Differential Evolution and Particle Swarm Optimization in Partitional Clustering

THIEMO KRINK

EVALife group, Dept. of Computer Science, Univ. of Aarhus, Denmark, e-mail: krink@daimi.au.dk

SANDRA PATERLINI

Dept. of Political Economics, Univ. of Modena and Reggio E., Italy, e-mail: paterlini@unimo.it

Abstract: In recent years, many partitional clustering algorithms based on genetic algorithms (GA) have been proposed to tackle the problem of finding the optimal partition of a data set. Surprisingly, very few studies considered alternative stochastic search heuristics other than GAs or simulated annealing. Two promising algorithms for numerical optimization, which are hardly known outside the heuristic search field, are particle swarm optimisation (PSO) and differential evolution (DE). In this study, we compared the performance of GAs with PSO and DE for a medoid evolution approach to clustering, which Paterlini and Minerva (2003) introduced in a previous paper. Moreover, we compared these results with the nominal classification, *k-means* and random search (RS) as a lower bound. Our results show that DE is clearly and consistently superior compared to GAs and PSO for hard clustering problems, both in respect to precision as well as robustness (reproducibility) of the results. Only for simple data sets, the GA and PSO can obtain the same quality of results in contrast to *k-means* and RS, and, as expected, for trivial problems all algorithms can obtain comparable results. Apart from superior performance, DE is very easy to implement and requires hardly any parameter tuning compared to substantial tuning for GAs and PSOs. Our study shows that DE rather than GAs should receive primary attention in partitional cluster algorithms.

Key-words: Cluster analysis, partitional clustering, differential evolution, particle swarm optimization, genetic algorithms.

1 Introduction

In the last decades, cluster analysis has played a central role in a variety of fields. Clustering is often used as a tool for preliminary and descriptive data analysis and for unsupervised classification. Its main purpose is to identify homogeneous groups by finding similarities between objects regarding their characterising attributes. Moreover, cluster analysis can be used to summarize the shared characteristics of a group of objects by calculation of their centroids or baricentres.

Partitional clustering algorithms determine a grouping solution by maximising the similarities among objects *within* the same groups while minimising the dissimilarities *between* different groups. Thus, the algorithmic task can be stated as an optimization problem. Statistical criteria that consider the within and the between variance scatter matrices can be used to quantify the goodness of the partitions and to determine the optimal one. Ideally, a clustering algorithm should be simple, efficient and capable of dealing with huge datasets. Moreover, it should be objective and robust for equivalent samples and able to detect different cluster shapes.

Nowadays, the *k-means* algorithm is one of the most popular partitional clustering algorithms, because it is easy to implement and very efficient, due to its linear time complexity. However, its main drawbacks are that it converges to arbitrary local optima and that it cannot deal well with non-spherical shaped clusters.

Many partitional clustering algorithms that have been introduced in recent years, are based on evolutionary algorithms, such as Genetic Algorithms (GA) (Holland 1975), which are stochastic search heuristics inspired by Darwinian evolution and genetics. The key idea is to create a population of candidate solutions to an optimization problem, which is iteratively refined by alteration and selection of good solutions for the next iteration. Candidate solutions are selected according to a so-called fitness function, which evaluates their quality in respect to the optimization problem. In case of GAs, the alteration consists of mutation to randomly explore solutions in the local neighbourhood of existing solutions and crossover to recombine information between different candidate solutions.

An important advantage of these algorithms is their ability to cope with local optima by maintaining, recombining and comparing several candidate solutions simultaneously. In contrast, local search heuristics, such as the stochastic simulated annealing algorithm, only refine a single candidate solution and are notoriously weak in coping with local optima. Deterministic local search, which is used in the *k-means* algorithm, always converges to the nearest local optimum from the starting position of the search. The only way to explore the search space better is to re-run the algorithm while initialising the search from different starting points.

Therefore, GAs are obviously an interesting alternative to *k-means* and simulated annealing in clustering. However, in the scientific community of heuristics, simple textbook GAs are known to have inferior performance compared to advanced versions and other modern optimization approaches and are rather used as a lower bound for performance comparison. Two promising and recently introduced approaches to numerical optimization, which are rather unknown outside the heuristic methods field, are particle swarm optimisation (PSO) and differential evolution (DE).

In this study, we compared the performance of GAs with PSO and DE as heuristic search methods for the medoid evolution algorithm previously introduced by Paterlini and Minerva (2003) regarding a set of artificial and real-world machine learning data sets. Moreover, we compared these results with the nominal classification, *k-means* and random search (RS) as a lower bound technique. To our knowledge, there have been only a few recent and rather unknown studies on PSOs (e.g. Xiao *et al*. 2003) and no known previous study on DE in clustering.

The remaining sections of the paper are organized as follows. Section 2 gives an overview of the application of GAs to clustering problems. Section 3 describes the medoid evolution approach and introduces the search heuristics. The following section 4 describes the experimental set-up regarding the algorithmic parameters, benchmark problems, and run schedule. Section 5 reports the main results, and finally, section 6 comments on our results and concludes our study.

2 Scientific Background

Genetic algorithms have been applied to partitional clustering in many ways, which can be grouped into three main categories: (i) direct encoding of the object-cluster association, (ii) encoding of cluster separating boundaries, and (iii) centroid/medoid and variation parameter encoding for each cluster.

To our knowledge, the first application of GAs to clustering was introduced by Raghavan and Birchand (1979) and it belongs to the first approach of using a direct encoding of the object-cluster association. The idea in this approach is to use a genetic encoding that allocates directly n objects to g clusters, such that each candidate solution consists of n genes with integer values in the interval $[1, g]$. For example, for *n=4* and *g=2* the encoding "2112" allocates the second and the third object to cluster 1 and the first and fourth object to cluster 2 and therefore the following clusters ({14}, {23}) are identified. Based on this problem representation, the GA tries to find the optimal partition according to a fitness function which measures the partition goodness. Since 1979, many authors have used this approach. It has been shown that such an algorithm outperforms *k-means* in the analysis of simulated and real datasets (e.g. Murthy and Chowdury 1996). However, the representation scheme has a major drawback because of its redundancy, for instance, "2112" and "1221" represent the same grouping solution ({14}, {23})). Falkenauer (1998) tackled this problem in an elegant way. In addition to the mentioned encoding of *n* genes representing each object-cluster association, they represent the group labels as additional genes in the encoding and apply *ad hoc* evolutionary operators on them.

The second kind of GA approach to partitional clustering is to encode cluster separating boundaries. Bandyopadhyay *et al.* (1995, 1998, 1999) used GAs to determine hyperplanes as decision boundaries, which divide the attribute feature space to separate the clusters. For this they encode the location and orientation of a set of hyperplanes with a gene representation of flexible length. Apart from minimizing the number of misclassified objects, their approach tries to minimize the number of hyperplanes required. Another interesting and more flexible approach by Bandyopadhyay and Maulik (2002b) is to determine the boundaries between clusters by connected linear segments instead of rigid planes. Sarafis *et al* (2002) introduced an approach that identifies clusters by evolving a representation of linear boundaries around clusters in the object attribute space, which they call rule-based data clustering.

The third way to use GAs in partitional clustering is to encode a representative variable (typically a centroid or medoid) and optionally a set of parameters to describe the extend and shape of the variance for each cluster. Srikanth *et al*. (1995) proposed an approach, which encodes the centre, extend, and orientation of an ellipsoid for each cluster. Moreover, many authors proposed cluster centroids, baricentres, or medoids as representation points to allocate each object to a specific cluster (e.g. Maulik and Bandyopadhyay 2000, Chiou and Lan, 2001, Bandyopadhyay and Maulik 2002a, Paterlini and Minerva 2003). The idea is to determine a representation point for each cluster and to allocate each object to the cluster with the nearest representation point, where 'nearest' refers to a distance measure, such as Euclidean distance. The fitness of a candidate solution is then computed as the adequacy of the identified partition according to a statistical criterion, such as the Marriott or variance ratio criterion (see section 3.2). Many studies have shown that this approach is more robust in converging towards the optimal partition than classic partitional algorithms (e.g. Maulik and Bandyopadhyay 2000, Chiou and Lan, 2001, Bandyopadhyay and Maulik 2002a, Paterlini and Minerva 2003).

Finally, some authors introduced hybrid clustering algorithms, which combine classic clustering techniques with GAs. For example, Krishna and Murthy (1999) introduced a GA with the direct encoding of object-cluster associations by Raghavan and Birchand (1979), but applied *k-means* to determine the quality of the GA candidate solutions. For this, each GA candidate solution is used as a starting point for a *k-means* run. The quality of the solution found by the *k-means* run is then used as the fitness of the GA candidate solution.

Compared to the great number of studies on partitional clustering with GAs, only a couple of applications using PSO (e.g.: Xiao *et al.* 2003) and no application using DE (to our knowledge) can be found in literature. Moreover, there have been substantial research efforts on GAs in hierarchical clustering (e.g. Kuncheva 1995, Tseng and Yang 2001) as well as applications of simulated annealing and other local search methods to clustering that are beyond the scope of this paper.

3 The Medoid Evolution Algorithm

3.1 The Clustering Problem

Let $\boldsymbol{O}=\{o_1, o_2, \ldots, o_n\}$ be a set of n objects and let $\boldsymbol{X}_{nxp}$ be the profile data matrix, with n rows and p columns. Each i-th object is characterised by a real-value p-dimensional profile vector $\boldsymbol{x}_i$ ($i=1,..,n$), where each element x_{ij} in $\boldsymbol{x}_i$ corresponds to the j-th real value feature ($j=1,\ldots,p$) of the i-th object ($i=1,\ldots,n$).

Given X_{nxp}, the goal of a non-hierarchical clustering algorithm is to determine a partition $G=\{C_1,C_2,\ldots,C_g\}$ $(i.e.: C_k \neq \emptyset, \forall k; C_k \cap C_h = \emptyset, \forall k \neq h; \bigcup_{k=1}^{g} C_k = \boldsymbol{O})$ such that objects which belong to the same cluster are as similar to each other as possible, while objects which belong to different clusters are as dissimilar as possible. For this, a measure of adequacy of the partition must be defined. The clustering problem is to find the partition G^* that has *optimal* adequacy with respect to all other feasible solutions $\boldsymbol{G}$=$\{G^1, G^2, \ldots, G^{N(n,g)}\}$ (*i.e.:* $G^i \neq G^j$,$i \neq j$) where $N(n,g) = \frac{1}{g!}\sum_{k=0}^{g}(-1)^k \binom{g}{k}(g-k)^n$ is the number of all feasible partitions. This is equivalent to

$$\underset{\boldsymbol{G}}{optimise} \;\; f(X_{nxp}, G)$$

where G corresponds to a single partition in $\boldsymbol{G}$ and $f(*)$ is a statistical-mathematical function that quantifies the goodness of the partition (see next section 3.2).

It has been shown that the clustering problem is NP-hard when the number of clusters exceeds three (Brucker 1978).

3.2 Statistical Clustering Criteria

Different statistical criteria have been proposed to measure the degree of adequacy of a partition and to allow comparison across different partitions (Marriott 1982). These criteria usually involve transformations, such as the trace or determinant, of the pooled-*within* groups scatter matrix (*W)* and of the *between* groups scatter matrix *(B)*.

The *pooled-within scatter matrix,* ***W****,* is defined as:

$\boldsymbol{W} = \sum_{k=1}^{g} W_k$ where W_k is the variance matrix of the objects' features allocated to cluster C_k $(k=1,\ldots, g)$. Thus, if $\mathbf{x}_l^{(k)}$ indicates the *l-th* object in cluster C_k and n_k the number of objects in cluster C_k.

$$W_k = \sum_{l=1}^{n_k} (\boldsymbol{x}_l^{(k)} - \overline{\boldsymbol{x}}^{(k)})(\boldsymbol{x}_l^{(k)} - \overline{\boldsymbol{x}}^{(k)})',$$

where $\overline{\boldsymbol{x}}^{(k)} = (\sum_{l=1}^{n_k} \boldsymbol{x}_l^{(k)})/n_k$ is the vector of the centroids for cluster C_k

The *between scatter matrix*, $\boldsymbol{B}$, is defined as

$$\boldsymbol{B} = \sum_{k=1}^{g} n_k (\overline{\boldsymbol{x}}^{(k)} - \overline{\boldsymbol{x}})(\overline{\boldsymbol{x}}^{(k)} - \overline{\boldsymbol{x}})' \text{ where } \overline{\boldsymbol{x}} = (\sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} \boldsymbol{x}_i)/n.$$

Then, the *total scatter matrix* ***T***, of the *n* observations can be decomposed as ***T***=***B***+***W***.

In our study, we consider three statistical criteria to measure the adequacy of the partition and define the optimisation problem $\underset{\boldsymbol{G}}{optimise} \;\; f(X_{nxp}, G)$ respectively as:

1) $\underset{\boldsymbol{G}}{minimise} \;\; trace(\boldsymbol{W})$

TRW – Trace Within Criterion (Friedman and Rubin 1967).

This criterion assumes implicitly a low correlation among measurements, gives equal importance to the variance within the groups, tends to create spherical clusters and allows orthogonal transformations of the data. It can be shown that minimizing *trace(**W**)* is equivalent to minimizing the sum of eigenvalues of ***W***.

2) $$\underset{\mathbf{G}}{maximise} \quad \frac{trace(\mathbf{B})/(g-1)}{trace(\mathbf{W})/(n-g)}$$

VRC - Variance Ratio Criterion (Calinski and Harabasz 1974).

(n-g) are the degrees of freedom of the *within* scatter matrix and *(g-1)* are the degrees of freedom of the *between* scatter matrix. As for the Trace Within Criterion, the Variance Ratio Criterion assumes implicitly a low correlation among measurements, gives equal importance to the variance within the groups, tends to create spherical clusters and allows orthogonal transformations of the data. Moreover, in many cases, it can be used to identify the optimal number of groups by comparison of maximization results for different values of g. However, for some data sets, this method fails if the results increase monotically with larger values of g.

3) $$\underset{\mathbf{G}}{minimise} \; g^2 \frac{det(\mathbf{W})}{det(\mathbf{T})}$$

MC - Marriott's criterion (Marriott 1971 and 1982)

The Marriott criterion addresses the correlation between variables, detects elliptical clusters with axes that are not parallel to the coordinates, and allows linear (not singular) transformations of the data. It can be shown that minimizing *det(**W**)* is equivalent to minimizing the product of the eigenvalues of ***W***. Marriott's criterion is commonly used to search for clusters characterized by such a strong internal correlation that one or more eigenvalues are equal to zero.

For further details about these and related criteria the reader is referred to Everitt (1993). Note that each cluster set C_k of *G* must at least contains one object, i.e., $C_k \neq \emptyset$, which is not guaranteed by the encoding of the medoid evolution algorithm as we will explain in the following section.

3.3 Fitness Evaluation and Search Space

Central to population-based heuristics, such as GAs, is the concept of a population of individuals, where each individual consists of an encoding of a candidate solution called chromosome (also: genes, genotype, or genome) and a fitness that indicates its quality. In our study, we used floating point arrays to encode representation points, hereafter called *medoids*, to be used in allocating objects to different clusters and therefore in determining a partition. Hence, if $\boldsymbol{X}_{nxp}$ is the profile matrix and g the number of clusters $\{C_1,C_2,,\ldots,C_g\}$ of the set of n objects $O=\{o_1,o_2,\ldots,o_n\}$, each chromosome in the population consists of p x g cells m_{kj} ($k=1,\ldots,g$, $j=1,\ldots,p$). Each group of p cells, that corresponds to the vector $\boldsymbol{m}_k$, identifies the *k-th* medoid coordinates in the R^p space of the measurements. The g groups of p cells that constitute the vector $\boldsymbol{m}$ represent the g medoids of the clusters. Figure 1 shows an example for a problem with 3 clusters and 4 features.

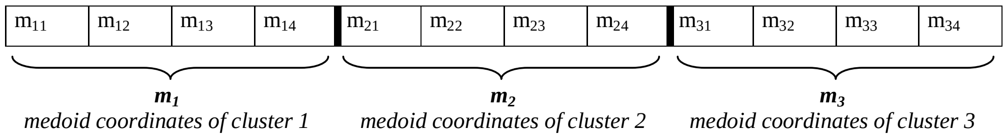
m11 m12 m13 m14
m21 m22 m23 m24
m31 m32 m33 m34
m1
medoid coordinates of cluster 1
m2
medoid coordinates of cluster 2
m3
medoid coordinates of cluster 3

Figure 1. Example of a cluster problem encoding with 3 clusters and 4 features.

In principle, any point in R^p could be considered as a possible choice for a medoid. However, it makes sense to restrict the search space to roughly the size of the profile matrix domain [$\boldsymbol{x}_{min}$, $\boldsymbol{x}_{max}$], which is more likely to contain good medoids.

In our study, we decided to define the medoid domain to be 40% larger than the profile matrix domain, i.e., [$\boldsymbol{x}_{min}-0.2|\boldsymbol{x}_{max}-\boldsymbol{x}_{min}|$, $\boldsymbol{x}_{max}+0.2|\boldsymbol{x}_{max}-\boldsymbol{x}_{min}|$].

It is then necessary to define a mapping between the medoid search space and the clustering search space $\boldsymbol{G}=\{G^1, G^2, \ldots, G^{N(n,g)}\}$. The mapping that we used is inspired by Forgy's approach of clustering (Forgy 1965). A partition H is determined by allocating each object to the nearest medoid, where 'nearest' refers to a distance metric, which is the Euclidean distance in our study. The adequacy of a feasible partition H is then evaluated by using one of the three statistical criteria (*TRW*, *MC*, *VRC*) described in section 3.2. Otherwise, if H is infeasible $(i.e.: C_k=\varnothing)$, we penalize the candidate solution with a fitness worse than the worst fitness of a feasible solution. More formally the fitness function is defined as:

$$F(\boldsymbol{X}_{nxp},\boldsymbol{m})=\begin{cases} f(\boldsymbol{X}_{nxp},H) & \text{if } H\subset \boldsymbol{G}=\left\{G^1,G^2,\ldots,G^{N(n,g)}\right\} \\ K & \text{if } H\not\subset \boldsymbol{G}=\left\{G^1,G^2,\ldots,G^{N(n,g)}\right\} \end{cases}$$

where ***m*** is the vector of the medoids of a candidate solution, *f(.)* is one of the statistical criteria *TRW, VRC* or *MC* and *K* is 10^8 if *f(.)* corresponds to *TRW* or *MC* and to -10^8 if *f(.)* corresponds to *VRC* respectively.

Note that there is no one-to-one correspondence between the search space and the space of feasible partitions: different medoid vectors can identify the same partition *H.* Moreover infeasible solutions might occur (e.g.: objects are allocated to less than *k* clusters).

3.4 Search Heuristics

3.4.1 The Genetic Algorithm (GA)

A GA is an evolutionary algorithm inspired by Darwinian evolution and genetics. Evolutionary algorithms have been originally introduced by Fogel *et al.* (1966) as evolutionary programming and Rechenberg (1973) and Schwefel (1975) as evolution strategies although the idea to use evolution as an inspiration for optimization dates back to the 1940s (for a complete overview see Fogel 1998). Later, John Holland (1975) introduced the term genetic algorithm (GA). The main algorithmic innovation in GAs is the introduction of a recombination operator called crossover that allows to recombine solutions of candidate solutions inspired by genetic reproduction. In our GA implementation (see table 1a), first a population of individuals containing the candidate solutions (encoded in floating point numbers) is created and the fitness of each individual is evaluated by the fitness function. The chromosomes of the start-up population are initialized with randomly chosen object feature vectors from the dataset.

```
void genetic_algorithm()
{
initialize();
evaluate();
determineAndProtectElite();
for (int i=0; i<numIterations; i++) {
  selectNewPopulation();
  apply_crossover();
  apply_mutation();
  evaluate();
  determineAndProtectElite();
}
}
```

Table 1a: Pseudo-code of the genetic algorithm (GA).

After initialization, the population is iteratively refined by selection of individuals for the next iteration, application of mutation and crossover operators, and re-evaluation of the new population according to the fitness function. For selection we use tournament selection of size 2, i.e., for each individual *j* we choose another individual *k* randomly from the population, compare the fitnesses, and substitute *j* by *k* in the new population if *k*'s fitness is better. Further, we use elitism with an elite size of 10, i.e., the 10 best individuals of the population in each generation are left unchanged by mutation and crossover. For this, we rank the individuals according to their fitness at the end of the evaluation phase. As the mutation operator, we use Gaussian mutation, such that

$$j_i = j_i + N(0,1) \cdot \sigma \cdot (x_{i,\max} - x_{i,\min}),$$

where j_i is the *i*-th gene of individual *j*, *N* is the Gaussian normal distribution, and σ the variance parameter of the mutation operator. The crossover operator in our algorithm is arithmetic crossover with

$$c_i = w_i \cdot a_i + (1 - w_i) \cdot b_i$$

where *c* is the offspring genome of the parent genomes *a* and *b*, w_i a random weight of the interval [0, 1] and *i = 1,..., n*, with $n = g \cdot p$ (number of genes). The application of the crossover operator to a genome *j* means that *j* becomes parent *a*, parent *b* is chosen randomly from the population and the offspring *c* substitutes *j*. Both operators are applied to each individual in the population, which is not in the elite, with a probability p_m for mutation and p_c for crossover respectively. The algorithm terminates after a fixed number of iterations. The optimization result is the candidate solution and the fitness of the best individual in the last generation.

3.4.2 Particle Swarm Optimization (PSO)

Particle Swarm Optimization, which was introduced by Kennedy and Eberhard (1995) is inspired by the swarming behaviour of animals and human social behaviour. A particle swarm is a population of particles, where each particle is a moving object that 'flies' through the search space and is attracted to previously visited locations with high fitness. In contrast to the individuals in evolutionary computation, particles neither reproduce nor get replaced by other particles.

Each particle consists of a position vector $\vec{x}$, which represents the candidate solution to the optimization problem, the fitness of solution $\vec{x}$, a velocity vector $\vec{v}$ and a memory vector $\vec{p}$ of the best candidate solution encountered by the particle with its recorded fitness.

The position of a particle is updated by

$$\vec{x}(t+1) \leftarrow \vec{x}(t) + \vec{v}(t+1)$$

and its velocity according to

$$\vec{v}(t+1) \leftarrow \chi(w\vec{v}(t) + \varphi_1(\vec{p} - \vec{x}(t)) + \varphi_2(\vec{p}_g - \vec{x}(t))),$$

where φ_1, φ_2 are uniform distributed random numbers within $[\varphi_{\min}, \varphi_{\max}]$ (typically $\varphi_{\min} = 0.0$ and $\varphi_{\max} = 2.0$) that determine the weight between the attraction to position $\vec{p}$, which is the best position found by the particle so far and $\vec{p}_g$ the overall best position found by all particles. A more general version of PSO considers $\vec{p}_g$ as the best position found in a certain neighbourhood of the particle, which does not generally contribute to performance improvements. Note that φ_1 and φ_2 are newly generated for each component of the velocity vector. Moreover, the so-called inertia weight w controls how much the particles tend to follow their current direction compared to the memorized positions $\vec{p}$ and $\vec{p}_g$. Finally, the so-called constriction factor χ can be used to manipulate the overall velocity of the swarm. In our preliminary parameter tuning experiments we focused on the control of the inertia weight, which was decisive for the performance of the PSO. Moreover, the speed of the particles is limited by a maximum velocity $\vec{v}_{\max}$, which is typically half of the domain size for each parameter in vector $\vec{x}$.

The algorithm works as outlined in the pseudo-code of table 1b.

```
void particle_swarm_optimization()
{
initialize();
evaluate();
updateParticleMemories();
for (int i=0; i<numIterations; i++) {
  updateVelocities();
  updatePositions();
  evaluate();
  updateParticleMemories();
}
}
```

Table 1b: Pseudo-code of the particle swarm optimization (PSO) algorithm.

The initialization of the algorithm corresponds to the description for the GA above, but additionally requires the initialization of the speed vectors, which are uniformly distributed random numbers in the interval [0, $\vec{v}_{\max}$]. After initialization, the memory of each particle is updated and the speed and position update rules are applied. If the speed exceeds $\vec{v}_{\max}$ it is truncated to this value. Moreover, if a new position vector is outside the domain, it is moved back into the search space by adding the negative distance with which it exceeds the search space to the position vector. This process is applied to all particles and repeated for a fixed number of iterations. The optimization result is the best recorded candidate solution ($\vec{p}_g$ in the last iteration) and fitness at the end of the run.

3.4.3 Differential Evolution (DE)

Differential evolution (Storn and Price 1995) is a rather unknown approach to numerical optimization, which is very simple to implement, requires little or no parameter tuning, and is known for remarkable performance. After generating and evaluating an initial population, as described for the GA above, the solutions are refined as follows (see table 1c): For each individual genome *j*, choose three other individuals *k*, *l*, and *m* randomly from the population (with *j≠k≠l≠m*), calculate the difference of the chromosomes in *k* and *l*, scale it by multiplication with a parameter *f* and create an offspring by adding the result to the chromosome of *m*. The only additional twist in this process is that not the entire chromosome of the offspring is created in this way, but that genes are partly inherited from individual *j*, such that

$$o.gene_i = \begin{cases} m.gene_i + f \cdot (k.gene_i - l.gene_i) & \text{if } U(0,1) < p_c \\ j.gene_i & \text{otherwise} \end{cases}$$

The proportion is determined by the so-called crossover probability p_c (an ill-conceived term), which determines how many genes of the difference vector on average are copied to the offspring. More precisely, the operator iteratively copies consecutive genes (from a random starting point on and continuing with the first gene after the last gene has past) of the difference vector to the offspring until $U(0,1) \geq p_c$. If the offspring o is better than j then j is substituted by o.

```
void differential_evolution()
{
initialize();
evaluate();
for (int i=0; i<numIterations; i++) {
  createDifferenceOffspringsAndReplaceParentsIfBetter();
  evaluate();
}
}
```

Table 1c: Pseudo-code of the differential evolution (DE) algorithm.

The process is repeated for a fixed number of iterations and the optimization result is the best recorded candidate solution and fitness at the end of the run.

3.4.4 Random Search (RS)

The random search algorithm guesses solutions instead of using a heuristic (see table 1d) and does not maintain a population. It iteratively generates random candidate solutions *j* by assigning random uniform distributed values within the medoid domain, i.e., $j = U(x_{\min}, x_{\max})$ and records the best candidate solution for a fixed number of iterations. No operators or selection schemes are applied to guide the search. Random search is often used as a lower bound algorithm for performance comparison with heuristic search methods.

```
void random_search()
{
initialize();
evaluate();
saveAsBestSolution();
for (int i=0; i<numIterations; i++) {
  createRandomSolution();
  evaluate();
  saveIfBetterSolution();
}
}
```

Table 1d: Pseudo-code of the random search (RS) algorithm.

4 Experimental Set-Up

4.1 Benchmark data

The algorithms have been tested in comparison with the nominal classification, random search and the *k-means* algorithm regarding the following artificial and real world datasets.

4.1.1 Artificial data

We generated six different types of artificial datasets (*s1*, *s2*, *s3*, *s4*, *s5*, *s6*) from multivariate normal distributions with different parameter configurations. For each type we generated five datasets. Table 8 in Appendix A describes the process that generated these data in detail. Datasets of type *s1*, *s3* and *s5* have three, six and nine non-overlapping spherical clusters of twenty objects with three features each. Datasets of type *s2*, *s4* and *s6* have three, six and nine partially overlapping spherical clusters of twenty objects with three features each respectively.

Figure 2(a) and 2(b) show the two and three dimensional plots of observations for one of the randomly generated datasets of type *s1*, and figure 2(c) and 2(d) show the two and three dimensional plots of observations in one of the randomly generated datasets of type *s2* respectively. Notice the non-overlapping and partially overlapping structure of the clusters.

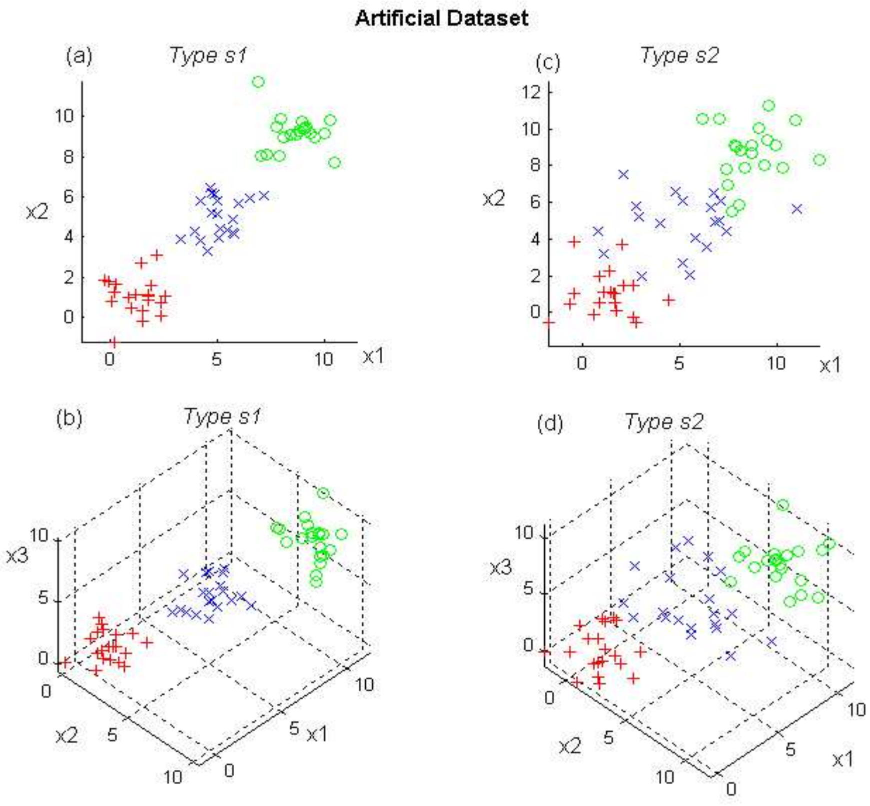

Figure 2: Example of two artificial datasets with three non overlapping (type s1 (a)-(b)) and overlapping clusters (type s2 (c)-(d)) of twenty objects with three features each.

4.1.2 Real world data

In addition to the artificial data, we used four well-known real-world datasets from the Machine Learning Repository (Merz et al. 1997) for further investigation. They are:

- <u>Fisher's Iris dataset</u> (n=150, p=4, g=3).

The dataset consists of three different species of iris flower: *Iris setosa, Iris virginica* and *Iris versicolour* (see figure 3). For each species, fifty samples with four features each (sepal length, sepal width, petal length and petal width) were collected.

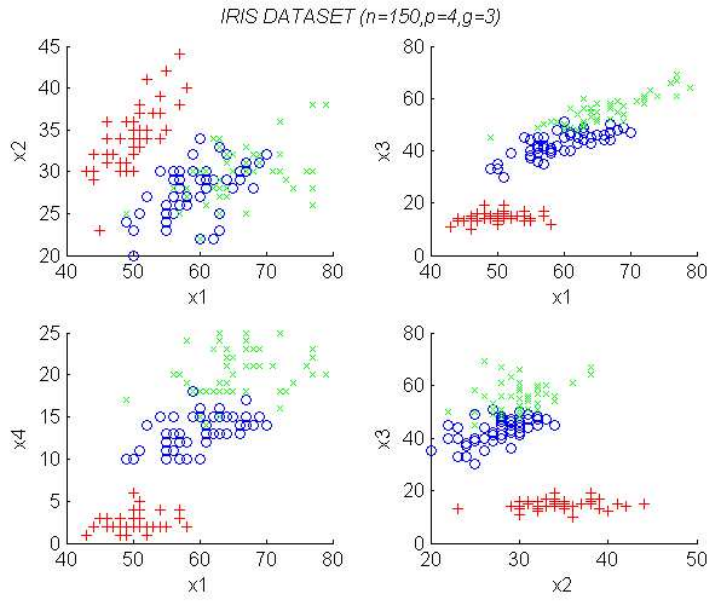

Figure 3: The iris dataset. *Iris setosa* (‘+’), *Iris versicolor* (’o’), *Iris virginica* (‘x’)

- Wisconsin Breast Cancer dataset (*n*=683, *p*=9, *g*=2).

This dataset consists of 683 objects characterized by nine features: clump thickness, cell size uniformity, cell shape uniformity, marginal adhesion, single epithelial cell size, bare nuclei, bland chromatin, normal nucleoli and mitoses. There are two categories in the data: malignant (444 objects) and benign (239 objects).

- Ripley’s glass dataset (*n*=214, *p*=9, *g*=6).

The data were sampled from six different type of glass: building windows float processed (70 objects), building windows non float processed (76 objects), vehicle windows float processed (17 objects), containers (13 objects), tableware (9 objects), headlamps (29 objects) with nine features each: refractive index, Sodium, Magnesium, Aluminum, Silicon, Potassium, Calcium, Barium and Iron.

- Vowel dataset (*n*=871, *p*=3, *g*=6).

This dataset consists of 871 Indian Telugu vowel sounds. The dataset has three features corresponding to the first, second and third vowel frequencies and six overlapping classes $\{\delta$ (72 objects), *a* (89 objects), *i* (172 objects), *u* (151 objects), *e* (207 objects), *o* (180 objects)$\}$.

Glass and Vowel datasets have clusters, which are strongly overlapping.

Table 2 below summarizes the main characteristics of the artificial and real-world dataset investigated.

Table 2: Characteristics of the datasets considered

Name	#Objects	#Clusters	#Features	Objects per Cluster
Artificial Data				
s1	60	3	3	20
s2	60	3	3	20
s3	120	6	3	20
s4	120	6	3	20
s5	180	9	3	20
s6	180	9	3	20
Real World Data				
Iris	150	3	4	50
Cancer	683	2	9	444.239
Glass	214	6	9	70,76,17,13,9,29
Vowel	870	6	3	72,89,172,151,207,180

4.2 Algorithmic settings

For the GA, PSO, and DE, we conducted several pre-experiments to determine one parameter setting per algorithm that yields the best performance with respect to all datasets.

The performance bottle-neck in all three search heuristics is the fitness evaluation of candidate solutions. Thus, for a fair performance comparison, all algorithms had the same number of fitness evaluations, which we set to 100.000. For the GA, PSO, and DE the number of fitness evaluations is the product of the population size times the number of iterations. For the random search algorithm (RS), we evaluated 100.000 randomly created points in the search space and recorded the best result.

For the GA, PSO, and DE we used the following parameter settings shown in table 3 (the RS algorithm has no parameters):

Table 3: Parameters of genetic algorithm, particles swarm optimization and Differential Evolution. The inertia weight of the PSO was linearly decreased from 1.0 to 0.7 during the 2000 iterations of the run.

GA		**PSO**		**DE**	
Parameter	**Value**	**Parameter**	**Value**	**Parameter**	**Value**
Population size	100	Population size	50	Population size	50
No. of iterations	1000	No. of iterations	2000	No. of iterations	2000
Crossover rate	1.0	Inertia weight	$1.0 \rightarrow 0.7$	Crossover rate	0.9
Mutation rate	1.0	$\varphi_{\min}$	0.0	Scaling factor f	0.3
Mutation sigma	0.05	$\varphi_{\max}$	2.0		
Elite size	10	χ	1.0		

In our preliminary experiments (not reported in this paper), we noticed that initialization with random candidate solutions for the medoids resulted in many infeasible solutions for the harder problems (often up to 90%), such as the glass data, which is a bad starting condition for population-based heuristics. Instead, we randomly selected data points of the datasets and used them as candidate solutions for the medoids during initialization in all final experiments, which yielded a much higher proportion of feasible candidate solutions and better overall results for the GA, PSO, and DE.

4.3 Implementation

We implemented our algorithms from scratch in C++ with Microsoft VisualStudio .NET except for using the MatLab C++ library for fast calculcation of matrix determinants. All experiments were run in Windows XP on a DELL latitude laptop PC and a DELL desktop PC both with Intel P4 2 GHz processors.

5 Results

5.1 Artificial data clustering

Regarding the artificial datasets, we conducted experiments for the thirty datasets (five variations of six data types), which we repeated 10 times for the stochastic algorithms RS, GA, PSO, and DE. Table 4 reports the mean and standard error of the mean best fitness over ten runs for each of the five datasets per data type.

Table 4: Mean Values and Standard Errors in the analysis of artificial datasets.
Column 1: data type (s1-s6), column 2: clustering criteria (MC = Marriott Criterion, TRW = Trace of the within matrix, VRC = Variance Ration Criterion), and columns 3-7: mean fitness with the standard error in brackets. The best fitness values are marked in bold.

dataset	criterion	k-means	RS	GA	PSO	DE
	MC		**0.229 (0)**	**0.229 (0)**	**0.229 (0)**	**0.229 (0)**
s1	TRW	218.50 (100.77)	**167.99 (0)**	**167.99 (0)**	**167.99 (0)**	**167.99 (0)**
	VRC		**332.13 (0)**	**332.13 (0)**	**332.13 (0)**	**332.13 (0)**
	MC		**0.516 (0)**	**0.516 (0)**	**0.516 (0)**	**0.516 (0)**
s2	TRW	**404.23 (0)**	404.26 (0.1733)	**404.23 (0)**	**404.23 (0)**	**404.23 (0)**
	VRC		**134.15 (0)**	**134.15 (0)**	**134.15 (0)**	**134.15 (0)**
	MC		0.8415 (0.1003)	**0.2343 (0)**	**0.2343 (0)**	**0.2343 (0)**
s3	TRW	670.76 (169.75)	665.49 (71.3147)	**335.48 (0)**	**335.48 (0)**	**335.48 (0)**
	VRC		573.59 (91.4623)	**1150.39 (0)**	1150.27 (0.5745)	**1150.39 (0)**
	MC		1.0956 (0.0684)	0.4615 (0.0026)	0.4650 (0.0146)	**0.4593 (0.0005)**
s4	TRW	983.21 (134.55)	1195.8 (77.2703)	804.53 (1.2116)	808.16 (13.8201)	**803.76 (0)**
	VRC		303.6 (15.0845)	481.2 (0.691)	481.38 (0.644)	**481.68 (0)**
	MC		*	0.3569 (0.1166)	0.5503 (0.1719)	**0.2183 (0)**
s5	TRW	1123.96 (224.43)	*	630.79 (85.5994)	798.64 (160.31)	**533.72 (0)**
	VRC		*	1998.71 (247.53)	1608 (319.5)	**2323.16 (0)**
	MC		1.872 (0.3243)	0.7345 (0.0808)	0.6213 (0.1125)	**0.4687 (0.0015)**
s6	TRW	1515.90 (193.68)	2817.92 (558.52)	1373.56 (71.0108)	1310.54 (94.9363)	**1192.2 (0.8476)**
	VRC		421.42 (73.2023)	898.63 (43.2029)	927.18 (74.279)	**1041.59 (0.6551)**

For the simple datasets of type *s1* and *s2*, GA, PSO and DE always reached the same mean values with standard error zero. Even RS obtained the same fitness with standard error zero in all experiments except one (type *s2*- *TRW* criterion), whereas *k-means* achieved the same result as GA, PSO, and DE for dataset *s2*, but failed in many runs for *s1*.

The more challenging datasets of type *s3*, *s4*, *s5* and *s6* revealed superior performance of GA, PSO and DE compared to RS and *k-means* by consistently obtaining better mean fitness values with lower standard errors. GA, PSO and DE converged consistently to the same candidate solutions with the same fitness and standard error zero for datasets of type *s3* (except for PSO-VRC). DE maintains such robust convergence also for the datasets of type s4 (except for MC) and datasets of type 5.

In conclusion, the harder the clustering problem, the more it pays-off to apply population-based heuristics. Moreover, comparing the population-based heuristics, DE clearly obtains the best results both in terms of accuracy (mean fitness) and robustness (variance of the repeated results).

5.2 Real-world data benchmarks

Apart from the artificial datasets, we conducted experiments with all algorithms for the four machine learning data benchmarks and repeated each run 30 times. The results are shown in table 5.

Table 5: Mean values of the optimal fitness values and standard Errors over thirty runs in the analysis of real world datasets. Columns 4-8 show the mean and standard errors (in brackets) of the best fitnesses over 30 runs. The best results are marked in bold.

Dataset	Criterion	k-means	RS	GA	PSO	DE
	MC		0.1995 (0.0002)	**0.1984 (0)**	**0.1984 (0)**	**0.1984 (0)**
iris	TRW	8950.18 (449.79)	**7885.14 (0)**	**7885.14 (0)**	**7885.14 (0)**	**7885.14 (0)**
	VRC		**561.63 (0)**	**561.63 (0)**	**561.63 (0)**	**561.63 (0)**
	MC		0.3733 (0.0019)	0.3565 (0.0027)	**0.3527 (3.51e-005)**	**0.3546 (0.0019)**
cancer	TRW	**19323 (0)**	19361 (3.3296)	**19323 (0)**	19324 (0.3944)	**19323 (0)**
	VRC		1022.28 (0.3555)	**1026.26 (0)**	**1026.26 (0.0002)**	**1026.26 (0)**
	MC		0.06463 (0.0017)	0.02661 (0.0009)	0.03176 (0.0010)	**0.01984 (0.0003)**
glass	TRW	366.98 (6.59)	393.21 (2.223)	341.09 (1.5999)	339.04 (0.9505)	**336.06 (0)**
	VRC		101.05 (0.9564)	121.94 (0.7799)	122.74 (0.578)	**124.62 (0)**
	MC		0.4563 (0.0051)	0.3199 (0.0015)	0.3032 (0.0024)	**0.2906 (0.0008)**
vowel	TRW	31687462 (229932)	34898130 (183388)	30943106 (13606)	30734068 (6467.23)	**30690785 (1816.64)**
	VRC		1278.7 (7.14)	1450.45 (0.9677)	1463.33 (0.3809)	**1465.55 (0.1156)**

Our results clearly show that Fisher's iris dataset is not sufficiently challenging to compare the performance between advanced clustering algorithms despite its great popularity in the clustering community. Even the random search algorithm could identify the same optimal partition in every run for *TRW* and *VRC*, whereas *k-means* frequently converged to local optima. By exhaustive search, we have verified that the best partition found corresponds to the global optimum of the corresponding fitness function.

Regarding the cancer data, *k-means* is as robust as GA and DE and converges to the same fitness value in every run, whereas for the TRW criterion, PSO has a mean value of 19324 and the approximated confidence interval at 95% level ([19323.21,19324.79]) does not include the optimal value 19323. For the *VRC*, GA, PSO and DE have equivalent results, whereas RS cannot compete. In contrast, for *MC*, PSO reaches the minimum mean with standard error close to zero. This is the only case in our experiments, where DE did not obtain the best mean value. However, the approximated confidence interval of the DE result at 95% level ([0.3508,0.3584]) contains the PSO mean fitness result, and thus the DE and PSO results are yet not significantly different. In all other experiments, DE always reaches the best mean fitness among all algorithms with a lower standard error than GA and PSO.

The second set of experiments on real-world data confirms the superior performance of the population-based heuristics. Simple datasets, however, can be analysed equally well with almost any kind of clustering algorithm, even random search. Substantial performance differences occur for challenging clustering problems with a large number of objects and clusters as well as overlapping cluster shapes. Moreover, DE turns out to be more robust and effective than GA and PSO.

Finally, we have computed the mean number of misclassified items, which is the average number of objects that were assigned to clusters other than according to the nominal classification. Table 6 reports the mean values and standard errors.

DE has a standard error smaller than one in all cases except for the cancer data with MC criterion (S.E.=1.71) and for vowel data with MC criterion (S.E.=5.85). Table 5 shows that in the five cases where DE does not identify the same partition in all 30 runs, the uncertainty in allocating objects to different groups is very small and related to few objects.

Table 6 Mean Values of the error with respect to the nominal classification and standard errors over thirty runs in the analysis of real world datasets.

dataset	criterion	k-means	RS	GA	PSO	DE
iris	MC	23.83 (3.31)	3.03 (0.16)	3 (0)	3 (0)	3 (0)
	TRW		16 (0)	16 (0)	16 (0)	16 (0)
	VRC		16 (0)	16 (0)	16 (0)	16 (0)
cancer	MC	27 (0)	79.17 (0.67)	80.43 (2.36)	81.27 (0.48)	81.13 (1.71)
	TRW		27.43 (0.31)	27 (0)	27 (0.05)	27 (0)
	VRC		28 (0.37)	27 (0)	26.87 (0.06)	27 (0)
glass	MC	104.60 (1.26)	106.9 (0.74)	101.87 (0.57)	104.33 (0.58)	107.43 (0.69)
	TRW		106.53 (0.71)	98.6 (0.25)	98.2 (0.19)	98 (0)
	VRC		105.2 (0.98)	98.83 (0.25)	98.43 (0.27)	98 (0)
vowel	MC	417.47 (5.95)	415.97 (7.07)	363.97 (11.18)	355.73 (9.44)	291.47 (5.85)
	TRW		420.77 (5.85)	454.4 (1.71)	449.13 (1.36)	451.4 (0.20)
	VRC		420.8 (5.29)	448.97 (2.08)	447.57 (1.90)	450.37 (0.78)

Moreover, table 6 shows that there was no case where the number of misclassified items was zero, which can be explained by the data shown in table 7. Columns 3 and 4 in table 7 show the fitness values for the three different criteria in correspondence of the nominal classification and the according best mean fitness by the clustering algorithms reported in table 6.

In all cases, the best mean values of the clustering algorithms are better than the corresponding fitnesses of the nominal classification for *TRW* and *MC* (minimization) as well as for *VRC* (maximization). This result can be explained by outliers, anomalous data and errors in collecting data. For example, figure 4 shows a

2 dimensional plot of the iris data (sepal length vs. petal width), where data have been grouped using different colors (red= '*setosa*', green='*versicolor*' , blue= '*virginica*') according to the nominal classification ('+') and to the optimal partition for MC ('o'). The three arrows in the figure mark those data points, where objects have been allocated to different groups by the MC optimal partition and the nominal classification. These data points are situated on the boundary between the two groups and could be reasonably considered to belong to both species. Moreover, the choice of an inappropriate fitness criterion can prevent the detection of the natural structure behind the data, such as a criterion assuming spherical shapes of the clusters when clusters are non-spherical.

Table 7: Comparison between the fitness value in correspondence of the nominal classification and the optimal mean values determined by the clustering algorithms.

dataset	**criteria**	**Nominal**	**Optimal Mean value**
iris	TRW	8929.70	7885.14
	MC	0.211	0.198
	VRC	487.331	561.628
cancer	TRW	20707	19323.2
	MC	0.6267	0.353
	VRC	912	1026.3
vowel	TRW	65984000	30690800
	MC	1.0829	0.291
	VRC	589.18	1465.55
glass	TRW	911.2	336.061
	MC	2.826	0.020
	VRC	19.702	124.616

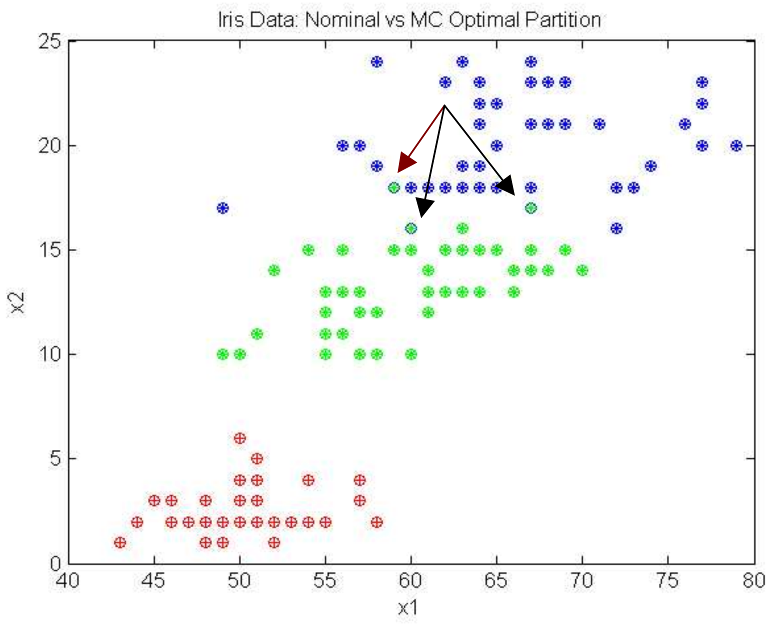

Figure 4: Iris Data: Nominal Classification vs. MC optimal Partition

7 Discussion and Conclusions

In this paper, we compared the performance of GAs, PSO and DE with *k-means* and RS for a medoid evolution approach to partitional clustering. Both, for artificial and real-world data, we found that all three population-based heuristics obtained very good results. The simple clustering problems of our datasets could be solved by any method, even RS which we used as a lower bound algorithm for comparison. However, for more complex datasets with many features, many clusters, and overlapping clusters, we found substantial performance differences between the investigated algorithms. Here, the three population-based heuristics were unquestionably superior compared to *k-means* and RS, which we expected for the GA according to earlier results reported in the literature.

The most important conclusion of our experiments is that DE is clearly and consistently superior compared to GAs and PSO both in respect to precision as well as robustness of the results (i.e. very similar results of repeated runs). The latter is an important characteristic from an end-user perspective, since a clustering algorithm must not only be accurate, but also produce reliable and reproducible results. Previous studies on DE in other domains of numerical optimization came to similar conclusions regarding its performance, but yet the algorithm is surprisingly unknown and has not been appropriately advertised. Our study shows for the first time the superior potential of DE in partitional clustering. Apart from superior performance, one should note that DE is very easy to implement (even simpler than PSO and GAs) and requires very little parameter tuning compared to substantial tuning for GAs and PSOs. Also in terms of run-time considering the same number of fitness evaluations, DE is faster than GA, in particular when the fitness evaluation is very short like in case of simple clustering problems.

Despite the convincing performance of all three population-based heuristics, none of the clustering experiments was without misclassification with respect to the nominal classification, which was what we expected. Interestingly, we found that the final fitness obtained by our algorithms was much better than the fitness of the nominal classification, which shows that the misclassification could not be explained by the optimization performance. Instead, misclassification is the result of the underlying assumptions of the clustering fitness criteria (such as spherical shape of the clusters), outliers in the dataset, errors in collecting data and human errors in the nominal solutions. This is indeed not a negative result. In fact, the differences of a clustering solution based on statistical criteria compared to the nominal classification can reveal interesting data points and anomalies in the dataset. In this way, a clustering algorithm can be used as a very useful tool for data pre-analysis.

Our study shows that DE rather than GAs should receive primary attention for numerical optimization in partitional cluster algorithms with great potential for data analysis tools.

Acknowledgements

The authors would like to thank the Danish Research Council for partial financial support of this study in context of the EVALife project. This work has been done while Sandra Paterlini was visiting EVALife Group, Aarhus Univ. (April - June 2003).

References

Bandyopadhyay S., Murthy C.A., and Pal S.K., 1995. Pattern classification with genetic algorithms, Pattern Recognition Letters, 16, 801-808.

Bandyopadhyay S., Murthy C.A., and Pal S:K., 1998. Pattern classification using genetic algorithm:determination of H, Pattern Recognition Letters, 19, 1171-1181.

Bandyopadhyay S., Murthy C:A., and Pal S.K., 1999. Theoretic performance of genetic pattern classifier, Journal of The Franklin Institute, 336, 387-422.

Bandyopadhyay S., Pal S.K., and Murthy C.A., 1998. Simulated Annneling based pattern classification, Journal of Information Sciences, 109, 165-184.

Bandyopadhyay S., and Maulik U., 2002a. An evolutionary technique based on K-means algorithm for optimal clustering in R^N, Information Sciences, 146, 221-237.

Bandyopadhyay S., and Maulik U., 2002b. Genetic clustering for automatic evolution of clusters and application to image classification, Pattern Recognition, 35, 1197-1208.

Brucker P., 1978. On the complexity of clustering problems. In: Beckmenn M., Kunzi H.P. (Eds.), Optimization and Operations Research, Lecture Notes in Economics and Mathematical Systems, Berlin, Springer, vol.157, 45-54.

Calinski T., and Harabasz J., 1974. A dendrite method for cluster analysis, Communication in Statistics, 3(1), 1-27.

Chiou Y.C., and Lan L.W., 2001. Theory and Methodology Genetic clustering algorithms, European Journal of operational Research, 135, 413-427.

Everitt B.S., 1993. Cluster Analysis. Halsted Press, third edition.

Falkenauer E., 1998. Genetic Algorithms and Grouping Problems, John Wiley and Son, Chichester.

Fogel D.B., 1998. Evolutionary Computation: The Fossil Record. IEEE Press, Piscataway, NJ.

Fogel L.J., Owens A.J. and Walsh M.J., 1966. Artificial intelligence through simulated evolution, New York, John Wiley.

Forgy E.W., 1965. Cluster Analysis of Multivariate Data: Efficiency versus Interpretability of classification, Biometrics, 21, 768-9.

Friedman H.P., and Rubin J., 1967. On some invariant criterion for grouping data, Journal of the American Statistical Association 63, 1159-1178.

Holland J.H., 1975. Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Harbor.

Kennedy J. and Eberhart R.C., 1995. Particle swarm optimisation.In: Proceedings of the 1995 IEEE International Conference on Neural Networks,vol. 4, IEEE Press, Piscataway, NJ, 1942-1948.

Krishna K., and Murty M.N., 1999. Genetic K-means Algorithm, IEEE Transaction on systems, man and cybernetics, 29.

Kuncheva L., 1995. Editing for the k-nearest neighbors rule by a genetic algorithm, Pattern Recognition Letters, 16, 809-814.

Marriott F.H.C., 1982. Optimization methods of cluster analysis, Biometrics, 69, 2, 417-422.

Maulik U., and Bandyopadhyay S., 2000. Genetic algorithm-based clustering technique, Pattern Recognition, 33, 1455-1465.

Merz C., Murphy P., and Aha D., 1997. UCI repository of Machine Learning databases. Department of Information and Computer Science, University of California, Irvine. http://www.ics.uci.edu/mlearn/MLRepository.html.

Murthy C.A., and Chowdury N., 1996. In search of optimal clusters using genetic algorithm, Pattern Recognition Letters, 17, 825-832.

Paterlini S., and Minerva T., 2003. Evolutionary Approaches for Cluster Analysis. In A. Bonarini, F. Masulli, G. Pasi (eds.) Soft Computing Applications. Springer-Verlag, Berlin. 167-178.

Raghavan V.V., and Birchand K., 1979. A clustering strategy based on a formalism of the reproductive process in a natural system. In: Proceedings of the Second International Conference on Information Storage and Retrieval, 10-22.

Rechenberg I., 1973. Evolutionsstrategie: Optimierung Technischer Systeme nach Prinzipien der Biologischen Evolution, Frommann-Holzboog, Stuttgart.

Sarafis I, Zalzala A.M.S, and Trinder P., 2002. A Genetic Rule-Based Data Clustering Toolkit. In: Fogel D.B., El-Sharkawi M.A., Yao X., Greenwood G., Iba H., Marrow P., and Shackleton M., 2002, Proceedings of the 2002 Congress on Evolutionary Computation CEC2002, IEEE Press, 1238-1243.

Schwefel, H-P., 1975. Evolutionsstrategie und Numerische Optimierung, Dissertation, Technical University of Berlin.

Srikanth R., George R., Warsi N., Prabhu D., Petri F.E., and Buckles B.P, 1995. A variable-length genetic algorithm for clustering and classification, Pattern Recognition Letters, 16, 789-800.

Storn R., and Kenneth P., 1995. Differential Evolution - a Simple and Efficient Adaptive Scheme for Global Optimization over Continuous Spaces. Technical Report TR-95-012, ICSI.

Tseng L.Y., and Yang S.B., 2001. A genetic approach to the automatic clustering problem, Pattern Recognition, 34, 415-424.

Xiao X., Dow E., Eberhart R., Ben Miled Z. and Oppelt R.J., 2003. Gene Clustering using Self-Organizing Maps and Particle Swarm Optimization. In: Proceeding of Second IEEE International Workshop on High Performance Computational Biology, Nice, France.

Appendix A

Table 8: Parameters configuration of the random generative processes of the artificial datasets. Column 1 reports the dataset types, column 2 the profile data matrix structure (e.g. the profile matrix ***X*** is made of three sub-matrices X_i (*i=1,..,3*) of size 20x3 with data generated from multivariate normal distribution with parameters as reported in column 4) , column 3 the size of the sub-matrices and the generative random process and column 4 the numerical parametric values for each random variable. Datasets are random realizations of the multivariate normal random variables $X_1, \dots X_9$, with parameters specified in column 4.

DATASET	DATA Matrix	Distribution	Moments
s1	$X = \begin{bmatrix} X_1 \\ X_2 \\ X_3 \end{bmatrix}$	X1~N(μ1,Σ1) (20x3)	μ1=[1,1,1],Σ1=I
		X2~N(μ2,Σ2) (20x3)	μ2=[5,5,5],Σ2=I
		X3~N(μ3,Σ3) (20x3)	μ3=[9,9,9],Σ3=I
s2	$X = \begin{bmatrix} X_1 \\ X_2 \\ X_3 \end{bmatrix}$	X1~N(μ1,Σ1) (20x3)	μ1=[1,1,1],Σ1=2I
		X2~N(μ2,Σ2) (20x3)	μ2=[5,5,5],Σ2=3I
		X3~N(μ3,Σ3) (20x3)	μ3=[9,9,9],Σ3=2I
s3	$X = \begin{bmatrix} X_1 \\ \dots \\ X_6 \end{bmatrix}$	X1~N(μ1,Σ1) (20x3)	μ1=[1,1,1],Σ1=I
		X2~N(μ2,Σ2) (20x3)	μ2=[5,5,5],Σ2=I
		X3~N(μ3,Σ3) (20x3)	μ3=[9,9,9],Σ3=I
		X4~N(μ4,Σ4) (20x3)	μ4=[-3,-3,-3],Σ4=I
		X5~N(μ5,Σ5) (20x3)	μ5=[-7,-7,-7],Σ5=I
		X6~N(μ6,Σ6) (20x3)	μ6=[-11,-11,-11],Σ6=I
s4	$X = \begin{bmatrix} X_1 \\ \dots \\ X_6 \end{bmatrix}$	X1~N(μ1,Σ1) (20x3)	μ1=[1,1,1],Σ1=2I
		X2~N(μ2,Σ2) (20x3)	μ2=[5,5,5],Σ2=3I
		X3~N(μ3,Σ3) (20x3)	μ3=[9,9,9],Σ3=2I
		X4~N(μ4,Σ4) (20x3)	μ4=[-3,-3,-3],Σ4=3I
		X5~N(μ5,Σ5) (20x3)	μ5=[-7,-7,-7],Σ5=2I
		X6~N(μ6,Σ6) (20x3)	μ6=[-11,-11,-11],Σ6=3I
s5	$X = \begin{bmatrix} X_1 \\ \dots \\ X_9 \end{bmatrix}$	X1~N(μ1,Σ1) (20x3)	μ1=[1,1,1],Σ1=I
		X2~N(μ2,Σ2) (20x3)	μ2=[5,5,5],Σ2=I
		X3~N(μ3,Σ3) (20x3)	μ3=[9,9,9],Σ3=I
		X4~N(μ4,Σ4) (20x3)	μ4=[-3,-3,-3],Σ4=I
		X5~N(μ5,Σ5) (20x3)	μ5=[-7,-7,-7],Σ5=I
		X6~N(μ6,Σ6) (20x3)	μ6=[-11,-11,-11],Σ6=I
		X7~N(μ7,Σ7) (20x3)	μ7=[-15,-15,-15],Σ7=I
		X8~N(μ8,Σ8) (20x3)	μ8=[13,13,13],Σ8=I
		X9~N(μ9,Σ9) (20x3)	μ9=[17,17,17],Σ9=I
s6	$X = \begin{bmatrix} X_1 \\ \dots \\ X_9 \end{bmatrix}$	X1~N(μ1,Σ1) (20x3)	μ1=[1,1,1],Σ1=2I
		X2~N(μ2,Σ2) (20x3)	μ2=[5,5,5],Σ2=3I
		X3~N(μ3,Σ3) (20x3)	μ3=[9,9,9],Σ3=2I
		X4~N(μ4,Σ4) (20x3)	μ4=[-3,-3,-3],Σ4=3I
		X5~N(μ5,Σ5) (20x3)	μ5=[-7,-7,-7],Σ5=2I
		X6~N(μ6,Σ6) (20x3)	μ6=[-11,-11,-11],Σ6=3I
		X7~N(μ7,Σ7) (20x3)	μ7=[-15,-15,-15],Σ7=2I
		X8~N(μ8,Σ8) (20x3)	μ8=[13,13,13],Σ8=3I
		X9~N(μ9,Σ9) (20x3)	μ9=[17,17,17],Σ9=2I