Auto-Clustering Using Particle Swarm Optimization and Bacterial Foraging

Jakob R. Olesen, Jorge Cordero H., and Yifeng Zeng

Department of Computer Science, Aalborg University, Selma Lagerlfs Vej 300 Aalborg 9220, Denmark
jpcordero@exatec.itesm.mx, yfzeng@cs.aau.dk, jro@cs.aau.dk
http://www.cs.aau.dk/∼yfzeng/

Abstract. This paper presents a hybrid approach for clustering based on particle swarm optimization (PSO) and bacteria foraging algorithms (BFA). The new method *AutoCPB* (Auto-Clustering based on particle bacterial foraging) makes use of autonomous agents whose primary objective is to cluster chunks of data by using simplistic collaboration. Inspired by the advances in clustering using particle swarm optimization, we suggest further improvements. Moreover, we gathered standard benchmark datasets and compared our new approach against the standard *K-means* algorithm, obtaining promising results. Our hybrid mechanism outperforms earlier PSO-based approaches by using simplistic communication between agents.

1 Introduction

How can we define a group of highly correlated genes while examining gene expression data? How can we find those relevant features in a dataset having numerous variables? How could we define a set of classes over a collection of observations? Basically, all these questions have been the fundamental motivation for one of the most popular branches of data mining: data clustering.

Clustering is one of the most important unsupervised learning problems that computer scientists, statisticians and mathematicians have tried to develop and improve for years. For more than two decades, clustering has taken special interest between the scientific community and it is probably in the jargon of other professionals that have no direct relationship with machine learning or even computer science. The reason for the later relies beneath its clear definition and interpretation.

The aim of data clustering is to recognize patterns in data and form groups (clusters C) of interdependent objects. According to [1], this task can be classified in four major paradigms:

- Model based methods.
- Hierarchical methods.
- Partitioning methods.
- Density estimation methods.

Besides of the previous classification, clustering approaches can also be independently classified in other terms. They can be either exhaustive or not exhaustive methods (exhaustive clustering associates every object to a given cluster, whereas in the second case some variables could not be included in any cluster at all). Moreover, a given clustering algorithm can produce disjoint or overlapping clusters.

In this work we focus on data clustering (exhaustive and disjoint) utilizing pairwise Euclidean distance between points in a m dimensional vectorial space.

Specifically, the objective is to partition a dataset $D = \{d_i | i = 1, \cdots, n\}$ with n records into a set of $k = |C|$ clusters according the following constraints: Every $d \in D$ has to be assigned to a cluster $C_i \in C$ such that $\forall_{C_i \in C}\ C_i \neq \emptyset$ and $\forall_{C_i, C_j \in C}\ C_i \cap C_j = \emptyset$. In this paper, we compare *AutoCPB* with the *K-means* [1] algorithm due to its popularity and robustness.

Several machine learning techniques have been applied to solve the problem of clustering. For example in [2], a neural network clusters data using entropy estimates. In [3], a genetic algorithm is combined with Nelder-Mead simplex search in order to produce a hybrid clustering algorithm. On the other hand, self organizing maps [4] have also been used for partitioning a dataset without specifying the number of clusters. However, much effort has not been dedicated to study evolutionary collaborative scheme for clustering. We propose an extension of the elemental particle swarm clustering algorithm for clustering.

Multiagent systems are used for simulating complex environments. In this paper, we propose a swarm intelligence clustering algorithm which takes advantage of simplistic communication and evolutionary methods in agents. Particle swarm optimization [5] is a class of evolutionary algorithms which aims to find a solution to a given optimization problem. Bacterial foraging algorithms [6] are a new paradigm in searching based on behavior of biological systems. In this paper we extend the advantages of social influence in PSO with the influence of bacterial foraging behavior.

The rest of this paper is organized as follows: Section 2 introduces background materials related to PSO and BFA. Section 3 describes our novel approach in detail. Section 4 depicts experimental results and some implementation details. Finally, Section 5 concludes our discussion and provides interesting remarks for future work.

2 Background

The fundamental idea of swarm intelligence algorithms [7] is that a set of individuals can cooperate in a decentralized manner increasing their productivity. Thus, the aim is to find mechanisms that can model complex systems, and represent them in a formal way [8].

2.1 Particle Swarm Optimization Exposed

Particle swarm optimization is a form of stochastic optimization based on swarms of simplistic, social agents [5]. Primary algorithms of particle swarm optimization perform search over a m dimensional space U by using a set of agents. In this lattice, an agent (particle) i occupy a position $x_i(t) = \{x_{i,j}(t) | j = 1, \cdots, m\}$ and has a velocity $v_i(t) = \{v_{i,j}(t) | j = 1, \cdots, m\}$ in an instant t, with a 1:1 correspondence (both $x_i(t)$ and $v_i(t)$ contain a set of components $\{j = 1, \cdots, m\}$ mapped to coordinates in U).

A simple PSO algorithm [7] works as follows: In the initialization phase, every agent takes positions around $x(0) = x_{min} + r(x_{max} - x_{min})$ and the velocities are set to 0 (x_{min}

and x_{max} are the minimal and maximal magnitudes in U and r is a real number between 0 and 1). Secondly, the algorithm enters in the search phase. The search phase consists of the following steps: Best cognitive/global position updating and velocity/position updating.

In the cognitive updating step every agent sets a value for the current (local) best position $y_i(t)$ that it has directly observed. The agent's local optima $y_i(t+1)$ is updated according to equation 1:

$$y_i(t+1) = \begin{cases} y_i(t) & \text{if } f(x_i(t+1)) \geq f(y_i(t)) \\ x_i(t+1) & \text{otherwise,} \end{cases} \tag{1}$$

whereas $f(x_i(t))$ is a fitness function which evaluates the goodness of a solution based on position $x_i(t)$.

In contrast, the global updating step sets in each iteration the best possible position Y_i observed by any agent. Equation 2 depicts the selection of the best global position.

$$Y_i(t+1) = \begin{cases} Y_i(t) & \text{if } f(y_i(t)) \geq f(Y_i(t)), \forall y_i(t) \\ y_i(t) & \text{if } \exists y_i(t) | f(y_i(t)) < f(Y_i(t)), \end{cases} \tag{2}$$

The velocity/position updating step selects new values for the position $x_i(t+1)$ and velocity $v_i(t+1)$ using equations 3 and 4 respectively:

$$x_i(t+1) = x_i(t) + v_i(t+1), \tag{3}$$

$$v_i(t+1) = v_i(t) + \underbrace{c_1 r_1 (y_i(t) - x_i(t))}_{\textbf{Cognitive component}} + \underbrace{c_2 r_2 (Y_i(t) - x_i(t))}_{\textbf{Global component}}, \tag{4}$$

where the c_1 and c_2 parameters are used to guide the search between local (cognitive component) and social (global component) observations. $r_1, r_2 \in [0,1]$ are random parameters which introduce a stochastic weight in the search. Finally, the algorithm stops until some convergence point is reached.

A suggested convergence test is proposed in [9]; it consists in testing whether $(f(Y_i(t)) - f(Y_i(t-1)))/f(Y_i(t))$ is smaller than a small constant ε for a given number of iterations. It is important to note that each particle in this multiagent system shares its knowledge (global best position) with all other particles by means of a neighborhood topology.

Several topologies have been proposed [7] (i.e. star, ring, clusters, Von Neumann, etc.). The difference between neighborhoods lies in how fast or slow (depending on connectivity) knowledge propagates through the swarm.

Further improvements have been proposed for the basic PSO algorithm [7,9]: Velocity clamping, inertia weight and constriction coefficient. All PSO-based algorithms in this study were implemented using the constriction coefficient (the most robust mechanism).

Velocity Clamping. A weakness of basic particle swarm optimization algorithms is that the velocity rapidly increases to unsuitable values. High velocity results in large position updates (in this case the particles may even leave the boundaries of the search space). This is specially true for particles occupying outlier positions. Velocity clamping is a simple restriction that imposes a maximal velocity. Equation 5 depicts this rule.

$$v_{i,j}(t+1) = \begin{cases} v_i'(t+1) & \text{if } v_i'(t+1) < V_{max} \\ V_{max} & \text{if } v_i'(t+1) \geq V_{max} \end{cases} \tag{5}$$

Obviously, large values of V_{max} facilitate exploration (small ones favor exploitation). V_{max} is frequently a fraction δ of the domain space for each dimension j in the search space U. Equation 6 presents this adjustment.

$$V_{max_j} = \delta(x_{max_j} - x_{min_j}), \tag{6}$$

whereas x_{max_j} and x_{min_j} represent the maximal and minimal magnitudes respectively.

Inertia Weight. The inertia weight is a constraint which aims to control the trade off between exploration and exploitation in a more direct fashion. Equation 7 introduces the inertia weight w that affects velocity updating.

$$v_i(t+1) = wv_i(t) + c_1r_1(y_i(t) - x_i(t)) + c_2r_2(Y_i(t) - x_i(t)) \tag{7}$$

$w \geq 1$ produces velocity increments over time (exploration), $w < 1$ decreases velocities over time (favoring exploitation). According to [7], the values $w = 0.7298$, $c_1 = c_2 = 1.49618$ have proved good results empirically, although in many cases they are problem dependent.

Constriction Coefficient. A similar method to inertia weight was proposed in [10], it is denominated the constriction coefficient. Equation 8 defines the new velocity updating mechanism.

$$v_i(t+1) = \chi(v_i(t) + \phi_1(y_i(t) - x_i(t)) + \phi_2(Y_i(t) - x_i(t))), \tag{8}$$

where $\phi = \phi_1 + \phi_2$, $\phi_1 = c_1r_1$, $\phi_2 = c_2r_2$ and

$$\chi = \frac{2k}{|2 - \phi - \sqrt{\phi^2 - 4\phi}|} \tag{9}$$

It is important to notice that $\phi \geq 4$ and $k \in [0,1]$ are necessary constraints to set χ in the range [0,1].

2.2 Bacterial Foraging Algorithms

Bacterial foraging algorithms are a new class of stochastic global search techniques [6]. Such algorithms emulate the foraging behavior of bacteria while situated in some nutrient substance. During foraging, a bacterium can exhibit two different actions: Tumbling or swimming.

The tumble action modifies the orientation of the bacterium. During swimming (chemotactic step) the bacterium will move in its current direction. After tumbling, the bacterium checks if it can find nutrients in its current direction, and if it can, then it will swim for a finite number of steps in that direction. After the bacterium has collected a given amount of nutrient, it will divide in two. The environment can also act in the bacteria population by eliminating or dispersing them.

A bacterial foraging algorithm can be defined as follows. Given a m dimensional search space U, each bacterium i has a position $x_i(t) = (x_{i,j}(t)|j = 1,\cdots,m)$ with m components at time t. It also has a chemotactic step size $C(i) > 0$ which influences the bacterium's step length. Initially the Bacteria is situated in several points in U. Then, each bacterium generates a random tumbling vector b_i consisting of m components. Following, the agent will enter into the swimming phase. Therefore, it will update its position one step at a time according to equation 10.

$$x_i(t+1) = x_i(t) + C(i) * b_i \tag{10}$$

swimming will continue for a number N_a of iterations iff $f(x_i(t+1)) < f(x_i(t))$ holds. Once that all agents finished tumbling and swimming, they reproduce. New generations are created only with the half of the *healthiest* agents. *Healthy* agents can be interpreted as the ones which performed the smallest number of chemotactic steps. Finally, an elimination/dispersal step deletes and reallocates a percentage of agents at random. The algorithm will run for a fixed number of iterations.

A simple improvement for tumbling/swimming is to make bacteria attract each other in some area; and then repeal each other as they consume nearby nutrients. The idea is to calculate the cell to cell fitness, and add it to the fitness position for each bacterium.

3 PSO/BFA Multiagent Clustering

Previously, we presented the theoretical foundations necessary in which the novel *AutoCPB* relies on. In this section we introduce in detail our multiagent system for clustering.

One of the main problems related with PSO algorithms is that they have a tendency to fall into suboptimal solutions, because of the lack of diversity in the swarm. One of our ambitions is to find a mechanism to improve the diversity.

An improvement of the original PSO algorithm [11] is to use either a memetic approach, or hybridizing it with another swarm intelligence method [12]. Given that *AutoCPB* is a hybrid heuristic algorithm, we opted to design preliminary prototype-algorithms. We start introducing a simplistic swarm-based clustering method and then we gradually present more elaborated developments.

3.1 PSO Clustering

We implemented a simple clustering algorithm denominated *ClusterP*, which was proposed in [11]. Such method combines PSO and *K-means* for grouping data.

The dataset $D = \{d_{1,m}, d_{2,m}, \cdots, d_{n,m}\}$ defines the number of components m and instances n in the search space U. Thus, each datum $d_{i,j} \in D$ can be seen as a point in

U. Since the aim is to find a set of k desired clusters C containing every element in D; it is easy to visualize that the main problem is to find the set of $O = \{o_1, o_2, \cdots, o_k\}$ centroids that minimize the fitness function (Euclidean distance) with respect to each point in D. In *clusterP*, every agent $p_l \in P$ represent a solution with the set of position $O_l = \{o_{l,j} | j = 1, \cdots, k\}$. Algorithm 3.1 shows the *ClusterP* technique.

Algorithm 3.1. The *ClusterP* Algorithm.

Input: Data $D = \{d_1, d_2, \cdots, d_n\}$, k.
Output: Clusters $C = \{C_1, C_2, \cdots, C_k\}$.

```
1: Initialize swarm $P$ (distribute $O$ ).
2: REPEAT:
3:     FOR $i = 1$ to $n$:
4:         FOR $j = 1$ to $k$:
5:             Calculate distances $dist(d_i, o_j)$.
6:         END FOR
7:         $C_h \Leftarrow d_i$ iff $argmin_{o_h \in O}(dist(d_i, o_h))$.
8:     END FOR
9:     FOR $l = 1$ to $|P|$:
10:        Update cognitive positions for $p_l$ using equation 1.
11:        Update global position for $P$ using equation 2.
12:    END FOR
13:    FOR $l = 1$ to $|P|$:
14:        Update velocities for $p_l$ using equation 4.
15:        Update positions for $p_l$ using equation 3.
16:    END FOR
17: UNTIL stopping condition holds.
```

ClusterP works as follows: It receives the data D and an integer k. First, it collocates every agent p in the environment, each containing a set of centroids O (line 1). Then, it assigns every record d_i to a cluster C_h iff the distance with its centroid $dist(d_i, o_h)$ is minimal (lines 3-8). Following, it obtains the cognitive and global position (lines 9-12), and updates every of its centroids velocities and positions (lines 13-16) as explained in Section 2.1. The algorithm stops after a fixed number of iterations or if no significant progress is made according to the fitness function.

3.2 Automatic PSO Clustering

ClusterP was extended in [13] in order to find the optimal number of clusters automatically. The new method is called *AutoCP*. It starts with a number k' of clusters ($k' \leq n$), and it deletes the *inconsistent* clusters.

We describe the form to detect inconsistent clusters as follows: First, for each cluster C_j we calculate its weight $W_j = \sum_{q=1}^{|C_j|} dist(o_j, d_q)$ as the sum of the distances from the center o_j to its points q_j where $q_j \neq o_j$. Immediately, all weights W from the clusters are sorted. Then, we normalize all weights W_j with respect to the cluster with lowest value W_s, such that the set of local thresholds become $th = \{th_j | j = 1,\cdots,k^{'}, th_j = W_s/W_j, W_s = argmin_{w \in W}(w)\}$. Finally, a cluster C_j is declared as *inconsistent* iff its threshold t_j is lower than the global threshold T, whereas T is in the range of [0,1]. Algorithm 3.2 introduces *AutoCP*.

Algorithm 3.2. The *AutoCP* Algorithm.

```
Input: Data $D = \{d_1, d_2, \cdots, d_n\}$.
Output: Clusters $C = \{C_1, C_2, \cdots, C_{k'}\}$.

1: Initialize swarm $P$ (distribute $O$).
2: REPEAT:
3:    Assign $D$ to clusters as in algorithm 3.1.
4:    Update cognitive/global positions as in algorithm 3.1.
5:    Update velocities/positions as in algorithm 3.1.
6:    FOR $j = 1$ to $k^{'}$:
7:        Calculate $W_j$ for each cluster $C_j$.
8:    END FOR
9:    Obtain $W_s = argmin_{W_j \in W}(W_j)$.
10:   FOR $j = 1$ to $k^{'}$:
11:       $th_j = W_s/W_j$.
12:       IF $(th_j < T)$
13:           Remove $C_j$ and $k^{'} = k^{'} - 1$.
14:       END IF
15:   END FOR
16: UNTIL stopping condition holds.
```

As previously mentioned, *AutoCP* simply adds a mechanism to *ClusterP* for automatic detection of clusters with no further modification (lines 6-15). At the end of each iteration, we select and discard inconsistent clusters.

3.3 AutoCB Clustering

In this section we present the method for automatic clustering using bacterial foraging algorithm *AutoCB*. It is an adapted version of the bacterial foraging algorithm for automatic clustering. Basically, it uses the *K-means* principle to add data to the closest centroids combined with a bacterial foraging search.

In algorithm 3.3, every agent observates a clustering solution represented by the positions of bacteria as centroids (notice that times are references to current $x_i(t)$ or posterior $x_i(t+1)$ positions in space. x_i in fact represent the tentative position for centroid o_i). In order to keep this algorithm as simple as possible, we decided not to add any complex initialization method. We simply exchanged the swimming mechanism to be executed firstly, followed by tumbling. In this way we use the fitness function to set up

Algorithm 3.3. The *AutoCB* Algorithm.

```
Input: Data $D = \{d_1, d_2, \cdots, d_n\}$.
Output: Clusters $C = \{C_1, C_2, \cdots, C_{k'}\}$.

1: Initialize $B$ (distribute $O$ randomly).
2: FOR Number of Elimination/Dispersal Steps
3:     FOR Number of Reproduction Steps
4:         FOR Number of Chemotactic Steps
5:             FOR Each Bacterium $i \in B$
6:                 WHILE $m < MaxSwimLength$
7:                     Delete inc. clusters as in algorithm 3.2.
8:                     Assign $D$ to $C$ as in algorithm 3.1.
9:                     IF $f(x_i(t+1), o_i) < f(x_i(t), o_i)$
10:                        $o_i = x_i(t+1)$.
11:                        $m = m+1$.
12:                    END IF
13:                    ELSE
14:                        $m = MaxSwimLength$.
15:                    END ELSE
16:                END WHILE
17:                $tumble(i)$, generate new $x_i(t+1)$.
18:            END FOR
19:        END FOR
20:        $reproduce(B)$.
21:    END FOR
22:    $eliminate(B), disperse(B)$.
23: END FOR
```

bacteria in good spots since the beginning. Initially, the algorithm distributes the positions for all centroids of every agent in B at random (line 1). For every bacterium i, the chemotactic part of the algorithm starts (lines 4-19): The agents will update their centroids positions for a maximal number of swims *MaxSwimLength* (lines 6-16). Firstly, the algorithm deletes inconsistent clusters as mentioned in lines 6-15 of algorithm 3.2 (line 7). Then, we assign each datapoint $d \in D$ to a cluster $C_j \in C$ according to the lines 3-8 of algorithm 3.1 (line 8).

Every agent observates the cells $x_i(t+1)$ in front of its centroids o_i. Then, they swim and update their centroids positions from o_i to $x_i(t+1)$ iff the Euclidean distance $f(x_i(t+1), o_i)$ to the new point is smaller than the current one $f(x_i(t), o_i)$ (lines 9-12). Swimming might immediately stop according to the control variable m in line 14; if the fitness of the tentative cell $f(x_i(t+1), o_i)$ is greater or equal to the one of the current position $f(x_i(t), o_i)$.

Once agent i has finished swimming, it will *tumble* (line 17). Thus, i will choose a tentative cell in a new direction $x_i(t+1)$ for each of its centroids $o_i = x_i(t)$. $x_i(t+1)$

is selected at random. Finally, the previous chemotactic process is encapsulated in a reproductive phase (line 3), and in a combined elimination/dispersion phase(line 2). The reproduction process deletes half of the agents having the smallest number of chemotactic steps. The elimination method destroys again a percentage α of agents at random. Dispersion reallocates a small percentage β of the remaining agents in U at random.

3.4 The AutoCPB Algorithm

The previous *AutoCB* algorithm is strictly based on the theory of bacterial foraging algorithms. However, we believe that we can improve its performance by sharpening the swimming and tumbling phase using a PSO-based algorithm.

In this section we describe in detail the hybrid algorithm for clustering based on PSO and BFA denominated *AutoCPB*. The algorithm follows the same structure of the AutoCB algorithm. Nevertheless, each bacterium agent i has assigned a position and a velocity to its centroids as previously seen in algorithm 3.1. The tumble is now guided by the swarm local and social beliefs.

A tentative cell $x_i(t+1)$ is deterministically chosen according to the best neighboring cell $y_i(t+1)$ and the global best position $Y_i(t+1)$. Swimming is still performed for a given number of iterations but now the agent's steps vary in dimension according to its velocity. *AutoCPB* contains a bacterial foraging skeleton and a particle swarm optimizator. In this algorithm, we simply replace tumbling in line 17 from algorithm 3.3 by a more elaborated PSO-based foraging mechanism (lines 18-24). Logically, swimming in lines 9-12 of algorithm 3.3 is also modified so we take advantage of the guided tumbling.

Specifically, the *AutoCPB* clustering algorithm works as follows: It collocates every centroid o_i for agent i in U at random ($x_i(t) = o_i$). Then, the modified tumbling/ swimming version is executed for all agents for a number of swims *MaxSwimLength* (lines 4-25): Inconsistent clusters are removed (line 7) as seen in algorithm 3.2. All datapoints $d \in D$ are assigned to the remaining clusters $C = \{C_j | j = 1, \cdots, k'\}$ (line 8). Then, all centroid positions $o_i = x_i(t)$ for agent i will be updated with the new position $x_i(t+1)$ iff $f(x_i(t+1)) < f(x_i(t))$ holds (lines 9-10). Otherwise, swimming stops in line 16. In this part of the algorithm, we swim by updating the value of $x_i(t+1)$ according equation 3 (line 11). We also record the best cognitive/local position $y_i(t+1)$ according to equation 1 for further tumbling computation (line 12). The final step of swimming is the update of the best global/social position $Y_i(t+1)$ (line 18).

In lines 21-24, PSO-based tumbling is executed. At this point, every centroid (with a velocity $v_i(t)$ and a position $x_i(t)$) starts observing its neighboring cells. It finally updates its own velocity to $v_i(t+1)$ and tentative position $x_i(t+1)$ by using equations 3 and 4 respectively. Notice that agents do not modify its current position during tumbling but only during the swimming phase (lines 9-14). Finally, once that we have clustered U into C, *AutoCPB* will reproduce (line 26), eliminate and disperse (line 28) agents in the same manner *AutoCB* does.

In the next part of this paper we comprehensively tested every approach obtaining promising results.

Algorithm 3.4. The *AutoCPB* Algorithm.

```
Input: Data $D = \{d_1, d_2, \cdots, d_n\}$.
Output: Clusters $C = \{C_1, C_2, \cdots, C_{k'}\}$.

1: Initialize $B$ (distribute $O$ randomly).
2: FOR Number of Elimination/Dispersal Steps
3:     FOR Number of Reproduction Steps
4:         FOR Number of Chemotactic Steps
5:             FOR Each agent $i \in B$
6:                 WHILE $m < MaxSwimLength$
7:                     Delete inconsistent clusters as in algorithm 3.2.
8:                     Assign $D$ to $C$ as in algorithm 3.1.
9:                     IF $f(x_i(t+1)) < f(x_i(t))$
10:                        $o_i = x_i(t+1)$.
11:                        Update $x_i(t+1)$ using equation 3.
12:                        Update $y_i(t+1)$ using equation 1.
13:                        $m = m+1$.
14:                    END IF
15:                    ELSE
16:                        $m = MaxSwimLength$.
17:                    END ELSE
18:                    Update $Y_i(t+1)$ using equation 2.
19:                END WHILE
20:            END FOR
21:            FOR Each agent $i \in B$
22:                Update $v_i(t+1)$ using equation 4.
23:                Update $x_i(t+1)$ using equation 3.
24:            END FOR
25:        END FOR
26:        $reproduce(B)$.
27:    END FOR
28:    $eliminate(B), disperse(B)$.
29: END FOR
```

4 Experimental Results

We proceed to analyze the performance of the aforementioned algorithms by testing them over well known benchmark datasets. Before we introduce the results, we describe the datasets, parameter settings and performed tests used for experimentation.

The algorithms were tested on nine datasets (D). Two domains were generated at random: Artificial1 (A1) and Artificial2 (A2). The rest of the datasets were taken from the UCI repository [14]: Iris (Ir), Wine (Wi), Pima (Pi), Haberman (Ha), BreastCancer (BC), Glass (Gl), and Yeast (Ye). Table 1 introduces the characteristics for each dataset in terms of its number of classes and dimensions.

Table 1. The datasets used for experimentation

D	Classes	Dimensions
A1	2	2
A2	3	3
Ir	3	4
Wi	3	13
Pi	2	8
Ha	2	3
BC	2	30
Gl	6	9
Ye	10	8

4.1 Environmental Settings

Each experiment is based on fifty consecutive runs of the same algorithm. For the *K-means* algorithm we set a maximal of 500 iterations (it terminates if no improvement is made). In the other algorithms, the assignation of points to clusters consists of a single iteration.

For *ClusterP* and *AutoCP* the stopping condition is the one described in Section 2.1. For *AutoCB* and *AutoCPB* we used 10 elimination/dispersal iterations, 20 reproduction steps and 20 chemotactic steps. In every method we used 30 agents. Every dataset has a predefined class. Therefore, we set k in *ClusterP* as the number of classes (in fact, we found that K-means obtains the best results in this fashion). For the multiagent-based automatic clustering algorithms we used 30 agents and an upper limit k' of 10 initial clusters.

4.2 Cluster Validation

According to [15] cluster validation can be divided into external, internal and relative validation. When using an external measure, we compare the solution with some optimal solution known a priori. Internal validation deals with judging the solution based on the intra-distance of a cluster. Relative measures compare the experimental clustering solution against another set of clusters previously found.

We used two validation measures, namely the inter cluster distance measure (ID) and the quantization error function (QEF). Both metrics have previously utilized to test cluster reliability [11]. The inter cluster distance calculates the average distances between the centroids and all their points in the cluster. It is calculated according to equation 11.

$$ID_C = \sum_{\forall o_i,o_j \in CEN, i \neq j} dist(o_i,o_j)/|C|, \tag{11}$$

where o_i and o_j are centroids, CEN is the set of all centroids, $|C|$ is the total number of clusters C and $dist()$ is the Euclidean distance. In essence, we calculate the average Euclidean distance between all pairs of centroids. The QEF is a global distance measure that evaluates the average distance from all datapoints to centroids in every cluster. Equation 12 present the QEF metric.

$$QEF_C = \Big(\sum_{j=1,k} \big(\sum_{\forall_{d_i \in C_j}} dist(d_i, o_j)/|C_j| \big) \Big)/k, \tag{12}$$

whereas k is the number of clusters, d_i is a datapoint contained in cluster C_j. All other variables are defined as in equation 11. The previous measures can be used to express the quality of a solution. Thus, we proceed to establish a discussion of the performance of the algorithms.

4.3 Benchmark Testing

Table 2 includes the average results for the inter cluster distance, quantization error function, number of clusters and the elapsed times (in milliseconds). Numbers in bold represent the best result for each dataset.

Even though, it can be reasoned as subjective to evaluate different clustering methods, we are confident that our test reflect some truth regarding the quality of the clustering. In fact, we believe that the ID and QEF tests express an acceptable comparison between clusterings for this investigation. Thus, every algorithm makes use of the *K-means* oriented mechanism for assigning points to clusters.

Specifically, every clustering method uses the **same** objective function. We add a datapoint to a cluster whose cluster centroid is the closest. The only difference between algorithms is their search mechanism, not contemplating any other characteristic for grouping. For all cases, we appreciate that *K-means* is the worst performing algorithm in terms of quality and the best in running time. Thus, every swarm based algorithm adds a refined search to *K-means*. However, in many cases we are more concerned in the quality of a solution. We can also improve the implementation of the algorithms in order to decrease the elapsed times.

We can conclude that with respect to the QEF metric; *AutoCP* and *AutoCPB* are the best algorithms, finding minimal values. *AutoCP* having a minimal QEF in A1, A2, Ir, Gl and Ye. *AutoCPB* is the most competitive in Ha, Pi, Wi, BC and Ye. However, both methods produce similar results. In every dataset any of them can be the first and the second most competitive method. This behavior is logical, and it is a common example of how the random element introduces some noise to a search technique. *AutoCPB* performs random steps (reproduction, elimination and dispersion).

For the ID metric we observe an almost identical trend. *AutoCB* outperforms all other methods in some cases (A1, A2). Indeed, *AutoCB* is the most randomized algorithm, it performs a nearest neighbor search with poor guidance but with a dynamical evolutionary mechanism. In every case, we can see that *ClusterP* is suboptimal and falls into local optima. The later is one of the reasons why the evolutionary method in the form of automatic clustering and bacterial foraging show a promising enhancement.

A common discussion in data mining is related with the selection of the *appropriate* number of clusters k for a given domain. A common assumption is that we should apply a given clustering technique depending on the particular problem/domain to analyze. However, with the advents in information technology and the analytical capabilities of

Table 2. Full clustering results. Swarm-based enhancements improve the performance of *K-means*

D	Method	QEF	ID	$\|C\|$	E. time
A1	*K-means*	11.64 +/-0.10	28.4 +/-0	2	**14.06 +/-5.75**
A1	*ClusterP*	11.99 +/-0.42	29.95 +/-2.03	2	1394.68 +/-18.7
A1	*AutoCP*	**4.91 +/-0.57**	4.13 +/-2.46	5.6 +1.01	2105.08 +/-427.96
A1	*AutoCB*	7.75 +/-1.18	**3.36 +/-2.31**	4.48 +/-1.27	8439.36 +/-423.15
A1	*AutoCPB*	6.2 +/-0.7	3.73 +/-2.51	4.8 +/-0.76	835.3 +/-74.6
A2	*K-means*	44.95 +/-33.56	28.89 +/-11.26	3	**95.3 +/-7.36**
A2	*ClusterP*	29.92 +/-2.27	35.62 +/-7.2	3	6914.04 +/-59.66
A2	*AutoCP*	**24.36 +/-1.13**	8.09 +/-3.46	4.9 +/-1.06	9446.92 +/-1376.23
A2	*AutoCB*	30.40 +/-3.07	**7.14 +/-3.87**	4.36 +/-1.03	11914.98 +/-269.82
A2	*AutoCPB*	24.45 +/-2.11	7.73 +/-3.94	4.22 +/-0.93	3658.74 +/-298.91
Ha	*K-means*	14.59 +/-6.69	8.85 +/-0.13	2	**135.94 +/-7.25**
Ha	*ClusterP*	10.23 +/-0.32	6.83 +/-1.96	2	13655.02 +/-171.18
Ha	*AutoCP*	9.18 +/-0.65	1.39 +/-0.84	3.26 +/-0.83	18916.88 +/-3962.37
Ha	*AutoCB*	12.41 +/-1.73	1.57 +/-0.82	2.98 +/-0.98	12055.92 +/-272.07
Ha	*AutoCPB*	**8.87 +/-0.76**	**1.2 +/-0.7**	3.14 +/-0.64	6930 +/-366.89
Pi	*K-means*	132.86 +/-46.72	111.77 +0	2	**742.2 +/-15.54**
Pi	*ClusterP*	73.85 +/-3.363	35.25 +/-11.43	2	72757.18 +/-1564.94
Pi	*AutoCP*	72.35 +/-3.29	11.43 +/-12.17	3.02 +/-1.12	107948.8 +/-20853.2
Pi	*AutoCB*	96.33 +/-12.68	12.06 +/-11.92	3.16 +/-0.89	33195.9 +/-1179.08
Pi	*AutoCPB*	**65.43 +/-6.73**	**6.65 +/-5.6**	2.96 +/-0.53	37445.02 +/-4117.43
Ir	*K-means*	1.58 +/-0.58	1.13 +/-0.49	3	**117.5 +/-7.89**
Ir	*ClusterP*	0.98 +/-0.14	1.19 +/-0.33	3	8390.32 +/-91.91
Ir	*AutoCP*	**0.78 +/-0.06**	0.26 +/-0.15	3.8 +/-0.81	11591.68 +/-1577.85
Ir	*AutoCB*	1.13 +/-0.17	0.25 +/-0.16	3.46 +/-1.16	14862.18 +/-297.83
Ir	*AutoCPB*	0.84 +/-0.09	**0.24 +/-0.13**	3.62 +/-0.81	4303.76 +/-324.01
Wi	*K-means*	216.67 +/-52.69	162.28 +/-63.28	3	**403.76 +/-13.2**
Wi	*ClusterP*	131.5 +/-8.60	68.74 +/-43.93	3	26301.88 +/-546.39
Wi	*AutoCP*	99.49 +/-7.02	24.79 +/-18.72	5.48 +/-1.05	37724.1 +/-3461.18
Wi	*AutoCB*	141.87 +/-17.68	32.1 +/-22.11	4.54 +/-1.11	42410.36 +/-1111.42
Wi	*AutoCPB*	**97.87 +/-10.51**	**23.29 +/-21.49**	4.88 +/-1.1	13421.86 +/-799.42
BC	*K-means*	563.32 +/-142.04	665.67 +/-0	2	**1949.08 +/-215.4**
BC	*ClusterP*	314.87 +/-11.53	141.46 +/-51.5	2	174520.6 +/-3574.57
BC	*AutoCP*	196.4 +/-14.18	44.3 +/-41.07	5.3 +/-1.04	259376.08 +/-34956.1
BC	*AutoCB*	292.89 +/-53.18	59.86 +/-50.86	4.32 +/-1.28	108388.1 +/-2871.43
BC	*AutoCPB*	**195.01 +/-22.4**	**34.44 +/-35.2**	4.62 +/-0.83	92403.2 +/-8281.34
Gl	*K-means*	9.23 +/-7.21	0.63 +/-0.29	6	**698.74 +/-41.74**
Gl	*ClusterP*	3.31 +/-0.42	0.39 +/-0.22	6	25520.92 +/-280.95
Gl	*AutoCP*	**1.23 +/-0.25**	0.29 +/-0.24	2.28 +/-0.5	45504.3 +/-24056.07
Gl	*AutoCB*	2.31 +/-0.4	0.24 +/-0.16	2.94 +/-1.27	29877.22 +/-651.5
Gl	*AutoCPB*	1.62 +/-0.31	**0.21 +/-0.2**	2.2 +/-0.45	11431.86 +/-1466.4
Ye	*K-means*	1.46 +/-1.09	0.04 +/-0.02	10	**6815.02 +/-62.67**
Ye	*ClusterP*	0.8 +/-0.12	**0.03 +/-0.01**	10	175838.7 +/-1397.52
Ye	*AutoCP*	**0.26 +/-0.02**	**0.03 +/-0.01**	2.12 +/-0.33	211558.7 +/-46305.2
Ye	*AutoCB*	0.34 +/-0.04	0.04 +/-0.02	2.42 +/-0.61	35482.18 +/-558.49
Ye	*AutoCPB*	**0.26 +/-0.02**	**0.03 +/-0.02**	2.22 +/-0.42	69598.58 +/-3044,74

the different branches of science we find enormous amounts of data from phenomena we can not truly understand (i.e. complex gene expression data, astronomical data). The later, is a true motivation for the development of automatic clustering algorithms.

From our results we can argue that in most domains, as the number of clusters increase, the number of data points diminish. However, this can not be true in the cases where the optimal clustering comprises a very segregated and marginal domain.

We can observe in Table 1 that a higher number of clusters (A1, A2, Ha, Pi, Ir, Wi and BC) tends to give a lower result in both ID and QEF. This might be because our tests do not express what we are really looking for, or perhaps, that we have discovered a new feature in the dataset. Thus, we can modify our assumptions about the optimal number of clusters in a dataset. Actually, since we use the same cluster assignment method in all techniques, we can conclude that we have also found the optimal number of clusters k given our objective function.

5 Final Discussion

In this work we described two swarm intelligence techniques, namely particle swarm optimization and bacterial foraging. We described in detail several algorithms based on these paradigms. The new algorithms improve the basic *ClusterP* method which in fact is based on *K-means*.

The essence of our work is that we managed to combine two major paradigms PSO and BFA in order to create a robust clustering algorithm. The resulting hybrid method *AutoCPB* showed improvements during experimentation. The original *AutoCP* algorithm was unable to reach the best results on its own. Actually, *AutoCB* is also suboptimal in non-random datasets because of its swarm model, which does not perform global search. Therefore, we sustained with empirical results that by properly combining the two techniques result in a true enhancement for clustering. Thus, we take the best feature of each algorithm (the *smart* foraging in PSO and the evolutionary rules from BFA).

The testing was possible due to our implemented framework, which enabled us to test all algorithm in well known datasets. In general, we believe that the application of hybrid PSO/BFA mechanisms is promising. We expect to improve AutoCPB to a major extent.

Future work consists in identifying a rule to minimize local optima in AutoCPB. This algorithm can be applied to other domains such as attribute clustering. This stochastic/evolutionary method can also show features not only in data but between variables in datasets. A more specific analysis of parameter setting is also considered as a tentative improvement.

References

1. Han, F., Kamber, K.: Data mining: Concepts and techniques, pp. 334–395. Academic Press, San Francisco (2001)
2. Tazutov, A., Kurenkov, N.: Neural network data clustering on the basis of scale invariant entropy. In: International Joint Conference on Neural Networks, pp. 4912–4918 (2006)

3. Chandra, S., Murthy, J.: An Efficient Hybrid Algorithm for Data Clustering Using Improved Genetic Algorithm and Nelder Mead Simplex Search. In: Proc. of the Int. Conf. on Comp. Int. and Mult. Appl., vol. 1, pp. 498–510 (2007)
4. Goncalves, M., Andrade, M.: Data Clustering using Self-Organizing Maps segmented by Mathematic Morphology and Simplified Cluster Validity Indexes: an application in remotely sensed images. In: International Joint Conference on Neural Networks, pp. 4421–4428 (2006)
5. Kennedy, J., Ebenhart, R.: Particle Swarm Optimization. In: Proceedings of the 1995 IEEE Int. Conf. on Neural Networks, pp. 1942–1948 (1995)
6. Passino, K.: Biomimicry of bacterial foraging for distributed optimization and control. Control Systems Magazine, 52–67 (2002)
7. Engelbrecht, A.: Fundamentals of Computational Swarm Intelligence. Wiley, Chichester (2005)
8. Heylighen, F., Joslyn, C.: Cybernetics and second-order cybernetics. Encyclopedia of Physical Science and Technology. Academic Press, New York (2001)
9. Bergh, F.: An analysis of particle swarm optimizers. PhD Thesis, University of Pretoria, South Africa (2002)
10. Clerc, M., Kennedy, J.: The particle swarm: Explosion, stability and convergence. IEEE Transactions on Evolutionary Computation 6, 58–73 (2002)
11. van der Merwe, D., Engelbrecht, A.: Data Clustering using particle swarm optimization. In: The 2003 congress on Evolutionary Computation, vol. 1, pp. 215–220 (2003)
12. Biswas, A., Dasgupta, S.: Synergy of PSO and bacterial foraging optimization: A comprehensive study on. Advances in Soft Computing Series, pp. 255–263. Springer, Heidelberg (2007)
13. Abraham, A., Roy, S.: Swarm intelligence algorithms for data clustering. Chp. 12, 279–313 (2008)
14. Asuncion, A., Newman, D.: UCI Machine Learning Repository. University of California, Irvine, School of Information and Computer Sciences (2007), `http://www.ics.uci.edu/~mlearn/MLRepository.html`
15. Theodoris, S., Koutroumbas, K.: Pattern Recognition. Academic Press, London (2006)