

An Adaptive Flocking Algorithm for Performing Approximate Clustering

Gianluigi Folino ^{a,*}, Agostino Forestiero ^a and
Giandomenico Spezzano ^a

^a*Institute of High-Performance Computing and Networking (ICAR)
National Research Council (CNR), Italy
Via Pietro Bucci 41C, I-87036 Rende (CS), Italy*

Abstract

This paper presents an approach based on an adaptive bio-inspired method to make state of the art clustering algorithms scalable and to provide them with an any-time behavior. The method is based on the biology-inspired paradigm of a flock of birds, i.e. a population of simple agents interacting locally with each other and with the environment. The flocking algorithm provides a model of decentralized adaptive organization useful to solve complex optimization, classification and distributed control problems. This approach avoids the sequential search of canonical clustering algorithms and permits a scalable implementation.

The method is applied to design two novel clustering algorithms based on the main principles of two popular clustering algorithms: DBSCAN and SNN. This approach can identify clusters of widely varying shapes and densities and is able to extract an approximate view of the clusters whenever it is required. Both the algorithms have been evaluated on synthetic and real-world data sets and the impact of the flocking strategy on performance has been evaluated.

1 Introduction

In recent years several approaches to knowledge discovery and data mining, and in particular clustering, have been developed, but only a few of them are designed using a decentralized approach.

* Corresponding author.

Email address: folino@icar.cnr.it (Gianluigi Folino).

Clustering data is the process of grouping similar objects according to their distance, connectivity, or relative density in space [11]. There are a large number of algorithms for discovering natural clusters in a data set, but they are usually implemented in a centralized way. These algorithms can be classified into partitioning methods [16], hierarchical methods [15], density-based methods [31] and grid-based methods [34]. Han, Kamber and Tung's paper [10] is a good introduction to this subject. In general, clustering algorithms focus on creating good compact representation of clusters using appropriate distance functions between data points. For this purpose, they generally need one or more parameters chosen by a user that indicate the characteristics of the expected clusters.

Centralized clustering is problematic when data to explore are large or widely distributed and/or in a streaming format. Centralized schemes require a high level of connectivity, impose a substantial computational burden, are typically more sensitive to failures than decentralized schemes, and are not scalable. Parallel and distributed computing is expected to relieve current mining methods from the sequential bottleneck, providing the ability to scale to massive datasets, and improving the response time. Achieving good performance on today's high performance systems is a non-trivial task. The main challenges include synchronization and communication minimization, work-load balancing, finding good data decomposition, etc. Some existing centralized clustering algorithms have been parallelized and the results are encouraging.

In the last few years, innovative algorithms based on biological models [3][20][21][24][4] have been introduced to solve the clustering problem in a decentralized fashion. These algorithms are characterized by the interaction of a large number of simple agents that sense and change their environment locally. Furthermore, they exhibit complex, emergent behavior that is robust with respect to the failure of individual agents. Ant colonies, flocks of birds, termites, swarms of bees etc. are agent-based insect models that exhibit a collective intelligent behavior (swarm intelligence) [1]. In Swarm Intelligence models, the emergent collective behavior is the outcome of a process of self-organization, in which insects are engaged through their repeated actions and interaction with their evolving environment. Intelligent behavior frequently arises through indirect communication between the agents using the mechanism of stigmergy [8]. Stigmergy is based on cooperation in insect societies. 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. 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 too much the behavior of the overall system.

In this paper, a bio-inspired method is presented using a flock of birds to make *state of art* clustering algorithms scalable and to provide them with an any-time behavior. The flocking algorithm is based on the simple rules defined by Reynolds with the extensions proposed by Macgill [22]. It is naturally decentralized, is suitable for distributed implementations and is robust to the failure of single individuals. Furthermore, this flocking algorithm combined with canonical clustering algorithms (i.e. DBSCAN [6] and SNN [5]) can be profitably applied as a data reduction technique to perform efficiently approximate clustering [19].

The method takes advantage of the collective search mechanism a flock implies, by which if a member of a flock finds an area of interest, the mechanics of the flock will draw other members to scan that area in more detail. So, while the standard clustering algorithms must scan the entire dataset to discover the clusters, this method can obtain a first approximation of the clusters, visiting only a portion of the points. Afterwards, the any-time property of the flocking algorithm will increase the size and the quality of the clusters as the system goes on running.

In this paper, the flocking-based approach is combined with two standard clustering algorithms in order to perform approximate clustering. Indeed, the flock performs a biased and incremental search on interesting points of the dataset, then a local strategy, based on standard clustering algorithms, can be used to merge the discovered points.

The first algorithm, named SPARROW (SPAtial ClusteRing AlgoRithm thrOugh SWarm Intelligence) combines the flocking algorithm with the density-based DBSCAN [6] algorithm. The other, named SPARROW-SNN (Shared Nearest-Neighbor similarity) couples the flocking algorithm with a shared nearest neighbor (SNN) [5] cluster algorithm to discover clusters with differing sizes, shapes in noise and high dimensional data. The two algorithms present a substantial difference, SPARROW, as it is based on a density approach, is not appropriate for discovering clusters with different densities, while SPARROW-SNN can cope with this kind of data.

A SWARM [23] simulation of both algorithms was built to investigate their properties and to assess the effectiveness of the two methods. In particular, it was evaluated the accuracy of the approximate solution, the impact of the number of agents on the foraging for clusters performance and was performed a qualitative comparison with a well known approximate clustering algorithm.

The rest of this paper is organized as follows: Section 2 presents the multi agent adaptive flocking algorithm, first introducing the classical model of Reynolds and then the modified rules of the flock algorithm that add an adaptive behavior to make it more effective for searching interesting objects; Section 3

shows how the stochastic search provided by the adaptive flocking algorithm can be used as a basis for clustering spatial data, combining it with a local merging strategy based respectively on DBSCAN and SNN to constitute the SPARROW and SPARROW-SNN algorithms; Section 4 discusses some related works concerning clustering algorithms based on the Swarm Intelligence paradigm. Section 5 presents the experimental results for the two approximate clustering algorithms and Section 6 reports discussions about the feasible strategies of parallelization, remarking advantages and drawbacks. Finally, section 7 draws some conclusions.

2 A multi agent adaptive flocking algorithm

In this section, a multi-agent stochastic search based on a classical flocking algorithm is presented, which has the advantage of being easily implementable on parallel and distributed machines and is robust compared to the failure of individual agents. First, the rules governing the flock model introduced by Reynolds are explained; then, the modified behavioral rules of the swarm agents are illustrated. They add an adaptive behavior to the flock and make it more effective in searching points, which have some desired properties in the space.

2.1 *The Reynolds' flock model*

The flocking algorithm was originally proposed by Reynolds [30] as a method for simulating the flocking behavior of birds on a computer both for animation and as a way to study emergent behavior. Flocking is an example of emergent collective behavior: there is no leader, i.e., no global control. Flocking behavior emerges from the local interactions. In the flock algorithm each agent has direct access to the geometric description of the whole scene, but reacts only to flock mates within a certain small radius. The basic flocking model consists of three kind of simple steering behavior:

Separation gives an agent the ability to maintain a certain distance from others nearby. This prevents agents from crowding too closely together, allowing them to scan a wider area.

Cohesion supplies an agent with the ability to cohere (approach and form a group) with other nearby 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.

Alignment gives an agent the ability to align with other nearby characters. Steering for alignment can be computed by finding all agents in the local neighborhood and averaging together the 'heading' vectors of the nearby agents.

2.2 An adaptive behavior flock for searching objects in spatial data

Different techniques can be used to cope with the problem of searching interesting object in spatial data. A multi-agent adaptive algorithm able to discover these points in parallel was introduced. This algorithm uses a modified version of the standard flocking algorithm, described above, which incorporates the capacity for learning that we can find in many social insects. In this algorithm, the agents are transformed into hunters with a foraging behavior that allows them to explore spatial data efficiently. This flocking algorithm was inspired by a work presented by Macgill [22]. However, the approach presented here has some optimizations explained better in the following that make it more appropriate for the task of clustering spatial objects.

The algorithm starts with a fixed number of agents that occupy a randomly generated position in this space. Each agent moves around the spatial data, testing the neighborhood of each location in order to verify whether a point can have some desired properties. Each agent follows the rules of movement described in Reynolds' model. In addition, this model considers four different kinds of agents, classified on the basis of some properties of data in their neighborhood. Different agents are characterized by a different color: red, revealing interesting patterns in the data, green, a medium one, yellow, a low one, and white, indicating a total absence of patterns.

The main idea behind this approach is to take advantage of the colored agent in order to explore more accurately the most interesting regions (signaled by the red agents) and avoid the ones without interesting points (signaled by the white agents). Red and white agents stop moving in order to signal this type of region to the others, while green and yellow ones fly to find denser zones. Indeed, each flying agent computes its heading by taking the weighted average of alignment, separation and cohesion (as illustrated in figure 1).

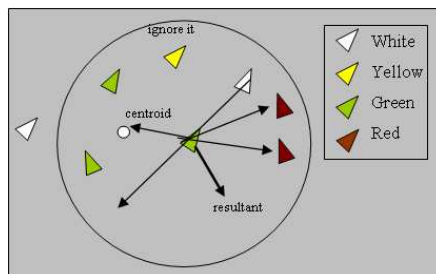


Fig. 1. Computing the direction of a green agent.

The following are the main features which make this model different from Reynolds' model:

- *Alignment* and *cohesion* do not consider yellow agents, since they move in a not very attractive zone.
- *Cohesion* is the resultant of heading towards the average position of the green flockmates (centroid), of the attraction towards reds, and of the repulsion from whites, as illustrated in figure 1.
- A *separation* distance is maintained from all the agents, apart from their color.

In figure 2, we summarized the pseudo code of the overall algorithm with the main functions better explained in the table 1.

```

for i=1 ... MaxIterations
    foreach agent (yellow, green)
        age=age+1;
        if (age > Max_Life)
            generate_new_agent();die();
        endif
        if (not visited (current_point))
            property = compute_local_property(current_point);
            mycolor= color_agent(property);
        endif
    end foreach
    foreach agent (yellow, green)
        dir= compute_dir();
    end foreach
    foreach agent (all)
        switch (mycolor){
            case yellow, green: move(dir, speed(mycolor)); break;
            case white: stop(); generate_new_agent(); break;
            case red: stop(); generate_new_close_agent(); break; }
    end foreach
end for

```

Fig. 2. The pseudo-code of the adaptive flocking algorithm.

Yellow and green agents will compute their direction, according to the rules previously described, and will move following this direction and with the speed corresponding to their color. Note that the color of the agent is assigned on the basis of the desired property at the point in which it falls; the assignment is made on a scale going from white ($property = 0$) to red ($property > threshold$), passing for yellow and green, corresponding to intermediate values.

| Function | Meaning |
|---|---|
| <code>generate_new_close_agent()</code> | generate a new agent in a random position (close to the current) |
| <code>generate_new_agent()</code> | generate a new agent in a random position of the search space |
| <code>stop()</code> | stop the agent |
| <code>die()</code> | kill the agent |
| <code>compute_local_property(P)</code> | compute the desired property at the point P |
| <code>color_agent(prop)</code> | color the agent on the basis of the property $prop$ |
| <code>speed(col)</code> | compute the speed of the agent depending on the color col |
| <code>compute_dir()</code> | compute the new direction of the agent using the modified Reynolds' model |
| <code>move(dir, s)</code> | move the agent with direction dir and speed s |

Table 1

Meaning of the main functions used in the pseudocode.

Agents will move towards the computed direction with a speed depending on their color: green agents more slowly than yellow agents since they will explore more interesting regions. An agent will speed up to leave an empty or uninteresting region whereas it will slow down to investigate an interesting region more carefully.

The variable speed introduces an adaptive behavior into the algorithm. In fact, agents adapt their movement and change their behavior (speed) on the basis of their previous experience and on the position of the red and white agents. Indeed, red and white agents will stop signaling to the others respectively the interesting and desert regions. Note that, for any agent that has become red or white, a new agent will be generated in order to maintain a constant number of agents exploring the data. In the first case (red), the new agent will be generated in a close random point, since the zone is considered interesting, while in the latter it will be generated in a random point over all the space. Anyway, this does not affect the overall performance of the system as the number of agents was not increased; in fact, white and red agents are not real agents, but only their position is stored. Finally, in the case where the agent falls in the same position as an older one it will be regenerated using the same policy described above.

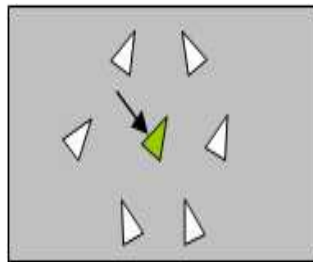


Fig. 3. The cage effect.

Yellow and green agents try to avoid the 'cage effect' (see figure 3); in fact, some agents could remain trapped inside regions surrounded by red or white agents and would have no way to go out, wasting useful resources for the exploration. So, a limit was imposed on their life. When their age exceeds a determined value (Max_Life) they die and are regenerated in a new randomly chosen position of the space.

Note that this algorithm presents some differences from that of Macgill. The *generate_new_agent* and *generate_new_close_agent* procedures maintain a constant number of agents exploring the search space respectively in all the space or in a neighborhood of interesting points. Aging limits the cage effect as described above and furthermore property of the point are computed only when the birds fall in a previously unvisited point.

2.3 Formal description of the flocking algorithm

In this subsection, a more formal description of the flocking algorithm is presented.

Consider the search space, in which the swarm moves, having dimension d . Let N be the number of birds and B be the set of all the birds $\{B_1, B_2, \dots, B_N\}$. Each bird B_k can be represented by three d -dimensional vectors: its position in this space $Pos_k : (x_k^1, x_k^2, \dots, x_k^d)$, its direction $Dir_k : (dir_k^1, dir_k^2, \dots, dir_k^d)$, where dir_k^i represents the component along the axis i of the direction of the bird and the color $Col_k \in \{white, yellow, green, red\}$, indicating the type of bird. We used as distance between two birds B_a and B_b , the euclidean distance between their respective positions: $dist(B_a, B_b) = \sqrt{\sum_{i=1}^d (x_a^i - x_b^i)^2}$.

We define as *dist_max* and *dist_min* respectively the radius indicating the limited sight of the birds and the minimum distance that must be maintained among them. $Neigh(B_k)$ denotes the neighborhood of a bird B_k , i.e. the set $\{B_\alpha \in B \mid dist(B_k, B_\alpha) \leq dist_max\}$, that is the set of the birds visible from the bird B_k . Furthermore, we define as $Neigh(col, B_k)$ the set $\{B_\alpha \in B \mid dist(B_k, B_\alpha) \leq dist_max, Col_\alpha = col\}$, that is the set of the birds, having color col , visible from the bird B_k .

Each bird moves with speed v , depending on the color of the agents (green agents' speed is slower, because they are exploring interesting zones). Then, for each iteration t , the new position of a bird B_k can be computed as:

$$\forall i = 1 \dots d \quad x_k^i(t+1) = x_k^i(t) + v \times dir_k^i \quad (1)$$

Note also that for each iteration the new direction of the agent k is obtained summing the three components of alignment, separation and cohesion:

$$dir_k^i = dir_al_k^i - dir_sep_k^i + dir_co_k^i. \quad (2)$$

Considering as $dir(B_a, B_b)$ the normalized direction of the vector between a bird B_a and a bird B_b , these components can be computed using the following

formulas (3, 4 and 5):

$$dir_al_k^i = \frac{1}{|Neigh(green, B_k)|} \cdot \sum_{B_\alpha \in Neigh(green, B_k)} dir_\alpha^i \quad (3)$$

and considering $centr(green, B_k)$ as the position of the centroid of the green agents in the neighborhood of k with generic coordinate i :

$\frac{1}{|Neigh(green, B_k)|} \cdot \sum_{B_\alpha \in Neigh(green, B_k)} x_\alpha^i$, then:

$$dir_co_k^i = dir(centr(green, B_k), B_k)^i + attr_red - rep_white \quad (4)$$

where $attr_red$ is equals to $\sum_{B_\alpha \in Neigh(red, B_k)} dir(B_\alpha, B_k)^i$ and rep_white is equals to $\sum_{B_\alpha \in Neigh(white, B_k)} dir(B_\alpha, B_k)^i$ i.e the sum of the attraction towards the centroid, of the attraction towards the red birds and of the repulsion from the white birds;

$$dir_sep_k^i = \sum_{B_\alpha \in Neigh(B_k), dist(B_\alpha, B_k) < dist_min} dir(B_\alpha, B_k)^i \quad (5)$$

3 Using the flocking algorithm for clustering spatial data

This section shows how the stochastic search provided by the adaptive flocking described in the previous section can be used as a basis for implementing algorithms for clustering spatial data. In particular, two clustering algorithms, named SPARROW and SPARROW-SNN, were implemented, which respectively combine the flocking strategy with the DBSCAN heuristics for discovering clusters with differing sizes, shapes in noise data, and with the SNN heuristics in order to discover clusters with differing densities. Note that the first, as it is based on a density approach, is not appropriate for discovering clusters with different densities, while the latter can cope with this problem.

While DBSCAN (and SNN) must scan the entire dataset to define all the clusters, our algorithm can obtain a first approximation of the clusters, visiting only a portion of the entire dataset. Hence, it can be applied to perform efficiently *approximate clustering* since the points, which are visited and analyzed by the agents, represent a biased sample of the entire dataset.

In the following, the DBSCAN algorithm and the SPARROW are described, then, SNN and SPARROW SNN are illustrated.

3.1 The DBSCAN algorithm

One of the most popular spatial clustering algorithms is DBSCAN, which is a density-based spatial clustering algorithm. A complete description of the algorithm and its theoretical basis is presented in the paper by Ester et al. [6]. In the following, the main principles of DBSCAN are briefly presented. The algorithm is based on the idea that all points of a data set can be re-grouped into two classes: *clusters* and *noise*. Clusters are defined as a set of dense connected regions with a given radius (*Eps*) and containing at least a minimum number (*MinPts*) of points. Data are regarded as noise when the number of points contained in a region falls below a specified threshold. The two parameters, *Eps* and *MinPts*, must be specified by the user and allow the density of the cluster to be controlled, which must be retrieved. The algorithm defines two different kinds of point in a clustering: *core points* and *non-core points*. A core point is a point with at least *MinPts* number of points contained in an *Eps*-neighborhood of the point. The non-core points in turn are either *border points*, if they are not core points, but are density reachable from another core point or *noise points* if they are not core points and are not density reachable from other points. Note that (1) a point p is directly density reachable from a point q if their distance is less than (*Eps*) and q is a core point; (2) a point p is density reachable from q if there is a chain of points $p_1, p_2, \dots, p_n, p_1 = q, p_n = p$ such that p_{i+1} is directly density reachable from p_i .

To find the clusters in a data set, DBSCAN starts from an arbitrary point and retrieves all points density reachable from that point using *Eps* and *MinPts* as controlling parameters. DBSCAN builds clusters in sequence (that is, one at a time), in the order in which they are encountered during the search. The retrieval of the density of a cluster is performed by successive spatial queries. These queries are supported efficiently by spatial access methods such as R*-trees.

3.2 SPARROW

SPARROW combines the multi-agent adaptive flocking algorithm with the DBSCAN clustering technique in order to discover clusters in parallel. It uses the previously described modified version of the standard flocking algorithm that incorporates the capacity for learning, which can be found in many social insects. Flocking agents are transformed into hunters with a foraging behavior that allow them to explore the spatial data and to search for clusters.

The algorithm follows the same pseudo-code described in figure 2, but the

compute_property function is derived from DBSCAN and computes the local density of the points of clusters in the data belonging to the neighborhood of the current point, then the *myColor* procedure chooses the color and the speed of the agents with regard to this local density. The algorithm is based on the same parameters used in the DBSCAN algorithm: *MinPts*, the minimum number of points to form a cluster and *Eps*, the maximum distance that the agents can look at. In practice, the agent computes the local density (called *density* in the following) in a circular neighborhood (with a radius determined by its limited sight, i.e. *Eps*) and then it chooses the color in accordance with the following simple rules:

$$\begin{aligned}
 \text{density} > \text{MinPts} & \Rightarrow \text{mycolor} = \text{red} & (\text{speed} = 0) \\
 \frac{\text{MinPts}}{4} < \text{density} \leq \text{MinPts} & \Rightarrow \text{mycolor} = \text{green} & (\text{speed} = 1) \\
 0 < \text{density} \leq \frac{\text{MinPts}}{4} & \Rightarrow \text{mycolor} = \text{yellow} & (\text{speed} = 2) \\
 \text{density} = 0 & \Rightarrow \text{mycolor} = \text{white} & (\text{speed} = 0)
 \end{aligned}$$

In the running phase, the yellow and green agents will compute their direction, according to the rules previously described, and will move following this direction and with the speed corresponding to their color.

In addition, new red agents will run a *merge* procedure, which will merge neighboring clusters. The merging phase considers two different cases: when points in the circular neighborhood have never been visited and when there are points belonging to different clusters. In the first case, the points will be labeled and will constitute a new cluster; in the second case, all the points will be merged into the same cluster, i.e. they will get the label of the cluster discovered first.

SPARROW suffers the same limitation as DBSCAN, i.e. it cannot cope with clusters of different densities. A new algorithm SPARROW-SNN, introduced in the next subsection, is more general and overcomes these drawbacks. It can be used to discover clusters with differing sizes, shapes and densities in noise data.

3.3 The SNN clustering algorithm

SNN is a clustering algorithm developed by Ertöz, Steinbach and Kumar [5] to discover clusters with differing sizes, shapes and densities in noise and high dimensional data. The algorithm extends the nearest-neighbor non-hierarchical clustering technique by Jarvis-Patrick [13] redefining the similarity between pairs of points in terms of how many nearest neighbors the two points share. Using this new definition of similarity, the algorithm eliminates noise and outliers, identifies representative points, and then builds clusters around the

representative points. These clusters do not contain all the points, but rather represent relatively uniform group of points. The SNN algorithm starts performing the Jarvis-Patrick scheme. In the Jarvis-Patrick algorithm a set of objects is partitioned into clusters on the basis of the number of shared nearest-neighbors. The standard implementation is constituted of two phases. The first is a pre-processing stage that identifies the K nearest-neighbors of each object in the dataset. In the subsequent clustering stage, two objects i and j join the same cluster if:

- i is one of the K nearest – neighbors of j ;
- j is one of the K nearest – neighbors of i ;
- i and j have at least K_{min} of their K nearest – neighbours in common;

where K and K_{min} are used-defined parameters. For each pair of points i and j is defined a link with an associate weight. The strength of the link between i and j is defined as:

$$strength(i, j) = \sum (K + 1 - m)(K + 1 - n) \quad where \quad i_m = j_n$$

In the equation above, K is the nearest neighbor list size, m and n are the positions of a shared nearest neighbor in i and j 's lists. At this point, clusters can be obtained by removing all edges with weights less than a user specified threshold and taking all the connected components as clusters. A major drawback of the Jarvis-Patrick algorithm is that the threshold needs to be set high enough since two distinct set of points can be merged into same cluster even if there is only link across them. On the other hand, if a high threshold is applied, then a natural cluster will be split into many small clusters due to the variations in the similarity in the cluster.

SNN addresses these problems introducing the following steps:

- (1) For every node (data point). it calculates the total strength (connectivity) of links coming out of the point;
- (2) It identifies representative points by choosing the points that have high connectivity ($> core_threshold$);
- (3) It identifies noise points by choosing the points that have low connectivity ($< noise_threshold$) and remove them;
- (4) It removes all links between points that have a weight smaller than a threshold ($merge_threshold$);
- (5) It takes connected components of points to form clusters, where every point in a cluster is either a representative point or is connected to a representative point.

The number of clusters is not given to the algorithm as a parameter. Also note that not all the points are clustered (they are implicitly considered as noise).

3.4 SPARROW-SNN

The SPARROW-SNN strategy of search, derived from the SNN algorithm main principles, permits to discovery clusters of arbitrary form and density. In practice, the flocking algorithm performs a biased sampling of the points of the dataset, as it focuses the search on interesting parts of the search space. Thus, the basic algorithm could be combined with many different clustering algorithms, for instance k-means. However, it is well known that algorithms as k-means are not effective in finding clusters with different densities or with globular shapes and hardly identify outliers. For instance, take a look at the example, found in literature, of figure 4; on the left correct clusters are showed, while on the right, the results of the execution of k-means. This algorithm clearly fails to identify the correct clusters; in the case a, because the clusters have different densities, while in the case b, because they are not globular. On the contrary, SPARROW-SNN is able to handle this kind of clusters; this statement will be clarified in the description of the algorithm and in the experimental results.

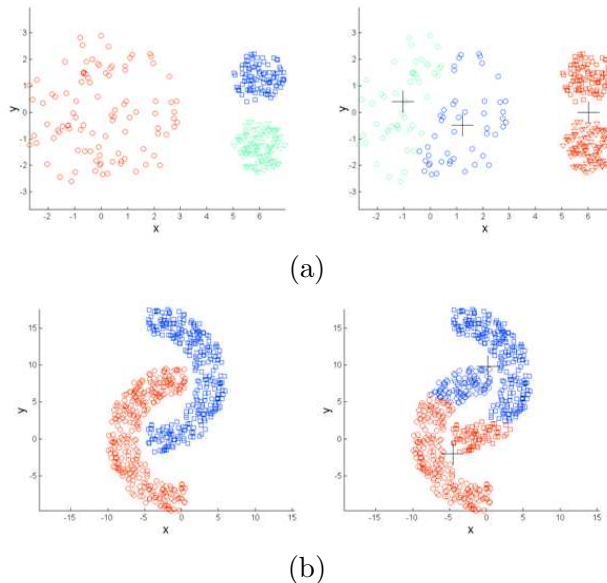


Fig. 4. Example of a) different densities and b) non globular clusters, not correctly individuated from the k-means algorithm (correct clusters on the left, clusters detected from k-means on the right)

To better understand as SPARROW-SNN works, the pseudocode was shown in figure 2. The algorithm starts with a fixed number of agents that will occupy a randomly generated position. From their initial position, each agent moves

around the spatial data testing the neighborhood of each location in order to verify whether the point can be identified as a *representative* (or core) point.

The *compute_property* function represents the connectivity of the point as defined in the SNN algorithm. In practice, when an agent falls on a data point A , not yet visited, it computes the connectivity, $conn(A)$, of the point, i.e. computes the total number of strong links the points has, according to the rules of the SNN algorithm. Note that SPARROW-SNN computes the nearest-neighbor list for each data element, using a *similarity-threshold* that reduces the number of data elements to be taken into consideration. The introduction of the similarity-threshold produces variable-length nearest-neighbor lists and therefore i and j must have at least P_{min} of the shorter nearest-neighbor list in common; where P_{min} is a user-defined percentage.

Points having connectivity smaller than a fixed threshold (*noise_threshold*) are classified as noise and are considered for removal from clustering. Then a color is assigned to each agent, on the basis of the value of the connectivity computed in the visited point, using the following procedure (called *color_agent()* in the pseudocode):

$$\begin{aligned}
 conn > core_threshold & \Rightarrow mycolor = red & (speed = 0) \\
 noise_threshold < conn \leq core_threshold & \Rightarrow mycolor = green & (speed = 1) \\
 0 < conn < noise_threshold & \Rightarrow mycolor = yellow & (speed = 2) \\
 conn = 0 & \Rightarrow mycolor = white & (speed = 0)
 \end{aligned}$$

The colors assigned to the agents are: red, revealing representative points, green, border points, yellow, noise points, and white, indicating an obstacle (uninteresting region). After the coloration step, the green and yellow agents compute their movement observing the positions of all other agents that are at most at some fixed distance (*dist_max*) from their and applying the same rules as SPARROW. In any case, each new red agent (placed on a representative point) will run the merge procedure, as described in subsection 3.2, so that it will include, in the final cluster, the representative point discovered, and together to the points that share with it a significant (greater than P_{min}) number of neighbors and that are not noise points. The merging phase considers two different cases: when points in the neighborhood have never been visited and when there are points belonging to different clusters. In the former, the same temporary label will be assigned and a new cluster will be constituted; in the latter, all the points will be merged into the same cluster, i.e. they will get the label corresponding to the smallest label. Thus clusters will be built incrementally.

4 Related works

Particle Swarm Optimization [18] (a population based optimization tool) is also inspired by the behavior of flocks of birds, but it is quite different from the approach used in this paper. PSO consists of a swarm of particles, each of them representing the solution to an optimization problem, moving in the problem search space. At the beginning, each particle has a random position and velocity. A particle moves on the basis of its own experience (its best past position) and of the best of particles in the swarm. Note that, not differently from our model, variants of PSO consider the best of a local neighborhood instead of all the swarm. Furthermore, the best particle is an attractor for the other particles as the red birds of our algorithm are the attractors for the other birds. However, the similarities with the flocking model ended here, since it moves for searching representative points and merge them in order to find clusters, while each PSO particle represents itself a solution to a problem and as a consequence it is less flexible in searching clusters (see later in this section). Moreover, in the flock algorithm, there is a stricter interrelation, while particles only look at the best particle in the group. Then PSO does not use any repulsion force to avoid uninteresting zones, while the white agents here are repulsive for the others.

The clustering problem has been faced in the recent past using algorithms based on the Swarm Intelligence paradigm. Termites, swarms of bees and especially ant colonies have polarized the attention of researchers, but the interesting properties of flock of birds have not been sufficiently exploited in the clustering task.

The first pioneering work on clustering using artificial ants was conducted by Deneuborg in [3]. Ants-like agents clusterize together objects, moving through a two-dimensional grid and picking and dropping them with a probability based on an estimation of the density of objects of the same type in the neighborhood. Lumer and Faieta in [21] extended this work including a distance function between data objects for the purpose of exploratory data analysis and applied these techniques to a database of customer profiles, projecting the space of attributes into a two-dimensional space. Moreover, they introduced ants with different moving speeds (fast ants group clusters on large scales, while slow ants work at smaller scales by placing data with more accuracy), a short term memory (ants can remember the last m dropped data, avoiding creating equivalent clusters) and behavioral switches (ants destroy clusters if they have not performed any deposit or pick up actions for a given number of time steps in order to escape local non-optimal spatial configurations).

Monmarché et al. in [24] proposed an algorithm that combines in an intelligent way clustering ants and k-means. Moreover, many data can be put on the same

cell and each cell with a non-zero number of items corresponds to a cluster. In the same way, each ant can carry more data up to a maximum capacity. Initially, ants cluster data objects, then k-means was applied to refine the clusters, and this procedure is repeated for a heap of objects. The evaluation on some synthetic and simple real world datasets gave good results in comparison with classical clustering algorithms. The ACluster algorithm overcomes some drawbacks and simplifies the model of Lumer and Faieta, introducing the concept of bio-inspired spatial transition probabilities that increase pheromone in locations where objects are more diffuse and adding two different threshold responses, one associated with the density of data and the other with their similarity. The algorithm is used for the task of image retrieval [27], for the clustering of textual documents [29] and extended in [28] for applying it to continuous data stream.

In [12], Handl et al. added, to the classical ant algorithm, adaptive scaling (when few pickup and drop action occurred over a specified time period the ants' sensitivity was reduced, in the opposite case it was increased), ant jumps (more efficient exploration of the space), stagnation control (after a number of unsuccessful dropping attempts an ant drops its load regardless of the neighborhood's similarity) and eager ants (ant was put immediately on a data point) to the classical model. They evaluated the performance of the algorithm with a number of standard techniques for clustering and a modified version of the algorithm was applied to the problem of and topographic mapping. Ghosh et al. [7] designed the APC (aggregation pheromone based) algorithm for performing data clustering. The system works in two steps: first ants, driven by the pheromone, form homogenous groups of clusters, then the average linkage agglomerative hierarchical clustering algorithm is applied for merging the resulting groups until the desired number of clusters is obtained. This algorithm was evaluated on a number of well-known benchmark data sets using different cluster validity measures. Results are compared with standard clustering techniques and with the previously cited algorithm used by Handl; experimental results showed the good performance both in terms of the quality of clustering as well as execution time. a limit of the approach is that the the number of cluster must be fixed in advance.

Kanade and Hall in [14] developed an ant algorithm that, in a first stage, moves the cluster centers (and not the data points) in the feature space. As a second stage, the found centers are evaluated using the Fuzzy C Means or Hard C Means algorithm. These two stages are repeated up to a fixed number of epochs. The results on some datasets are superior to the behavior of FCM and HCM initialized randomly.

In [33] an adaptive ant clustering algorithm (A2CA) was introduced. Among the novelties in comparison with the other algorithms, the concept of progressive vision is interesting, which permits coping with clusters of different

dimension, making ants augment their vision when they encounter a big cluster. The algorithm was also evaluated on a real world bioinformatics dataset in which the classical ant clustering algorithms fails, while A2CA worked quite well.

The first idea of employing a flock of birds to perform the task of clustering can be found in [25]. The authors used a flock of colored birds to drive a geographical analysis machine (GAM); in practice, a flock of circles of different sizes explore the search space in order to find interesting geographical patterns. The system was validated on the city of Baltimore for discovering crime patterns. In this approach the flock suggests geographic regions of interest, but other methods are necessary to separate and identify clusters. In [2], Cui and others introduce a Multiple Species Flocking algorithm suitable for a decentralized approach and apply it to the problem of document clustering. The authors discuss some communication schemes and show a simple distributed implementation on three nodes and its execution time. Another flocking algorithm was presented in [26]. The algorithm was successfully applied to a real database concerning the field of Biometrics. Furthermore, agents and clusters can be visualized by LCD stereoscopic glasses. In both the two flocking based algorithms, differently from the system used here, each bird represents a point of the dataset and therefore it is scalable with difficulty to large datasets.

Kazemian et al. [17] introduced a data clustering strategy based on Flower Pollination by Artificial Bees (FPAB) in which the flowers represent the data and the gardens the clusters and pollen is used to move the flowers with the lowest growth (probably not belonging to that cluster) from one garden to another. Then, the Fuzzy C Means algorithm is applied to refine the clusters and the new centers found are used as new flowers and the algorithm is repeated until an appropriate number of clusters is reached. FPAB is applied to some simple datasets and was compared with an ant algorithm.

Particle Swarm Optimization was also applied to the clustering task, but the point of view is quite different from the swarm based one. In fact, PSO was applied to discover clusters using an approach in which each particle represents the cluster centroids, as in [32], for example. This limits the applicability of the algorithm and presents all the problems of the centroid based approach, i.e. it is necessary to know the number of clusters a priori.

5 Experimental results

In the next subsections, the experimental results obtained for the two clustering algorithms are presented. Both algorithms were implemented using Swarm, a multi-agent software platform for the simulation of complex adaptive sys-

tems. In the Swarm system, the basic unit of simulation is the swarm, a collection of agents executing a schedule of actions. **Swarm** provides object oriented libraries of reusable components for building models and analyzing, displaying, and controlling experiments on those models. More information about Swarm can be obtained from [23]. All the experiments were averaged over 30 tries. Where not differently specified, the algorithm was run using 50 agents and the visibility radius of the agents (`dist_max`) set at 6 and the minimum distance among the agents (`dist_min`) at 2.

5.1 Test suite used in the experiments

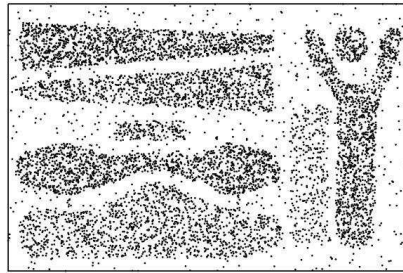


Fig. 5. The DS1 dataset.

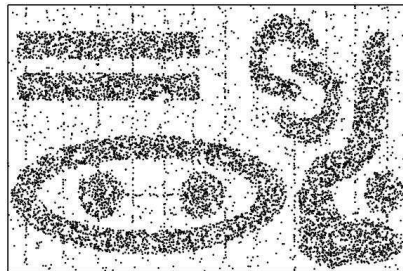


Fig. 6. The DS4 dataset.



Fig. 7. The North-East dataset.

SPARROW and SPARROW-SNN are based on the merge technique used respectively in DBSCAN and SNN. Differently from the basic algorithms,

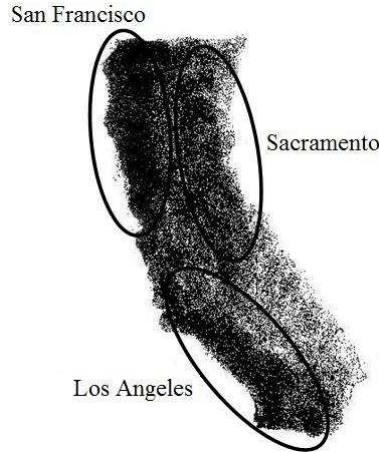


Fig. 8. The Sequoia dataset.

they employ a clever technique to explore the search space more efficiently and perform approximate clustering.

Specifically, the following experiments were conducted:

- (1) determination of the accuracy of the approximate solution that would be obtained if the cluster algorithm were run only on a reduced percentage of points, as opposed to running the DBSCAN (SNN) clustering algorithm on the entire dataset;
- (2) determination of the impact of the number of agents on the clustering performance;
- (3) a qualitative comparison with a hybrid technique based on a uniform sampling or a biased sampling combined with a well known hierarchical clustering algorithm.

Two synthetic data sets (DS4 and DS1) and two real (North-East and Sequoia) were used for conducting the above described experiments. Each point of the data sets has two attributes that define the x and y coordinates. Furthermore, all the data sets have a considerable quantity of noise.

The artificial datasets, DS1 (see figure 5) and DS4 (see figure 6), contain respectively 8000 and 8843 points. Note that DS1 presents clusters with different densities and hence only SPARROW-SNN can succeed in finding the clusters. In fact, the first was used only for testing SPARROW-SNN, while the latter only for SPARROW. The third dataset, called North-East, showed in figure 7, contains 123593 points representing postal addresses of three metropolitan areas (New York, Boston and Philadelphia) in the North East States, It comprises a lot of noise represented from distributed rural areas and smaller towns. The fourth called SEQUOIA, showed in figure 8 is composed of 62556 names of landmarks (and their coordinates), and was extracted from the US Geological Survey's Geographic Name Information System. This dataset presents

| | Perc. of data points for cluster | | | |
|--------------|---|---------|---------|---------|
| | 7% | 12% | 22% | 70% |
| Philadelphia | 40.92 % | 62.09 % | 76.63 % | 97.21 % |
| New York | 34.26 % | 50.33 % | 64.89 % | 95.12 % |
| Boston | 44.87 % | 64.21 % | 79.87 % | 97.89 % |

Table 2

Number of clusters and number of points for clusters for North-East data set (percentage in comparison to the total point for cluster found by DBSCAN) when SPARROW analyzes 7%. 12%. 22% and 70% points.

three main clusters that correspond respectively to the areas of S. Francisco, Sacramento and Los Angeles.

5.2 SPARROW results

As the algorithm under consideration is based on the merge procedure of DBSCAN, it was decided to compare it with a publicly available version of this algorithm. Our approach uses the same parameters as DBSCAN. Therefore, if all the points of the dataset were visited, the same results would be obtained. Then, the cluster points found by DBSCAN (note DBSCAN visit all the points), in the experiments were considered as 100%. It is necessary to verify how this percentage was approached visiting only a portion of the entire dataset. As for the parameters, Eps was set at 9 for all the datasets and MinPts at 20 for DS4 and 40 for Sequoia and the North-East.

Table 2, 3 and 4 show, for the three data sets, the number of clusters and the percentage of points for each cluster found by DBSCAN and SPARROW. Although DBSCAN and SPARROW would produce the same results if all points of the data set were examined, these experiments showed that SPARROW can obtain, with an average accuracy about 78% on DS4, the same number of clusters with a good percentage of points for each cluster using only 22% of the spatial queries used by DBSCAN. The same behavior can be observed in the tables of the two real datasets. On the other hand, DBSCAN can not detect clusters, without visiting all the points, because of the different strategy of attribution of the points to the clusters. In fact, if DBSCAN was stopped before it had performed the spatial queries on all the points, a correct number of points would be obtained for the clusters already individuated and probably a smaller number of points for the cluster that it was building, but obviously, not all the clusters would be discovered.

Finally, the impact of the number of agents on the foraging for clusters performance is presented. Figures 9, 10 and 11 give the number of core points

| | Perc. of data points for cluster | | | |
|---|---|--------|--------|---------|
| | 7% | 12% | 22% | 70% |
| 1 | 51.16% | 70.99% | 78.86% | 95.76 % |
| 2 | 45.91% | 64.74% | 74.40% | 95.45 % |
| 3 | 40.68% | 59.36% | 81.95% | 97.55 % |
| 4 | 44.21% | 60.66% | 81.67% | 98.05 % |
| 5 | 54.65% | 58.72% | 71.54% | 94.99 % |
| 6 | 48.77% | 59.91% | 78.10% | 97.76 % |
| 7 | 54.29% | 66.43% | 79.18% | 96.12 % |
| 8 | 51.16% | 70.99% | 78.86% | 96.33 % |
| 9 | 45.91% | 64.74% | 74.40% | 95.25 % |

Table 3

Number of clusters and number of points for clusters for DS4 data set (percentage in comparison to the total point for cluster found by DBSCAN) when SPARROW analyzes 7%. 12%. 22% and 70% points.

| | Perc. of data points for cluster | | | |
|--------------|---|--------|--------|--------|
| | 7% | 12% | 22% | 70% |
| S. Francisco | 48.12% | 66.22% | 79.32% | 98.88% |
| Sacramento | 44.03% | 61.11% | 80.34% | 97.56% |
| Los Angeles | 51.21% | 68.65% | 81.92% | 98.32% |

Table 4

Number of clusters and number of points for clusters for Sequoia data set (percentage in comparison to the total point for cluster found by DBSCAN) when SPARROW analyzes 7%. 12%. 22% and 70% points.

found in 100 time steps (iterations) for 25, 50 and 100 agents. A comparative analysis reveals that a 100-agents population discovers a larger number of points than the other two populations with a smaller number of agents (the scalability is almost linear). This scalable behavior of the algorithm would determine a faster completion time, in the case of parallel or distributed implementation, because a smaller number of iterations would be necessary to produce a solution of the same quality.

5.3 SPARROW-SNN results

For the experiments a synthetic dataset and two real life datasets were used, as described in subsection 5.1.

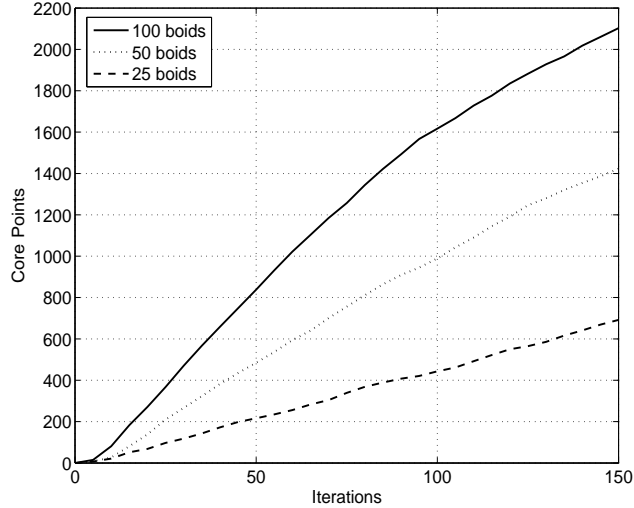


Fig. 9. The impact of the number of agents on the foraging for clusters strategy (DS4).

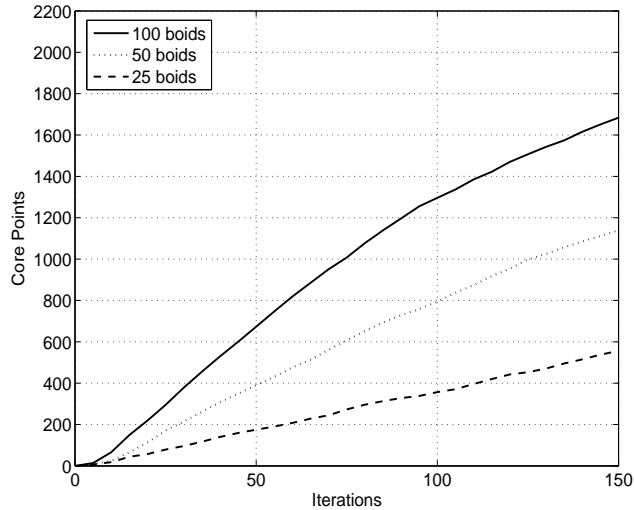


Fig. 10. The impact of the number of agents on the foraging for clusters strategy (Sequoia).

Noise_threshold and core_threshold were set respectively at 5 and 10 for the DS1 dataset and at 10 and 20 for Sequoia and the North East. K and P_{min} were set respectively at 30 and 0.8 for all the datasets.

The accuracy of the SPARROW-SNN algorithm is firstly illustrated in comparison with the SNN algorithm when SPARROW-SNN is used as a technique for approximate clustering. To this purpose, SNN was implemented and the number of clusters and the number of points for cluster were computed for the above described datasets. Tables 5, 6 and 7 present a comparison of these results with respect to ones obtained from SPARROW-SNN when a population of 50 agents has visited respectively 7%, 12% and 22% and 70% of the entire dataset.

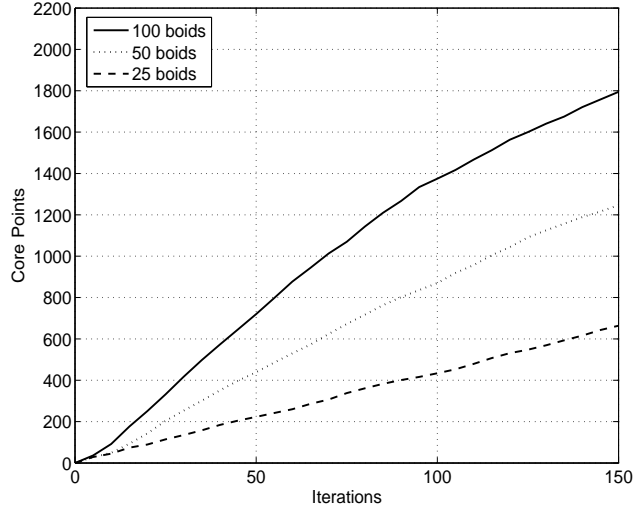


Fig. 11. The impact of the number of agents on the foraging for clusters strategy (North-East).

| | Perc. of data points for cluster | | | |
|---|----------------------------------|---------|---------|---------|
| | 7% | 12% | 22% | 70% |
| 1 | 41.35 % | 58.31 % | 70.37 % | 98.25 % |
| 2 | 30.08 % | 53.58% | 60.72 % | 97.02 % |
| 3 | 29.28 % | 40.99 % | 53.02 % | 91.10 % |
| 4 | 20.93 % | 30.51 % | 51.41 % | 89.65 % |
| 5 | 53.38 % | 65.36 % | 76.56% | 99.46 % |
| 6 | 56.89 % | 69.87 % | 73.63 % | 98.90 % |
| 7 | 33.89 % | 43.5 % | 61.58 % | 99.59 % |

Table 5

Number of clusters and number of points for clusters for the DS1 data set (percentage in comparison to the total point for cluster found by SNN) when SPARROW-SNN analyzes 7%, 12%, 22% and 70% points.

Note that with only 7% of points a clear vision is obtained of the found clusters and with a few more points a good percentage representative of all the clusters is achieved. This trend is well-marked in the North-East dataset. For the DS1 dataset, the results are not so well-defined, because the clusters called number 3 and 4 have very few points, so they are very hard to discover. For the real datasets, only the results for the three main clusters representing respectively the towns of Boston, New York and Philadelphia (Sequoia) and the regions of S. Francisco, Sacramento and Los Angeles (North-East) were reported.

Furthermore, for the two real datasets (i.e. Sequoia and North-East), a two-tailed paired t-test ($\alpha=0.05$) was performed for the sake of comparison be-

| | Perc. of data points for cluster | | | |
|--------------|----------------------------------|----------------|----------------|----------------|
| | 7% | 12% | 22% | 70% |
| Philadelphia | 42.48 % | 65.22 % | 79.44 % | 99.65 % |
| New York | 38.71 % | 52.27 % | 67.56 % | 96.61 % |
| Boston | 46.52 % | 68.61 % | 82.29 % | 99.94 % |

Table 6

Number of clusters and number of points for clusters for the North-East data set (percentage in comparison to the total point for cluster found by SNN) when SPARROW-SNN analyzes 7%, 12%, 22% and 70% points.

| | Perc. of data points for cluster | | | |
|--------------|----------------------------------|---------------|---------------|---------------|
| | 7% | 12% | 22% | 70% |
| S. Francisco | 49.11% | 67.75% | 80.38% | 99.55% |
| Sacramento | 44.81% | 61.72% | 80.86% | 98.22% |
| Los Angeles | 51.03% | 71.99% | 84.68% | 98.86% |

Table 7

Number of clusters and number of points for clusters for the Sequoia data set (percentage in comparison to the total point for cluster found by SNN) when SPARROW-SNN analyzes 7%, 12%, 22% and 70% points.

tween Sparrow and Sparrow-SNN. The values in bold reported in table 6 and 7 highlight the cases in which Sparrow-SNN outperforms SPARROW, i.e. SPARROW-SNN obtains an error (meaningful with respect to the statistical test) lower than that obtained by SPARROW. Note that, for the North-East dataset, the SNN version outperforms the other version for almost all the cases. This is mainly due to the different densities present in the dataset hardly detectable from SPARROW.

Finally, in order to study the scalability of our approach, the impact of the number of agents on the foraging for clusters performance is presented. Figures 12, 13 and 14 give, respectively for the DS1, North-East and Sequoia dataset, the number of clusters found in 100 time steps (iterations) for 25, 50 and 100 agents. A comparative analysis reveals that a 100-agents population discovers a larger number of clusters than the other two populations with a smaller number of agents and the trend is almost linear. This scalable behavior determine a faster completion time, in the case of parallel or distributed implementation, because a smaller number of iterations is necessary to produce a solution of the same quality.

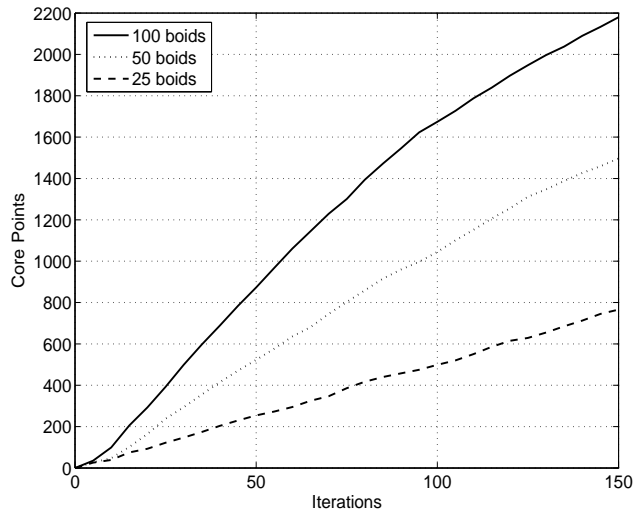


Fig. 12. The impact of the number of agents on the foraging for clusters strategy (DS1).

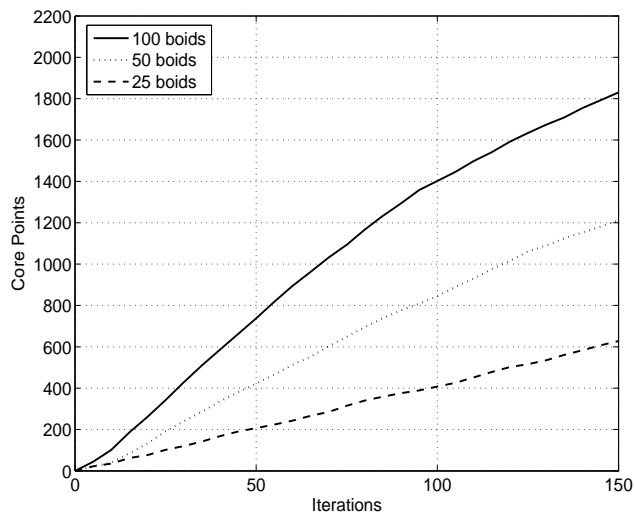


Fig. 13. The impact of the number of agents on the foraging for clusters strategy (North-East).

5.4 Comparison with a hybrid clustering technique

Kollios et al. in [19] adopted a hybrid method for performing approximated clustering, presenting interesting analogies with our approach. In order to cope with large datasets, they replaced the random sampling technique adopted by the CURE algorithm [9] with a biased sampling based on the density of the dataset. In fact, first the biased sampling is applied on the original dataset for improving the quality of the obtained clusters and then CURE is applied to the sampled points for merging the clusters. Indeed, even our method performs a kind of biased sampling by means of the flocking algorithm and then a traditional merging technique is applied (i.e. DBSCAN or SNN). For the sake

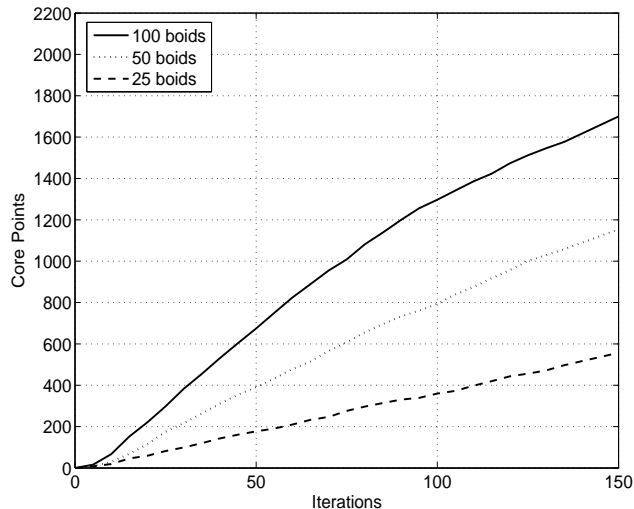


Fig. 14. The impact of the number of agents on the foraging for clusters strategy (Sequoia).

of comparison, we show the results of the phase of sampling performed by our flock, and from the random sample and the biased sample used in the previously mentioned algorithm.

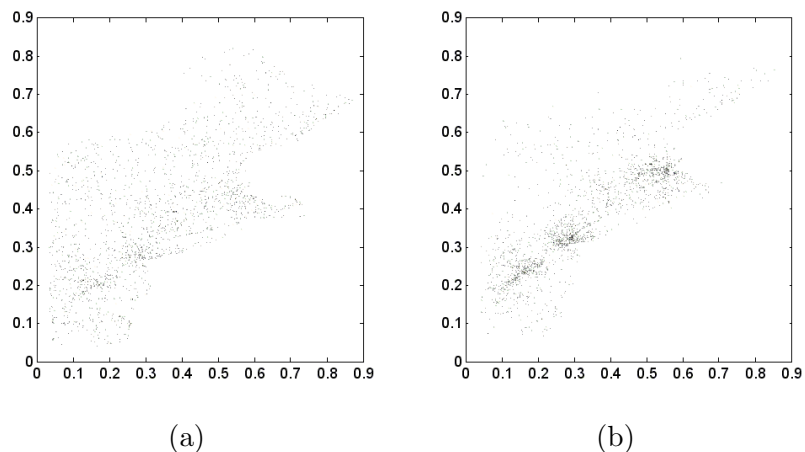


Fig. 15. Comparison of performance of the algorithm for the North-East dataset using a) a random sample of 1000 points. b) A biased sample adopted in the paper [19]

Figures 15 a and b show respectively the results concerning a random sample and a biased sample of 1000 points for the North-East dataset. On figure 16 a and b, they are shown the clusters obtained for the two samples using CURE with 10 representative points for each cluster. Clearly the biased sample permits to detect the correct clusters, while the uniform sample fails to handle the different density of the clusters. Figures 15 and 16 are taken from [19] and in the same paper is shown that the uniform sample fails even if 4000 points are used. Using our clustering algorithm, we can not fix the number

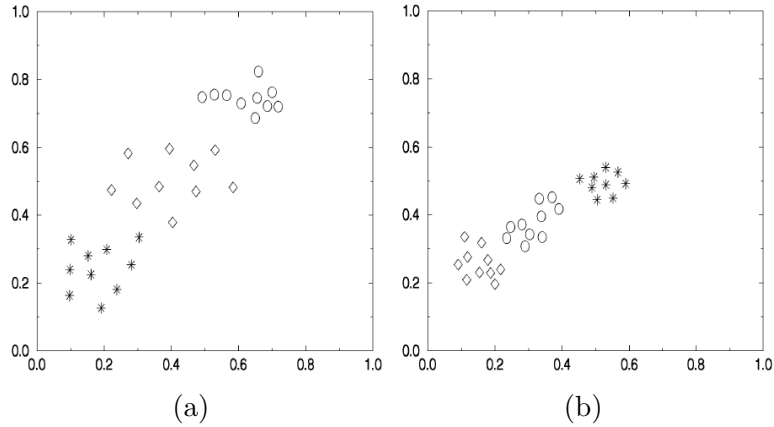


Fig. 16. Comparison of performance of the algorithm for the North-East dataset using a) Hierarchical clustering of 15 a using 10 representative points. b) Hierarchical clustering of 15 b using 10 representative points.

of representative points, so, in figure 17, we show the results concerning our biased sample and the center (illustrated with a circle) of the clusters discovered from SPARROW using a) 1000 b)2000 and c)4000 points. In the boxes are shown the representative points discovered for each cluster. Analogue results were obtained for the Sequoia dataset and they are not shown here for brevity. Effectively, even our algorithm is able to identify the correct cluster exploring only 1000 points and, sampling more points, incrementally improve the quality of the clusters, differently from the uniform random sample and with results comparable to the biased sample performed by the algorithm of Kollios et al..

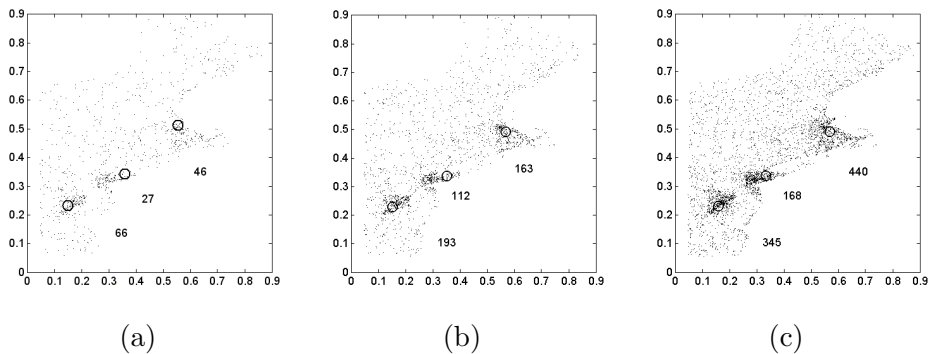


Fig. 17. Comparison of performance of SPARROW for the North-East dataset using a) 1000 points. b) 2000 points c) 4000 points.

5.5 The Noise

To better understand how our algorithm works, we analyzed its behavior in presence of noise points. In figure 18 a and b, they are represented points

belonging to clusters and noise points vs number of visited points for the Sparrow algorithm, respectively for the Sequoia and the North-East dataset.

An interesting property of our strategy consists in finding more points belonging to clusters than noise points. In fact, if we observe the figures, we can note that until the 30% of visited points for North-East (and about 20% for Sequoia), Sparrow is able to find a large percentage of cluster points, about 80% of the total. After this threshold, a few cluster points are discovered and, on the contrary, many noise points are found. So, it is not so convenient go on searching beyond this threshold value, unless you want to estimate more accurately the percentage of noise points.

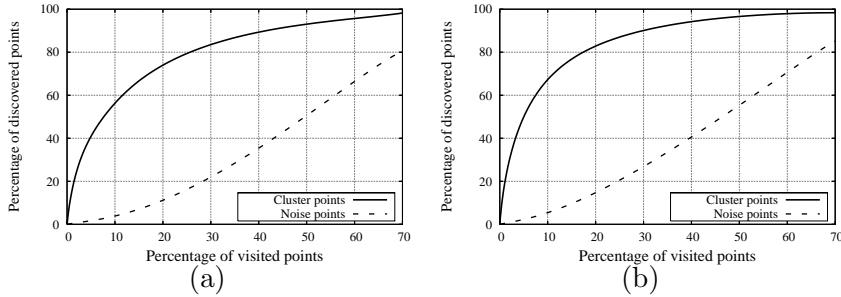


Fig. 18. Number of noise and cluster points found for SPARROW vs. number of visited points for a) the SEQUOIA dataset. b) the North-East dataset

5.6 Final remarks

Experimental results showed the efficacy of the two algorithms in performing approximate clustering. SPARROW is based on a density approach (DBSCAN) and therefore not appropriate for discovering clusters with different densities, while SPARROW-SNN can cope with this problem. In fact, the latter is able to discover clusters in the DS4 dataset, presenting cluster having different densities. Furthermore, SPARROW-SNN performs better in terms of accuracy, as showed for the North-East and Sequoia real datasets, where dense clusters are not easy to discover, as high density data must be discriminated from quite dense noise zones.

6 Parallel/Distributed implementation issues

The flocking-based clustering algorithm presented is naturally adapt to be implemented on parallel/distributed architectures. Two different methodologies could be taken into account, decomposing the search space or using a distributed flock and maintaining the data on a central server. Obviously, the

best approach is to distribute the data (search space) among different processors as using a central server for maintaining the data could be a serious bottleneck both for containing a large dataset in the memory of a single node and for the continuous access to this node for checking the data. In fact, as for a computational point of view, the overhead of the flock movement is negligible in comparison with the execution of multiple queries on a large dataset.

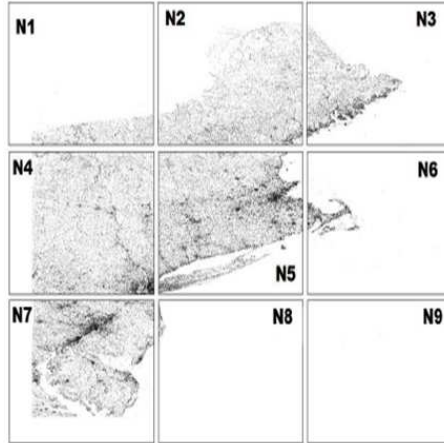


Fig. 19. Search space decomposed among 9 processors (using original spatial position).

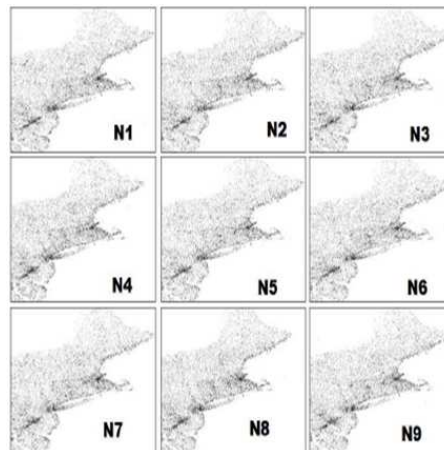


Fig. 20. Search space decomposed among 9 processors (randomly sampling).

Thus, using this methodology, the distribution of the original data among different nodes can be undertaken mainly using two different approaches, each one having different advantages and drawbacks. For the sake of clearness, we take as an example the dataset North-East (see figure 7). Spatial data can be decomposed in accordance to their original spatial position (for instance, see the simple distribution of figure 19) or they can be randomly sampled and distributed over nodes (see figure 20). Using the former approach, load balancing issues can be quite complicated. In fact, for instance in figure 19, node 9 does not work at all and nodes 1, 6 and 8 works on a very limited portion of data. To obtain a better load balancing, choosing more complicated

decomposition would help, but that would require a deeper knowledge of the domain of data and would bring to a more complex communication pattern among neighboring nodes, increasing the communication overhead. Anyway, in situations in which for reasons of privacy or because data are too large to be moved or because data are already logically and physically distributed, this solution should be strongly taken into account. For instance, imagine the scenario, in which huge amounts of data are stored in autonomous, geographically distributed sources over networks with limited bandwidth and large number of computational resources.

On the contrary, using the latter approach, the load would be certainly well balanced, but other problems could raise. First of all, it is necessary to choose the right threshold (i.e. the right number of nodes on which divide the data), otherwise, by dividing data among the nodes, some clusters with a low density could be not discovered. Furthermore, while in the previous approach, only information on the borders must be exchanged among the neighboring nodes (i.e. birds moving from a node to another or representative points placed on the border between two nodes), in this case, it is present a much higher communication overhead. In fact, the different population of flocks placed on the different nodes could act independently, but, in order to merge the clusters, representative points must be massively exchanged among the nodes.

7 Conclusions

This paper describes an adaptive flocking algorithm and its application to the problem of clustering spatial data. The approach is based on the use of swarm intelligence techniques. Two novel algorithms that combine density-based and shared nearest neighbor clustering strategy with a flocking algorithm are presented. An adaptive classical flock algorithm is formalized and optimized and is combined with state of the art clustering algorithms as DBSCAN and SNN, in order to perform approximate clusters. In this way, our approach presents interesting characteristics such as decentralization, fault tolerance and good performance in clustering spatial data of different size, shape and densities, also in the presence of noise. The algorithms were implemented in SWARM and evaluated using synthetic and real word datasets. Measures of accuracy of the results show that the flocking algorithm can be efficiently applied as a data reduction strategy to perform approximate clustering. Moreover, the algorithm presents a good scalable behavior and it is suitable for distributed implementations. Note that the two implementations present the same advantages and the same drawback as the DBSCAN and SNN methods, however the basic methodology presented here could be also used to implement other different clustering algorithms.

References

- [1] Eric Bonabeau, Marco Dorigo, and Guy Theraulaz. *Swarm intelligence: from natural to artificial systems*. Oxford University Press, New York, NY, USA, 1999.
- [2] Xiaohui Cui, Jinzhu Gao, and Thomas E. Potok. A flocking based algorithm for document clustering analysis. *J. Syst. Archit.*, 52(8):505–515, 2006.
- [3] J. L. Deneubourg, S. Goss, N. Franks, Sendova-Franks, A. C. Detrain, and L. Chretien. The dynamics of collective sorting robot-like ants and ant-like robots. In *From Animals to Animats: Proc. of the 1st Int. Conf. on Simulation of Adaptive Behaviour*. MIT Press/Bradford Books, 1990.
- [4] Issmail Ellabib, Paul Calamai, and Otman Basir. Exchange strategies for multiple ant colony system. *Inf. Sci.*, 177(5):1248–1264, 2007.
- [5] L. Ertoz, M. Steinbach, and V. Kumar. A new shared nearest neighbor clustering algorithm and its applications. In *Workshop on Clustering High Dimensional Data and its Applications at 2nd SIAM International Conference on Data Mining*, pages 105–115, 2002.
- [6] Martin Ester, Hans-Peter Kriegel, Jorg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proc. 2nd Int. Conf. on Knowledge Discovery and Data Mining*, pages 226–231, 1996.
- [7] Ashish Ghosh, Anindya Halder, Megha Kothari, and Susmita Ghosh. Aggregation pheromone density based data clustering. *Inf. Sci.*, 178(13):2816–2831, 2008.
- [8] P.P. Grass. *La Reconstruction du nid et les Coordinations Inter-Individuelles chez Beellicositermes Natalensis et Cubitermes sp. La Thorie de la Stigmergie : Essai d'interprétation du Comportement des Termites Constructeurs in Insect. Soc. 6*. Morgan Kaufmann, 1959.
- [9] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. CURE: an efficient clustering algorithm for large databases. In *ACM SIGMOD International Conference on Management of Data*, pages 73–84, June 1998.
- [10] J. Han, M. Kamber, and a. K. H. Tung. *Spatial Clustering Methods in Data Mining: A Survey, Geographic Data Mining and Knowledge Discovery*. Morgan Kaufmann, 2001.
- [11] Jiawei Han and Micheline Kamber. *Data Mining: Concepts and Techniques (The Morgan Kaufmann Series in Data Management Systems)*. Morgan Kaufmann, September 2000.
- [12] J. Handl, J. Knowles, and M. Dorigo. Ant-based clustering and topographic mapping. *Artificial Life*, 12(1):35–61, 2006.

- [13] R. A. Jarvis and E. A. Patrick. Clustering using a similarity measure based on shared nearest neighbors. *IEEE Transactions on Computers*, C-22(11), 1973.
- [14] P.M. Kanade and L.O. Hall. Fuzzy ant clustering by centroid positioning. In *IEEE International Conference on Fuzzy Systems*, pages 371–376, Budapest, 2004. IEEE Computer Society.
- [15] George Karypis, Eui-Hong Han, and Vipin Kumar. Chameleon: Hierarchical clustering using dynamic modeling. *IEEE Computer*, 32(8):68–75, 1999.
- [16] L. Kaufman and P. J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. John Wiley, 1990.
- [17] Majid Kazemian, Yoosef Ramezani, Caro Lucas, and Behzad Moshiri. Swarm clustering based on flowers pollination by artificial bees. In *Swarm Intelligence in Data Mining*, pages 191–202. 2006.
- [18] J. Kennedy and R. C. Eberhart. Particle swarm optimization. In *Proceedings of the IEEE International Conference on Neural Networks*, pages 1942–1948, Perth, Australia, 1995. IEEE Service Center.
- [19] George Kollios, Dimitrios Gunopulos, Nick Koudas, and Stefan Berchtold. Efficient biased sampling for approximate clustering and outlier detection in large data sets. *IEEE Transactions on Knowledge and Data Engineering*, 15(5):1170–1187, 2003.
- [20] Pascale Kuntz and Dominique Snyers. Emergent colonization and graph partitioning. In *SAB94: Proceedings of the third international conference on Simulation of adaptive behavior : from animals to animats 3*, pages 494–500, Cambridge, MA, USA, 1994. MIT Press.
- [21] E. D. Lumer and B. Faieta. Diversity and adaptation in populations of clustering ants. In *From Animals to Animats: Proc. of the Third Int. Conf. on Simulation of Adaptive Behaviour*, pages 501–508. MIT Press, 1994.
- [22] James Macgill. Using flocks to drive a geographical analysis engine. In M. A. Bedau J. S. McCaskill N. H. Packard S. Rasmussen, editor, *Artificial Life VII: Proceedings of the Seventh International Conference on Artificial Life*, pages 446–453. The MIT Press, Cambridge, Massachusetts, 2000.
- [23] N. Minar, R. Burkhart, C. Langton, and M. Askenazi. The swarm simulation system, a toolkit for building multi-agent simulations, 1996.
- [24] Nicolas Monmarché, M. Slimane, and Gilles Venturini. On improving clustering in numerical databases with artificial ants. In *ECAL '99: Proceedings of the 5th European Conference on Advances in Artificial Life*, pages 626–635, London, UK, 1999. Springer-Verlag.
- [25] S. Openshaw and J. Macgill. The use of flocks to drive a geographical analysis machine. In *GeoComputation 98, Proceedings of the 3rd International Conference on GeoComputation*, Bristol, United Kingdom, September 1998.

- [26] Fabien Picarougne, Hanene Azzag, Gilles Venturini, and Christiane Guinot. A new approach of data clustering using a flock of agents. *Evol. Comput.*, 15(3):345–367, 2007.
- [27] V. Ramos, F. Muge, and P. Pina. Self-organized data and image retrieval as a consequence of inter-dynamic synergistic relationships in artificial ant colonies. In Ajith Abraham, Javier Ruiz-del-Solar, and Mario Köppen, editors, *Soft Computing Systems - Design, Management and Applications*, Frontiers in Artificial Intelligence and Applications Vol. 87, pages 500–512. IOS Press Amsterdam, Berlin, Oxford, Tokyo, Washington D.C., 2002.
- [28] Vitorino Ramos and Ajith Abraham. Swarms on continuous data. In *CEC03 - Congress on Evolutionary Computation*, pages 1370–1375, Canberra, Australia, December 2003. IEEE Computer Society.
- [29] Vitorino Ramos and Juan J. Merelo. Self-organized stigmergic document maps: Environment as mechanism for context learning. In *AEB 2002 - 1st Spanish Conference on Evolutionary and Bio-Inspired Algorithms*, pages 284–293, Mrida, Spain, September 2002.
- [30] Craig W. Reynolds. Flocks, herds and schools: A distributed behavioral model. In *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pages 25–34, New York, NY, USA, 1987. ACM Press.
- [31] Jorg Sander, Martin Ester, Hans-Peter Kriegel, and Xiaowei Xu. Density-based clustering in spatial databases: The algorithm gbscan and its applications. *Data Min. Knowl. Discov.*, 2(2):169–194, 1998.
- [32] D. W. Van der Merwe and A. P. Engelbrecht. Data clustering using particle swarm optimization. In Ruhul Sarker, Robert Reynolds, Hussein Abbass, Kay Chen Tan, Bob McKay, Daryl Essam, and Tom Gedeon, editors, *Proceedings of the 2003 Congress on Evolutionary Computation CEC2003*, pages 215–220, Canberra, 8-12 December 2003. IEEE Press.
- [33] André L. Vizine, Leandro Nunes de Castro, Eduardo R. Hruschka, and Ricardo R. Gudwin. Towards improving clustering ants: An adaptive ant clustering algorithm. *Informatika (Slovenia)*, 29(2):143–154, 2005.
- [34] Wei Wang, Jiong Yang, and Richard R. Muntz. Sting: A statistical information grid approach to spatial data mining. In *VLDB'97, Proceedings of 23rd International Conference on Very Large Data Bases, August 25-29, 1997, Athens, Greece*, pages 186–195. Morgan Kaufmann, 1997.