Information Sciences 373 (2016) 321–336

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Self-organizing anomaly detection in data streams

Agostino Forestiero

Institute of High-Performance Computing and Networking (ICAR), National Research Council of Italy (CNR), Via Pietro Bucci 7/11 C, 87036 Rende (CS), Italy

ARTICLE INFO

Article history:
Received 21 January 2015
Revised 3 September 2016
Accepted 3 September 2016
Available online 5 September 2016

Keywords:
self organizing
multi agent
anomaly detection

ABSTRACT

Many distributed systems continuously gather, produce and elaborate data, often as data streams that can change over time. Discovering anomalous data is fundamental to obtain critical and actionable information such as intrusions, faults, and system failures. This paper proposes a multi-agent algorithm to detect anomalies in distributed data streams. As data items arrive from whatever sources, they are associated with bio-inspired agents and randomly disseminated onto a virtual space. The loaded agents move on the virtual space in order to form a group following the flocking algorithm. The agents group on the basis of a predefined *concept of similarity* of their associated objects. Only the agents associated to similar objects form a flock, whereas the agents associated with objects dissimilar to each other do not group in flocks. Anomalies are objects associated with isolated agents or objects associated with agents belonging to flocks having a few number of elements. Swarm intelligence features of the approach, such as adaptivity, parallelism, asynchronism, and decentralization, make the algorithm scalable to very large data sets and very large distributed systems. Experimental results for real and synthetic datasets confirm the validity of the proposed model.

1. Introduction

In many distributed applications, such as network flow monitoring, fraud detection, telecommunications, data management, the data flow produced is continuous. It is fundamental to analyze streaming data, as they are an important source of knowledge that enable us to take important decisions at the right time. Monitoring of data flows plays an important role in many domains since it allows detection of events and anomalies in observed systems. So, it is possible to take actions, such as execute a recovery program or notify an administrator. Some significant fields and applications are [26]: (i) Sensor monitoring and surveillance, network traffic, Web logs and Web page click streams; (ii) Trend of workload in an öe-commerceÆ server, which can help in dynamically fine tuning the server in order to obtain better performance; (iii) To analyze meteorological data, by observing how spatial-meteorological points evolve over time; and (iv) the evolution of the spread of illnesses. Finding how system evolution can identify sources responsible for the spread of illness.

These systems continuously produce a huge amount of data often as data streams that can be change over time. Discovering anomalous data (anomaly/outlier detection) is fundamental to obtain critical and actionable information such as intrusions, faults, and system failures. Achieving systems that can handle the endless flow of data by being incremental, has been addressed so far. They are fast and clever enough to approximate results with a fixed level of accuracy, but an important challenge is to detect the unusual objects in the data stream without storing all data. Outlier detection tech-

E-mail address: forestiero@icar.cnr.it

http://dx.doi.org/10.1016/j.ins.2016.09.007
0020-0255/

niques in [44] has been grouped into four categories: (i) statistical approaches; (ii) distance-based methods; (iii) profiling methods; and (iv) model-based approaches. Clustering-based approaches have also been used to detect outliers either as an algorithm to individuate points that do not belong to clusters or as clusters that are significantly smaller than others [37]. Using clustering algorithms to analyze data stream and detect anomalies in data, has evolved as a new form of online data analysis where the information in the data tends to change over time. These algorithms perform cluster analysis of data streams to monitor the results in real time. The aim of the algorithm is to recognize the evolution and provide a result that dynamically adapts to the data. The algorithm can examine data only once, as it arrives, and it must take into account data evolution. It is necessary to design a mechanism to remember old data, such as compression or summarize old information. Novel algorithms that are able to produce models of the data in an online way, are required to analyze the data flow. These algorithms look at each datum only once and within a limited amount of time. Standard data analysis algorithms are a useful starting point, but they must be adapted to work in the stream environment.

In data stream applications, data objects arrive with very high rates, which means that the analysis must be performed very rapidly and efficiently. In such applications, data volumes are huge, so it is not always possible to keep all the data in memory. A sliding window strategy can be used to maintain a percentage of the data in memory. Sliding windows strategy can be based on two concepts: (i) a count-based window in which the **n** most recent objects are maintained, and (ii) a time-based window in which all objects arriving in the last **t** time intervals are stored. Objects maintained during the sliding window are named active objects. The object that leaves the window is deleted from the collection of active objects. So, the algorithms must be designed for outlier monitoring, considering the sliding window. The stream-based algorithms must consider the memory space required for auxiliary information and the storage consumption must be as low as possible. Possibly, they should be able to enlarge the sliding window, to accommodate more objects.

In this paper a bio-inspired method algorithm to detect anomalies in distributed data streams, is proposed. In this approach, originally proposed in [21] as clustering algorithm, data items arrive by several distributed source and are associated with self-organizing agents. The agents are randomly disseminated onto a unique bi-dimensional virtual space. The loaded agents move and work together following a swarm intelligence model known as the flocking algorithm [15,40]. The flocking algorithm is a bio-inspired computational model to simulate a group of entities that mimic a flock of birds. In this model each entity makes movement decisions without any communication with others. The agents follow a small number of simple rules based on their neighboring entities and the obstacles present in the environment. The complex global behavior of the entire group emerges from the interaction of these simple rules. The rules followed by agents are: (i) *separation* - steering to avoid collision with neighbors - (ii) *alignment* - steering toward the average heading and velocity of neighbors - and (iii) *cohesion* - steering toward the average position of neighbors. These rules are applied to calibrate its position when an agent encounters at least one other agent in its neighborhood. The agent does not interact with the agents outside its visibility radius. If the visibility radius does not contain other agents it moves following its previous direction.

Each loaded agent moves into the virtual space following the above flocking rules. When an agent encounters other agents in its visual radius, it evaluates the *similarity* measure and then it decides whether to form the flock or no. Two agents are similar when the respective associated objects are similar. The similarity measure between two data objects can be defined in accordance with the kind of data analyzed. The similarity measure is a function that quantifies the similarity between two objects. Similarity functions can be, but not only, the Euclidean distance, the squared Euclidean distance, the Cosine similarity or the Manhattan distance. In the proposed model, all the rules - cohesion, alignment and separation - are applied in order to form a flock with similar agents, otherwise when two agents are dissimilar only the separation rule is applied. In particular, if two agents are dissimilar, the separation rule overrides the cohesion rule and the alignment rule. Whereas, if two agents are similar, all the rules are applied with the aim to form a flock composed of only similar agents. Thanks to the flocking rules, similar agents will attempt to lie close by exploiting of an attracting force which is proportional to the distance between the two agents. A similar concept, introduced by [14], was proposed to simulate flocking behavior among an heterogeneous population of entities. Cui et al. in [13] use the flocking model to cluster streams of documents. Each agent carries a feature vector of data point, in this case a document. The periodical changing the feature vector of each agent, simulates the stream of documents. But, they neither summarize nor take into account the past information. Old documents are just discarded and the new documents are considered to generate new clusters. In the approach proposed in this paper, the anomalies/outliers are represented by: (i) data associated with agents belonging to flocks with a number of components smaller than a fixed threshold and (ii) data associated with agents isolated agents. The proposed approach undertakes a search for dissimilar points by exploiting an efficient parallel multi-agent search instead of an exhaustive search. Swarm intelligence features boasted of the algorithm makes the approach completely decentralized as each agent acts independently and communicates only with its neighbors. Thanks to the characteristics of locality, parallelism, decentralization and asynchronism, the algorithm is scalable for very large data sets. Experimental results on real and synthetic data sets confirm the validity of the approach. The remainder of the paper is organized as follows: Section 2 illustrates the related works, while Section 3 introduces the density-based clustering algorithm whereon the approach presented in this paper is based. Section 4 introduces the basic concepts of swarm intelligence and flock algorithm. In Section 5 the outlier algorithm is detailed, while in Section 6 a set of experimental results are depicted.

2. Related work

Discovering unusual objects in a set of data is a challenge tackled with several approaches and some examples can be found in [2,23] and [46]. In static contest, for example [10], introduces a distance-based outlier detection algorithm named ORCA. In this algorithm, a simple pruning rule and randomization was used to enhance the naive block nested loop algorithm, obtaining a near linear scaling on large and high dimensional data sets. The CPU time is often approximately linear in the dataset size in this nested loop schema. Ramaswamy et al. [39] extended the above-mentioned approach by giving a rank to the outliers. Given k and n, a data point is an outlier if no more than n-1 other points have higher value for D^k than p, where D^k is the distance of the k^{th} point nearest neighbor of p. Distance-based outlier detection algorithms are not suitable for clusters with varying densities. They are planned to work under the assumption that the whole data set is stored in secondary memory and therefore they are not suitable for data streams. A distance-based approach based on two main parameters R and k was proposed by Knorr and Ng in [27]. Here, a data point is an outlier if less than k points are within distance R from it. A deviation-based technique for outlier detection, is described by Arning [7]. If the features of a data point differ from the features of other data points, it is considered an outlier. While, Aggarwal in [4], proposed an algorithm to find outliers in high-dimensional data. The proposed approach says that a point is an outlier if it is present in some lower dimensional projection in a local region of irregularly low density. Differently Eskin et al. in [18] proposed a clustering-based outlier detection technique that involves fixed-width clustering with a fixed radius.

The common characteristic of the majority of the proposed algorithms is that they work in a static fashion. Thus, the algorithm must be executed from scratch if there are changes in the data objects, leading to performance degradation when updates are frequent. When the objects arrive in a streaming fashion, and usually in high rates, new issues emerge [9]. The problem of the impossibility of revising a data stream during the computation was addressed by Aggarwal (2006) at al. in [3]. They proposed that a stream clustering algorithm should be separated in an online component, to collect appropriate summary statistics on the data, and an offline component that makes use of the information stored to provide the results. In static datasets, a lot of approaches for outlier detection have been used, but only a small amount of work has been done for outlier detection in data streams. While, Elahi et al., to detect outliers in dynamic data stream, introduced in [16] an efficient clustering-based algorithm. The approach of the algorithm is to split the stream in clusters and chunks. When the number of clusters is steady, k-mean approach is used by the chunks.

An important area of research concerns the analysis of data in distributed environments, and in particular outliers detection. Moreover, the challenge of the researchers is to develop techniques that are asynchronous, scalable, and robust when the characteristics of the network change. In [43] the problem of outlier detection over multiple data stream was addressed. In particular, they consider an hierarchical network architecture where a unique node collect data from others node. The approach starts from a more standard definition of an outlier in a wireless sensor network. Otey et al. in [35] starts from the concept of the impracticality of collecting all the data to one site and the dynamic nature of the data. Their algorithm uses complete passes over all distributed data and requires a form of synchronization among node/sensors. In [24] a decentralization method in which the nodes of the network make decisions about communication and processing of the data, was proposed. In this approach, the node/sensors only send information to the leader node if the detected value is outside the normal range, so as to reduce bandwidth consumption. Distributed deviation detection in a distributed system, in [36] was proposed. The approach to the sensor network and targets misbehaving sensors, was tailored. The authors propose a method to maintain density estimates of values from sensors, and flags the sensor if its value significantly deviates from the previously observed value. This operation is executed in a distributed fashion (close to the sensor) and only the result is reported to the central node when needed. Distributed outlier detection is useful exploited for network intrusion detection. In [38] an approach for collaborative intrusion detection in large networks, was proposed. In particular, the authors propose a distributed protection of the network through a hierarchy of surveillance systems. In [30] some techniques to enable different organizations to collaborate for enhanced network intrusion detection, were examined. Each involved organization can gather more information and build an enhanced model of global network activity, and better precise models of attacks.

Innovative algorithms based on swarm intelligence models have been introduced to solve real world problems in a decentralized fashion such as the clustering problems [17,20,33] and outliers detection [6,25,31]. Some study addresses swarm intelligence-based approaches in data quality detection are surveyed in [29]. An ACO-based approach of clustering was proposed as a data preprocessing procedure, during which outliers can be detected, was presented in [25]. The approach collects into the next nodes the continuity of ants for similar data points and dissimilar data points. Each ant of the data points compares their property according to the initial data point set, checks the importance of the data points and iteratively updates the values of the pheromone deposited by other ants. Finally, the data point selection matrix for obtaining final results using another algorithm is generated; at the same time, those unselected data points are outliers. In [45] an intrusive analysis model based on the design of honeypots and an ant colony, was introduced; the ACO is applied to trace the trail of attack and analyse the habits and interests of aggressors; while Soroush et al. in [42] present one of the pioneering studies, in which ACO is used for intrusion detection, unlike previous approaches in which it was used for intrusion response. In [6] a novel swarm intelligence based clustering technique called Hierarchical Particle Swarm Optimization Based Clustering with the aim to detect outliers, is proposed. The technique consists in performing Hierarchical Agglomerative Clustering for outlier detection where a swarm of particles evolves through different stages to identify outliers and normal clusters. In [31] the outlier detection problem is converted into an optimization problem. A Particle Swarm Optimisation (PSO) based approach to outlier detection, is proposed. PSO is used to automatically optimize the distance measures. The approach en-

hances the scope of PSO and allows new insights into outlier detection. In [5] an optimization algorithm is used to improve the computation of Non-Reduct in order to detect outliers. The algorithm, namely Binary Particle Swarm Optimization, optimizes the rules generated from Rough-Outliers algorithm by using Binary PSO algorithm,giving significant outliers object detected. The detection of outliers process is then enhanced by hybriding it with Negative Association Rules. In [32] the outlier detection problem is converted into an optimization problem. A Particle Swarm Optimisation (PSO) based approach to outlier detection is then applied, which expands the scope of PSO and enables new insights into outlier detection.

3. A density-based clustering over an evolving data stream

In this section is presented a density-based clustering algorithm for evolving data streams [12]. The algorithm extracts some *statistics* in order to generate clusters. The method to summarize a group of data point with similar characteristics has inspired the algorithm presented in this paper. However, some changes/semplifications, detailed in the following, were introduced to the original version. The concept of *core-micro-cluster* is the main notion exploited in the algorithm proposed in this paper. Indeed, as [12] use this concept to obtain the statistics for representing the clusters, in this paper this concepts is used to summarize the characteristics of the data items coming from the distributed sources. In the model presented in [12], the *damped window model* was considered to clusters data stream and the weight of each data decreases exponentially with time t through a fading function $f(t) = 2^{-\lambda t}$, where $\lambda > 0$. The weight of the data stream is a constant $Weight = \frac{v}{1-2^{-\lambda}}$, where v is the number of points arrived in one time unit, i.e *speed of the stream*. Historical data reduces its importance when λ assumes higher values.

The concept of *core point*, introduced in *DBSCAN* [19], to store an approximate representation of a group of data, was extended coining the notion of *micro-cluster*. A core point is an object in which ε neighbors have overall weight at least μ. A cluster is a group of core points with the same label. Micro-clusters can be: the *core-micro-cluster*, the *potential core-micro-cluster*, and the *outlier micro-cluster*. A *core-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 $CMC(weight, center, radius)$, where:

$$weight = \sum_{j=1}^{n} f(t - T_{i_j}) \tag{1}$$

with $weight \geq \mu$.

$$center = \frac{\sum_{j=1}^{n} f(t - T_{i_j}) p_{i_j}}{weight} \tag{2}$$

and

$$radius = \frac{\sum_{j=1}^{n} f(t - T_{i_j}) dist(p_{i_j}, center)}{weight} \tag{3}$$

with $radius \leq \omega$.

$dist(p_{i_j}, center)$ is the Euclidean distance between the point p_{i_j} and the *center*. Note that the weight of a micro-cluster must be above a predefined threshold μ in order to be considered core. A cluster with arbitrary shape in a data stream can be described by a set of c-micro-clusters.

A *potential 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 $\{\overline{CF^1}, \overline{CF^2}, weight\}$, with $weight \geq \beta\mu$.

β, $0 < \beta \leq 1$ is a parameter defining the outlier threshold relative to c-micro-clusters.

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

is the weighted linear sum of the points,

$$\overline{CF^2} = \sum_{j=1}^{n} f(t - T_{i_j}) p_{i_j}^2 \tag{5}$$

is the weighted squared sum of the points.

For each p-micro-cluster will be:

$$center = \frac{\overline{CF^1}}{weight} \tag{6}$$

and

$$radius = \sqrt{\frac{\overline{CF^2}}{weight} - \left(\frac{\overline{CF^1}}{weight}\right)^2} \tag{7}$$

A p-micro-cluster is a set of points that could become a micro-cluster. An *outlier 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 $\{\overline{CF^1}, \overline{CF^2}, weight, t_0\}$. The *weight*, CF^1, CF^2, center and radius are the same as the p-micro-cluster. $t_0 = T_{i_1}$ denotes the creation of the o-micro-cluster. In an outlier micro-cluster the *weight* must be below the fixed threshold, thus $weight < \beta\mu$. However, it could grow into a potential micro-cluster when, adding new points, its weight exceeds the threshold.

The algorithm starts with an online phase in which the micro-clusters are maintained and updated as new points arrive online and then an 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, the algorithm 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. $r_p \leq \epsilon$. If the 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* is then checked if $weight \geq \beta\mu$. 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 gradually decay. When it is below $\beta\mu$, c_p becomes an outlier.

In the off-line part of the algorithm a variant of the DBSCAN algorithm in which the potential micro-clusters are considered as virtual points, as exploited. The concepts of density-connectivity and density reachable, adopted in DBSCAN, are used by the algorithm to generate the final result. The algorithm 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.

4. Swarm intelligence

Swarm Intelligence (SI), defined in [11] as "The emergent collective intelligence of groups of simple agents", is an innovative computational method for solving problems that cannot be efficiently tackled with traditional techniques. Swarm Intelligence originally took its inspiration from the biological examples provided by social insects such as ants, termites, bees. These systems are typically made up of a population of simple agents interacting directly or indirectly (by acting on their local environment) with each other. Indirect interaction, i.e. when an individual modifies the environment and the other responds to this change, is named *stigmergy* [22]. This mechanism permits direct communication to be reduced among agents, and must be taken into account when designing artificial systems. In practice, an agent deposits something in the environment that makes no direct contribution to the task being undertaken, but is used to influence the subsequent behavior that is task related. Although there is normally no centralized control structure dictating how individual agents should behave, local interactions among such agents often lead to the emergence of global behavior. Examples of systems like these can be found in nature, including ant colonies, bird flocking, animal herding, bacteria molding and fish schooling. The advantages of SI are twofold: firstly, it offers intrinsically distributed algorithms that can use parallel computation quite easily; secondly, the use of multiple agents supplies a high level of robustness, as the failure of a few individuals does not alter the behavior of the overall system too much.

4.1. Flocking model

Flocking model is an example of emergent collective behavior: there is no leader, i.e., no global control. Flocking behavior emerges from the local interactions of independent entity. Each entity, by only exploiting a geometric description, interacts with the nearby elements which are placed inside its visibility range. The distance between two agents cannot be less than a prefixed value. To simulate the flocking behavior of birds on a computer both for animation and as a way to study emergent behavior, Reynolds proposed firstly in [40] the flocking model. In this basic model the author referred to each individual as a "boid". Three simple steering rules each boid needs to execute to follow the flocking behavior: (i) *separation* (to avoid collision with neighbors); (ii) *alignment* (to follow the velocity and the direction of neighbors); *cohesion* (to approach the neighbors). Thanks to these rules, the action performed by the boid to react to the movement of its neighbor, can be described. The visibility range of boids is defined by the locality degree. The boids are not influenced of entities outside of its visibility range.

A model to better simulate flocking behavior among an heterogeneous population of entities was developed in [14]. This model, namely *Multiple Species Flocking* (MSF), includes a similarity rule that allows each boid to discriminate among its neighbors and to group only with those similar to itself. The similarity rule enables the flock to organize groups of heterogeneous entities into homogeneous subgroups consisting only of individuals of the same species. The authors use the concept of *velocity vector* of the flock to describe the similarity, alignment, cohesion and separation rules. The approach proposed in this paper is inspired to the concepts of Multiple Species Flocking model, but it does not exploit the similarity and dissimilarity rules proposed in [14]. In fact, the model simply modifies the basic flocking rules to take into account the similarity of an entity with its neighbors. In particular, if there are two dissimilar agents, the separation rule overrides the cohesion rule and the alignment rule. Whereas, if two agents are similar, all the rules are applied with the aim of forming a flock composed only of similar agents.

The virtual space V_s where agents are scattered and move according to the flocking rules is a two dimensional Cartesian space, let $\mathcal{R}^2$, while the data stream point are represented in a n-dimensional feature space, let $\mathcal{R}^n$. Let R be the radius of the visibility range of the agent and let r be the minimum distance that must be maintained among the agents, with

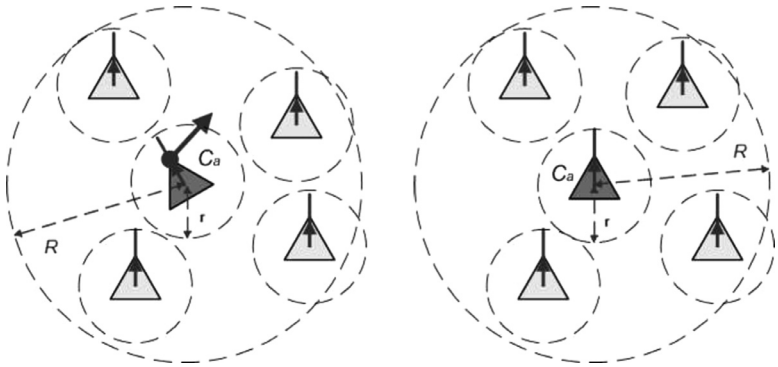

Fig. 1. Alignment rule.

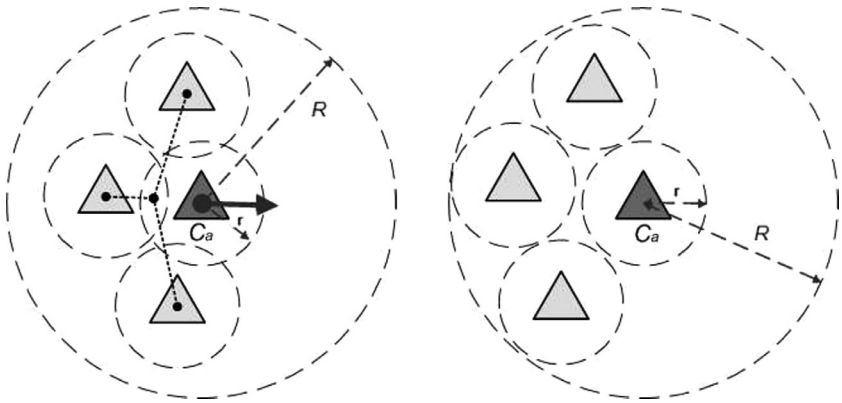

Fig. 2. Separation rule.

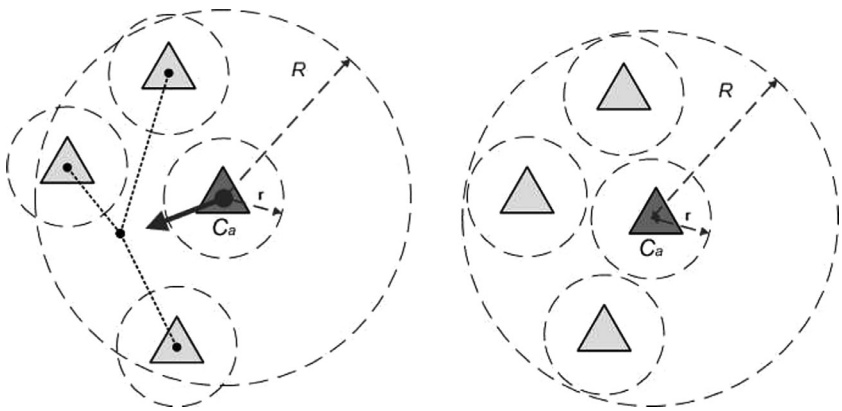

Fig. 3. Cohesion rule.

$R > r$. The *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. The basic flocking rules are depicted in Figs. 1, 2 and 3, where R is the visibility range, r is the minimum distance between two agents, and d(B, C_a) is the distance between the flock mate B, a member of the flock, and the current agent C_a.

The alignment rule, depicted in Fig. 1, gives an agent the capability to align with other nearby agents contained in its visibility range. The agent can compute the steering for alignment by finding all agents in the local neighborhood and averaging together the 'heading' vectors of the nearby agents. The component of the velocity vector owing to the alignment

rule, applied when $d(B, C_a) \leq R \wedge d(B, C_a) \geq r$, can be formally described as:

$$\vec{V}_{alignment} = \frac{1}{n}\sum_{i}^{n}\vec{v}_i \tag{8}$$

where $d(B, C_a)$ is the distance between its neighbor flock mate B and the current agent C_a, $\vec{V}_{alignment}$ is the velocity driven by the alignment rule, $\vec{v}_i$ is the velocity of the agent B_i, and n is the number of neighbor agents.

The separation rule, depicted in Fig. 2, gives an agent the capacity to maintain an established distance from others nearby. This prevents agents from crowding too closely together, allowing them to scan a wider area. The component of the velocity vector owing to the separation rule, applied when $d(B, C_a) \leq 2r$, can be formally described through the following formula:

$$\vec{V}_{separation} = \sum_{i}^{n}\frac{\overline{\vec{v}_i + \vec{v}_c}}{d(B_i, C_a)} \tag{9}$$

where $\vec{v}_c$ is the velocities of the current agent, $\vec{v}_i$ is the velocity of the i-th flock mate and $\vec{V}_{separation}$ is the separation velocity.

The cohesion rule, depicted in Fig. 3, provides an agent with the ability to approach and form a group with other agents. Steering for cohesion can be computed by finding all agents in the local neighborhood and computing the average position of the nearby agents. The steering force is then applied in the direction of that average position. The component of the velocity vector owing to the cohesion rule, applied when $d(B, C_a) \leq R \wedge d(B, C_a) \geq r$, can be formally described through the following formula:

$$\vec{V}_{cohesion} = \sum_{i}^{n}(P_i - P_c) \tag{10}$$

where P_i and P_c are the positions of the current agent and $\vec{V}_{cohesion}$ is the cohesion velocity.

The overall flocking behavior will be expressed by a linear combination of the velocities calculated by all the rules that represent the velocity vector of the agents in the virtual space, i.e. $\vec{V} = \vec{V}_{alignment} + \vec{V}_{cohesion} + \vec{V}_{separation}$.

5. Self-organizing data algorithm

The rules of the flocking algorithm were exploited and enriched to define a bio-inspired algorithm for discovering anomalies in data streams. Each item that comes from a data stream is associated to an agent. A similarity rule that allows agents to discriminate among its neighbors and to group only with agents similar to itself, was added in the model. Two agents are considered similar if the corresponding associated data items are similar. In particular, if two agents are dissimilar the separation rule overrides the cohesion rule and the alignment rule. Whereas, if two agents are similar, all the rules are applied with the aim of forming a flock composed only of similar agents. The agents are deployed and move according to the modified rules on a bi-dimensional Cartesian space representing the virtual space. The virtual space is implemented as a two-dimensional toroidal grid of fixed size and it can be assumed independently discrete or continuous. In this paper, it is assumed to be discrete and not continuous for the sake of simplicity. The dimension of the virtual space is related to the number of data items, N_p, processed every time unit. Indeed, since each data item is associated to an agent in the virtual space, they must have sufficient space for their movement. Experimentally it was noted that the dimension of the virtual space must be at least $4 \cdot Np$. Therefore, the dimension of the virtual space must be $dim \times dim$ such that $dim \times dim \geq 4 \cdot Np$. Each multidimensional data point $dp = x1, ..., xn$ arrived from a source is associated with an agents and the agent with a velocity vector $\vec{v} = (m, \alpha)$ initialized such that $m = 1$ and the angle α with a random value in the interval [0, 360]. Each cell of the virtual space with coordinates (x, y) contains only one agent with position $P = (x, y)$. The agents associated with a data item are identified as *simpleAgent*s. The agents are grouped based on the similarity in data items space instead of their space.

The agents use a *similarity function* to check whether the carried data items are similar. To evaluate the similarity between two data items several similarity measure can be used. The similarity measure depends on the data items input to the algorithm. In this paper the Euclidean distance to measure the similarity between two data items, is used. It is assumed that two agents, $A1$ and $A2$, are similar if their Euclidean distance of the associated data items $d(dp1, dp2) \leq \epsilon$. The agents move following the modified flocking model. Like birds in the real world, similar agents - associated with similar objects - form a flock, whereas dissimilar agents move away from the flock. This algorithm assumes that an agent moves one cell at a time in the virtual space.

When a prefixed number of iterations is elapsed, that is each agent has performed a given number of movements, the system is analyzed. A variable number of flocks of similar agents was born in the virtual space. The core concept of the algorithm is to summarize the characteristics of every flock and represent it with a virtual data item associated to a new agent in the virtual space. The agents that, move in the virtual space to form a new flock and carry a virtual data item, are denoted as *summaryAgent*. The concepts illustrated in Section 3 are exploited to summarize the characteristics of a flock and create a virtual data item to associate to *summaryAgent*. The concepts of micro-cluster was borrowed to represent a group of data points through a unique data point. The obtained virtual data item is assigned to a *summaryAgent*, deployed in the virtual space and exploited for successive iterations of outlier detection.

In this way, a restricted amount of memory to store the value of old data points, is employed. In fact, for each flock of agents, then for each group of similar data points, only one data point will be stored for the successive iterations. A set of new agents are inserted into the virtual space when a new data stream arrives. During the first time unit only *simpleAgent*s are present in the virtual space, whereas in the successive time units, both kinds of agent are present and the new flock can be composed of both kind of agents. Indeed, at the end of every time unit, the flocks with given characteristics are replaced with a *summaryAgent* with associated a virtual data points. The behavior of both agents is the same, indeed, when a *simpleAgent*/*summaryAgent* meets a *simpleAgent*/*summaryAgent* in its visibility range, the modified rules are applied. If the agents are similar, they try to form a flock, whereas, if the agents are dissimilar, they go away. Both kinds of agent work together to form new flocks based on the similarity concept. The algorithm was designed to update the summary of the *summaryAgent* when each step of the iteration finishes (each time units), because a *simpleAgent* could be dropped if the flock, during its movement, encounters another agent more similar to it. The algorithm, indeed, exhibits an *adaptive behavior* since an agent can leave the group it participates in and join another group on the basis of the agents it encounters during its motion. Thus, during this predefined number of iterations, the points join and leave the groups forming different flocks.

To better illustrate how the algorithm works, a simple synthetic dataset was built and the behavior of the algorithm was detailed. A set of about 100 documents was considered and, each document, is represented through a vector of features according to the Vector-space Model. This model relies on the concept that the meaning of a document is derived from the document's constituent terms. Documents are represented as vectors of terms, $doc = \{t_1, t_2, t_3, \ldots., t_n\}$, where t_i is a non-negative value denoting the single or multiple occurrences of term t in document *doc*. Each unique term in the document collection corresponds to a dimension in the space. The dataset's documents are tailored so that all are similar except 3, which represent the anomalies. To simulate the stream, about 10 documents are inserted in the system every step, i.e the vectors representing the documents are assigned to the agents and the agents are spread on the virtual space. The similarity between two documents is achieved exploiting the cosine similarity function as reported in formula 11.

$$similarity(u, v) = \cos(\vec{d_1}, \vec{d_2}) = \frac{\vec{d_1} \cdot \vec{d_2}}{|\vec{d_1}| \times |\vec{d_2}|} \tag{11}$$

where $\vec{d_1} \cdot \vec{d_2}$ indicates the *dot-product* between the vectors $\vec{d_1}$ and $\vec{d_2}$.

Fig. 4 shows a set of illustrations of how the process evolves in the virtual space. In particular it is reported, for each step, the initial state (left side of the arrow) and the final state (right side of the arrow) of the virtual space. To evolve from the initial state to the final state of every step, the system performs a number *Iter* of iteration (movements of agents). In this case *Iter* is equals to 1000. At the end of every step, each flock composed at least of *Nag* agents is replaced by a *summaryAgent* and inserted onto the virtual space in the successive step. In this experiment *Nag* in set to 4. To calculate the features of the *summaryAgent*s, the concepts illustrated in Section 3, are exploited. The isolated agents and the flocks composed of a number of agents lower than *Nag* are inserted onto the virtual space for successive step. When the stream is finished, all similar agents are unified and summarized, whereas dissimilar agents (associated to dissimilar documents) are isolated. The flock's searching mechanism of the flocking algorithm is the main advantage that helps agents to form a flock quickly. Since the agents continuously move in the virtual space and join the flock constituted of agents more similar to themselves, new results can be quickly re-generated when adding or deleting agents at run time. Thanks to this characteristic, the flocking algorithm can be profitably applied to analyze dynamically a changing information stream.

6. Experimental results

In order to investigate the effectiveness on real and synthetic datasets a Java prototype of the algorithm was implemented. All the experiments were performed on an Intel(R) Core (TM) i7 with 8 Gb of memory. First of all, a brief description of the dataset employed is given. In particular two synthetic datasets were used, namely Gauss and STREAM, and three real datasets, Forest Covertype dataset and Internet Advertisements dataset, both available at UCI Machine Learning Repository [8] and 1998 DARPA Intrusion Detection, available at [28].

Since the source of the data items of the distributed system is not relevant for the algorithm, to transform a dataset in a sequence of distributed data streams, the input order of the dataset was considered as the streaming order. The synthetic dataset namely Gauss is made up of a set of about 100,000 instances. It consists of a combination of three Gaussian distributions with uniform noise. The synthetic dataset STREAM is generated using the three datasets showed in Fig. 5. Each of them contains 10,000 points, and to generate the data stream, each dataset was randomly chosen 10 times, thus generating an evolving data stream of total length 100,000.

Forest Covertype dataset represents the prevision forest cover type through the cartographic variables. The dataset describes the environment in which trees were observed. It is composed of 581,012 instances represented by 54 geological and geographical attributes. Internet Advertisements dataset represents a set of possible advertisements on Internet pages. The features encode the geometry of the image (if available) as well as phrases occurring in the URL, the image's URL and alt text, the anchor text, and words occurring near the anchor text. The dataset is composed of 3279 instances represented by 1558 attributes. DARPA Intrusion Detection Evaluation Data consists of network connection records of several intrusions simulated in a military network environment. The TCP connections were elaborated to create a dataset of 23 numerical features. Since the amount of available data is huge, the data from two weeks are used, about 100,000 TCP connection records.

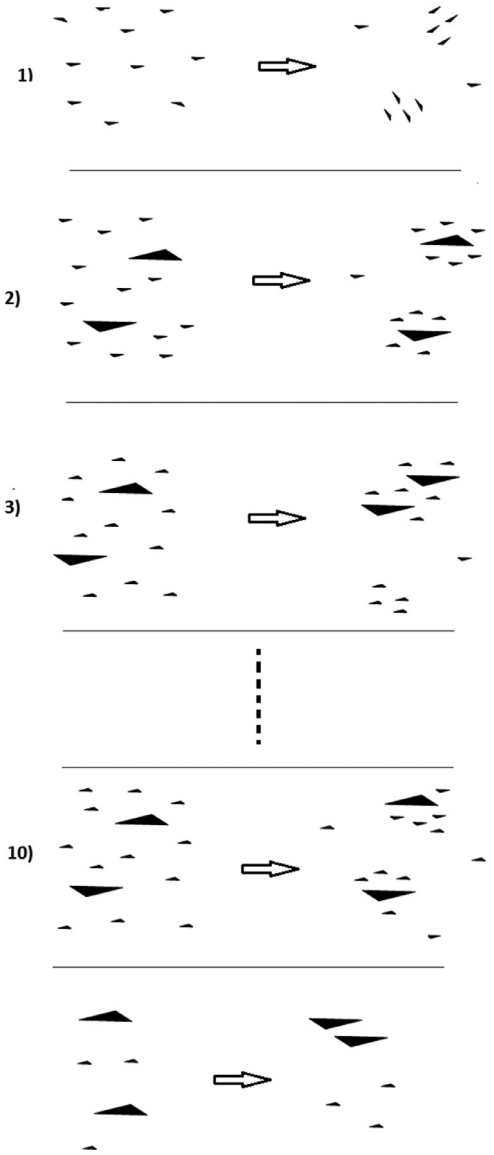

Fig. 4. Illustrations of how the process evolves in the virtual space. For each step, the initial state (left side of the arrow) and the final state (right side of the arrow).

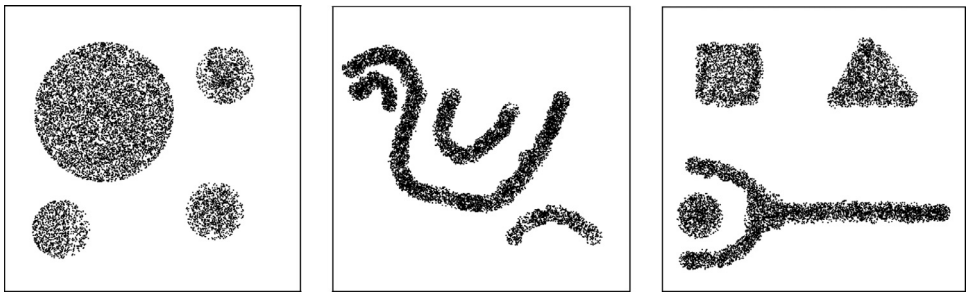

Fig. 5. The datasets used to generate the data stream STREAM.

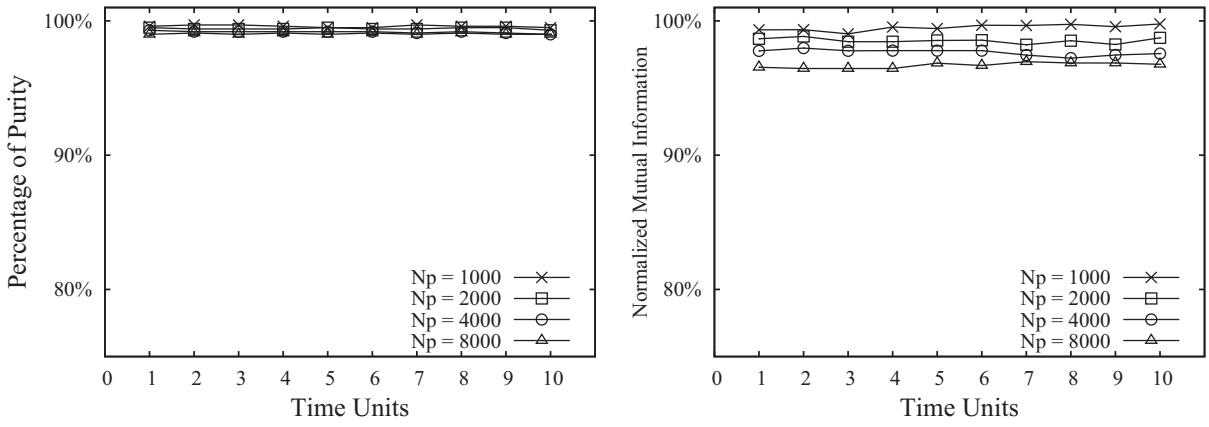

Fig. 6. The average purity and normalized mutual information for Gauss dataset, when the values of the number of points analyzed for each time units N_p, ranges from 1000 to 8000.

The right composition of *summaryAgents* was evaluated exploiting the average *purity* and the *normalized mutual information*. Since, for each considered dataset, the true data label is known, it is possible evaluate the goodness of the grouping operation performed by agents, and consequently the dissimilarity of the outliers. The dimension of the virtual space *dim* must be such that $dim \times dim \geq 4 \cdot Np$, thus in these experiments was set to 200, because *Np* up to 8000. While to generate a *summaryAgent*, a group of agents has to be composed by at least 100 entities.

The average purity *P* is defined as:

$$P = \frac{\sum_{i=1}^{N_S} \frac{|N_{right}(i)|}{|N_{total}(i)|}}{N_S} * 100\%; \tag{12}$$

where N_S indicates the number of *summaryAgents* for each time interval, $|N_{right}|$ indicates the number of points really similar in *summaryAgent* **i**, and $|N_{total}|$ indicates the total number of points in *summaryAgent* **i**.

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, $NMI(A, B) = 0$.

Fig. 6 shows the values of average purity and normalized mutual information for Gauss dataset respectively, when the number of points N_p, introduced for each time unit, ranging from 1000 to 8000 points. It should be noted that the values of average purity and of normalized mutual information are very satisfactory and constant independently both of the time and of the number of points inserted each time unit N_p. Fig. 7 shows the values of average purity and normalized mutual information for STREAM dataset respectively, when the number of points N_p, introduced for each time unit, ranges from 1000 to 8000 points. As well in this second artificial dataset, it is possible to note that the values of average purity and of normalized mutual information are very good and constant independently both of the time and of the number of points inserted each time unit N_p.

The same observations can be made for the values of average purity and normalized mutual information of the real datasets. Fig. 8 shows the values of average purity and normalized mutual information for Forest Covertype dataset respectively, when the number of points N_p, introduced for each time unit, ranges from 1000 to 8000 points. While Fig. 9 shows the values of average purity and normalized mutual information for DARPA dataset respectively, when the number of points N_p, introduced for each time unit, ranges from 1000 to 8000 points. As well for both real dataset, it should be noted that the values of average purity and of normalized mutual information are very adequate and constant independently both of the time and of the number of points inserted each time unit N_p.

Successively, two measures, namely Precision and Recall, were used to further validate the algorithm. Precision represents the fraction of the values reported by algorithm as outliers that are true outliers. Recall represents the fraction of the true outliers that the algorithm identified correctly.

Fig. 10 shows the values of Precision and Recall for the Gauss and STEAM datasets, when the number of points analyzed N_p is set to 5000 for each time unit. Notice that the results obtained by the model are very satisfactory for both datasets. The same observations can be made for the graphs showed in Fig. 11.

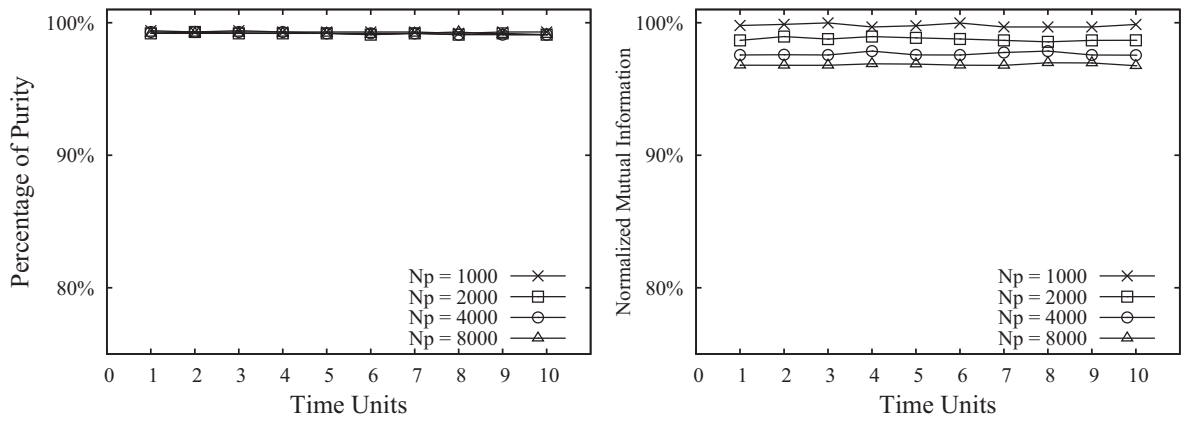

Fig. 7. The average purity and normalized mutual information for STREAM dataset, when the values of the number of points analyzed for each time units N_p, ranges from 1000 to 8000.

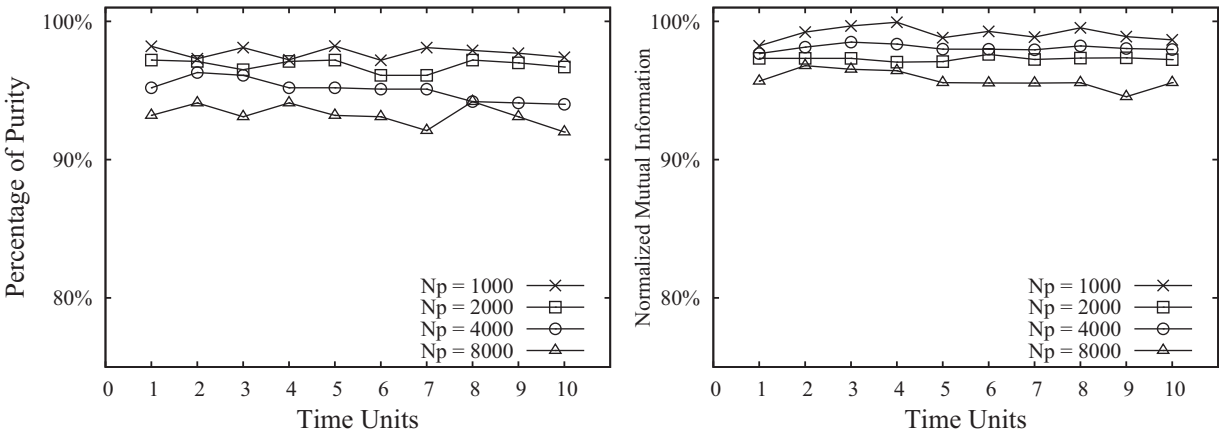

Fig. 8. The average purity P for Forest Covertype dataset, when the values of the number of points analyzed for each time units N_p, ranges from 1000 to 8000.

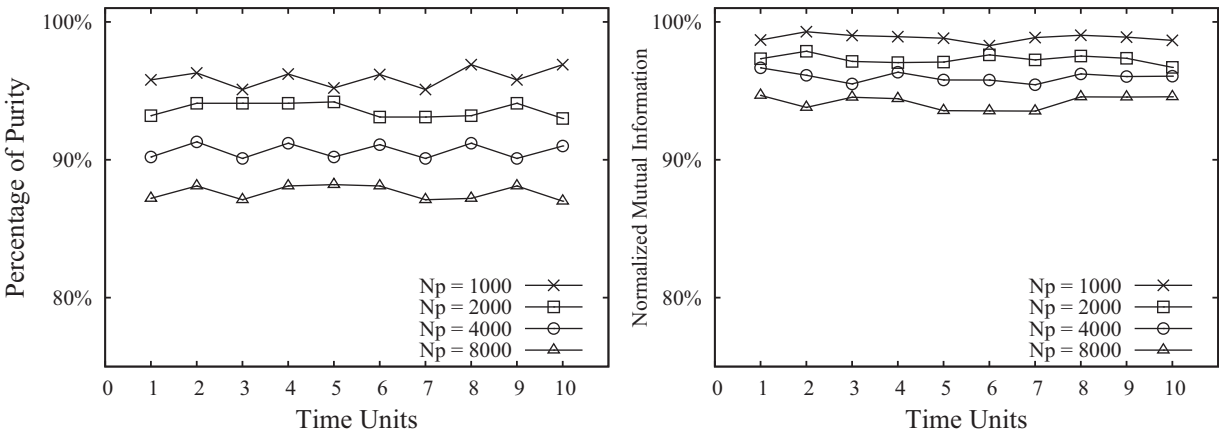

Fig. 9. The average purity P for DARPA dataset, when the values of the number of points analyzed for each time units N_p, ranges from 1000 to 8000.

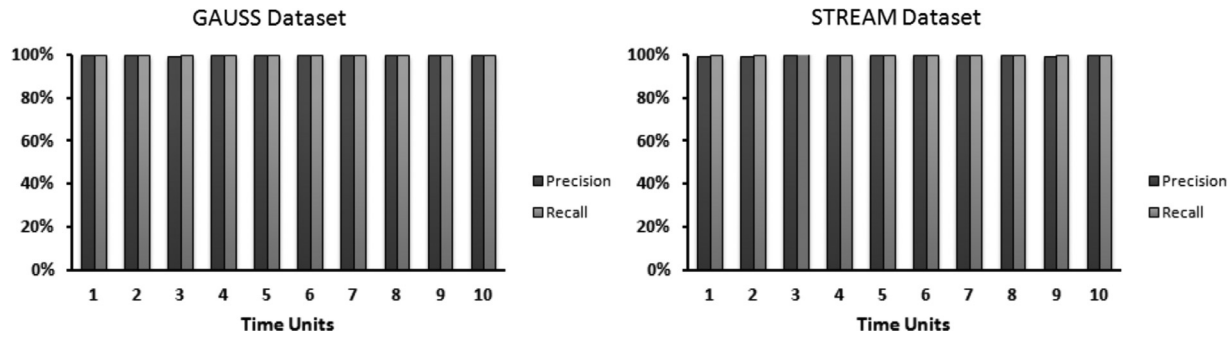

Fig. 10. Precision and Recall for the Gauss and STREAM datasets, when the number of points analyzed N_p is set to 5000 for each time unit.

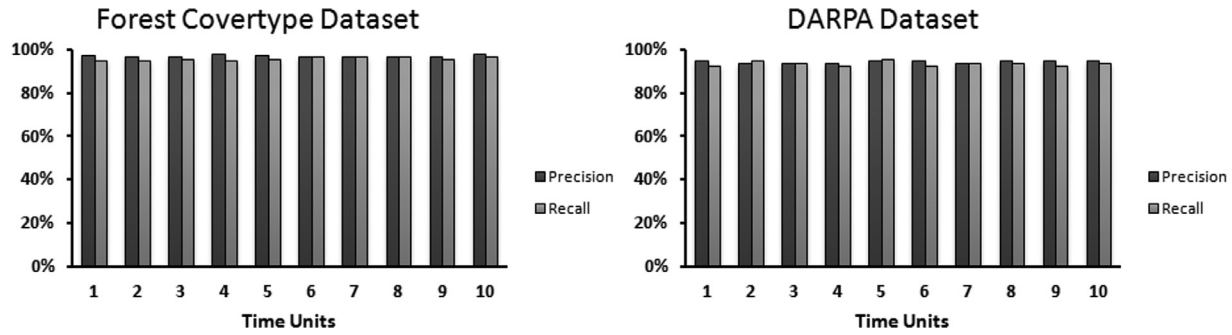

Fig. 11. Precision and Recall for the Forest Covertype and DARPA datasets, when the number of points analyzed N_p is set to 5000 for each time unit.

Table 1
Precision and Recall for Internet Advertisements datasets, when the number of points analyzed N_p is set to 300 for each time unit.

Time Units	Precision	Recall
1	94.80	92.80
2	93.90	94.80
3	93.60	93.80
4	93.90	92.80
5	94.80	95.70
6	94.80	92.80
7	93.80	93.70
8	94.90	93.80
9	94.80	92.80
10	94.90	93.90

These figures depict the values of Precision and Recall for the Forest Covertype dataset and DARPA datasets, respectively, when the number of points analyzed N_p is set to 5000 for each time unit.

Table 1 reports the values of Precision and Recall for Internet Advertisements datasets, when the number of points analyzed N_p is set to 300 for each time unit. The algorithm has to individuate whether an image is an advertisement (“ad”) or not (“nonad”), i.e. an outlier. Notice that the values obtained by the model are very satisfactory.

In order to analyze the approach proposed in this paper, a comparison of the performances obtained with other similar methods, was performed. The datasets utilized for the comparison are the FisherÆs Iris dataset and BUPA dataset. Both the datasets are taken from UCI Machine Learning Repository [8]. FisherÆs Iris, extensively used in clustering and classification applications, consists of 150 records, 3 classes (Iris setosa, Iris versicolor and Iris viginica) and 4 dimensions. BUPA dataset comprises of 345 records, 2 classes and 6 dimensions.

In Fig. 12a comparison among some partition-based algorithms and the algorithm proposed in this paper, is reported. Enhanced versions of the algorithms PAM, CLARA and CLARANs, introduced in [34] to improve the outlier detection mechanism, were considered. In [1] was established that there are 10 outliers in the class Iris viginica of Fisher's Iris dataset and

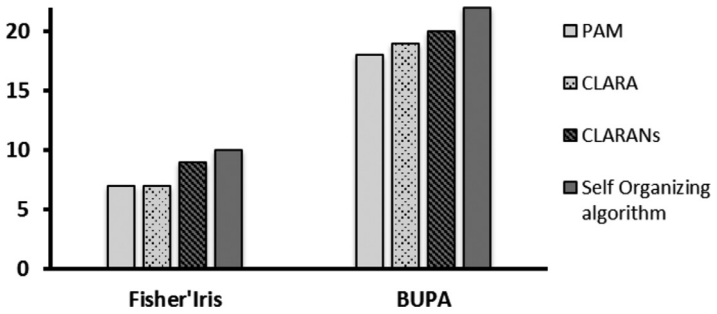

Fig. 12. Number of outliers detected of the enhanced versions of the algorithms PAM, CLARA and CLARANs and the Self Organizing algorithm proposed in this paper on Fisher's Iris dataset and BUPA dataset.

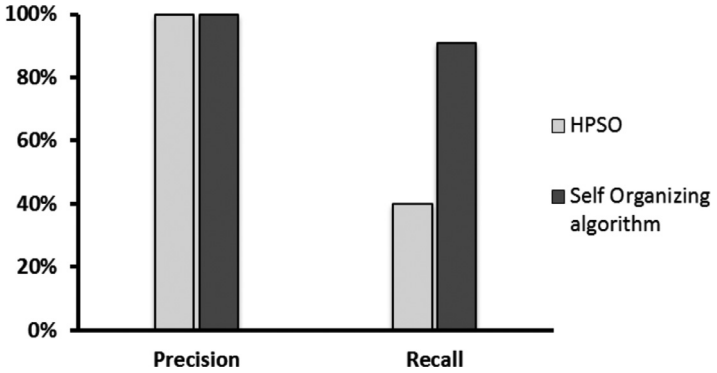

Fig. 13. Precision and Recall on Fisher's Iris dataset for HPSO algorithm and the Self Organizing algorithm proposed in this paper.

22 outliers in the class 1 of BUPA dataset. The comparison was achieved on the basis of the number of outliers detected of every algorithm. Notice that the values obtained by the algorithm proposed in this paper are is very satisfactory.

Fig. 13 depicts a comparison, in terms of Precision and Recall, between an algorithm for outlier detection based on Particle Swarm Optimization (PSO) introduced by Alam at al. in [6] and the self-organizing algorithm here proposed. PSO, proposed by Kennedy and Eberhart [15], is a swarm intelligence based meta-heuristic inspired to the cooperation and communication of swarm behavior of bees, ant, birds, fish and other insects. The mechanism simulates their collective behavior to solve their foraging search and communication problem. PSO, thanks to high decentralization and simple implementation is efficiently applicable to optimization problems. The algorithm, namely HPSO-clustering and previously proposed in [41], uses a partitional-based method for generating a hierarchy of clusters: the levels are processed as a generation of the swarm and the first generation is the entire swarm. The *centroid* of the cluster is associated to the particle of the swarm and, by merging two clusters in each generation, the swarm evolves in a single cluster. Outlier detection is achieved by discovering outliers at different levels of the clustering solution. So a hierarchy of clusters is generated, and the suspicious observations are detected. We can see how, while the value of Precision is similar, the value of the Recall, related to the algorithm proposed in this paper, is significantly better.

6.1. Complexity and scalability

A naive implementation of an algorithm would require $O(n^2)$ time, where n is the number of objects. In fact, each object should be compared with all the other objects in order to compute the similarity and decide to be similar. Here, each boid has a spatial position and a visibility range, thus it has a quick access to the other objects by visiting the nearby cells of the toroidal grid. Though it is not possible a priori to evaluate the number of groups 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. Then, an important performance index is the number of comparison (distance computations) performed by the agents to form the group of similar entities. Fig. 14 show the mean number of comparisons on Gauss dataset and DARPA dataset, respectively.

In particular, these figures show the number of distance computations performed by the algorithm to form the group of similar agents at the generic iteration on a synthetic dataset (Gauss) and on a real dataset (DARPA). It is worth noting that, thanks to the intrinsic characteristics of swarm-inspired algorithms, such as parallelism, asynchronism and decentralization,

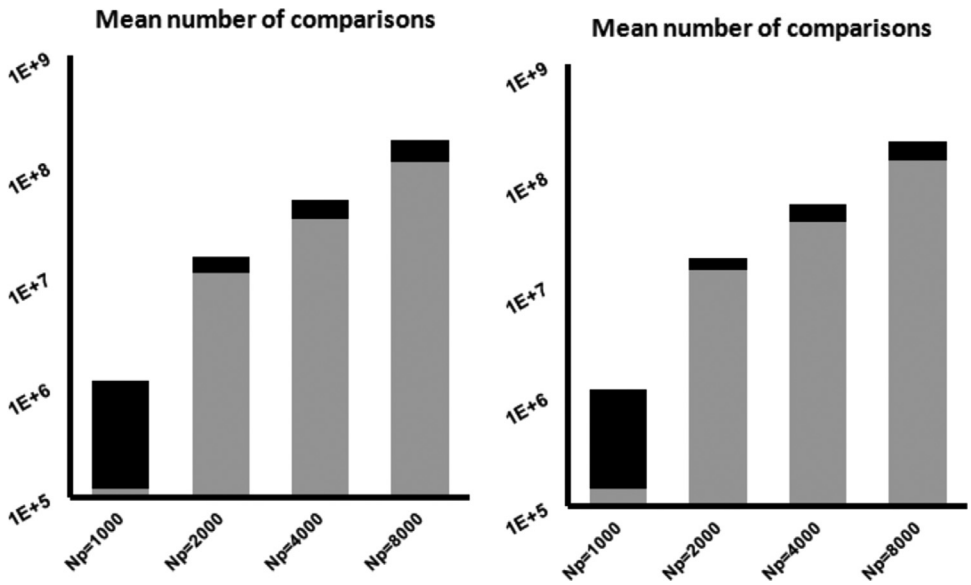

Fig. 14. Mean number of comparisons on Gauss and DARPA datasets performed by the algorithm, when the number of points analyzed N_p ranges from 1000 to 8000.

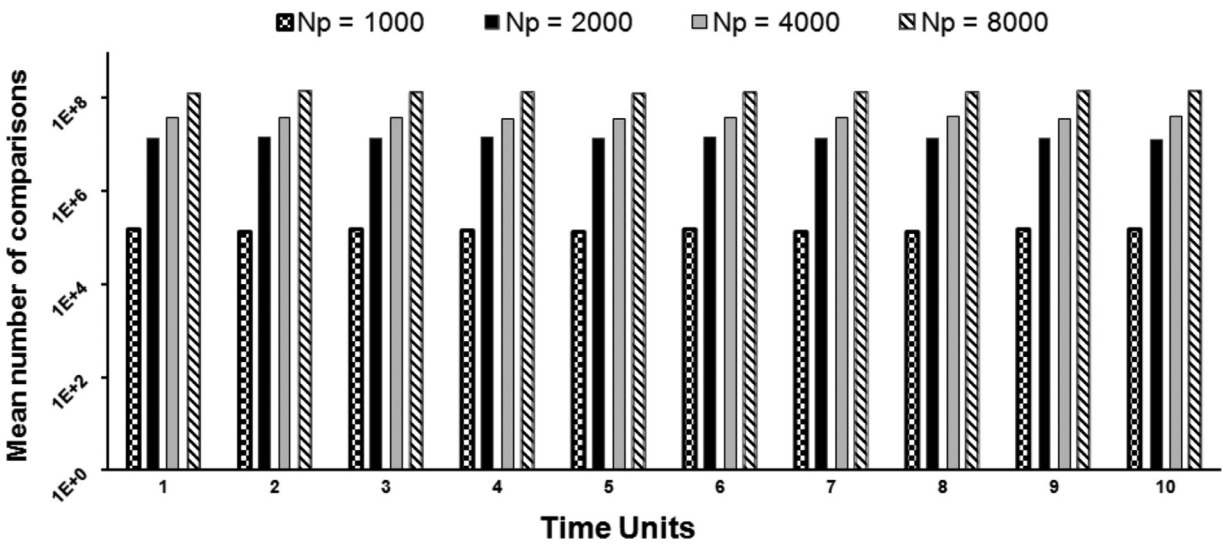

Fig. 15. Mean number of comparisons on DARPA dataset versus time units performed by the algorithm when the number of points analyzed N_p ranges from 1000 to 8000.

the number of comparisons needed to individuate the group of similar objects is much lower with respect to the number of comparisons needed when all objects are compared with all objects. In Fig. 14, the gray bars represent the number of comparisons performed by the algorithm, whereas the black bars represent the comparisons performed when all objects are compared with all objects. Fig. 15 shows the mean number of comparisons DARPA dataset, i.e. the distance computations, versus time units, performed by the algorithm when the number of points analyzed N_p ranges from 1000 to 8000.

To investigate the scalability feature of the algorithm proposed in this paper, the execution time when the number of data points/agents put in the virtual space changes, was considered. This kind of study also represents how the algorithm reacts when the number of sources of the distributed data streams changes. Fig. 16 shows the execution time in seconds on the analyzed datasets, when the number of points *Np* ranges from 2000 to 16,000 data items. The dimension of the virtual space *dim* was set to 250.

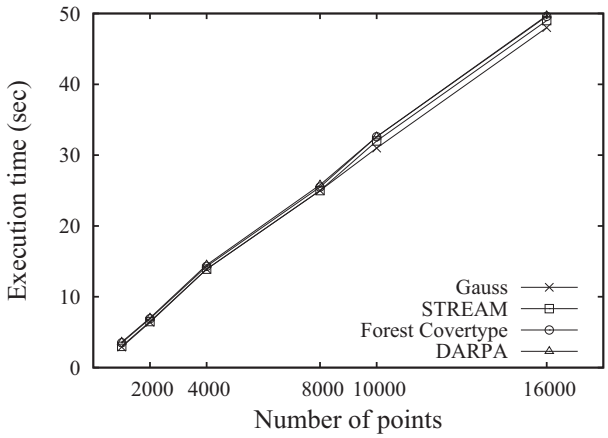

Fig. 16. Execution time of the algorithm on the analyzed datasets, when the number of points *Np* ranges from 1000 to 16,000 data items.

Notice that the execution time increases linearly with respect to the number of points (agents) deployed on the virtual space. It is necessary to highlight that a low number of iterations distorts group formation. In fact, every time a new group of agents is spread onto the virtual space, they must have an adequate time interval to meet each other and to decide to join together. The execution time of the algorithm is influenced both by the number of data points processed at each time unit and the number of iterations allowed for each agent to move onto the virtual space, every time unit.

7. Conclusion

To detect anomalies in data stream, a bio-inspired clustering algorithm has been proposed. Each multidimensional data item is coupled with a bio-inspired agent and they are spread onto a virtual space independently from the sources they arrive. The agents, following a flocking based exploration strategy, work autonomously and they interact only with immediate neighbor entities. The flocks are only composed of *similar* agents, whereas the dissimilar agents do not join any group. The similarity between two agents depends on the similarity of the associated objects. This approach can be profitably exploited both as clustering method and for anomaly detection, and both in distributed and centralized systems. The algorithm is very scalable and suitable for large data sets thanks to features such as adaptivity, parallelism, asynchronism, and decentralization. Experimental results on real and synthetic data sets confirm the validity of the approach proposed. Future work will aim to extend the approach for real life applications.

References

[1] E. Acuna, C. Rodriguez, A meta analysis study of outlier detection methods in classification, Technical paper, Department of Mathematics, University of Puerto Rico at Mayaguez, 2004.
[2] C.C. Aggarwal, Outlier analysis, Springer Science & Business Media, 2013.
[3] C.C. Aggarwal, J. Han, J. Wang, P.S. Yu, On clustering massive data streams: A summarization paradigm, in: Data Streams - Models and Algorithms, 2007, pp. 9–38.
[4] C.C. Aggarwal, P.S. Yu, Outlier detection for high dimensional data. In: Proceedings of the 2001 ACM SIGMOD International Conference on Management of Data, SIGMOD '01, ACM, New York, NY, USA, 2001, pp. 37–46.
[5] A. Ahmad, F. Shaari, Z.A. Long, Outlier detection method based on hybrid rough: Negative using pso algorithm, Proceedings of the 8th International Conference on Ubiquitous Information Management and Communication, ICUIMC '14, ACM, New York, NY, USA, 2014, pp. 108:1–108:5.
[6] S. Alam, G. Dobbie, P. Riddle, M.A. Naeem, A swarm intelligence based clustering approach for outlier detection, in: Evolutionary Computation (CEC), 2010 IEEE Congress on. IEEE, 2010, pp. 1–7.
[7] A. Arning, R. Agrawal, P. Raghavan, A linear method for deviation detection in large databases, in: KDD, 1996, pp. 164–169.
[8] A. Asuncion, D. Newman, in: Uci machine learning repository, 2007.
[9] B. Babcock, S. Babu, M. Datar, R. Motwani, J. Widom, Models and issues in data stream systems, Proceedings of the Twenty-first ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS '02, ACM, New York, NY, USA, 2002, pp. 1–16.
[10] S.D. Bay, M. Schwabacher, Mining distance-based outliers in near linear time with randomization and a simple pruning rule, Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining, ACM, 2003, pp. 29–38.
[11] E. Bonabeau, M. Dorigo, G. Theraulaz, Swarm intelligence: from natural to artificial systems. Vol. 4, Oxford university press, New York, 1999.
[12] F. Cao, M. Ester, W. Qian, A. Zhou, Density-based clustering over an evolving data stream with noise, in: Proceedings of the 2006 SIAM International Conference on Data Mining, 2006, pp. 328–339.
[13] X. Cui, J. Gao, T.E. Potok, A flocking based algorithm for document clustering analysis, J. syst. archit. 52 (8) (2006) 505–515.
[14] X. Cui, T.E. Potok, A distributed agent implementation of multiple species flocking model for document partitioning clustering, Lecture Notes in Computer Science, 4149, Springer, 2006, pp. 124–137.
[15] R.C. Eberhart, Y. Shi, J. Kennedy, Swarm Intelligence, Morgan Kaufmann, 2001.
[16] M. Elahi, K. Li, W. Nisar, X. Lv, H. Wang, Efficient clustering-based outlier detection algorithm for dynamic data stream, in: FSKD (5). IEEE Computer Society, 2008, pp. 298–304.
[17] I. Ellabib, P.H. Calamai, O.A. Basir, Exchange strategies for multiple ant colony system, Inf. Sci. 177 (5) (2007) 1248–1264.
[18] E. Eskin, A. Arnold, M. Prerau, L. Portnoy, S. Stolfo, A geometric framework for unsupervised anomaly detection: Detecting intrusions in unlabeled data, in: Applications of Data Mining in Computer Security, Kluwer, 2002.

[19] M. Ester, H.P. Kriegel, J. Sander, X. Xu, A density-based algorithm for discovering clusters in large spatial databases with noise, in: Proc. 2 nd Int. Conf. on Knowledge Discovery and Data Mining(KDD ' 96), 1996, pp. 226–231.
[20] G. Folino, A. Forestiero, G. Spezzano, An adaptive flocking algorithm for performing approximate clustering, Inf. Sci. 179 (18) (2009) 3059–3078.
[21] A. Forestiero, C. Pizzuti, G. Spezzano, Flockstream: A bio-inspired algorithm for clustering evolving data streams, in: ICTAI. IEEE Computer Society, 2009, pp. 1–8.
[22] P. Grasse, in: La reconstruction du nid et les coordinations inter-individuelle chez bellicoitermes natalenis et cubitermes sp la theorie de la stigmergie: Essai d'interpretation des termites constructeurs. insectes sociaux 6, 1959.
[23] M. Gupta, J. Gao, C. Aggarwal, J. Han, Outlier detection for temporal data: A survey, Knowl. Data Eng. IEEE Trans. 26 (9) (2014) 2250–2267.
[24] L. Huang, X. Nguyen, M. Garofalakis, M.I. Jordan, A. Joseph, N. Taft, In-network pca and anomaly detection, in: Advances in Neural Information Processing Systems, 2006, pp. 617–624.
[25] R. Jindal, S.D. Sharma, M. Manoj Sharma, A new technique to increase the working performance of the ant colony optimization algorithm, Int. J. Innovative Technol. Exploring Eng. 3 (2) (2013) 128–131.
[26] M. Khalilian, N. Mustapha, Data stream clustering: Challenges and issues, 2010. CoRR abs/1006.5261.
[27] E.M. Knorr, R.T. Ng, Algorithms for mining distance-based outliers in large datasets, in: VLDB, Morgan Kaufmann, 1998, pp. 392–403.
[28] R. Lippmann, J.W. Haines, D.J. Fried, J. Korba, K. Das, The 1999 darpa off-line intrusion detection evaluation, Comput. Netw. 34 (4) (2000) 579–595.
[29] B. Liu, M. Cai, J. Yu, Swarm intelligence and its application in abnormal data detection, Informatica 39 (1) (2015).
[30] M.E. Locasto, J.J. Parekh, S. Stolfo, V. Misra, in: Collaborative distributed intrusion detection, 2004.
[31] A.W. Mohemmed, M. Zhang, W.N. Browne, Particle swarm optimisation for outlier detection, in: GECCO, ACM, 2010, pp. 83–84.
[32] A.W. Mohemmed, M. Zhang, W.N. Browne, Particle Swarm Optimisation for Outlier Detection, Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation, GECCO '10, ACM, New York, NY, USA, 2010, pp. 83–84.
[33] N. Monmarché, M. Slimane, G. Venturini, On improving clustering in numerical databases with artificial ants, in: ECAL, in: Lecture Notes in Computer Science, 1674, Springer, 1999, pp. 626–635.
[34] P. Murugavel, M. Punithavalli, Improved hybrid clustering and distance-based technique for outlier removal, Int. J. Comput. Sci. Eng. (IJCSE) 3 (1) (2011) 333–339.
[35] M.E. Otey, A. Ghoting, S. Parthasarathy, Fast distributed outlier detection in mixed-attribute data sets, Data Mining Knowl. Discovery 12 (2–3) (2006) 203–228.
[36] T. Palpanas, D. Papadopoulos, V. Kalogeraki, D. Gunopulos, Distributed deviation detection in sensor networks, ACM SIGMOD Record 32 (4) (2003) 77–82.
[37] D. Pokrajac, A. Lazarevic, L.J. Latecki, Incremental local outlier detection for data streams, in: CIDM, IEEE, 2007, pp. 504–515.
[38] P.A. Porras, P.G. Neumann, Emerald: Event monitoring enabling response to anomalous live disturbances, in: Proceedings of the 20th national information systems security conference, 1997, pp. 353–365.
[39] S. Ramaswamy, R. Rastogi, K. Shim, Efficient algorithms for mining outliers from large data sets, in: W. Chen, J.F. Naughton, P.A. Bernstein (Eds.), SIGMOD Conference, ACM, 2000, pp. 427–438. SIGMOD Record 29(2), June 2000.
[40] C.W. Reynolds, Flocks, herds and schools: A distributed behavioral model, in: M.C. Stone (Ed.), SIGGRAPH, ACM, 1987, pp. 25–34.
[41] A. Shafiq, D. Gillian, P. Riddle, An evolutionary particle swarm optimization algorithm for data clustering, in: Swarm Intelligence Symposium. IEEE, IEEE, 2008, pp. 1–6.
[42] E. Soroush, M.S. Abadeh, J. Habibi, A boosting ant-colony optimization algorithm for computer intrusion detection, in: Proceedings of the 2006 International Symposium on Frontiers in Networking with Applications (FINA 2006), 2006.
[43] L. Su, W. Han, S. Yang, P. Zou, Y. Jia, Continuous adaptive outlier detection on distributed data streams, in: High Performance Computing and Communications, Springer, 2007, pp. 74–85.
[44] J. Tang, Z. Chen, A.W.-C. Fu, D.W. Cheung, Capabilities of outlier detection schemes in large datasets, framework and methodologies, Knowl. Inf. Syst. 11 (1) (2007) 45–84.
[45] C.-L. Tsai, C.-C. Tseng, C.-C. Han, Intrusive behavior analysis based on honey pot tracking and ant algorithm analysis, in: Security Technology, 2009. 43rd Annual 2009 International Carnahan Conference on. IEEE, 2009, pp. 248–252.
[46] A. Zimek, E. Schubert, H.-P. Kriegel, A survey on unsupervised outlier detection in high-dimensional numerical data, Stat. Anal. Data Min. 5 (5) (2012) 363–387.