ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 2015

Seria: MATEMATYKA STOSOWANA z. 5 Nr kol. 1945

Anna REICHEL[1], Iwona NOWAK[2]

[1]Faculty of Applied Mathematics
Silesian University of Technology

[2]Institute of Mathematics
Silesian University of Technology

PROBABILISTIC MODEL-BUILDING ALGORITHMS AS TOOL TO FIND OPTIMUM OF A FUNCTION

Summary. The aim of this paper is to present the probabilistic model-building heuristics which is a modification of an evolutionary algorithm. the Probabilistic-Based Incremental Learning (*PBIL*) and the compact Genetic Algorithm (*cGA*) is presented as a example of the probabilistic model building algorithms dedicated to the binary problems. Both heuristics are tested on three functions that allow to investigate the advantages, disadvantages and limitations of methods under consideration.

ALGORYTMY Z MODELEM PROBABILISTYCZNYM JAKO NARZĘDZIE OPTYMALIZACJI FUNKCJI

Streszczenie. Celem niniejszego artykułu jest przedstawienie heurystyk wieloagentowych wykorzystujących model probabilistyczny. W artykule omówiono dwie metody: the Probabilistic-Based Incremental Learning (PBIL) oraz the compact Genetic Algorithm (cGA), będące przykładami heurystyk z modelem probabilistycznym. Obie metody są przeznaczone do rozwiązywania problemów binarnych. W ramach pracy metody te testowano na trzech funkcjach zdefiniowanych w przestrzeni ciągów binarnych. Testy miały zbadać zalety, wady oraz ograniczenia obu prezentowanych heurystyk populacyjnych.

2010 Mathematics Subject Classification: 65K10, 65K99.
Keywords: population based algorithms, heuristic methods, optimization.
Corresponding author: A. Reichel (anna.reichel@onet.pl).
Received: 23.10.2015.

1. Introduction

Recently, in a large class of population-based algorithms, methods using probabilistic models are playing an increasing role.

The general structure of those methods is similar to the genetic algorithm. The difference is that the next generation of candidate solutions is generated on the basis of a probabilistic model instead of the crossover and mutation. The model promotes solutions that lead to the aim and uses them to create successive generations of individuals with increasing fitness.

The candidate solutions are pseudo-randomly generated taking the probabilistic model into account. It means, that in this kind of algorithms, the way how to build the probabilistic model is responsible for both the convergence to the global optimum and for its rate.

To fully exploit the advantages of these methods, it has to be ensured so that the construction of a probabilistic model, with the effective convergence, will not loose possibilities to correctly search the space. If the current population will affect too strong the model changes in subsequent iterations, it can lead to a rapid convergence of the population and incorrect space exploration. On the other hand, too slow change of the model will made the optimization method close to random search. The way how the probabilistic model will be constructed is crucial for this type of methods. Remaining elements of the algorithm, such as succession, usually have a classic form – known from the genetic algorithm.

In this paper two methods of optimization, *PBIL* (Population Based Incremental Learning) and *cGA* (Compact Genetic Algorithm) will be presented. Both heuristics are methods being a genetic algorithm modification and using probabilistic model. In *PBIL* and *cGA* technique, there is assumed that the optimum is searched in the set of binary strings.

The Population Based Incremental Learning algorithm was proposed by Baluja in [1]. In [2] he provides a detailed comparison between PBIL and GA. The theoretical analysis of the *PBIL* technique and proof of its convergence were presented by Höhfeld and Rudolph in [8]. Similar considerations were also conducted in [6].

The Compact Genetic Algorithm was introduced by Harik at al. in [7]. The comparison of cGA with genetic algorithm on standard binary string optimization problems is also presented there. A more general discussion is carried in [11] where different approaches for building the probabilistic models are presented. Pelikan in his review work [10] mentioned, between many others methods, the algorithms

in question. Both heuristics are presented there as steps towards the creation of Bayesian Optimization Method.

In this paper results obtained for three test functions ($trap_n$, $3-deceptive$, $MaxDiversity$ problem) are presented. The choice of test functions was dictated by the ability of comparison of the results obtained in calculations with the known exact solution. The $trap_n$ and $3-deceptive$ are standard, commonly used for testing binary functions [3,4,12]. The $MaxDiversity$ problem was formulated by Kuo, Glover and Dhir in [9], but authors did not consider its optimization in an algorithmic way. Gallego, Duarte, Laguna and Marti have discussed similar task, the searching for approximate solution using the scatter search procedure [5]. As it was mentioned in work [10], there are many details connected with algorithms disused in this work, but it does not contain a systematic comparison of the methods.

In the presented work, the main aim of this study was the comparison of *PBIL* and *cGA* methods and to indicate the advantages and disadvantages of both of them. The specificity of heuristics has forced search in space of binary strings and selection of functions for tests. In both methods, a probabilistic model which is characteristic for this group, will be responsible by probability of the occurrence of zero or one in the binary sequence.

2. Probabilistic model-building algorithms

2.1. *PBIL* method

The first method presented in this paper is the *Population-Based Incremental Learning* (*PBIL*). The algorithm uses a learning process based on observation of the best solution in current population.

In this method, individuals for the next population/generation are created on the basis of the vector $\mathbf{p} = [p_1, p_2 \ldots, p_k]$. Components p_i determine the probability of occurrence of number one on i-th position in generated individual (i.e. binary sequence length m). It means that in the *PBIL* method the vector $\mathbf{p}$ plays role of the probabilistic model.

It is characteristic for the method discussed that in order to upgrade the vector $\mathbf{p}$, the best individual in current generation is used only. Therefore, the probabilistic model is created on the basis of the single but the most promising solution denoted by $\mathbf{b}$.

At the beginning of the iteration process, it is assumed that all components of vector $\mathbf{p}$ equal $\frac{1}{2}$. This implies that the starting population is generated with uniform distribution. Initial population (and all next generations) consists m binary sequences of length k.

In subsequent iterations components of vector $\mathbf{p}$ are updated according to the following formula:

$$p_i^{(k+1)} = (1-\lambda) \cdot p_i^{(k)} + \lambda b_i, \tag{1}$$

where $p_i^{(k)}$ means i-th coordinate of vector $\mathbf{p}$ in k-th generation, b_i – i-th bit of current vector $\mathbf{b}$ and λ – learning rate.

Individuals of $(k+1)$st population are always randomly choose according to the current vector of probabilities. Opposite to the standard genetic algorithm, *PBIL* does not keep the best individual for the next generation, but the specific of procedure gives a great chance that it will be generated. The probabilistic model is modified on the basis on the best solution and in consequence it promotes generation of individuals better than solutions in the previous generation. Therefore, there is a big chance that in next population the bigger number of promising individuals (from the objective function point of view) will appear.

The parameter λ (called *the learning rate*) is also very important for the method discussed. Its value is fixed at the beginning of the iterative process and it affects on its progress. A small value of the learning rate slows down the modification of probabilistic model while too big may interfere with too rapid unification of the population. The coefficient λ should be selected to balance the ability for focused exploration and the possibility of space exploitation.

Below the scheme of the *PBIL* method is presented:

---

procedure *PBIL*:

1. *Random choice of initial generation, initialization of the probability vector* $\mathbf{p}$, $(p_i = 0.5, \forall i = 1, \ldots, n)$

2. *Evaluation of fitness of individuals, selection of the best solution (denoted by* $\boldsymbol{b}$*).*

3. *Modification of the probability vector coordinates according to the formula:*

$$p_i = (1-\lambda) \cdot p_i + \lambda \cdot b_i, \quad i = 1, 2, \ldots k,$$

where λ *– learning rate.*

4. *Random generation of a new population according to the model represented by the current vector* $\boldsymbol{p}$.

5. *If the end condition is satisfied – stop, otherwise go to step 2.*

2.2. cGA method

Next algorithm discussed in the paper is the *compact Genetic Algorithm* (*cGA*), the kind of modification of Genetic Algorithm using probabilistic model.

Similarly to the *PBIL* method, in the*cGA* the subsequent generations of individuals are generated according to the probabilistic model. Here, the model is updated on the basis on the best and the worse individuals in the current population. The vector $\mathbf{p}$ serves the role of probabilistic model and its components are modified according to the formula:

$$p_i = \begin{cases} p_i + \frac{1}{m}, & x_i = 1 \quad \wedge \quad y_i = 0, \\ p_i - \frac{1}{m}, & x_i = 0 \quad \wedge \quad y_i = 1, \qquad i = 1, 2, \ldots k, \\ p_i, & \text{in other cases,} \end{cases} \tag{2}$$

where $\mathbf{x} = [x_1, \ldots, x_k]$ and $\mathbf{y} = [y_1, \ldots, y_k]$ are the best and the worst individuals in population, respectively and m is the population size.

Similarly to the previous method, the components of vector $\mathbf{p}$ define the probability of occurrence of the number 1 on i-th position of a binary sequence being the individual generated to the next generation. The coefficient $\frac{1}{m}$, in the formula (2), plays role of the learning rate but it can be replaced by any other fixed parameter.

Using the best and the worst individuals of current population, the *cGA* method can effectively create the probabilistic model. The procedure of the vector $\mathbf{p}$ construction increases the probability of generating individual close to the best one and decreases chance of creation solutions with low fitness.

Below the *cGA* method scheme is presented.

procedure *cGA*:

1. *Random choice of the initial generation, initialization the probability vector* $\mathbf{p}$ $(p_i = 0.5, \forall i = 1, \ldots, n)$.

2. *Evaluation of fitness of individuals, selection of the best and the worst solution (denoted by* $\mathbf{x} = [x_1, \ldots, x_k]$ *and* $\mathbf{y} = [y_1, \ldots, y_k]$*, respectively).*

3. *Modification the probability vector coordinates according to the formula:*

$$p_i = \begin{cases} p_i + \frac{1}{m}, & x_i = 1 \wedge y_i = 0, \\ p_i - \frac{1}{m}, & x_i = 0 \wedge y_i = 1, \\ p_i, & \text{in other cases,} \end{cases} \qquad i = 1, 2, \ldots k.$$

   where m *– the population size.*

4. *Random generation of a new population according to the current vector* $\boldsymbol{p}$.

5. *If the end condition is satisfied – stop, otherwise go to step 2.*

3. Test functions

In the work presented, three testing functions were optimized. In all cases it was assumed that space of acceptable solutions is a set of binary strings of length k. This assumption was forced by specificity of presented methods of optimization.

Each of the test function has different nature. They have been chosen in order to recognize the advantages and disadvantages of the presented heuristics.

Because all used test functions were so-called *benchmarks* it was easy to determine the quality of the obtained solutions.

3.1. $trap_n$

The $trap_n$ was the first function used for tests. It is given by the formula:

$$f_{trap_n}(\mathbf{u}) = \begin{cases} n - 1 - u_1, & \text{if } u_1 < n, \\ n, & \text{in other cases,} \end{cases} \tag{3}$$

where n is an order of function and u_1 the number of ones occurring in the binary vector $\mathbf{u}$.

It is usually assumed that the order of function $trap_n$ is the same as the size of problem i.e. $n = k$. In such situation function $trap_n$ has global maximum equals 1 obtained for string of ones on each position. In contrast to the *OneMax* function, classically used for tests, values of the function $trap_n$ do not depend linearly on the number of zeros in the vector $\mathbf{u}$ (Fig. 1). It may results additional difficulties in optimization process. It is sufficient to note that the optimum value lies just next to the worst possible solution (obtained for strings with single bit different than 1).

In some approaches the string length can be greater than function degree n. In this situation, function $trap_n$ reaches equivalent global maximum at several points. Any vector containing more than n ones is treated as optimal solution.

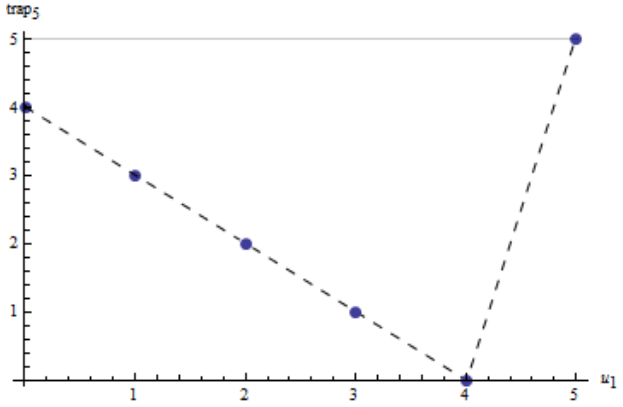

Fig. 1. Graph of $trap_5$ function

Rys. 1. Wykres funkcji testowej $trap_5$

3.2. $3-deceptive$

The second, used in this paper, test function is the $3-deceptive$ function given by the formula:

$$f_{3dec}(\mathbf{u}) = \begin{cases} 0.9, & \text{if } u_1 = 0, \\ 0.8, & \text{if } u_1 = 1, \\ 0, & \text{if } u_1 = 2, \\ 1, & \text{otherwise,} \end{cases} \tag{4}$$

where u_1 means the number of ones in the binary string $\mathbf{u}$.

The $3-deceptive$ function has one global minimum and two slightly different local maximum. Furthermore, if the length k of vector $\mathbf{u}$ is greater than 3, the global maximum is not unique (Fig. 2). Such a situation makes the optimization difficult, because two, usually significantly different, solutions have the same quality.

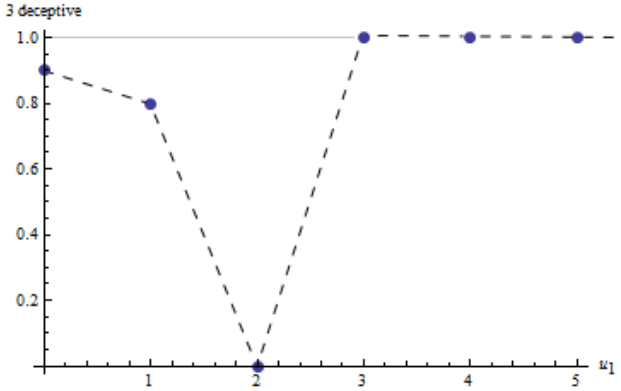

Fig. 2. Graph of $3-deceptive$ function
Rys. 2. Wykres funkcji testowej $3d-eceptive$

3.3. $MaxDiversity$

The $MaxDiversity$ problem was the last and the most interesting numerical test. In such a problem, the aim of the optimization is to find, in a given set of points A, the k-element subset S such that the sum of the distances between points in S is the greatest.

Notice, that the problem of finding a subset of the fixed power can be reduced to the problem of searching the space of binary vectors. For this purpose, firstly the elements of given set A are arbitrarily arranged in sequence, ie.

$$A = (A_1, A_2, \ldots, A_n).$$

Any subset of A can be represented by the binary string $\mathbf{x}$ of length n. One at i-th position means that i-th element of set A belongs to the subset S. More specifically:

$$x_i = \begin{cases} 1 & \text{if } A_i \in S, \\ 0 & \text{if } A_i \notin S, \end{cases} \quad i = 1, \ldots, n. \tag{5}$$

It means that $MaxDiversity$ problem can be reduced to searching binary string (of length n in which on k positions number 1 occurs) representing the subset S that meets the assumptions of optimal solution.

The objective function of the $MaxDiversity$ problem is defined by:

$$\mathrm{MD}(S, A) = \sum_{i=1}^{m-1} \sum_{j=i+1}^{m} d(A_i^S, A_j^S), \tag{6}$$

where S is any $m-$element subset of set A, points A_i^S, A_j^S are points belonging to S and d is the distance function. In general, distance can be defined in any way, but in this work the Euclidean distance was used.

For example, assuming that A is the set of vertices of unit square $WXYZ$ and looking for subset S which consists of two elements, we should get pair of opposite vertices which distance equals $\sqrt{2}$. Such a problem has two equivalent optimal solutions represented by $\mathbf{x}_{1,opt} = (1, 0, 1, 0)$ and $\mathbf{x}_{2,opt} = (0, 1, 0, 1)$[1].

4. Numerical results

The purpose of the numerical tests was comparison of *PBIL* and *cGA* methods, their evaluation and, if possible, determination of optimal parameters. Both methods were used to solve the test functions presented in previous section. In calculations, the influence of the population size on result accuracy and the convergence of iteration process was observed. Furthermore the *PBIL* method was tested for various learning rates.

In a single test, 100 trials were made. A single trial shall be understood as an iterative loop carried out until the exact solution is found, but not longer than for 100 iterations. The average (in 100 trials) number of iterations required for finding the optimal solution was treated as the result of test.

[1]The order of bits in string is compatible with classical numbering of square vertices.

If during 100 iterations exact result[2] was not found, the test was regarded as incorrect. The number of tests in which a solution was found was marked by m. After a hundred trials (for fixed parameters), the average number of iterations required to obtain the exact value of extrema was calculated. In this purpose the following formula was used:

$$\text{Mean number of iterations} = \frac{1}{m}\sum_{i=1}^{m} it_{k_i},$$

where $it_{k_i} \in \{1, \ldots, 99\}$ – number of iterations of k_i-th positive test. For tests considered as incorrect, their average error was calculated by the following formula:

$$\text{Mean error} = \sum_{i=1}^{100-m} \mid y_{opt} - y_i \mid,$$

where y_{opt} – global optimum, y_i – solution obtained in 100 iterations of i-th incorrect test.

For *cGA* algorithm, dependence of the average number of iterations for the parameter λ was observed additionally.

4.1. Function $Trap_n$

As part of tests carried out, the maximum of function $trap_n$ was searched for different values of n.

The obtained results i.e. average number of iterations necessary to achieve the solution are given in Tables 1–4. If 100 steps of iteration was not enough to determine the global maximum, the information about the average error of solutions in 100 experiments is also provided. The configurations for which the exact solution was found in fewest number of iterations are additionally highlighted.

On the basis of the results shown in Tables 1–4, it is easy to notice that in a few cases the test was regarded as incorrect.

Summarizing, in the case of functions $trap_n$, *cGA* method seems to be more effective than *PBIL*. Regardless of the order of functions, *compact Genetic Algorithm* cope well with optimization, even using small populations. The proper action of method is proved by the fact, that the increase of population size makes less the number of iterations needed to obtain the correct result. Use of the method *PBIL* is more troublesome, because selection of the learning rate λ is necessary. As follows from the foregoing statements, wrong selection of this parameter may

[2]All test functions are benchmarks what means that the exact solution is known.

negatively affect the performance of the method. The tests seem to indicate that the safest value of λ is 0.01, because for this value, the risk of an inexact result was the lowest.

Table 1

Results – function $trap_5$

	Number of iterations (error)					
	cGA	*PBIL*				
N_{pop}		$\lambda = 0.5$	$\lambda = 0.2$	$\lambda = 0.1$	$\lambda = 0.01$	$\lambda = 0.0001$
5	**2.97**	24.62 (0.58)	6.03 (0.04)	3.94	5.94	6.77
20	1.77	1.63	**1.59**	1.88	2.02	2.6
50	**1.16**	1.2	1.18	1.21	1.32	1.21

Table 2

Results – function $trap_6$

	Number of iterations (error)					
	cGA	*PBIL*				
N_{pop}		$\lambda = 0.5$	$\lambda = 0.2$	$\lambda = 0.1$	$\lambda = 0.01$	$\lambda = 0.0001$
5	**3.55**	47.91 (1.29)	13.76 (0.3)	6.76 (0.03)	10.6	12.46 (0.01)
20	2.65	3.84 (0.06)	**2.26**	2.6	3.05	3.44
50	1.71	**1.42**	1.46	1.53	1.63	1.71
100	1.24	1.22	1.25	**1.15**	1.19	1.19

Table 3

Results – function $trap_7$

N_{pop}	*Number of iterations (error)*					
	cGA	*PBIL*				
		$\lambda = 0.5$	$\lambda = 0.2$	$\lambda = 0.1$	$\lambda = 0.01$	$\lambda = 0.0001$
5	**4.86**	63.61 (1.84)	34.72 (0.92)	19.53 (0.39)	14.99	25.91 (0.02)
20	**3.08**	15.71 (0.41)	4. (0.03)	3.76	5.61	7.89
50	2.42	2.81 (0.03)	**2.06**	2.18	2.59	3.08
100	1.61	1.43	**1.42**	1.44	1.65	1.76
200	1.22	**1.16**	1.26	1.19	1.26	1.2

Table 4

Results – function $trap_{10}$

N_{pop}	*Number of iterations (error)*					
	cGA	*PBIL*				
		$\lambda = 0.5$	$\lambda = 0.2$	$\lambda = 0.1$	$\lambda = 0.01$	$\lambda = 0.0001$
5	**5.42**	141.76 (3.6)	71.36 (2.59)	56.78 (1.8)	42.21 (0.06)	78.45 (0.97)
20	**6.18**	84.58 (1.97)	30.72 (0.92)	12.78 (0.18)	20.21	(0.13) (0.13)
50	6.27	37.64 (0.77)	7.78 (0.12)	**4.79**	12.49	21.14 (0.01)
100	5.76	7.94 (0.12)	**2.86**	3.85	7.42	10.82
200	4.78	4.78 (0.06)	**2.22**	2.79	4.7	4.88
500	2.49	1.79	**1.66**	1.69	2.16	2.28

4.2. Function $3-deceptive$

As it was mentioned in one of previous chapters, the function *3-deceptive* reaches its global maximum equals 1, if at least 3 components of the vector $\mathbf{u}$ equals one.

In presented paper, 3 variants of function specified by the general formula (4) was considered. In the first variant, a solution was searched in a set of vectors of length 3, which implies the existence of exactly one, unique global optimum. In all other cases, the *3-deceptive* function adopts optimum at several different points in space. More specifically, for the n-element vectors,

$$f_{3dec}^{max}(\mathbf{u}) = 1,$$

for any $\mathbf{u} \in A$, where A is set of binary strings containing at least three bits equal to one.

Interpretation of the results shown in Table 5 and Table 6 is the same as provided in subsection 4.1.

Table 5

Results – function $3-deceptive$; space of vectors of length 3

	Number of iterations (error)					
	cGA	*PBIL*				
N_{pop}		$\lambda = 0.5$	$\lambda = 0.2$	$\lambda = 0.1$	$\lambda = 0.01$	$\lambda = 0.0001$
5	1.8	6.26 (0.01*)	**1.36**	1.84	1.77	1.89
20	**0.92**	1.01	0.97	0.93	1.07	0.94
50	0.87	0.87	**0.84**	0.86	0.88	0.88
100	0.89	0.84	0.86	0.88	0.91	**0.83**
200	0.87	0.86	**0.85**	**0.85**	0.86	0.89
500	**0.82**	0.91	0.86	**0.82**	0.93	0.84

* – algorithm *PBIL* for $\lambda = 0.5$ and a very small population (5 individuals) returned an inaccurate value.

Table 6

Results – function 3 – *deceptive*; space of vectors of length 5

	Number of iterations (error)					
	cGA	*PBIL*				
N_{pop}		$\lambda = 0.5$	$\lambda = 0.2$	$\lambda = 0.1$	$\lambda = 0.01$	$\lambda = 0.0001$
5	**0.41**	0.61	0.47	0.53	0.55	0.57
20	0.51	0.43	**0.40**	**0.40**	0.49	0.61
50	0.53	0.58	**0.47**	0.48	0.59	**0.47**
100	0.50	0.52	0.5	0.48	**0.43**	0.48
200	**0.48**	0.61	0.51	0.54	0.5	0.54
500	0.52	0.47	**0.46**	0.58	0.55	0.5

1st variant – binary strings of lenght 3

The optimization of function 3 – *deceptive* in 1st variant has proceeded without major problems for both methods. In the vast majority of cases results were obtained after only a few steps. It is certainly a consequence of the specifics of the search space (vectors with 3 components) and the relatively high chance for generation the optimal or close to optimal solution in initial population.

The only test in which optimization had ran less smoothly was a case in which relatively high learning rate ($\lambda = 0.5$) and very small population were used. It certainly results from the unification of the population which the cause is too high a value of λ and in consequence too rapid domination of one individual in population.

Generally, for formulated tasks, no significant difference between working of algorithms *CGA* and *PBIL* was observed.

2nd variant – binary strings of lenght 5

As in the previous variant, relatively good results were obtained in both algorithms. There was no significant influence of the coefficient λ on the final results. In many experiments the exact/optimal result was obtained in the zero iteration. It was assumed that in the calculations different solutions for which the objective function takes the same value are not distinguished.

3th variant – binary strings of lenght 10 or more

Solving the *3-deceptive* problem in the space of vectors of length 10 or more does not allow to observe how discussed method work. The probability that vector with at least three ones will be generated in the initial population is very high and increases with the length of the vector. The cardinality of a set of optimal solutions (generated in initial population) for the vector length n is

$$\|A\| = \sum_{i=3}^{n} \binom{n}{i},$$

what means that probability of generating such solutions tends to 1 i.e.

$$P = \frac{\sum_{i=3}^{n} \binom{n}{i}}{2^n} \underset{n\to\infty}{\longrightarrow} = 1.$$

4.3. *MaxDiversity* problem

In the *MaxDiversity* problem, the goal is to find subset S containing k elements of the given set A (of cardinality n). The solution is represented by binary string of length n. One on ith position means that ith element of set A belongs to a subset S.

As in previous tests, the criterion adopted to stop the algorithm is to find the optimal solution or the execution of maximum number of iterations (in this situation as the solution, the best obtained result is accepted).

Due to the different nature of the problem, Tables 7 and 8 (except average number of iterations required to reach a solution) contain additionally percentage error of final solutions. This value should be understood as a percent of correct solutions in 100 experiments.

A – a set of vertices of the unit square (in 2D space)

The goal of the task is to find a k-element subset S of given set A. The set $A = \{(0,0),(0,1),(1,0),(1,1)\}$ is a set of vertices of unit square. The total distance between points belonging to optimal subset S has to be the biggest.

As a first test, the problem with $k = 2$ was solved. In this case, the solution is not unique. Two subsets $S_1 = \{(0,0),(1,1)\}$ and $S_2 = \{(1,0),(0,1)\}$ are optimal with respect to the target of problem. The solutions obtained are represented by vectors $\mathbf{x}_1 = (1,0,0,1)$ and $\mathbf{x}_2 = (0,1,1,0)$, respectively. For both subsets:

$$\mathrm{MD}^{max}(S_l, A) = \sqrt{2},$$

where $l = 1, 2$.

Table 7

Results of $MaxDiversity$ – vertices of the unit square ($k = 2$)

	Nomber of iterations					
	cGA	*PBIL*				
N_{pop}		$\lambda = 0.5$	$\lambda = 0.2$	$\lambda = 0.1$	$\lambda = 0.01$	$\lambda = 0.0001$
3	**2.23**	2.26 (0.41)	2.53	2.57	2.31	2.4
5	1.72	2.22	2.14	2.23	2.17	2.16
20	1.65	2.00	2.00	2.00	2.00	2.00
50	1.66	2.00	2.00	2.00	2.00	2.00
100	1.67	2.00	2.00	2.00	2.00	2.00

Evaluating methods used in the work, it is easy to noted that both algorithms work correctly, without too much trouble finding one of two optimal solution. Selection of parameters, such as the population size and the value of learning coefficient, not really matter, but according to expectations very small populations did not guarantee a solution.

A – a set of 10 randomly selected points in the unit ball with center at (0,0)

In second test of $MaxDiversity$ problem, it was assumed that (Fig. 3):

$$\begin{aligned} A \quad = \{&(0.5, -0.5),\ (0.4, 0.1),\ (-0.9, -0.1),\ (0.1, 0.12),\\ &(-0.32, 0.14),\ (-0.1, 0.58),\ (0.911, 0.2),\ (-0.77, 0.58),\\ &(0.14, -0.85),\ (-0.14, -0.13)\}. \end{aligned}$$

In such a problem the accurate solution

$$S = \{(0.911, 0.2),\ (-0.77, 0.58),\ (0.14, -0.85)\},$$

is represented by a vector $\mathbf{x} = (0, 0, 0, 0, 0, 0, 1, 1, 1, 0)$ for which

$$MD^{max}(A, S) \simeq 4.721.$$

The results obtained in numerical experiments are collected in Table 8.

Optimization process proceeded in a satisfactory manner both for *cGA* and *PBIL* method. For appropriate size of population, the global optimum was found in all tests. Additionally, it seems that, in the case of using *PBIL*, a smaller population requires a lower coefficient learning.

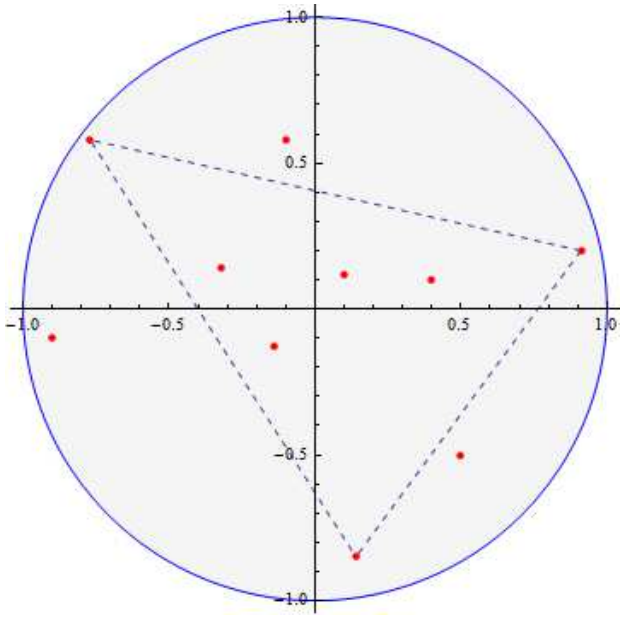

Fig. 3. The solution of problem on the background of set A

Rys. 3. Rozwiązanie problemu na tle zbioru A

Table 8

Results of $MaxDiversity$ – set of 10 points in unit ball ($k = 3$)

	Number of iterations					
	cGA	*PBIL*				
N_{pop}		$\lambda = 0.5$	$\lambda = 0.2$	$\lambda = 0.1$	$\lambda = 0.01$	$\lambda = 0.0001$
3	7.13 (0.43)	5.48 (0.67)	10.54 (0.49)	16.6 (0.42)	29.33 (0.30)	32.55 (0.33)
5	6.93 (0.51)	5.03 (0.54)	9.34 (0.37)	11.39	19.53	18.71 (0.15)
20	5.16	4.20 (0.33)	5.01	5.57	6.56	7.7
50	3.83	3.03	3.13	3.39	3.62	3.62
100	2.87	2.55	2.5	2.66	2.51	2.7

5. Conclusions

The tests carried out as part of presented work showed satisfactory efficacy of both discussed methods: *cGA* and *PBIL*. On this basis, some interesting conclusions can be formulated.

In many cases, the algorithm *cGA* gives better results than *PBIL*. Comparing the working time of both algorithms, it should be remembered that *cGA* returns the result in a shorter time than the *PBIL*. Unlike to *PBIL*, the algorithm *cGA* does not require selection of any additional parameter. Calculations performed using the *PBIL* method have shown that both high and very low value of learning rate λ reduces the effectiveness of algorithm. Tests show that the most optimal learning coefficient is 0.1. With proper selection of λ, the results obtained by *cGA* and *PBIL* are similar.

The numerical tests and observations made on the basis of them indicate that in the case of the problem for which the optimum is not known, the compact Genetic Algorithm should be recommended.

References

1. Baluja S.: *Population-based incremental learning. A method for integrating genetic search based function optimization and competitive learning.* Technical Report CMU-CS-94-163, School of Computer Science, Carnegie Mellon University, Pittsburgh 1994.
2. Baluja S., Caruana R.: *Removing the genetics from standard genetic algorithm.* Proceedings of the First International Conference on Machine Learning, Morgan Kaufmann 1996, 36–46.
3. Chen Y., Hu J., Hirasawa K., Yu S.: *Solving Deceptive Problems Using a Genetic Algorithm with Reserve Selection.* IEEE Congress on Evolutionary Computation, IEEE 2008, 884–889.
4. Deb K., Goldberg D.E., Whitley L.D.: *Analyzing deception in trap functions.* Found. Genetic Algorithms **2** (1993), 93–108.
5. Gallego M., Duarte A., Laguna M., Marti R.: *Hybrid heuristics for the maximum diversity problem.* Comput. Optim. Appl. **44** (2007), 411–426.
6. González C., Lozano J.A., Larranaga P.: *The convergence behavior of the PBIL algorithm: a preliminary approach.* Artifical Neural Nets ang Genetic Algorithms, Springer, Vienna 2001, 228–231.

7. Harik G.R., Lobo F.G., Glodberg D.E.: *The compact genetic algoritm.* IEEE Trans. Evolutionary Computation **3**, no. 4 (1999), 287–297.
8. Höhfeld M., Rudolpf G.: *Toward the theory of population based incremental learning.* Proceedings of the IEEE Conference on Evolutionary Computation, IEEE Press 1997, 1–5.
9. Kuo C.-C., Glover F., Dhir K.S.: *Analyzing and modeling the maximum diversity problem by zero-one programming.* Decis. Sci. **24** (1993), 1171-1185.
10. Pelikan, M.: *Bayesian optimization algorithm: from single level to hierarchy.* PhD thesis, University of Illinois at Urbana-Champaign, Urbana, Illinois 2002.
11. Pelikan M., Goldberg D.E., Lobo F.G.: *A survey of optimization by building and using probabilistic models.* Comput. Optim. Appl. **21** (2002), 5–20.
12. Sivanandam S.N., Deepa S.N.: *Introduction to Genetic Algorithms*, Springer Verlag, Berlin 2008.