Babeş - Bolyai" University, Cluj-Napoca
Faculty of Mathematics and Computer Science
Computational intelligence Reports
2005

A New Algorithm within Genetic Chromodynamics. Applications to Function Optimization and Classification

Catalin Stoean and D. Dumitrescu

Abstract

A recently developed radii-based evolutionary algorithm designed to solve multimodal optimization problems is presented. The approach can be placed within the genetic chromodynamics framework. The basic motivation for modifying the original algorithm was to preserve its ability to search for many optima in parallel while increasing convergence speed, especially for more complex problems, by adopting generational selection and different replacement schemes.

Presented algorithm is applied to function optimization and used as an engine for a learning classifier system; the latter is applied for two classification problems. Obtained experimental results encourage further investigation.

1. Introduction

Current report presents a recently created radii-based evolutionary algorithm and proves its efficiency in solving various problems. The algorithm is inspired from a metaheuristic presented in [Dum00b], namely Genetic Chromodynamics (GC), that has been largely used in recent years for function optimisation, clustering or classification ([Dum00b], [Gor04], [Sto04], [Sto05a]).

The new algorithm (presented in this report) within the GC framework, called Elitist Generational GC (EGGC), is an algorithm that speeds up convergence and, at the same time, looks into the search space for more accurate approximations of the solutions. The new algorithm is applied for function optimization, as well as for two classification problems; results obtained are compared with those of the classical algorithm in the GC framework and with those of other models applied for the same ([Sto05a], [Sto05b], [Sto05c], [Sto05d]).

The report starts with a presentation of the classical algorithm in the GC framework in order to introduce the changes that lead to EGGC in section 3. Section 4 contains applications of the new algorithm to the optimization of several multimodal, *n*-dimensional functions. Next section contains the integration of EGGC into a learning classifier system and application to two classification problems.

2. Genetic Chromodynamics Framework

GC belongs to the family of radii-based multimodal evolutionary frameworks, as it builds and maintains subpopulations connected each to local or global optima of the problem to be solved. This is achieved by introducing a set of restrictions such as the way selection is applied or the way recombination takes place. For selection, each chromosome represents a *stepping-stone* for the forming of the new generation – each chromosome is taken into account for reproduction. If a chromosome has no similar individuals to it, then it mutates. Consequently, for reproduction a local interaction principle is considered, meaning that only chromosomes similar under a given threshold recombine. After either recombination or mutation takes place, the offspring fights for survival with the stepping-stone parent.

GC introduces a new operator that *merges* very similar chromosomes into a single one that is often chosen to be the best one of them with respect to fitness evaluation (Algorithm 1). It is a very useful operator as it leads to a better computational time, obtained by reducing

the size of the population, meaning thus less fitness evaluations. Consequently, subpopulations independently evolve and become better separated with each iteration and lead, at convergence, each one to an optimum.

```
Begin
      Repeat
            A chromosome c is considered to be the current one;
            Select all m individuals in the merging region of c,
            including itself;
            Remove all but the best chromosome from the selection;
      Until merging cannot be applied at all
End
```

Algorithm 1. Merging procedure in GC

GC is able to concentrate search on many basins of attraction in parallel, so that several optima are found simultaneously. The evolutionary process takes place as follows. First, the initial population is randomly generated. Next, every chromosome is taken into account in the forming of the new generation; mating regions around each chromosome are determined by a radius. Therefore, only neighbouring chromosomes are recombined. When no mate is found in the mating region of the current chromosome, the latter produces one offspring by mutation, with a step size that still keeps the descendant in the mating region of its parent. If there is more than one chromosome *near* the current, the mate is determined using proportional selection. Then, if the offspring has better fitness than the current chromosome, it replaces the latter in the population. Figure 1 illustrates the way crossover, mutation and merging take place.

In conclusion, there are two important parameters:

- *mating radius* parameter that is used for finding a mate for crossover for the current chromosome; in Figure 1, the mating radius represents the radius of the circle that has the centre in c_2.
- *merging radius* parameter used for detecting the chromosomes that will be merged.

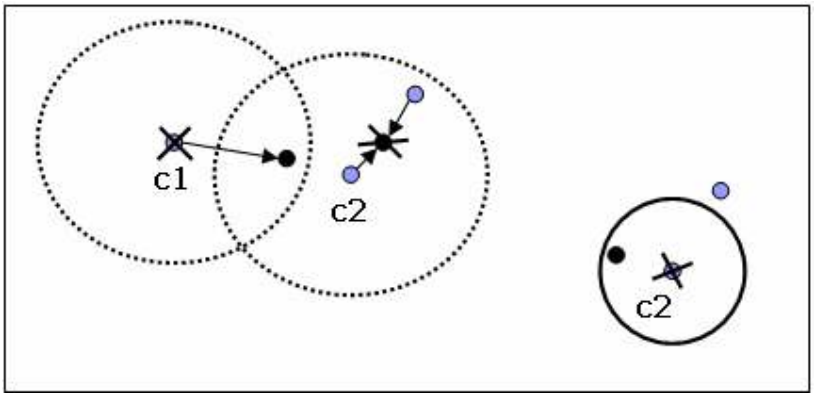


Figure 1. Mating (left) and merging (right) within GC. Chromosome c_1 produces one offspring by mutation, chromosome c_2 selects another chromosome from its mating region (dotted circle) and produces one offspring by crossover. Crossed lines indicate replaced chromosomes with worse fitness. During merging, c_2 is deleted because there is another chromosome with better fitness in its merging region (solid circle).

Figure 2 presents a general scheme of the GC algorithm, as flow-chart.

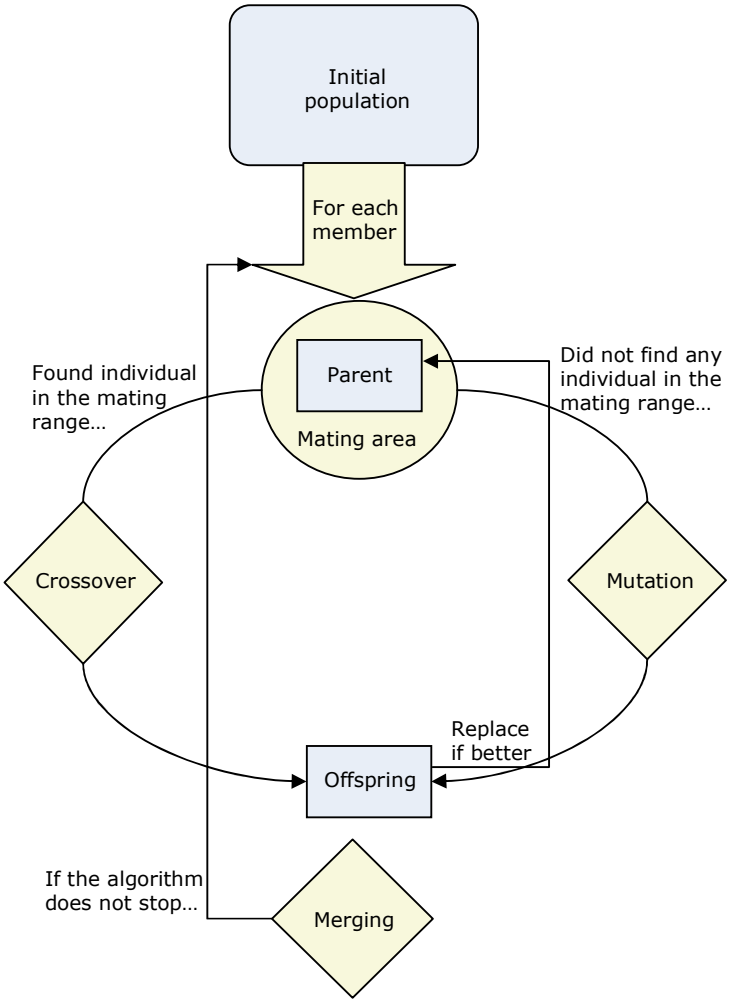


Figure 2. General scheme of a GC algorithm

By reducing potential partners for crossover of a chromosome to those lying in its mating region, only chromosomes that are close to each other recombine, favouring the appearance and maintenance of subpopulations. Offspring replaces the current chromosome only if fitter. Therefore, after a few generations, the chromosomes will concentrate on the most promising regions of the search space, *i.e.* those connected to the optima. Algorithm 2 outlines the pseudocode of GC.

```
Begin
      Initialize population;
      While termination condition is not satisfied
         ▪ Evaluate each chromosome;
         ▪ For all chromosomes c in the population do
               o If mating region of c is empty then
                     ▪ Apply mutation to c;
                     ▪ If obtained chromosome is fitter than c
                       then replace c;
                     ▪ End if
               o Else
                     ▪ Select one chromosome from the mating
                       region of c for crossover;
                     ▪ Obtain and evaluate one offspring;
                     ▪ If offspring is fitter than c then replace
                       c;
                     ▪ End if
               o End if
         ▪ End for
         ▪ Merging
      End while
End
```

Algorithm 2. GC Algorithm

The stop condition of the algorithm may refer to a pre-specified number of steps (generations) for which the algorithm will run or to a fixed number of generations without any improvement for which the algorithm should run or to a specified accuracy of the optimum that the algorithm must reach etc.

3. Elitist Generational Genetic Chromodynamics

The new approach achieves a better exploitation of the search space for the three changes made in the classical algorithm (Algorithm 2). First, it proposes a random search mechanism, instead of the deterministic stepping stone principle. Second, it uses a generational strategy for selection, *i.e.*, when a new chromosome is obtained, it immediately enters the current population. Finally, the descendant obtained after crossover replaces the worst chromosome in its region instead of its parent.

Consequently, the algorithm has an increased random character, as the chromosomes that form the next generation are randomly taken from the population. And as the newly generated chromosomes are immediately entering the current population and moreover, as they are replacing the worst chromosomes in their region, the model achieves an increased convergence speed.

In the initialization of the population, a number n of chromosomes are considered – the values for genes are randomly taken from their intervals. The distance between two chromosomes is computed. Let a chromosome c be considered: the distance between c and all the other chromosomes in the population is calculated. The mating region of c contains all chromosomes that are at a distance from c of less than a given threshold that represents the mating radius.

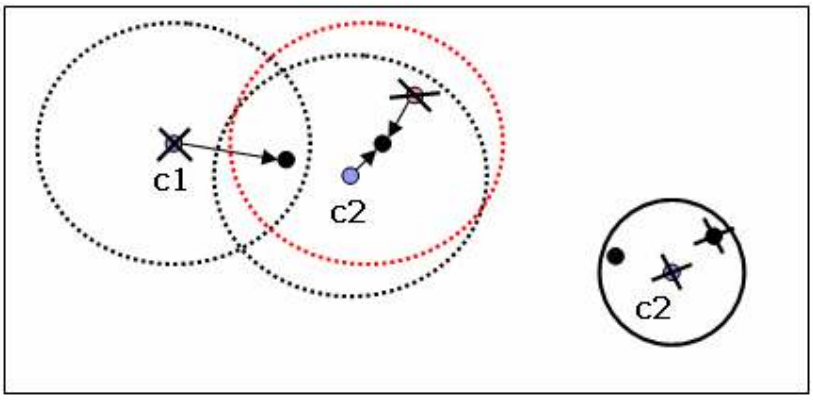


Figure 3. Mating (left) and merging (right) within EGGC. As in Figure 1, c_1 and c_2 each produce one offspring. This time, the second offspring replaces its other parent – one of the chromosomes in its replacement area (red dotted circle) - because the latter is the worst chromosome within its replacement region. During merging, two chromosomes are removed, c_2 and one offspring, because now three chromosomes are within merging radius from c_2.

Like in GC case, crossover is usually considered to take place only between pairs of two chromosomes and one offspring is obtained from these two parents. In order to enter the current generation, the offspring fights with the chromosomes in its personal replacing region. Mutation causes only minor perturbation to a chromosome. The local interaction, crossover, mutation and merging principles still hold. Radii-based evolution in the new context of EGGC is depicted in Figure 3 and the complete procedure is outlined in Algorithm 3.

```
Begin
      Initialize population;
      While termination condition is not satisfied
         ▪ Evaluate each chromosome;
         ▪ For i = 1 to n do
               o Randomly pick a chromosome c;
               o If mating region of c is empty then
                     ▪ Apply mutation to c;
                     ▪ If obtained chromosome is fitter than c
                       then replace c;
                     ▪ End if
               o Else
                     ▪ Select one chromosome from the mating
                       region of c for crossover;
                     ▪ Obtain and evaluate offspring d;
                     ▪ Find worst chromosome w within replacement
                       radius of d;
                     ▪ If d is fitter than w then replace w;
                     ▪ End if
               o End if
         ▪ End for
         ▪ Merging
      End while
End
```

Algorithm 3. EGGC algorithm

In EGGC algorithm, the original scheme was modified in order to achieve increased convergence speed based on better exploitation of the search space. That is obtained especially by the way in which offspring resulting from crossover enters the population. It does not replace the first parent, but the worst chromosome in its replacement radius.

Therefore, weak chromosomes are removed more aggressively (alongside with the effect merging has in this respect) from the current population. This is the motivation for not applying the stepping stone principle here, but *n* (where *n* is the number of chromosomes in the population) random chromosomes are selected instead. Now chromosomes may be replaced by some offspring without ever being selected for recombination. An important aspect of the algorithm is the choice of the replacement radius value. If picked properly, this new parameter may lead to improved convergence speed.

Selection for replacement adopts a generational scheme, as already stated. As the effect of quasi-generational selection for replacement, used by the classical GC, can be noticed only when another stepping-stone does not find the initial chromosome in its mating region but sees the offspring instead, the new algorithm is totally generational. This means that the offspring that replaces its parent might be selected for reproduction many times in the same generation or might vanish within that generation. Thus, the generational scheme leads to increased exploitation.

4. Application of EGGC to Function Optimization

The EGGC algorithm was tested on three bi-dimensional functions and an *n*-dimensional one (Table 1) and obtained results indicate that it has good accuracy, stability, low computational time and thus provides a good method for multimodal function optimization.

FUNCTION	F	EXPRESSION	DOMAIN
Six-Hump Camel Back	f_1	$-[(4-2.1x^2+\frac{x^4}{3})x^2+xy+4(y^2-1)y^2]$	$[-2, 2]^2$
Schaffer	f_2	$-0.5 - \frac{\sin^2(\sqrt{x^2+y^2})-0.5}{(1+0.001(x^2+y^2))^2}$	$[-20, 20]^2$
Himmelblau	f_3	$200-(x^2+y-11)^2-(x+y^2-7)^2$	$[-4, 4] \times [-6, 6]$
Schwefel	f_4	$\sum_{i=1}^{n} x_i \sin(\sqrt{\lvert x_i \rvert})$	$[-500, 500]^2$

Table 1. The four considered multimodal test problems

The aim for the *Six-Hump Camel Back* function is to locate the two global optima and the four other local optima; they are reviewed in Table 2.

Optimum type	Coordinates (x, y)	$f_1(x, y)$
Global	(-0.0898, 0.7126)	1.0316
Global	(0.0898, -0.7126)	1.0316
Local	(-1.7036, 0.7961)	0.2155
Local	(1.7036, -0.7961)	0.2155
Local	(-1.6071, -0.5687)	-2.1043
Local	(1.6071, 0.5687)	-2.1043

Table 2. Values for the global and local optima of the *Six-Hump Camel Back* function

The major difficulty for the f_1 function (Figure 4) is that the two local optima with the same value -2.1043 can easily be missed by a multimodal evolutionary algorithm. For instance, in [Lee01], several algorithms were used (a genetic algorithm using gradient information, a local optimization method, a multi-start local optimization method and a conventional EA), but each time only one local or global optimum was detected with an accuracy of $\varepsilon = 10^{-3}$; in [Zah04], not all optima were found every time the algorithm was run.

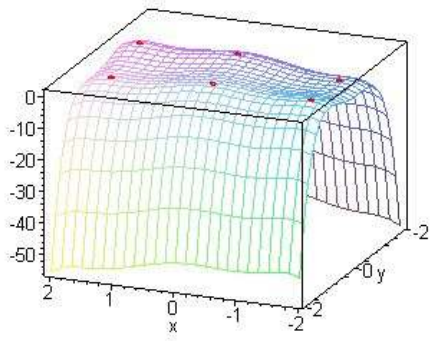


Figure 4. Six-Hump Camel Back function and the optima points located by proposed model

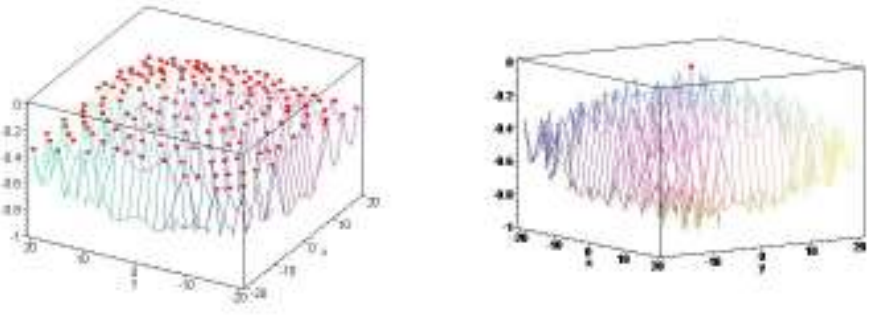
Because of lack of information, direct comparisons to other algorithms could not be conducted. In Table 3, results obtained by the original GC algorithm, EGGC and two algorithms presented in [Zah04] are outlined. The values for the EGGC parameters used for f_1 are shown in Table 8 The success rate was computed as the ratio between the number of cases

when all optima have been located with the desired accuracy and the total number of runs. For the two algorithms from [Zah04], local optima accuracies were not specified. Although the number of generations in EGGC was lower than that in the classical algorithm of GC, the number of evaluations in EGGC was a little higher; the explanation is that after each crossover, there are some more evaluations that are performed for all chromosomes in the replacement area of the offspring. In conclusion, for this function, the two algorithms perform in a very similar manner.

Measures	**CDE and MCDE**	**MMDE**	**GC**	**EGGC**
No. of runs	30	30	30	30
Accuracy global optima	10^{-5}	10^{-5}	10^{-5}	10^{-5}
Accuracy local optima	-	-	10^{-4}	10^{-4}
Success rate (%)	100	100	100	100
Mean evaluation calls	62 645	14 610	19 923	19 980

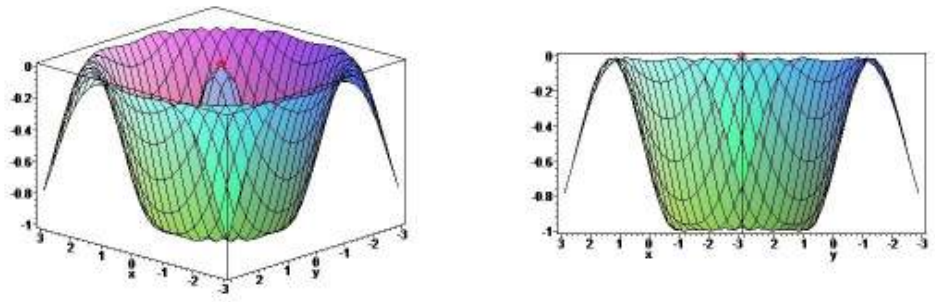
Table 3. Comparisons of algorithms for the Six-Hump Camel Back function

The aim for the *Schaffer* function (Figure 5) is to detect the global optimum $f_2(x, y) = 0$ that can be escaped because of the high number of local optima around it and because the difference between the values of the local optima and the value of the global optimum is very small (of the order 10^{-3}). This is the reason why parameters for f_2 were chosen such (Table 8) that the entire search space was covered.



(a) (b)

Figure 5. The Schaffer function. (a) All optima detected by EGGC algorithm (b) The global optimum detected by the algorithm.



(a) (b)

Figure 6. Same Schaffer function represented on a smaller interval, $[-3, 3]^2$. There are two different view points here: (a) shows the detected global optimum (the hill), its surrounding valley and the nearby encircling hills.; (b) illustrates the slight difference between the global optimum and the surrounding local optima.

While in Figure 5 it cannot be easily noticeable how small the difference between the global optimum and the very close local optima around it is, Figure 6 takes us closely to the global optimum area, by reducing the function interval representation. As noticeable from Table 4, the original GC algorithm, although performing well, needs many more evaluations than EGGC.

Measures	**GC**	**EGGC**
No. of runs	30	30
Accuracy global optimum	10^{-6}	10^{-6}
Success rate (%)	100	100
Mean evaluation calls	658 211	349 712

Table 4. Performance of GC and EGGC on the Schaffer function

The *Himmelblau* function has four global optima (Figure 7), $f_3(x, y) = 200$.

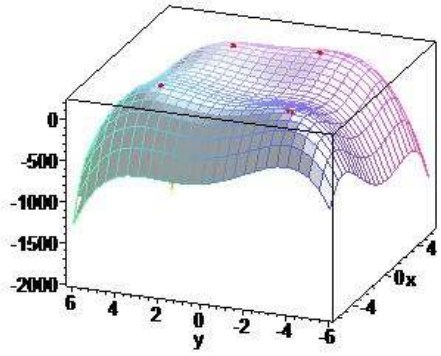


Figure 7. Himmelblau function and its global optima found by EGGC algorithm

In [Bea93], the optimization of Himmelblau function was tested using a sequential niche technique; again, obtained results cannot be directly compared with those obtained by GC/EGGC due to the fact that in [Bea93], in order to detect all the four optima, several runs (6.1, on average) of the algorithm were necessary. Additionally, the desired accuracy was not specified. Using the EGGC algorithm, all four optima were detected for each of the 30 runs with the accuracy $\varepsilon = 10^{-5}$. A performance comparison between GC and EGGC is given in Table 5.

Measures	GC	EGGC
No. of runs	30	30
Accuracy global optima	10^{-4}	10^{-4}
Success rate (%)	100	100
Mean evaluation calls	87 307	97 817

Table 5. Performance of GC and EGGC on the Himmelblau function

Schwefel function (Figure 8 illustrates the two-dimensional case) is a well known benchmark function which is very misleading for evolutionary algorithms; it has a high number of local optima and the global one can easily be missed. The aim for it is to find the global optimum $f_4(\bar{x}) = 418.9829n$. In this paper, f_4 was considered in turn for $n = 1, 2, \ldots 100$. Although radii-based evolutionary algorithms usually have difficulties with high-

dimensional test problems, EGGC algorithm detects the optimum with an accuracy of $\varepsilon = 10^{-2}$, even for n = 100, in all 100 runs, but at the expense of a high number of fitness evaluations. Accuracy improves when decreasing the value of n for the function.

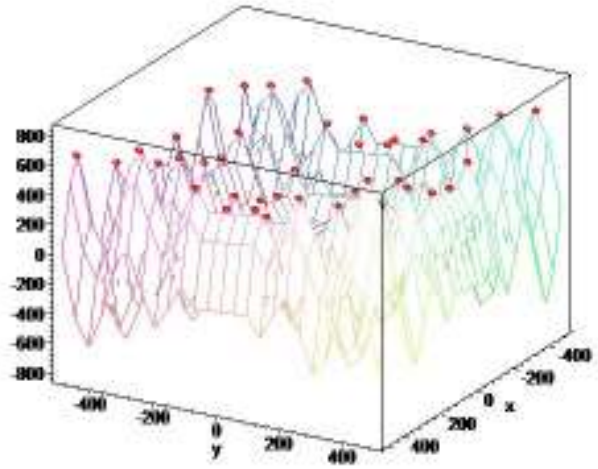


Figure 8. Schwefel function for n = 2 and the optima detected by EGGC algorithm

The Schwefel function was also tested with the original GC algorithm which detected the optimum for n = 100 with similar accuracy, but needed even more fitness evaluations than EGGC. We present a comparison of both algorithms and different performance measures for n = 10 and $\varepsilon = 10^{-2}$ in Table 4.6.

Measures	GC	EGGC
No. of runs	30	30
Mean evaluation calls	1 819 593	1 545 752
Mean generations	48 747	39 043
Mean best fitness	4189.827822	4189.827956

Table 6. GC and EGGC performance on the Schwefel function when n = 10

Comparisons of results obtained after applying EGGC for the Schwefel function were performed to those in [Gord93]. In the latter, authors presented the performance of four global evolutionary algorithms, four island algorithms and one cellular algorithm, on the shifted

Schwefel function with n = 1, 2,..., 10. In the first category, a simple genetic algorithm (SGA), an elitist SGA (ESGA), the parallel CHC (pCHC) and the Genitor algorithm were present. The second category contained island-SGA (I-SGA), island elitist SGA (I-ESGA) island-pCHC (I-pCHC) and island-Genitor (I-Genitor). The algorithms were run for a set number of generations and the number of runs (out of 30) in which the global optimum was found (denoted by *ns*) was reported, along with the mean best fitness of the 30 runs (denoted by MBF in Table 7). Note that fitness values are normalized so that the global optimum is at 0.0.

Algorithm	Ns	MBF
SGA	0	17.4
ESGA	16	17.3
pCHC	15	5.9
Genitor	20	13.2
I-SGA	9	16.5
I-ESGA	13	2.6
I-pCHC	28	0.2
I-Genitor	24	0.9
Cellular	26	0.7

Table 7. Performance of different evolutionary techniques on the Schwefel function taken from [Gord93]

Table 8 contains the empirically determined EGGC parameter values for the four functions. Mutation with normal perturbation was used in all cases; the value of the mutation strength directly depends on the size of the interval and implicitly on the search space size. In the case of the Schwefel function, higher mutation strength was chosen in the beginning, to escape local optima. The value was afterwards decreased by $n/100$ every 100 generations, where n represents the number of dimensions. The values for mating and merging radii were chosen in the same manner, thus depending on the search space size. Replacement radius for a given chromosome is generally chosen to be equal to the mating value of that individual. However, for problems with large plateaus in the fitness landscape, by choosing a different value for the replacement from that of the mating radius, faster convergence can be achieved (see Tables 6 and 8 vs. Table 9 for f_4). On the other hand, for problems with very close optima, its value must be chosen again not equal to the mating radius (see f_1 in Table 8), in

order to obtain all optima. The last parameter from Table 8 gives the number of generations without any improvement which is necessary to determine termination of the algorithm. For the Schaffer and Schwefel functions, the stop condition was established to be the moment when the algorithm achieves the desired accuracy because only the global optimum had to be determined.

Parameters	f_1	f_2	f_3	f_4
No. of chromosomes	100	2 000	150	500
Mutation strength	0.1	0.1	0.5	25n
Mating radius	0.14	0.1	2	15n
Replacement radius	0.1	0.1	2	15n
Merging radius	0.14	0.1	0.02	12n
Mutation probability	0.4	0.4	0.4	0.4
No. improvement times	100	-	50	-

Table 8. Parameters of the EGGC algorithm for all functions

To strengthen the assumption that for the tested high dimensional problem, EGGC is really faster than the original GC algorithm, a recent parameter tuning method, SPO, developed by colleagues at Dortmund University, ([Bar04], [Bar05]), was applied for the 10 and 20-dimensional Schwefel function. The response value Y (quality criterion) was set to the average number of evaluations (AES) until reaching the global optimum with accuracy 10^{-2} (Table 9).

Parameter name	**N/R**	**Min**	**Max**	**GC**	**EGGC**
No. of chromosomes	**N**	10	500	10	13
Mating radius	$\mathbf{R}_+$	0.01	5.0	3.6752	2.9366
Replacement radius	$\mathbf{R}_+$	0.01	5.0	-	4.748
Merging radius	$\mathbf{R}_+$	0.01	5.0	1.275	3.9197
Mutation strength	$\mathbf{R}_+$	1.0	250.0	238.45	231.08

Mutation probability	$\mathbf{R}_+$	0.0	1.0	0.6185	0.8575
Mutation scaledown	**N**	10	1000	368	625
No. improvement times	**N**	1	200	167	46
AES, 32 runs	-	-	-	1.69E+5	1.45E+5
AES, standard deviation	-	-	-	5.92E+4	3.63E+4

Table 9 Algorithm design for optimizing the 10-dimensional Schwefel function with GC and EGGC. The two last columns give the best found configurations, the last lines the resulting AES measures with standard deviations.

Results show that for both algorithms the AES decreased considerably, to ≅10% of the original values reported in Table 6. A t-test confirms that the true means for GC and EGGC are different with 95% confidence (p-value 0.036), so one may conclude that EGGC is indeed faster on this problem. However, the difference is quite small - around 15% - and may be statistically significant, but of little importance for practical uses. Surprisingly, many optimized parameter values are very similar for the two algorithms, *e.g.* number of chromosomes, mating radius and mutation strength and probability. In contrast to that, the merging radii are chosen differently; larger than the mating radius for EGGC and smaller for GC. Nevertheless, other good configurations found during tuning indicate that smaller merging radii also work for EGGC.

From these first results, and supported by our previous findings, we learned that EGGC may be advantageous for higher dimensional problems. Consequently, we performed a second comparison for the same test function in 20 dimensions, this time allowing for larger radius values. The outcome (Table 10) unambiguously favours EGGC over GC, thereby validating our assumption. However, the maximum radii tested may still be too small; increasing them may lead to even better speedup.

Parameter name	**N/R**	**Min**	**Max**	**GC**	**EGGC**
No. of chromosomes	**N**	10	500	24	10
Mating radius	$\mathbf{R}_+$	0.01	15.0	11.695	11.02
Replacement radius	$\mathbf{R}_+$	0.01	15.0	-	9.6561
Merging radius	$\mathbf{R}_+$	0.01	15.0	14.748	3.81

Mutation strength	$\mathbf{R}_+$	1.0	250.0	242.16	237.43
Mutation probability	$\mathbf{R}_+$	0.0	1.0	0.6705	0.6185
Mutation scaledown	**N**	10	1000	269	238
No. improvement times	**N**	1	200	8	167
AES, 32 runs	-	-	-	3.32E+5	1.07E+5
AES, standard deviation	-	-	-	1.10E+5	2.26E+4

Table 10. Algorithm design for optimizing the 20-dimensional Schwefel function with GC and EGGC; the maximum radii are increased compared to Table 9.

Through the increased exploitative nature of EGGC, it seems that a better equilibrium between exploration and exploitation than in the standard GC algorithm is established. This leads to two advantages in using EGGC.

First, it proves to be very accurate and stable in very hard multimodal cases. A first situation is that of the difficulty in locating the local optima, *e.g.* the Six-Hump Camel Back function. Another hard situation would be very distant optima, *e.g.* the Schwefel function, where other algorithms often miss the global optimum. Another situation is having more global optima which are not easily distinguishable in the fitness landscape, like in the case of the Himmelblau function. An even harder situation is that of local optima lying too close to the global optimum - thus an evolutionary algorithm usually gets stuck in a local optimum; this is the case with the Schaffer function. Some of the evolutionary algorithms fail in some of these cases, some fail in other cases. The classical GC algorithm, on the other hand, also performs well in this respect, but its speed of convergence is not very good for functions with many local optima very near to the global optimum, *e.g.* the Schaffer function and, especially in the case of more complex problems (n-dimensional problems), *e.g.* the Schwefel function.

As the question of computational time arises, for many low-dimensional multimodal functions, both algorithms are surely not competitive to other, namely non-evolutionary algorithms. For the other cases, *i.e.* local optima and global optima almost indistinguishable and n-dimensional function optimization with high values for n, for instance, both EGGC and GC perform accurate and stable; however, EGGC is significantly faster.

Another important feature of proposed algorithm is the new parameter, the replacement radius. Together with the generational scheme, it seems to have brought more power to EGGC. For instance, for the Schwefel function, where the optima are far from each

other, this parameter promotes rapid movement of the chromosomes through the search space. In the Six-Hump Camel Back function, where the optima are very close to each other, the parameter is responsible for the small steps necessary not to escape them.

5. EGGC as Engine for a Learning Classifier System

Instead of using an unimodal genetic algorithm for the development of a learning classifier system (LCS), EGGC is chosen to be applied for classifying data by means of machine learning; it looks like a good choice from many points of view (that are to be discussed later in this report) but especially because its multimodality; thus, multiple rules may be built for classifying the objects in the considered data sets. This section continues with a short description of LCSs.

5.1. Learning Classifier Systems

A learning classifier system represents a machine learning system that uses an evolutionary algorithm as a rule discovery component [Dum00a], [Mic92]. The rules (or productions, which are simple if-then rules) represent a population that is evolved by an appropriate evolutionary algorithm; the rules cover the space of possible inputs and they are then evolved in order to successfully be applied to the problem to be solved - the problems may range from data mining to robotics.

There are two important families of LCSs: the Pittsburgh and Michigan approaches. In a Pittsburgh-type LCS, each individual represents an entire set of rules. The individuals compete among themselves and only the strong ones survive and reproduce. This is achieved by means of natural, proportional selection and variation operators; therefore, Pittsburgh approach uses a typical evolutionary algorithm (EA) for the learning problem. What remains to be solved is the representation problem and the way individuals adapt to their environment. Usually, in a chromosome there also appear operators from propositional logic, like disjunction and/or conjunction.

In a Michigan-style LCS, each individual of the population represents a unique, distinct rule, so the EA evolves a set of rules; the population represents the rule set needed to solve the problem. The goal here is not to obtain the best individual, but to find the best set of chromosomes (rules) in the end of the algorithm. Usually, chromosomes representation is

divided into two parts - one is the condition part and contains the values for the attributes that appear in the condition of the rule and the other part consists of the conclusion of the rule. A credit assignment system is used in order to reward the better rules in a higher proportion or, at the same time, to penalise the worse rules. When new entities (rules) enter the population through mutation and/or recombination, usually crowding methods are utilised in order to introduce them; in this way, they replace only very similar individuals in the population. This technique is especially used because, in this problem case, not the best chromosome is desired most, but a set of chromosomes (rules) that, when applied, gain an optimum result for the problem to be solved.

5.2. EGGC Learning Classifier System

An LCS inspired from the Michigan family is presented. It is applied for the classification of two data sets that come from the UCI repository of machine learning databases ([Pre94]); one of them is *Pima Indians Diabetes Database* and the other one is the *Iris Plants Database*. Based upon some values the patients have for their attributes, the first task is to predict whether they suffer of diabetes or not and the second task is, again, based on the values of the attributes the iris flower have, to predict their classes.

In each of the two cases, the data set, containing patients or flowers, respectively, is divided in two parts – a *training set* and a *test set*. The LCS uses the objects in the training set and the values for their attributes for producing classification rules that are used in the decision-making process. The rules are finally used to predict the class for each of the object (patient or flower, depending on the data set) in the test set.

Present model offers an easier alternative to the credit assignment system proposed by Holland in order to solve the multimodal optimization problem within the Michigan approach. And proposed replacement is EGGC, since it has many times proven to be a very efficient way in determining multiple optima. Thus, the combination between the Michigan approach and the GC engine seems like a good match in the domain of learning classifier systems.

5.3. Application to Pima Indian Diabetes Database

All patients in the data set are females of at least 21 years old, of Pima Indian heritage, living near Phoenix, Arizona, USA. There are eight attributes (either discrete or continuous)

containing personal data, *e.g.* age, number of pregnancies, and medical data, *e.g.* blood pressure, body mass index, result of glucose tolerance test etc.

The last attribute is a discrete one and it offers the diagnosis, which is either 0 (negative) or 1 (positive). 34.9% of the patients in the dataset are assigned diabetes positive. The total number of cases is 768. The data is complete, according to its documentation; nevertheless, there are some 0 values of attributes that were not reported as missing data, but look a bit strange. No replacement or deletion of these values was undertaken in present paper.

Each chromosome will encode an IF-THEN rule. A chromosome contains therefore nine genes, one for each attribute and one for the outcome; first eight genes are real valued while the last is a binary one and it gives the output of the chromosome (conclusion of the rule encoded). In conclusion, the condition of the rule is a conjunction of personal data and symptoms and its conclusion is the diagnosis.

The rules (chromosomes) are evolved against the training set. The fitness of a chromosome is computed as its distance to all patients in the training set that have the same outcome. The aim is to minimize distances, conceiving thus good rules for the diagnosis they represent. A rule is considered of high-quality if it matches the condition part of the data in the training set with same outcome as itself.

Having a chromosome $c = (c_1, c_2, \ldots, c_8, c_9)$ and a patient from the training set $p = (p_1, p_2, \ldots, p_8, p_9)$, the distance between c and p is computed by:

$$d(c, p) = \sum_{i=1}^{8} \frac{| c_i - p_i |}{b_i - a_i} \qquad (1)$$

where a_i and b_i represent the lower and upper bounds of the i-th attribute. As the values for the eight attributes belong to different intervals, the distance measure has to refer to the interval bounds.

Convex crossover was used, with the coefficients biased by the fitness of the two parents involved. Mutation is with normal perturbation. Values for the parameters of the EGGC algorithm are specified in Table 12.

dif_i denotes the difference between the bounds of the interval corresponding to attribute i. Replacement radius was taken equal to the mating radius for both applications. The stop condition of the algorithm is given by a number of generations that can pass without any improvement for the solutions; this value was considered to be 10.

At termination of the algorithm, two chromosomes are obtained, one for each class. These chromosomes are tested against the data in the test set and accuracy is computed. The

ratio between training and test sets was set to 75%-25%, as established by Prechelt in [Pre94] with respect to the diabetes task.

Three kinds of tests were conducted with different possibilities of choosing the data that would go into training and test, respectively. The two sets are obviously disjoint. First, test sample cross-validation was performed. The first 75% of the data were taken to compose the training set and the remaining data were assigned to the test set, as in the standard manner of using this data set, according to [Pre94]. The obtained mean accuracy for the test set in 100 runs was 75.06%. Second, another test was done according to rules of splitting that should be used for this data set, as established by Prechelt in [Pre94]. The data set is sequentially split into 75% training - 25% test to give 4 different combinations of these two sets, *i.e.*:

- first 75% of the cases for training and last 25% for test
- reversely, first 25% data for test and last 75% for training
- first 50% data for training, next 25% for test and last 25% for training, as well
- first 25% data for training , next 25% for test and last 50% for training, as well

The algorithm is subject to 100 trials again. The mean accuracy obtained for the test set was 69.672%. Last, random cross-validation was performed, *i.e.* the training set containing 75% data and the test set containing 25% data were randomly generated in each run. The algorithm was applied 100 times and the obtained mean accuracy was of 69.515%.

However, in many tests, it was noticed that when the chromosome pool still has four chromosomes left and has not converged yet, a higher accuracy of 80% is obtained. This leads to the idea that in the structure of each of the two obvious clusters two other subclusters are included. Thus, with the best instead of last accuracy, better results can be obtained. Note that applying SPO parameter tuning here did not lead to any improvement.

Literature reports accuracy on the diabetes task ranging from 62% to 80.7%. Unfortunately, not many papers specify variables of the testing environment; thus, proposed algorithm cannot be objectively compared with them. Yet, there are some papers which specify them, although they differ in training/test sets sizes and method of assigning data to each of them. Authors also mention they did not delete any of the lines containing missing data. When not specified, the number of runs is presumed to be ten.

In [Smi88] a neural network algorithm to forecast the onset of diabetes mellitus was used. From the 768 samples, an equal number of 170 samples were selected randomly to represent each of the two possible results of diabetes test: positive and negative. The remaining 428 were used as validating samples. The mean of five runs was 75.12%. In

[Au01], a total of 30% of the records were randomly selected as test set. Rules were mined from the remaining 70% of the data. The algorithm was applied ten times. If the authors were to define a baseline accuracy to mean the accuracy obtained by simply assigning the most frequently occurring values to the attributes being predicted, it is 65.1%.

One approach our results can be directly compared to is [Smt04]. Using a neural network, 75% training - 25% test, the rules established by Prechelt, 100 trials and no replacement or deletion of missing data, the mean accuracy was obtained as 65.55%. Another approach that allows for objective comparison of results is [Yao97]. A new evolutionary system to evolve artificial neural networks was proposed, test sample cross-validation was used, 30 runs of the algorithm were conducted and again no replacement or deletion of missing data was done. The obtained mean accuracy on the test set was of 77.6%, the best accuracy 80.7%.

In a succinct wrapping up, Table 11 contains a comparison between the accuracy obtained by EGGC versus those of the two latter algorithms. The original GC algorithm was not applied.

Algorithm	Number of runs	Accuracy (%)
EGGC with test sample cross-validation	100	**75**
EGGC with sequential splitting	100	**69.67**
EGGC with random cross-validation	100	**69.5**
EGGC best accuracy instead of last	100	**80**
Neural Network (NN) with sequential splitting	100	65.5
Evolved NN with test sample cross-validation	30	77.6
Evolved NN with cross-validation - best result within specified number of runs	30	80.7

Table 11. Results of different techniques for the Diabetes Diagnosis Problem in comparison to EGGC

5.4. Application to Iris Database

Same LCS that uses EGGC as an engine is applied to another classification problem. The Iris Plants Database contains 3 classes (3 types of iris plants), with 50 instances for each class and 4 numerical attributes which represent length and width of the petals and sepals,

respectively. The three classes are equally distributed. According to database documentation, one class is linearly separable from the other two.

There are two ways of dividing the database into training and test sets: on the one hand, two thirds of each of the three classes was considered as training set and the rest as test set; on the other hand, the training set was randomly chosen and the test set consisted of the remaining instances. In the first case, the three classes are equally distributed into training and test sets.

Chromosome representation is similar to the one from the diabetes diagnosis problem, *i.e.* $c = (c_1, c_2, \ldots, c_4, c_5)$, where the first four genes correspond to the attributes of an iris plant and the last one embodies its class. Same distance as in (1) - with four instead of eight as the upper bound of the sum - was used for computing differences between individuals. Same variation operators were considered. The values that were used for the parameters of the evolutionary algorithm are given below in Table 12.

Number of chromosomes	Mutation probability	Mating region	Merging radius	Mutation strength	No improvement times
100	0.4	0.3	0.03	dif_i / 100	10

Table 12. Parameters of the EGGC algorithm for both classification problems

The final result of the LCS has to consist of at least three rules (chromosomes), that is at least one for each class. The accuracy for the first mode of fixing the training and test set varies between 94% and 98%, while for the second way of selecting the two disjoint sets the accuracy ranges from 88% to 96%. Besides accuracy from the third row in Table 13, where the percent is the best found in 100 runs (obtained even during these runs and not necessarily at the end of them), all other values in the third column are obtained by computing the average for the accuracies obtained at the end of each of the 100 runs.

Unfortunately, in some literature approaches that also used this database there were no clear descriptions of the way training and test sets were chosen, so direct comparisons between their results and those obtained by EGGC are not very accurate. For instance, in [Vee96], a genetic programming model was tested on the Iris database, while in [Yan91], some neural networks algorithms (backpropagation algorithm, denoted by BP, and cascade-correlation algorithm, in short CCA) were applied in this respect. Results are outlined in Table 13.

Algorithm	Number of runs	Accuracy (%)
EGGC with equally distributed cross-validation	100	**95.76**
EGGC with random cross-validation	100	**92.84**
EGGC best accuracy instead of last	100	**98**
Genetic programming ([Vee96])	100	92.7
Neural Network BP with random cross-validation	10	93.2
Neural Network CCA with random cross-validation	10	92.6
Neural Network modified CCA with random cross-validation	10	97

Table 13 Results of different techniques for the Iris Plants Database in comparison to EGGC

6. Conclusions and Future Work

A multimodal evolutionary algorithm has been tested for the optimization of four multimodal functions and has been used as an engine for a LCS; the obtained classifier system was applied to two classification problems.

Referring to the modified LCS, some possible progresses are discussed in what follows; some of them are already being tested and promising early results were obtained, while some others are to be experimented in the near future.

One possibility, suggested at the HCMC conference ([Sto05b]), is to build separate rules for different decades; therefore, rules are built depending on the *age* attribute. The explanation was that people that are young are very likely to have different reasons to suffer of diabetes than the others or they may have a different type of diabetes.

Experiments have been conducted and results showed indeed an important increase of the classification accuracy: 77.08% of the patients in the test set were correctly classified, using test sample cross-validation, in comparison to 75% when plain EGGC was used. Two decades were considered, so that all patients of age between 21 and 51 are in the first decade and patients of age between 52 and 81 in the second one. Consequently, (at least) four rules are obtained, that is (at least) two for each decade: one for the ill patients and one for the healthy ones.

Another possibility (suggested at the same HCMC conference) is to weight each of the eight attributes. In this manner, not all attributes have the same importance; the degrees of importance are detected either using a typical genetic algorithm or they may be evolved

together with the values for attributes using EGGC. In the first case, the EGGC algorithm provided the rules to be applied, that is the values for the eight attributes in each of the two cases; then, independently, a typical genetic algorithm evolved eight values, each between 0 and 1, in order to find the proper weights for the attributes of the rules. The weights are evaluated with respect to the accuracy obtained by application of the rules to the training set – so the higher the accuracy on the training set, the better the weights are. This algorithm was tested in both situations, with two decades and without them. Without decades, the algorithm provides accuracy around 77%; what is unusual is that, when using decades, the result is not very stable – it varies from 74.48% to 79.68%. Evolving the weights together with the values for the attributes using EGGC did not offer good results.

A third possibility concerns the way the fitness function is applied to the individuals; as all individuals represent rules, the fitness value of a certain chromosome *c* could be given by:

$$f(c) = e^{\frac{x}{1+y}}$$

where *x* represents the number of patients from the training set that are correctly classified by the rule (chromosome) *c* and *y* says how many patients are wrongly classified by the same rule; *f* is the fitness function. If *c* does not classify correctly any patient from the training set then its fitness is considered to be zero. At the end of the EGGC algorithm, obtained rules are applied to the test set and accuracy is computed.

This mode of computing fitness evaluation should to be more efficient as it seems more convenient that a chromosome is closer (regarding distance) to only a part of the patients from training (and consequently test) set(s) and not to all of them as it was computed in the present model. Curiously, this mode of computing fitness for chromosomes did not lead to major improvement regarding accuracy on the test set; results were so far very similar to those outlined in present paper.

A further improvement that is still to be done is to easily adjust the way merging takes place: it should occur only if the obtained set of rules provides a better accuracy on the training set.

7. References

[Au01] W. H. Au, K. C. C. Chan, *Classification with Degree of Membership: A fuzzy Approach*. In Proc. of the 1st IEEE Int'l Conference on Data Mining, San Jose, CA, 2001.

[Bar04] T. Bartz-Beielstein, K. E. Parsopoulos, M. N. Vrahatis, *Design and Analysis of Optimization Algorithms using Computational Statistics*, Applied Numerical Analysis & Computational Mathematics (ANACM), Vol. 1, No. 2, pp. 413-433, 2004.

[Bar05] T. Bartz-Beielstein, *New Experimentalism Applied to Evolutionary Computation*, Ph.D. dissertation, University of Dortmund, 2005.

[Bea93] D. Beasley, R. Bull, R. R. Martin, *Sequential niche technique for Multimodal Function Optimization*, Evolutionary Computation, Vol. 1, No. 2, pp. 101-125, 1993.

[Dum00a] D. Dumitrescu, B. Lazzerini, L.C. Jain, A. Dumitrescu, *Evolutionary Computation*, CRC Press, Boca Raton, Florida, 2000.

[Dum00b] D. Dumitrescu, *Genetic Chromodynamics*, Studia Universitatis Babes-Bolyai Cluj-Napoca, Ser. Informatica, vol. 45, no. 1, pp. 39-50, 2000.

[Gord93] V. S. Gordon, L. D. Whitley, *Serial and Parallel Genetic Algorithms as Function Optimizers*, Proceedings of the 5th International Conference on Genetic Algorithms, San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., pp. 177-183, 1993,.

[Gor04] R. Gorunescu, P. H. Millard, *An Evolutionary Model of a Multidisciplinary Review Panel for Admission to Long-term Care*, ICCC 2004, Baile – Felix, Oradea, 2004, pp. 181 - 185.

[Mic92] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs*, 2nd Edition, Springer-Verlag, 1992.

[Lee01] K. - Y. Lee, M. I. Roh, *An Efficient Genetic Algorithm using Gradient Information for Ship Structural Design Optimization*, 161 - 170, Schiffstechnik Bd. 48 – 2001/Ship Technology Research Vol. 48 - 2001.

[Pre94] L. Prechelt, *Proben 1 – a Set of Benchmark and Benchmarking Rules for Neural Network Training Algorithms*. University of Karlsruhe, Institute for Program Structures and Data Organization (IPD), Tech. Rep. 21/94, 1994.

[Smi88] J. W. Smith, J. E. Everhart, W. C. Dickson, W. C. Knowler, R. Johannes, *Using the ADAP Learning Algorithm to Forecast the Onset of Diabetes Mellitus*,

Proceedings of 12th Symposium on Computer Applications in Medical Care, R. A. Greenes, Ed. IEEE Computer Society Press, pp. 261–265, 1988.

[Smt04] R. Smithies, S. Salhi, N. Queen, *Adaptive Hybrid Learning for Neural Networks*, Neural Computation, vol. 16, no. 1, pp. 139–157, 2004.

[Sto04] C. Stoean, R. Gorunescu, M. Preuss, D. Dumitrescu, *An Evolutionary Learning Spam Filter System*, SYNASC 2004, Timisoara, Romania, Symbolic and Numeric Algorithms for Scientific Computing, International participation and committee, D. Petcu, V. Negru, D. Zaharie, T. Jebelean (Eds.), Mirton Press, pp. 512-522, September 26-31, 2004.

[Sto05a] C. Stoean, M. Preuss, R. Gorunescu, D. Dumitrescu, *Elitist Generational Genetic Chromodynamics - a New Radii-Based Evolutionary Algorithm for Multimodal Optimization*, The 2005 IEEE Congress on Evolutionary Computation - CEC 2005, Edinburgh, UK, September 2-5, pp. 1839 - 1846, 2005.

[Sto05b] C. Stoean, R. Stoean, M. Preuss, D. Dumitrescu, *Diabetes Diagnosis through the Means of a Multimodal Evolutionary Algorithm*, Proceedings of the First East European Conference on Health Care Modelling and Computation - HCMC 2005, Craiova, Romania, pp. 277-289, 2005.

[Sto05c] Catalin Stoean, D. Dumitrescu, *Elitist Generational Genetic Chromodynamics as a Learning Classifier System*, Annals of University of Craiova, Mathematics and Computer Science Series, in press.

[Sto05d] Catalin Stoean, D. Dumitrescu, *New Algorithms within Genetic Chromodynamics. Applications to Function Optimization and Classification*, Technical Report, Department of Computer Science, Faculty of Mathematics and Computer Science, Babes-Bolyai University.

[Vee96] C. J. Veenman, *Positional Genetic Programming: Genetic Algorithms with Encoded Tree Structures*, Master's thesis, Vrije Universiteit, Amsterdam, 1996.

[Yan91] J. Yang, V., Honavar, *Experiments with the Cascade-Correlation Algorithm*, Tehnical Report 91-16, Iowa State University, USA, 1991.

[Yao97] X. Yao, Y. Liu, *A New Evolutionary System for Evolving Artificial Neural Networks*. IEEE Transactions on Neural Networks 8(3), pp. 694-713, 1997.

[Zah04] D. Zaharie, *Extensions of Differential Evolution Algorithms for Multimodal Optimization*, Symbolic and Numeric Algorithms for Scientific Computing, Timisoara, Romania, pp. 523-534, 2004.