ed) (2007) Data streams—models and algorithms. Springer, Boston

Aggarwal CC, Han J, Wang J, Yu P (2003) A framework for clustering evolving data streams. In Proceedings of 29th international conference on very large data bases (VLDB'03). Morgan Kaufmann, San Francisco, pp 81–92

Aggarwal CC, Han J, Wang J, Yu P (2006) On clustering massive data streams: a summarization paradigm. In: Aggarwal CC (ed) Data streams—models and algorithms. Springer, Boston pp 11–38

Azzag H, Monmarché N, Slimane M, Guinot C, Venturini G (2003) AntTree: a new model for clustering with artificial ants. In: Banzhaf W, Christaller T, Dittrich P, Kim JT, Ziegler J (eds) Advances in artificial life—Proceedings of the 7th European conference on artificial life (ECAL). Lecture notes in artificial intelligence, vol 2801. Springer, Berlin, pp 564–571

Babock B, Datar M, Motwani R, O'Callaghan L (2003) Maintaining variance and k-medians over data stream windows. In: Proceedings of the 22nd ACM symposium on principles of data base systems (PODS 2003), San Diego, pp 234–243

Barbará D (2002) Requirements for clustering data streams. SIGKDD Explor Newslett 3(2):23–27

Beringher J, Hullermeier E (2006) Online clustering of parallel data streams. Data Knowl Eng 58(2):180–204

Cao F, Ester M, Qian W, Zhou A (2006) Density-based clustering over evolving data stream with noise. In: Proceedings of the sixth SIAM international conference on data mining (SIAM'06), Bethesda, pp 326–337

Charikar M, O'Callaghan L, Panigrahy R (2003) Better streaming algorithms for clustering problems. In: Proceedings of the 35th annual ACM symposium on theory of computing (STOC'03), San Diego, pp 30–39

Chen Y, Li T (2007) 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), ACM, New York, pp 133–142

Cui X, Potok TE (2006a) A distributed agent implementation of multiple species flocking model for document partitioning clustering. In: Cooperative information agents, Edinburgh, pp 124–137

Cui X, Potok TE (2006b) A distributed flocking approach for information stream clustering analysis. In: Proceedings of the ACIS international conference on software engineering, artificial intelligence, networking, and parallel/distributed computing (SNPD'06), Las Vegas, pp 97–102

Dai B, Huang J, Yeh M, Chen M (2006) Adaptive clustering for multiple evolving streams. IEEE Trans Knowl Data Eng 18(9):1166–1180

Eberhart RC, Yuhui S, James K (2001) Swarm intelligence (the Morgan Kaufmann series in artificial intelligence). Morgan Kaufmann, San Francisco,

Ester M, Kriegel H-P, Jrg S, Xu X (1996) 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), Portland, pp 373–382

Folino G, Forestiero A, Spezzano G (2009) An adaptive flocking algorithm for performing approximate clustering. Inform Sci 179(18):3059–3078

Guha S, Mishra N, Motwani R, O'Callaghan L (2000) Clustering data streams. In: Proceedings of the annual IEEE symposium on foundations of computer science, Redondo Beach, pp 359–366

Guha S, Meyerson A, Mishra N, Motwani R, O'Callaghan L (2003) Clustering data streams: theory and practise. IEEE Trans Knowl Data Eng 15(3):515–528

Hamdi A, Monmarché N, Alimi A, Slimane M (2008) SwarmClass: a novel data clustering approach by a hybridization of an ant colony with flying insects. In: Dorigo M, Birattari M, Blum C, Clerc M, Stützle T, Winfield A (eds) Ant colony optimization and swarm intelligence—6th international conference, ANTS 2008. Lecture notes in computer science, vol 5217, September 22–24 2008. Springer, Berlin, pp 411–412

Handl J, Meyer B (2007) Ant-based and swarm-based clustering. Swarm Intell 1(2):95–113

Li Tu, Chen Y (2009) Stream data clustering based on grid density and attractions. ACM Trans Knowl Discov Data 3(3):12:1–12:27

Li W, Ng WK, Yu PS, Zhang K (2009) Density-based clustering of data streams at multiple resolutions. ACM Trans Knowl Discov Data 3(3):14:1–14:28

Liu S, Dou Z-T, Li F, Huang Y-L (2004) A new ant colony clustering algorithm based on DBSCAN. In: 3rd international conference on machine learning and cybernetics, Shanghai, pp 1491–1496

Nasraoui O, Coronel CR (2006) Tecno-streams: tracking evolving clusters in noisy data streams with a scalable immune system learning model. In: Proceedings of the 6th SIAM international conference on data mining (SDM'06), Bethesda, pp 618–622

Nasraoui O, Uribe CC, Coronel CR, González FA (2003) Tecno-streams: tracking evolving clusters in noisy data streams with a scalable immune system learning model. In: Proceedings of the 3rd IEEE international conference on data mining (ICDM'03), Melbourne, pp 235–242

O'Callaghan L, Mishra N, Mishra N, Guha S (2002) Streaming-data algorithms for high quality clustering. In: Proceedings of the 18th international conference on data engineering (ICDE'01), San Jose, pp 685–694

Reynolds CW (1987) Flocks, herds and schools: a distributed behavioral model. In: SIGGRAPH '87: Proceedings of the 14th annual conference on computer graphics and interactive techniques. ACM, New York, pp 25–34

Sanghamitra B, Giannella C, Maulik U, Kargupta H, Liu K, Datta S (2006) Clustering distributed data streams in peer-to-peer environments. Inform Sci 176(214):1952–1985

Tan P-N, Steinbach M, Kumar V (eds) (2006) Introduction to data mining. Perason International Edition, Boston

Wang Z, Wang B, Zhou C, Xu X (2004) Clustering data streams on the two-tier structure. In: Advanced Web technologies and applications, Springer, New York, pp 416–425

Zhou A, Cao F, Qian W, Jin C (2007) Tracking clusters in evolving data streams over sliding windows. Knowl Inform Syst 15(2):181–214