Journal of Heuristics, 9: 113–129, 2003

Self-Adaptive Genetic Algorithm for Clustering

JUHA KIVIJÄRVI
Turku Centre for Computer Science (TUCS), Department of Information Technology, University of Turku, FIN-20014 Turku, Finland
email: juhkivij@utu.fi

PASI FRÄNTI
Department of Computer Science, University of Joensuu, PB 111, FIN-80101, Joensuu, Finland

OLLI NEVALAINEN
Turku Centre for Computer Science (TUCS), Department of Information Technology, University of Turku, FIN-20014 Turku, Finland

Abstract

Clustering is a hard combinatorial problem which has many applications in science and practice. Genetic algorithms (GAs) have turned out to be very effective in solving the clustering problem. However, GAs have many parameters, the optimal selection of which depends on the problem instance. We introduce a new self-adaptive GA that finds the parameter setup on-line during the execution of the algorithm. In this way, the algorithm is able to find the most suitable combination of the available components. The method is robust and achieves results comparable to or better than a carefully fine-tuned non-adaptive GA.

Key Words: clustering, evolutionary computing, genetic algorithms, self-adaptation

1. Introduction

Clustering is a *combinatorial optimization problem* where the aim is to partition a set of data objects into a predefined number of clusters in such a way that similar objects are grouped together and dissimilar objects in separate groups (Everitt, 1992; Kaufman and Rousseeuw, 1990). Clustering has many applications in social sciences, numerical taxonomy, computer science, data compression and image analysis. The size and dimensionality of the data are often very high in these applications, which makes manual processing practically impossible. High quality computer based clustering is therefore desired.

The general clustering problem includes three subproblems: (1) selection of the evaluation function, (2) decision of the number of groups, and (3) selection of the clustering algorithm. Here we consider the last subproblem and assume that the number of clusters has been fixed a priori. The evaluation function depends on the application and the type of data objects. Minimization of intra-cluster diversity is widely used and it is therefore applied here as well.

There are several established methods for generating a clustering algorithmically (Everitt, 1992; Kaufman and Rousseeuw, 1990; Gersho and Gray, 1992). The most cited and widely used method is the *k-means algorithm* (McQueen, 1967). It begins with an initial solution, which is iteratively improved using two different optimality criteria in turn until a local minimum has been reached. The algorithm is easy to implement and it gives reasonable results in most cases.

Clusterings of very good quality have been obtained by *genetic algorithms* (GA) (Fränti et al., 1997). However, there are many parameters in genetic algorithms that should be tuned. Since the running times are typically high, the tuning of the parameters is time-consuming. The situation becomes even more complicated because the optimal parameter values depend on each other and on the problem instance in question. For example, in Fränti et al. (1997) it was observed that the optimal selection method depends on the crossing method used. Furthermore, the optimal parameter values may change during the execution of the algorithm.

In this paper we propose a *self-adaptive genetic algorithm* (SAGA) for the clustering problem. The aim is to give an effective and straight-forward algorithm which obtains good solutions for the optimization problem without explicit parameter tuning. The self-adaptation takes place at individual level (Hinterding, Michalewicz, and Eiben, 1997). This means that each individual of the GA population contains a set of parameter values. These parameters are used in the reproduction process.

The idea of SAGA is that in addition to the solutions, also the parameter values go through genetic optimization. In particular, we adapt the *crossover method*, the *mutation probability* and the *noise range*. We consider six crossing methods of which the best one is supposed to become dominative. The algorithm controls also the amount of stochastic disturbance (which is divided into mutations and noise) applied to a solution. It is expected that these parameters may have different values at different stages of the solution process.

We will see that the results of SAGA are fully comparative with those of a carefully refined non-adaptive genetic algorithm. The major benefit of SAGA is that the refining phase can be omitted completely. The algorithm finds the best or at least a satisfying parameter combination and thus is able to obtain a good solution practically always.

The rest of the paper is organized as follows. We begin in Section 2 by giving a formal definition of the clustering problem. The basic genetic algorithm is described in Section 3. In Section 4 we show how to modify the genetic algorithm in order to make it self-adaptive. The performance of the algorithm is studied experimentally in Section 5. Conclusions are drawn in Section 6.

2. Clustering problem

2.1. *Definition of the problem*

The clustering problem is defined as follows. Given a set of N data objects x_i, partition the data set into M clusters in such a way that similar objects are grouped together and objects with dissimilar features belong to different groups. Partition (P) defines a clustering by giving for each object the index (p_i) of the cluster where it is assigned to. Each cluster j is described by its representative data object (c_j).

2.2. *Object dissimilarity*

Each object x_i has K attributes x_i^k ($k = 1, 2, \ldots, K$), which together form a *feature vector*. For simplicity, we assume that the attributes are numerical and have the same scale. If this is not the case, the attributes must first be normalized. We suppose that the dissimilarity (distance) between two objects x_1 and x_2 is measured by the *Euclidean distance* of the two feature vectors. Euclidean distance is a widely used distance function in the clustering context, and it is calculated as

$$d(x_1, x_2) = \sqrt{\sum_{k=1}^{K} \left(x_1^k - x_2^k\right)^2}. \tag{1}$$

Note that our method is not limited to Euclidean distance.

2.3. *Evaluation of clustering*

The most important choice in the clustering method is the objective function for evaluating the quality of a clustering. The choice of the function depends on the particular application in question and there is no universal solution which could be applied. A commonly used objective criterion is to minimize the sum of squared distances of the data objects to their cluster representatives. Given a partition $P = (p_1, p_2, \ldots, p_N)$ and the cluster representatives $C = (c_1, c_2, \ldots, c_M)$, the *mean squared error* (*MSE*) $e(P, C)$ is calculated as

$$e(P, C) = \frac{1}{NK} \sum_{i=1}^{N} d\left(x_i, c_{p_i}\right)^2. \tag{2}$$

Given a partition P, the optimal choice for the cluster representatives C minimizing (2) are the cluster *centroids*, calculated as

$$c_j = \frac{\sum_{P_i = j} x_i}{\sum_{P_i = j} 1}, \quad 1 \le j \le M, \quad 1 \le i \le N. \tag{3}$$

2.4. *Generating clustering*

Once the objective function has been selected, the clustering problem can be formulated as a combinatorial optimization problem. The problem of finding an optimal partition P equals to the problem of finding the set of optimal cluster centroids C. This is due to the fact that if one of these is given, an optimal choice for the other one can be constructed. This is formalized in the following two optimality conditions (Gersho and Gray, 1992):

– *Nearest neighbour condition*: For a given set of cluster centroids, any data object can be optimally classified by assigning it to the cluster whose centroid is closest to the data object in respect to the distance function.

– *Centroid condition*: For a given partition, the optimal set of cluster representatives (i.e., the one minimizing the distortion) is the set of cluster centroids.

These optimality conditions are used for example in the k-means algorithm, where they are applied in turn. Thus, k-means never moves to a worse solution. When used as a stand-alone method, k-means tends to end up at a possibly weak local optimum. However, the method is highly useful as a fine-tuning operation in more sophisticated methods (Fränti et al., 1997; Fränti, Kivijärvi, and Nevalainen, 1998; Fränti and Kivijärvi, 2000; Kaukoranta, Fränti, and Nevalainen, 1998)

2.5. *Number of clusters*

In applications such as *vector quantization* (Gersho and Gray, 1992), the number of clusters (M) is merely a question of resource allocation. It is therefore a parameter of the clustering algorithm. In some other applications the number of clusters must also be solved. The choice of the correct number of clusters is an independent problem and should be considered separately. The decision is typically made by a researcher of the application area but analytic methods also exist (Gyllenberg, Koski, and Verlaan, 1997; Dubes and Jain, 1987) for helping in doing this. A common approach is to generate several clusterings for various values of M and compare them against some evaluation criteria. In the rest of this paper we assume that the number of cluster is fixed a priori.

3. Genetic algorithm

In this section we describe the traditional genetic algorithm on which we base the self-adaptive genetic algorithm. This genetic algorithm produces solutions of very high quality. The initial setup has only a minor effect on the results of GA. The method is therefore robust and, unlike the k-means algorithm, it does not get stuck at a local optimum but examines the solution space on a wider range.

An individual ι in the traditional genetic algorithm codes a solution ω_ι of the clustering problem. In this case the solution is everything an individual contains. The reason for this conceptual distinction between individual and solution is that an individual in SAGA will also include additional parameter information, as we will see.

Solution ω_ι consists of partition P_{ω_ι} and cluster centroids C_{ω_ι}. The general structure of the traditional genetic algorithm is following (Fränti et al., 1997):

```
1. Generate $S$ random individuals to form the initial generation.
2. Iterate the following $T$ times.
    (a) Select $S_B$ surviving individuals for the new generation.
    (b) Select $S - S_B$ pairs of individuals as the set of parents.
    (c) For each pair of parents $(\iota_a, \iota_b)$ do the following:
        (i) Create the solution $\omega_{\iota_n}$ of the offspring $\iota_n$ by crossing the solutions of the parents.
        (ii) Mutate $\omega_{\iota_n}$ with probability $\psi$.
```

(iii) Apply k-means to ω_{ι_n}.
(iv) Add ι_n to the new generation.

(d) Replace the current generation by the new generation.

3. Output the best solution of the final generation.

Note that the overall best solution is preserved in the final generation because $S_B > 0$ (step 2a).

3.1. Selection method

A selection method defines a way a new generation of S individuals is constructed from the current one. Selection method consists of the following three steps:

– determining S_B survivors
– selecting a crossing set of S_C individuals
– selecting pairs for crossover from the crossing set

We utilize two selection methods: *roulette wheel selection* and *elitist selection*. In the roulette wheel selection method only the best individual survives ($S_B = 1$) and the crossing set consists of all the individuals ($S_C = S$). For the crossover, $S - 1$ pairs of individuals are randomly chosen. The probability for the individual ι_i to be selected in the crossover is

$$p(\iota_i) = \frac{\frac{1}{1+e(\omega_{\iota_i})}}{\sum_{j=1}^{S} \frac{1}{1+e(\omega_{\iota_j})}} \tag{4}$$

where ι_j stand for the individuals in the current population and ω_{ι_j} the corresponding solutions.

In the elitist selection method, the crossover set consists of the surviving S_B individuals ($S_C = S_B$). All the individuals in the crossover set are crossed with each other so that the crossover phase produces $\frac{S_C(S_C-1)}{2}$ new individuals. This results in population size $S = \frac{S_C(S_C+1)}{2}$. Other population sizes could be implemented simply by discarding a suitable number of individuals. Here we use $S_C = 9$, giving $S = 45$.

3.2. Crossover operations

The crossover operation produces a new solution ω_n from two parent solutions ω_a and ω_b. We have implemented six crossover methods in this study. A brief description of these methods follows.

Method 1: *Random multipoint crossover* (Delport and Koschorreck, 1995) is performed by picking $\frac{M}{2}$ randomly chosen cluster centroids from each of the two parents in turn. Duplicate centroids are rejected and replaced by repeated picks.

Method 2: In the *centroid distance crossover* (Pan, McInnes, and Jack, 1995) the clusters are sorted according to their distances from the centroid of the entire data set. The new solution is created by taking the central clusters from ω_a and the remote clusters from ω_b.

Method 3: The *largest partitions crossover* (Fränti et al., 1997) aims to assign the cluster centroids to areas with many data objects. This is done by selecting the centroids of the largest clusters, i.e. the clusters with the greatest number of data objects. The data objects that belong to the clusters already selected are not considered in further calculations.

Method 4: In the *multipoint pairwise crossover* (Fränti et al., 1997) the clusters of solutions ω_a and ω_b are first paired in a greedy manner in such a way that each cluster of ω_a has a pair in ω_b so that the centroids of these clusters are close to each other. One cluster centroid is then randomly picked from each pair.

Method 5: *One-point pairwise crossover* (Fränti et al., 1997) differs from the multipoint pairwise crossover in the selection of the cluster centroids. Instead of selecting one random cluster centroid from each pair, the first $\frac{M}{2}$ cluster centroids are taken from ω_a and the rest cluster centroids from ω_b. The latter are selected in such a way that the clusters do not form a pair with the already selected cluster centroids.

Method 6: In the *pairwise nearest neighbour crossover* (Fränti et al., 1997) the parents are merged into one solution of $2M$ clusters. The solution is then reduced to size M using the *pairwise nearest neighbour algorithm* (PNN) (Equitz, 1989) for clustering. This can be implemented effectively so that we avoid the full scale recalculation of the partition after the crossover phase (Fränti, 2000).

The largest partitions crossover, one-point pairwise crossover and pairwise nearest neighbour crossover are deterministic in the sense that for two fixed parent solutions ω_a and ω_b they always end up with the same solution. Thus, with these methods the genetic variation can be lost if sufficient randomization, such as mutation, is not applied in the other steps of the algorithm.

3.3. *Mutations*

Each cluster centroid is replaced by a randomly chosen data object with probability ψ. This operation is performed after the crossover. Formally, a mutation makes the change $C_i = (c_1, \ldots, c_M) \to C_i^* = (c_1^*, \ldots, c_M^*)$, where $c_j^* = c_j$ with probability $1 - \psi$ and $c_j^* = x_{r_j}$ with probability ψ. In the previous formula, r_j is randomly chosen from $[1, N]$ and thus x_{r_j} is randomly selected from the set of all data objects. In essence, mutations implement a single randomized step of local search.

3.4. *Noise*

After the mutation, noise is added to the centroids. This is done by adding a random vector to each centroid. The component values of this vector are in the range $[-\nu, \nu]$. Typically one forces ν to decrease during the execution of the algorithm. Noise adding implements stochastic relaxation (Zeger and Gersho, 1989), if the amount of noise decreases as a function of time.

Note that the concepts of noise and mutation are rather close to each other. It is quite common to use only one of them. In this paper we describe and use both operations to give the algorithm more freedom of operation.

3.5. *Fine-tuning by the k-means algorithm*

Finally, G iterations of the k-means algorithm are applied for fine-tuning the new solutions. This compensates the fact that the crossover, mutation and noise operations are not very likely to produce better solutions as such. We will discuss later the effect of the value of G to the quality of the result.

4. SAGA—Self-adaptive genetic algorithm

The selection of optimal parameter values for the genetic algorithm is tedious. Thus an adaptive system, where the parameter values are automatically adapted on the basis of the problem instance, would be highly desired. We use the *individual level self-adaptation* (Hinterding, Michalewicz, and Eiben, 1997), where each individual has a set of parameters of its own. The parameter values affect this individual only. In the crossover and mutation steps the parameters are changed in the same manner as the crossover and mutation change the solution.

We consider three parameters: *crossover method* γ, *mutation probability* ψ and *noise range* ν. Thus, an individual ι is of the form $\iota = (\omega_\iota, \gamma_\iota, \psi_\iota, \nu_\iota)$ where $\omega_\iota = (P_{\omega_\iota}, C_{\omega_\iota})$. The parameters of an individual ι are used for creating the solution ω_ι. The general structure of the self-adaptive genetic algorithm is following:

1. Generate S random individuals to form the initial generation.
2. Iterate the following T times.
   - (a) Select S_B surviving individuals for the new generation.
   - (b) Select $S - S_B$ pairs of individuals as the set of parents.
   - (c) For each pair of parents (ι_a, ι_b) do the following:
     - (i) Determine the parameter values $(\gamma_{\iota_n}, \psi_{\iota_n}, \nu_{\iota_n})$ for the offspring ι_n by crossing the parameters $(\gamma_{\iota_a}, \psi_{\iota_a}, \nu_{\iota_a})$ and $(\gamma_{\iota_b}, \psi_{\iota_b}, \nu_{\iota_b})$ of the two parents.
     - (ii) Mutate the parameter values of ι_n with the probability Ψ (a predefined constant).
     - (iii) Create the solution ω_{ι_n} by crossing the solutions of the parents. The crossing method is determined by γ_{ι_n}.
     - (iv) Mutate the solution of the offspring with the probability ψ_{ι_n}.
     - (v) Add noise to ω_{ι_n}. The maximal noise is ν_{ι_n}.
     - (vi) Apply k-means iterations to ω_{ι_n}.
     - (vii) Add ι_n to the new generation.
   - (d) Replace the current generation by the new generation.
3. Output the best solution of the final generation.

The next three subsections describe the parameters from the viewpoint of adaptation. The parameters are divided into three classes: adaptive parameters, adaptation control parameters and non-adaptive parameters.

4.1. Adaptive parameters

The SAGA adapts three parameters: crossover method γ, mutation probability ψ and noise range ν. These have a direct effect on the individual. The key questions regarding the adaptive parameters are the effect of the parameters and the method used for mutating them. The crossing of the parameters is done simply by selecting the corresponding parameter value randomly from one of the parents.

The *crossover method* parameter γ specifies the crossover method used to create the new solution. The mutation of the crossover method parameter is implemented simply as a random selection of one of the six crossover methods.

The *mutation probability* parameter ψ specifies the probability of a mutation. In the mutation, each cluster centroid is replaced by a randomly selected data object with probability ψ. When the mutation probability parameter is mutated, a random number drawn from a Gaussian distribution is added to the current value of ψ. The mutation probability parameter is restricted to the range $[\psi_{min}, \psi_{max}]$.

The *noise range* parameter ν specifies the range of the noise to be added to the cluster centroids. The mutation of the noise range parameter is done in the same way as the mutation of the mutation probability parameter: a random number is drawn from a Gaussian distribution and added to the current parameter value. The value is restricted to the range $[\nu_{min}, \nu_{max}]$.

4.2. Adaptation control parameters

The adaptation of the parameters brings along new parameters that control the adaptation. The actual settings of these *adaptation control parameters* are, however, not very critical on the performance of the algorithm. From the viewpoint of a user, the parameters in 4.1 have been replaced by these parameters. They have mostly obvious settings or only a minor effect on the outcome of the algorithm, as we will show later.

The *limits of the mutation probability* ψ_{min} and ψ_{max} specify the range for the mutation probability parameter. We fix the parameters to the most obvious values. We let $\psi_{max} = 1$, which means that all the cluster centroids are mutated for this parameter value. The lower limit is $\psi_{min} = 0$, which means that no mutations occur.

Similarly the *noise range limits* ν_{min} and ν_{max} specify the range for the values of the noise range parameter. The lower limit is set to the obvious $\nu_{min} = 0$ since the inclusion of noise is a rather radical operation and it is thus not always desired. This is true especially in the later stages of the algorithm. The upper limit ν_{max} is set to the maximal value of a feature.

The most important new parameter and the only one which has no obvious setting is the *parameter mutation probability* Ψ. It specifies the probability with which a parameter of a new individual is mutated after its value has been decided in the parameter crossover. The probability should be large enough to make sure that new parameter combinations occur and small enough to enable slow convergence of the parameter values. This parameter should be of minor importance on the outcome of the algorithm compared to the original parameters.

4.3. Non-adaptive parameters

Several parameters are not adapted. The reasons to this and the selection of these parameter values are discussed next.

The *number of individuals in the population* (S) is not adapted. If S is increased, better results will be obtained at the cost of a higher running time. Our adaptation method does not adapt the parameters on the basis of the computation time elapsed. Furthermore, the population size is a population level parameter, not an individual level parameter as the previous parameters.

A k-means iteration never weakens a solution. Thus, in order to be able to adapt the *number of k-means iterations*, we should have a criterion which would allow us to decrease this number. Time could again be such a criterion. Because we do not consider the running time in the adaptation, the number of k-means iterations is not adapted but fixed as $G = 2$, which was found out to be a good choice in Fränti et al. (1997).

The effect of a *selection method* can only be seen at the population level, not at the individual level. Thus adapting it at the individual level is implausible. We have experimented on two selection methods, roulette wheel selection and elitist selection, see Section 5.

5. Test results

We use the following default parameter values: parameter mutation probability $\Psi = 5\%$, number of the k-means iterations $G = 2$, population size $S = 45$ and the roulette wheel selection method. Each test run consists of $T = 1000$ generations and the results are averages of $R = 20$ test runs. Each test run has a different randomly selected initial population. Initial solutions were formed by randomly drawing M distinct data objects from the data set and using these as cluster representatives.

We have used four test problems, three of which originate from vector quantization. *Bridge* consists of 4×4 spatial pixel blocks sampled from a gray-scale image with image depth of 8 bits per pixel. *Bridge2* has the blocks of *Bridge* after a BTC-like quantization into two values according to the average pixel value of the block. The cluster representatives are rounded to binary vectors. *Lates mariae* records 215 data samples from pelagic fishes of Lake Tanganyika. The data originates from a biological research, in which the occurrence of 52 different DNA fragments was tested for each fish sample using RAPD analysis and a binary decision was obtained whether the fragment was present or absent. The cluster representatives are real vectors. *Miss America* has been obtained by subtracting two subsequent image frames of an video image sequence and constructing 4×4 spatial pixel blocks from the residuals. Table 1 shows the dimensions of these test problems.

Table 2 shows results of several clustering methods for the test problems. The MSE values for Bridge2 have been multiplied by the number of attributes (16) so that the error can be interpreted as average number of distorted attributes per data object. Included methods are k-means (McQueen, 1967), Splitting method with local repartitioning (SLR) (Fränti,

Table 1. Dimensions of the test problems.

Data set	Attributes	Objects	Clusters
Bridge	16	4096	256
Bridge2	16	4096	256
Lates mariae	52	215	8
Miss America	16	6480	256

Table 2. Comparison of several clustering methods. $R = 100$ for k-means and SR, $R = 1$ for SLR and Ward's method, $R = 20$ for others.

	Bridge		Bridge2		Lates mariae		Miss America	
	Av. MSE	St. dev.	Av. MSE	St. dev.	Av. MSE	St. dev.	Av. MSE	St. dev.
k-means	180.073	1.442	1.489	0.015	0.0703	0.0054	5.963	0.056
SLR	170.221	0.000	1.362	0.000	0.0662	0.0000	5.398	0.000
Ward's method	169.253	0.000	1.429	0.000	0.0627	0.0000	5.507	0.000
SR	162.607	0.275	1.469	0.014	0.0683	0.0045	5.265	0.013
RLS-2	164.220	0.251	1.264	0.004	0.0626	0.0000	5.262	0.019
GA	161.403	0.089	1.263	0.004	0.0626	0.0000	5.108	0.004
SAGA	161.183	0.102	1.252	0.003	0.0626	0.0000	5.100	0.003

Kaukoranta, and Nevalainen, 1997), Ward's method (Ward, 1963), Stochastic relaxation (SR) (Zeger and Gersho, 1989), Randomised local search (RLS-2) (Fränti and Kivijärvi, 2000), Genetic algorithm (GA) (Fränti et al., 1997) and SAGA. The results of SLR and Ward's method were not repeated since the methods are deterministic. The results of k-means and Stochastic relaxation were repeated 100 times. We can see that the results of GA and SAGA are in most cases better than those of the other methods included. The exceptions are for Bridge2 and Lates mariae. Even though most methods are especially unable to find a good solution for Bridge2 since only binary vectors are allowed to represent clusters, RLS-2 is able to find as good results as GA. In the case of Lates mariae, GA, SAGA and RLS-2 reach the same solution every time. Since the problem is a relatively easy one, this solution is likely to be the global optimum.

Table 3 shows a more detailed comparison between SAGA and the ordinary GA for Bridge. In GA we use the best parameter setup found in Fränti et al. (1997): the probability of the mutations is 1%, PNN serves as the crossover method, and no noise is added to the solutions. We have also included the results of GA using random crossover and the results of SAGA using only PNN crossover. The results show that SAGA achieves consistently better clustering results that GA, which indicates that the adaptation gives extra benefit besides finding the best parameter combination. Even though the differences are rather small, the deviation of the results is very small and thus the difference in results is statistically very significant when tested with Student's t-test ($p < 10^{-7}$). The differences on Bridge2 ($p < 10^{-10}$) and Miss America ($p < 10^{-7}$) sets have a similar significance, see Table 2.

Table 3. Comparison between SAGA and the ordinary genetic algorithm for Bridge. $R = 20$.

	Av. MSE	Best MSE	St. dev.	Av. time
SAGA	161.183	161.049	0.102	15:56:50
SAGA (PNN only)	161.404	161.219	0.107	18:06:04
GA (PNN crossover)	161.403	161.289	0.089	11:31:49
GA (random crossover)	174.089	173.602	0.246	10:43:22

Table 4. The effect of parameter mutation probability for Bridge. $R = 20$.

	Av. MSE	Best MSE	St. dev.	Av. time
$\Psi = 0\%$	166.829	161.137	5.860	20:02:22
$\Psi = 0.5\%$	161.177	160.936	0.150	14:13:04
$\Psi = 5\%$	161.183	161.049	0.102	15:56:50
$\Psi = 50\%$	161.871	161.353	0.216	15:18:41

SAGA can also utilize the different crossover techniques, which is verified by the results of SAGA using only PNN crossover. Even though PNN crossover is the most effective crossover method for Bridge (Fränti et al., 1997), the results of SAGA using all the available crossover methods are statistically better ($p < 10^{-7}$) than the results with only PNN crossover. The results of GA using random crossover show that a good parameter setup is indeed vital.

The differences in running times are explained by the optimizations in the k-means implementation (Kaukoranta, Fränti, and Nevalainen, 2000). Since some noise is added to solutions in SAGA, practically all the cluster representatives change and the optimizations do not speed up the calculation of the partition considerably. The usage of faster crossover methods compensates the time difference somewhat. This is confirmed by considering the running times of SAGA with only PNN crossover.

Next we study how the non-adaptive parameters (4.3) and adaptation control parameters (4.2) affect the output of SAGA. Table 4 demonstrates the effect of changing the parameter mutation probability Ψ. It turns out that the parameter mutations are necessary as the results are clearly inferior for $\Psi = 0\%$. However, the exact value of Ψ is not crucial but all the tested values ($\Psi > 0$) gave good results. We selected $\Psi = 5\%$ as the default value based on preliminary testing. The final results show that there is no statistically significant difference between the results of $\Psi = 0.5\%$ and $\Psi = 5\%$.

Table 5 shows the effect of the population size. In order to make the results comparable, we have adjusted the number of iterations so that in each case an approximately equal number of solutions is evaluated. A larger population size results in a more robust algorithm. Sizes from 15 up seem to work well. If the population size is very small, SAGA is not always able to find a good parameter setup and thus may end up with a poor result. The increased deviations demonstrate this very well. Also, a larger population is capable of maintaining more genetic variation and therefore able to improve the result longer.

Table 5. The results of different population sizes for Bridge. $R = 20$.

	Iterations	Av. MSE	Best MSE	St. dev.	Av. time
$S = 3$	7000	178.098	161.069	32.372	4:33:33
$S = 6$	2800	163.701	160.745	8.306	4:05:15
$S = 10$	1556	165.997	160.750	10.223	4:54:34
$S = 15$	1000	161.780	160.879	3.319	4:37:55
$S = 28$	519	161.180	161.008	0.137	5:00:07
$S = 45$	318	161.342	161.083	0.143	5:12:51

Table 6. The effect of k-means iteration count G for Bridge. $R = 20$.

	Av. MSE	Best MSE	St. dev.	Av. time
$G = 0$	161.344	161.088	0.148	13:11:00
$G = 1$	161.253	160.981	0.117	11:23:22
$G = 2$	161.183	161.049	0.102	15:56:50
$G = 4$	161.148	160.982	0.108	22:02:47
$G = 6$	161.130	160.998	0.094	26:21:46
$G = 8$	161.124	160.843	0.125	28:43:46
$G = 10$	161.183	161.050	0.094	32:05:22

Table 7. Comparison of the selection methods for Bridge. $R = 20$.

	Iterations	Av. MSE	Best MSE	St. dev.	Av. time
Roulette wheel	818	161.205	161.051	0.110	13:05:22
Elitist	1000	161.443	161.223	0.171	5:28:36

Table 6 demonstrates the effect of the number of k-means iterations G. It turns out that G has a rather small effect on the object function. Even without k-means the results are very good, which is mainly due to the efficiency of the PNN crossover method. The differences in the results when $G > 1$ are quite insignificant, but the running time steadily increases along the number of k-means iterations. Thus our default choice $G = 2$ seems to be a good one.

The roulette wheel selection and elitist selection methods are compared in Table 7. The elitist selection method gives significantly shorter running times but the premature convergence weakens the results in comparison to the roulette wheel selection method. The number of iterations has again been adjusted to balance the difference in the number of evaluated solutions. The shorter running time of the elitist method is mainly due to our effective k-means implementation (Kaukoranta, Fränti, and Nevalainen, 2000) which benefits from the fact that after convergence the changes in the solutions are very small.

Figures 1–3 demonstrate the power of adaptation by showing the development of active values of the crossover method parameter for typical test runs. The graphs show the frequency of each crossover technique applied at different generations. These figures clearly demonstrate the power of the adaptation. For Bridge, PNN is the dominating crossover method. In figure 1, PNN is the most frequently used method at the beginning and at the end of the solution process. However, the one-point pairwise method dominates the generations from 300 to 850. In the second run, see figure 2, the PNN crossover is rather dominative most of the time and never completely disappears.

In figure 3 the elitist selection method is used for Bridge2. This shows two important differences to the previous two runs. Firstly, the PNN crossover does not dominate in any

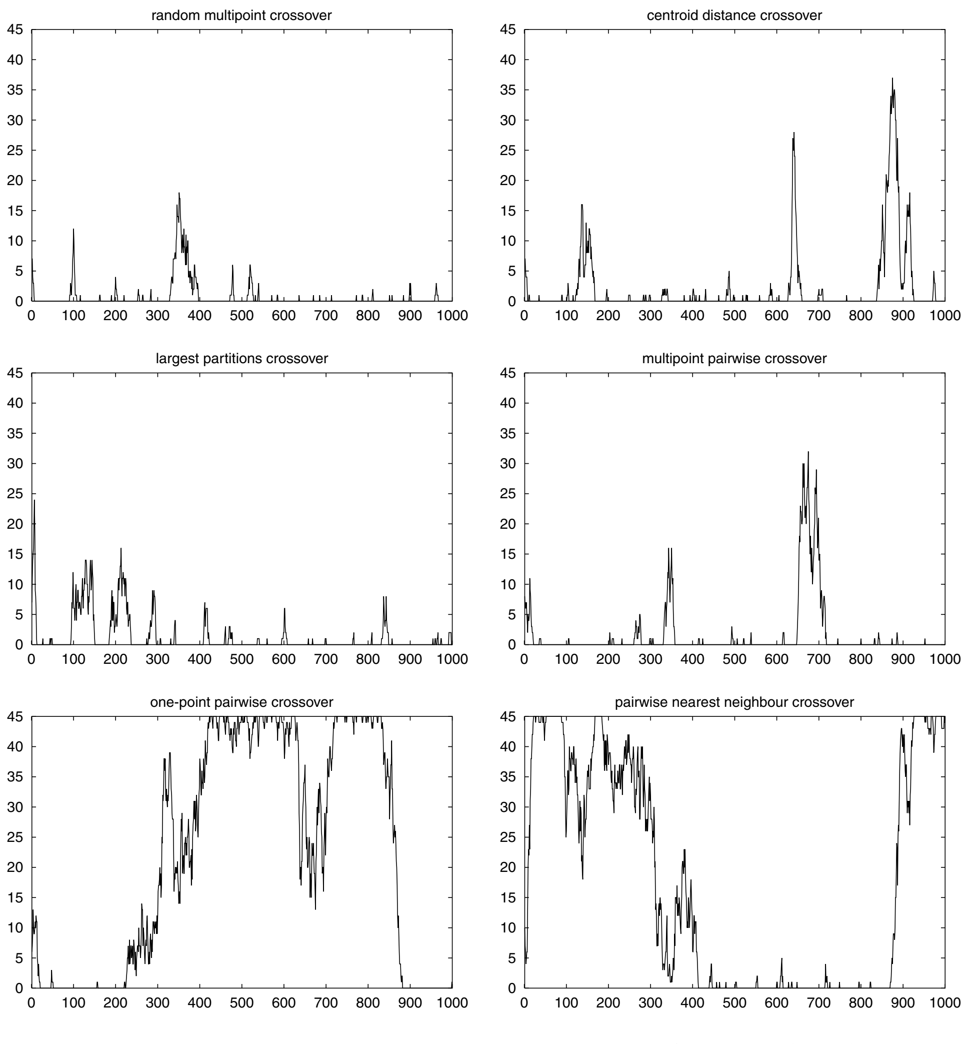

Figure 1. Frequency of different crossover methods for Bridge, with $\Psi = 0.5\%$. MSE = 160.9726.

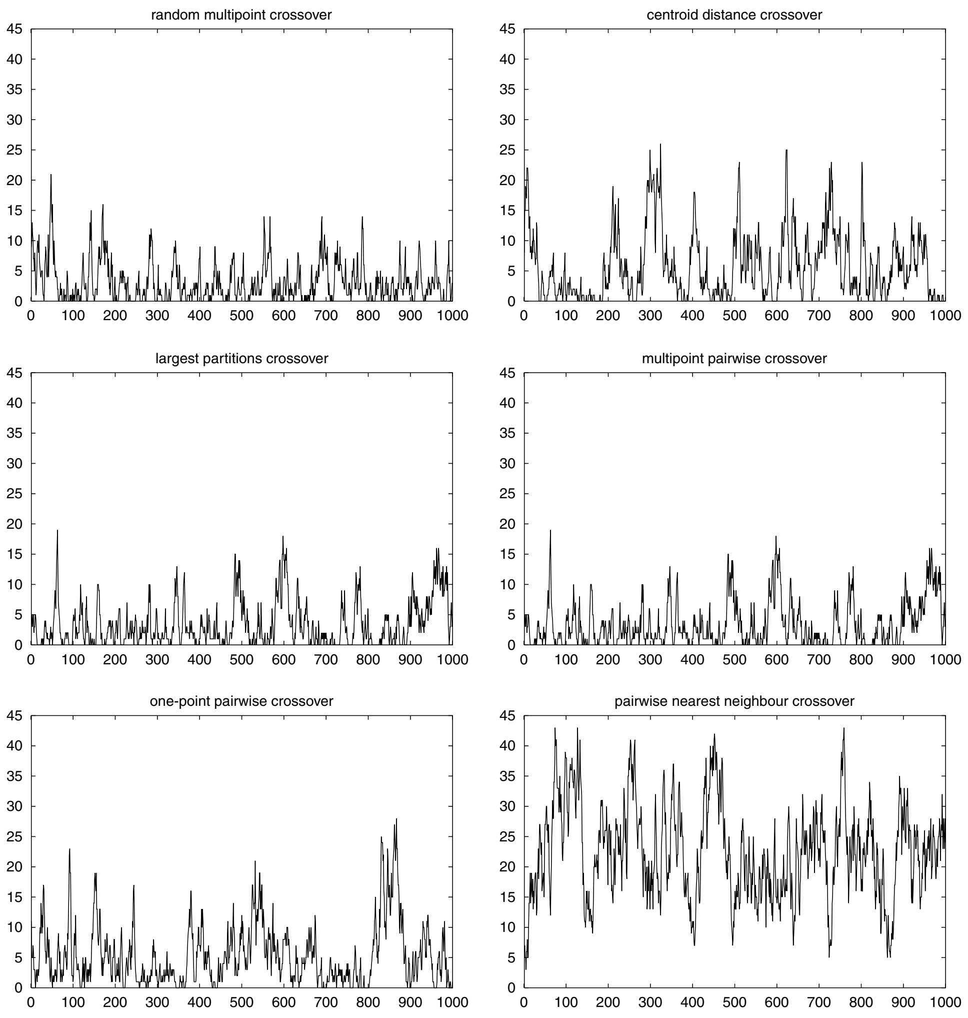

Figure 2. Frequency of different crossover methods for Bridge, with $\Psi = 5\%$. MSE = 161.0574.

stage. Instead, the centroid distance, one-point pairwise and largest partitions crossover methods each have their dominative phases. Secondly, the changes in the frequencies are very sudden. This is due to the elitist selection method which allows a successful crossing method rapidly become more popular.

In Tables 8–10 we analyze the successfulness of the crossing methods (see Section 3). The tables show for different generation ranges the number of times the individual crossover methods produce a new locally best solution. The runs are the same as in figures 1–3. We see that the PNN crossover is very effective for Bridge in the beginning of the run. In the later generations the situation may be different. For Bridge2, most of the improvements in

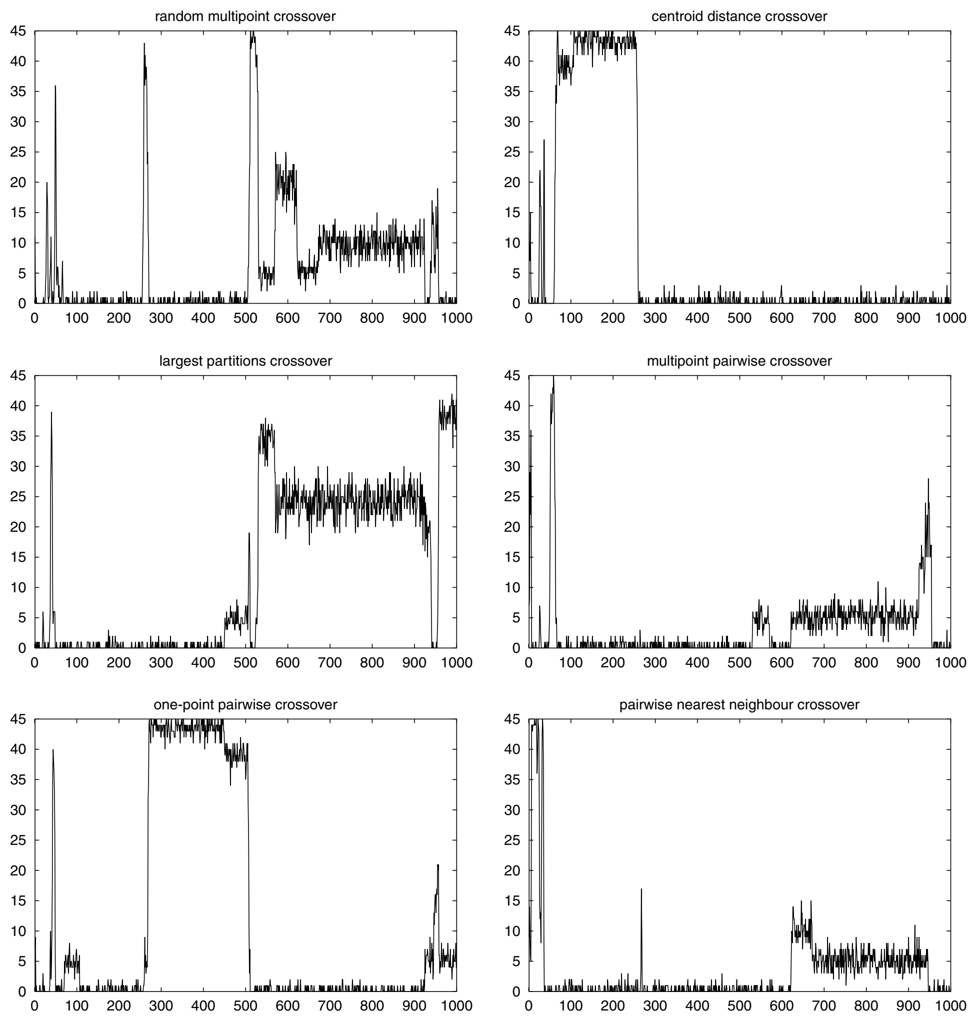

Figure 3. Frequency of different crossover methods for Bridge2. Elitist selection method. MSE = 1.2561.

the later stages are obtained by the random multipoint crossover. However, other methods have also succeeded.

6. Conclusions

A self-adaptive genetic algorithm was proposed for the clustering problem. The method automatically adjusts its parameters during the execution of the algorithm. SAGA was able to give consistently better results than a carefully fine-tuned non-adaptive GA. In addition, the results on a binary test set demonstrate that the optimal parameter combination may be

Table 8. Frequencies of the crossing methods improving the best solution. See figure 1.

	Crossover method					
Iterations	1	2	3	4	5	6
1–200	0	0	0	0	1	34
201–400	0	0	0	0	12	2
401–600	0	0	0	0	3	0
601–800	0	0	0	0	1	0
801–1000	0	2	0	0	6	4

Table 9. Frequencies of the crossing methods improving the best solution. See figure 2.

	Crossover method					
Iterations	1	2	3	4	5	6
1–200	1	1	0	1	0	33
201–400	0	0	0	0	0	8
401–600	0	0	0	0	0	4
601–800	0	0	0	0	0	5
801–1000	0	0	0	0	0	3

Table 10. Frequencies of the crossing methods improving the best solution. See figure 3.

	Crossover method					
Iterations	1	2	3	4	5	6
1–200	7	12	1	12	1	17
201–400	4	0	0	0	1	0
401–600	7	0	1	0	1	0
601–800	2	0	0	0	0	0
801–1000	4	0	1	1	0	0

different for different problem instances, and therefore self-adaptation can be highly useful in such situations.

The experiments show that the parameter settings of SAGA have minor effect on the results. Parameter mutation is necessary, but the exact amount is non-essential. Small population sizes sometimes lead into a situation where a good parameter combination was not found and the result was inferior. Population sizes from 15 up worked very well. The effects of a large population size are increased robustness and better preservation of genetic variation. The number of k-means iterations was quite insignificant. Elitist selection method converges fast and is thus unable to achieve as good results as roulette wheel selection.

To sum up, the new self-adaptive clustering algorithm gives very high quality results for hard problem instances. On the other hand one must pay for this with a rather large time consumption of the solution process.

An interesting topic of further research could be the population level adaptation (Hinterding, Michalewicz, and Eiben, 1997), which allows the consideration of the running time. Thus, also the parameters in Section 4.3 could be adapted. Another benefit of the population level adaptation would be the possibility to optimize the time usage of the algorithm.

References

Delport, V. and M. Koschorreck. (1995). “Genetic Algorithm for Codebook Design in Vector Quantization.” *Electronics Letters* 31, 84–85.

Dubes, R. and A. Jain. (1987). *Algorithms that Cluster Data*. New Jersey: Prentice-Hall, Englewood Cliffs.

Equitz, W.H. (1989). “A New Vector Quantization Clustering Algorithm.” *IEEE Trans. Acoustics, Speech Signal Processing* 37, 1568–1575.

Everitt, B.S. (1992). *Cluster Analysis*, 3rd edn. London: Edward Arnold/Halsted Press.

Fränti, P. (2000). “Genetic Algorithm with Deterministic Crossover for Vector Quantization.” *Pattern Recognition Letters* 21, 61–68.

Fränti, P., T. Kaukoranta, and O. Nevalainen. (1997). “On the Splitting Method for VQ Codebook Generation.” *Optical Engineering* 36, 3043–3051.

Fränti, P. and J. Kivijärvi. (2000). “Randomised Local Search Algorithm for the Clustering Problem.” *Pattern Analysis & Applications* 3, 358–369.

Fränti, P., J. Kivijärvi, T. Kaukoranta, and O. Nevalainen. (1997). “Genetic Algorithms for Large Scale Clustering Problems.” *The Computer Journal* 40, 547–554.

Fränti, P., J. Kivijärvi, and O. Nevalainen. (1998). “Tabu Search Algorithm for Codebook Generation in VQ.” *Pattern Recognition* 31, 1139–1148.

Gersho, A. and R.M. Gray. (1992). *Vector Quantization and Signal Compression*. Dordrecht: Kluwer Academic Publishers.

Gyllenberg, M., T. Koski, and M. Verlaan. (1997). “Classification of Binary Vectors by Stochastic Complexity.” *Journal of Multivariate Analysis* 63, 47–72.

Hinterding, R., Z. Michalewicz, and A.E. Eiben. (1997). “Adaptation in Evolutionary Computation: A Survey.” In *Proceedings of the 4th IEEE International Conference on Evolutionary Computation*, Indianapolis, pp. 65–69.

Kaufman, L. and P.J. Rousseeuw. (1990). *Finding Groups in Data: An Introduction to Cluster Analysis*. New York: John Wiley & Sons.

Kaukoranta, T., P. Fränti, and O. Nevalainen. (1998). “Iterative Split-and-Merge Algorithm for VQ Codebook Generation.” *Optical Engineering* 37, 2726–2732.

Kaukoranta, T., P. Fränti, and O. Nevalainen. (2000). “A Fast Exact GLA Based on Code Vector Activity Detection.” *IEEE Transactions on Image Processing* 9, 1337–1342.

McQueen, J.B. (1967). “Some Methods of Classification and Analysis of Multivariate Observations.” In *Proc. 5th Berkeley Symp. Mathemat. Statist. Probability* 1, Univ. of California, Berkeley, USA, pp. 281–296.

Pan, J.S., F.R. McInnes, and M.A. Jack. (1995). “VQ Codebook Design Using Genetic Algorithms.” *Electronics Letters* 31, 1418–1419.

Ward, J.H. (1963). “Hierarchical Grouping to Optimize an Objective Function.” *J. Amer. Statist. Assoc.* 58, 236–244.

Zeger, K. and A. Gersho. (1989). “Stochastic Relaxation Algorithm for Improved Vector Quantiser Design.” *Electronics Letters* 25, 896–898.