STUDIA INFORMATICA
Nr 1/2(7) Systemy i technologie informacyjne 2006

Function optimization using metaheuristics

Marek Pilski[1], Franciszek Seredyński[1,2,3]
[1] Institute of Computer Sciences, University of Podlasie,
Sienkiewicza 51, 08-110 Siedlce, Poland
[2] Polish-Japanese Institute of Information Technology,
Koszykowa 86, 02-008 Warsaw, Poland
[3] Institute of Computer Science, Polish Academy of Sciences,
Ordona 21, 01-237 Warsaw, Poland

Abstract. The paper presents the results of comparison of three metaheuristics that currently exist in the problem of function optimization. The first algorithm is Particle Swarm Optimization (PSO) - the algorithm has recently emerged. The next one is based on a paradigm of Artificial Immune System (AIS). Both algorithms are compared with Genetic Algorithm (GA). The algorithms are applied to optimize a set of functions well known in the area of evolutionary computation. Experimental results show that it is difficult to unambiguously select one best algorithm which outperforms other tested metaheuristics.

Keywords. Particle Swarm Optimization, Artificial Immune System, Genetic Algorithm, function optimization.

1 Introduction

There is a wide range of problems, which can be reduced to solving the problem of multivariable function. Usually, solving this category of problems boils down to finding the optimum of a given function. Unfortunately, finding the optimum, especially in case of non-linear functions, may be very computationally complex, and in some cases it may sometimes be impossible to find it in an analytical way or when determinist methods are not known. In such case, various heuristics may be useful, which, however, do not always guarantee that the optimal solution be found. Nevertheless, they can find a solution close to the best one within a reasonably short period of time.

In the article we shall examine new algorithms: Particle Swarm Optimization (PSO) and Artificial Immune Systems (AIS) as an alternative to the classical Genetic Algorithm (GA).

An inspiration to invent a PSO metaheuristic was the ability of living creatures such as birds or fish to travel in groups (flocks of birds and schools of fish)

in harmony. Terminology concerning the "swarm" was proposed by Millonas [7] in his book, which describes the modelling of artificial life. Optimization of PSO comprises very simple notions and paradigms, which can easily be coded and simulated using a computer. The algorithm demands that basic mathematical operators be used and is not computationally expensive as regards memory usage and speed. Hence, PS is becoming increasingly popular, and numerous implementations, applications and modifications appear [3][12][13][14]. AIS, in turn, is built basing on rules governing the functioning of immune system of vertebrates. Mechanisms governing immune systems are used to build AIS, dedicated to data analysis, optimization, as well as the construction of anomaly detecting systems [15][16]. Recently, they have also become popular as regards numerous implementations, improvements and applications. In turn, GA is based on analogy with the evolution process found in nature, and is becoming increasingly popular and successful [2].

The aim of the current study is thus not to check if one algorithm is in general „better" than another one, but to analyze the behaviour of three different algorithms on a given set of optimization functions with a great number of variables, which are extremely difficult to solve, and to provide new insights on using these algorithms for the proposed objective functions.

We shall try to compare all the heuristics in the common testing environment of well-known functions, which constitute an experimental firing ground for many optimization methods.

The paper is organized in the following way. Next section presents PSO algorithm. Section 3 describes AIS and section 4 outlines GA. Section 5 presents a set of test functions and section 6 shows results of experiment study. Last section contains conclusion.

2 Particle Swarm Optimization

2.1 Description of functioning PSO

Initial position of individuals is chosen at random from the solutions. Next, a single particle may move in the direction described by the equation:

$$\begin{cases} v^{t+1}[k] = c_1 r_1 v^t[k] + c_2 r_2 (y^t[k] - x^t[k]) + c_3 r_3 (y^{*t}[k] - x^t[k]) \\ x^{t+1}[k] = x^t[k] + v^{t+1}[k] \end{cases}, \quad (1)$$

where v^t – speed of a molecule at time t; x^t – position of a molecule at time t; y^t – the best position found so far by a molecule for time t; y^{*t} – the best position fund so far by the neighbours for time t; c_1, c_2, c_3 – weight coefficients defining the influence of each of the three elements of the compromise, which respectively define how much a molecule trusts: c_1 – itself at time, c_2 – its own experience, c_3 – its neighbours and

their experience; *[k]* – k-th vector coordinates *x*, *v* and *y* of the length equal to the number of dimensions of the space of solutions.

Coefficients c_1, c_2, c_3 are multiplied by random values r_1, r_2 and r_3, which belong to $<0,1>$ range and are defined for every generation.

The second line of the equation (1) means that the speed v is represented by the number defining the distance a molecule can travel in a time $t=1$ so we can add up the values of variables *x* and *v* assuming that their units are identical.

The value of the vector of speed is changeable, which prevents the travelling of an individual through a space of solutions, along the straight line. The change in the vector's value is calculated in the following way:

$$v_i(t) = \chi(v_i(t-1) + \rho_1(p_i - x_i(t-1)) + \rho_2(p_g - x_i(t-1))), \tag{2}$$

where: v_i – vector of speed of an individual in an iteration *i*; p_i – the best position of a given individual (*pbest*i – value for the position p_i); p_g – position of the best individual in the whole population (*gbest* – its value). Moreover, parameters ρ_1 and ρ_2 may influence the vector of speed of a molecule. First of them influences p_i value, the second on p_g. A change in these parameters changes the force of influence of the best values found so far on molecules. The direction of movement is influenced by the best position of a given individual (p_i) and position of the best individual in the whole population (p_g).

The speed of a molecule should be great enough to make it possible for a molecule to leave a local minimum and, at the same time, small enough to provide a division into search areas. It is recommended that the value be chosen in accordance to the problem in question [8]. It is done by introducing an additional parameter called inertia weight:

$$\chi = \frac{\kappa}{abs\left(\frac{1-\frac{\rho}{2}-\sqrt{abs(\rho^2-4\rho)}}{2}\right)}, \tag{3}$$

where κ – coefficient, $\kappa \in (0,1>$; ρ – coefficient, which is the sum of ρ_1 and ρ_2.

A similar solution was proposed in paper [5], where the authors worked out a new method using Random Number Inertia Weight. Alternative methods of setting inertia weight by using a fuzzy variable [13][14], which give better results, are proposed.

2.2 Particle Swarm Optimization – pseudocode

Individual: $x=\{x_1, x_2, ..., x_N\}$ (structure representing a solution)
Population: *S*

Optimization criterion: function *f(x)*
Initialize *S*
DO
  FOR every individual
    Count the value of function
    IF ($f(xi)<pbest_i$) THEN $p_i=x_i$
    IF ($f(x_i)<gbest$) THEN $p_g=x_i$

  END
FOR every individual
  Count the value of vector of speed
  Count the position of particle
END
WHILE the maximum number of iteration or the smallest error it does not it be becomes reach
Problem_solution = the best individual from *S*

3 Artificial Immune System

3.1 Description of functioning AIS

The idea of the algorithm presented in this paper was originally proposed by L.N. de Castro and F. J. Von Zuben'a [4] who called it CLONALG. To use the algorithm, it was necessary to assume that it does not refer to a predetermined, public set of antigens because the set of antigens is constituted by unknown maxima of function *f(x)*. As affinity of antibody *p* and antigen we shall take the value of function *f(p)*. In the algorithm, an antibody is a vector of floating point numbers.

In each iteration of the algorithm, the function of adjustment is counted for every antibody. Next, *n* best antibodies are chosen from population *Ab* to a new population of antibodies – *Abn*. Moreover, in each iteration of the algorithm, in the new population *Abn* the following operations are carried out sequentially for every antibody:

Cloning:

The number of clones depends on the degree of adjustment, the higher the degree of adjustment (the value of function *f*) the larger the number of clones. The total number of clones is generated for all these *n* antibodies according to the following formula:

$$N_c = \sum_{i=1}^{n} round\left(\frac{\beta \cdot N}{i}\right), \tag{4}$$

where: β – multiplying factor of the clones' number, n – total number of individuals chosen for cloning, N – size of population.

E. g. for N=100 and β=1, the highest affinity (i=1) will produce 100 clones, while the second highest (i=2) affinity produces 50 clones, etc.

Hipermutation:

For vector $x=[x_1,x_2,...,x_i]$ output vector $z=[z_1,z_2,...,z_i]$ is calculated from:

$$z_i=x_i \pm \Delta x=X*\alpha*Rand<0\text{-}1>, \quad (5)$$

where: X – the absolute value from the range of the function in question; α – parameter defining the degree of mutation calculated from:

$$\alpha=exp(-\rho*D), \quad (6)$$

where: ρ – mutation coefficient; D – fitness coefficient equal $1-n/N$; n – position of an antibody in a vector of solutions of the length N sorted according to affinity.

Then from population *Abn* a few best individuals are chosen and they replace individuals with low adjustment in population *Ab*.

Finally, the result is the best antibody from population *Ab*.

3.2 Artifical Immune System – pseudocode

Antibody: $x=\{x_1, x_2, ..., x_N\}$ (structure representing a solution)
Initialize population *Ab()* (at random the N of antibodies);
DO
  FOR every antibody *Ab*
    Count the function of adjustment;
  END
  Sort population *Ab* according to adjustment;
  Choose n ($n<N$) the best antibodies from population *Ab* to a new population *Abn*.
  FOR every antibody *Abn*
    Cloning;
    Hipermutation;
    Count the function of adjustment;
  END
  Sort population *Abn* according to adjustment;
  Choose the N-n new individuals from *Abn* and replace individuals with low adjustment in population *Ab* by them;
WHILE the maximum number of iterations or the smallest error is not reached.
Problem_solution = the best antibody from *Ab*.

4 Classical Genetic Algorithm

4.1 Description of functioning GA

In the algorithm we have an initial population *P()* consisting of properly chosen chromosomes, which are assessed by a known function of objective. In each iteration of the algorithm, called a generation, the algorithm sequentially performs three basic genetic operations:

- *Selection* – selection of the fittest individual from a present population (the value of the function), there are three methods of selection:
  - roulette wheel selection,
  - tournament selection,
  - ranking selection,
- *Crossover (one point crossover)* – exchange of genetic material between pairs of individuals coupled/matched during selection. A crossover point on the parent organism string is selected. All data beyond that point in the organism string is swapped between the two parent organisms.
- *Mutation* – the change of value of a gene chosen at random in a chromosome.

Next, new individuals are assessed – calculating the value of the function of objective. Evolutionary process lasts until the solution of the problem is reached at a satisfying level. Then, the best individual out of a given population *P(t)* is the result.

4.2 Classical Genetic Algorithm – pseudocode

Chromosome: $x=\{x_1, x_2, ..., x_N\}$ (structure representing a solution)
Population: *P()*
Optimization criterion: function *f(x)*
t = 0
Initialize *P(t)*
Evaluate *P(t)*
WHILE termination condition NOT TRUE
    t = t + 1
    Select *P(t)* from *P(t – 1)*
    Crossover in *P(t)*
    Mutate in *P(t)*
    Evaluate *P(t)*
DO
Problem_solution = the best chromosome from *P(t)*

5 Test function

Five test functions were used to carry out the experiments. The functions to be minimized are presented below:

5.1 Sphere model

A continuous, convex, unimodal function

$$f_1(x) = \sum_{i=1}^{n} x_i^2; \quad x \in R^n, \tag{7}$$

minimum $x^*=(0, 0, ..., 0)$, $f_1(x^*)=0$

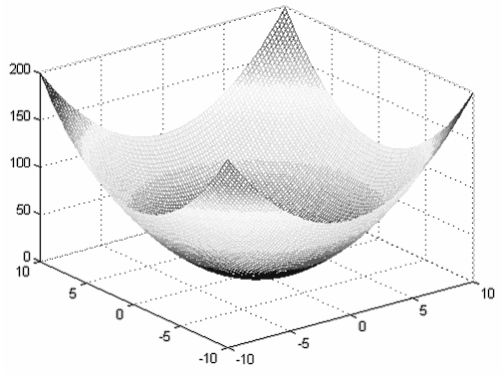

Figure 5.1. Sphere model

Griewank's function

A continuous, multimodal function

$$f_2(x) = \frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1; \qquad x \in R^n, \tag{8}$$

minimum $x^*=(0, 0, ..., 0)$, $f_2(x^*)=0$

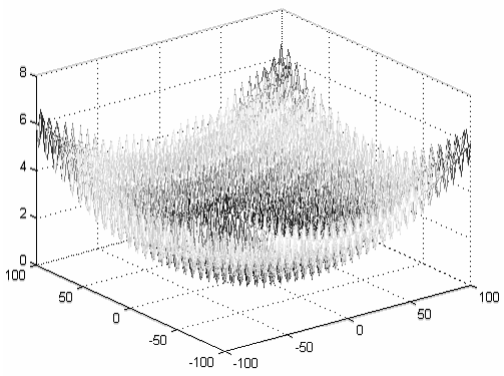

Figure 5.2. Griewank's function

Rastrigin's function

A continuous, multimodal function

$$f_3(x) = \sum_{i-1}^{n} \left(x_i^2 - 10 \cdot \cos(2\pi x_i) + 10\right); \qquad x \in R^n, \tag{9}$$

minimum $x^*=(0, 0, ..., 0)$, $f_3(x^*)=0$

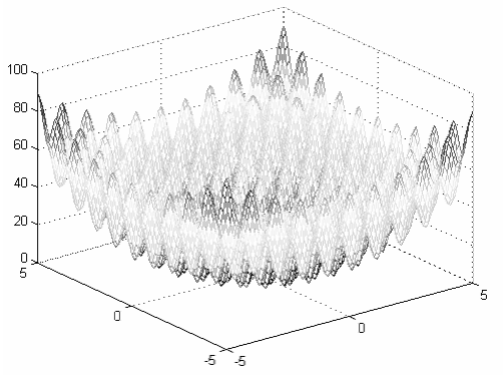

Figure 5.3. Rastrigin's function

Rosenbrock's function

A continuous, unimodal function

$$f_4(x) = \sum_{i=1}^{n} (100(x_i^2 - x_{i+1})^2 + (1 - x_i)^2); \qquad x \in R^n, \tag{10}$$

minimum $x^*=(1, 1, ..., 1)$, $f_4(x^*)=0$

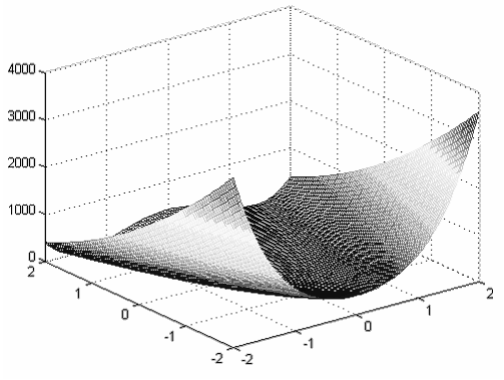

Figure 5.4. Rosenbrock's function

Schwefel's function

A continuous, unimodal function

$$f_5(x) = \sum_{i=1}^{n}\left(\sum_{j=1}^{i} x_j\right)^2; \qquad x \in R^n, \tag{11}$$

minimum $x^*=(0, 0, ..., 0)$, $f_5(x^*)=0$

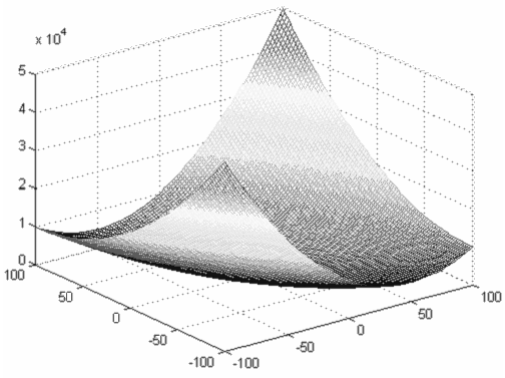

Figure 5.5. Minimization of Schwefel's function

6 Experimental study

The algorithms discussed (PSO, AIS and GA) were tested using functions presented in Chapter 5. In the experiments, which were carried out, the precision of results was lower than 10^{-6}. Experiments were carried out for the following number

of variables n=5, 10, 20, 30 but the results were presented only for n=30. All the algorithms were run for 200 generations.

The experiments, which were not presented in this paper, aimed at setting parameters of the algorithms. Parameters (proper for each algorithm), which were set, are: size of population, total number of antibodies chosen for cloning, multiplying factor, degree of mutation, selection methods, probability of crossover, coefficients of molecule speed change. Each algorithm was run many times for different values of the parameters mentioned above, and the number of variables of optimized function was increased from n=5 to n=30. When regulating parameters, the computational cost of the algorithm was taken into consideration. The best results were memorized. On this basis, the mean value of the parameters, which gave the best results for a given function, was calculated. For the sake of the presentation, each experiment was repeated 25 times, with parameter values predefined, and the best result was presented for each generation.

For every algorithm the optimum values of parameters were well-chosen for every function separately.

For algorithm PSO the values of ρ_1 and ρ_2 were near or equal to 2 and the size of the population was about 50 individuals (only for Rastrigin's function the population consisted of 100 individuals).

For AIS algorithm the total size of population for the all the functions tested consisted of 100 individuals. The number of antibodies chosen for cloning was equal 100, only for Schwefel's function the value was equal 70. The multiplying factor β for Rastrigin's function was equal 10, for sphere model and Griewank's function β=9, for Schwefel's function β =7 and for Rosenbrock's function β=6.

For GA the values of parameters were characterized by the greatest variety. The size of population ranged from 40 individuals for Griewank's function to 90 for Rastring's function. For Griewank's, Rastring's and Rosenbrock's function, ranking method of selection proved to be the best one. Mutation coefficient was smaller than 0,01. As for crossover coefficient, it did not exceed 0,8 (it was high – 0,95 only for Rosenbrock's function).

Figures 6.1-6.5 show mean results of experiments.

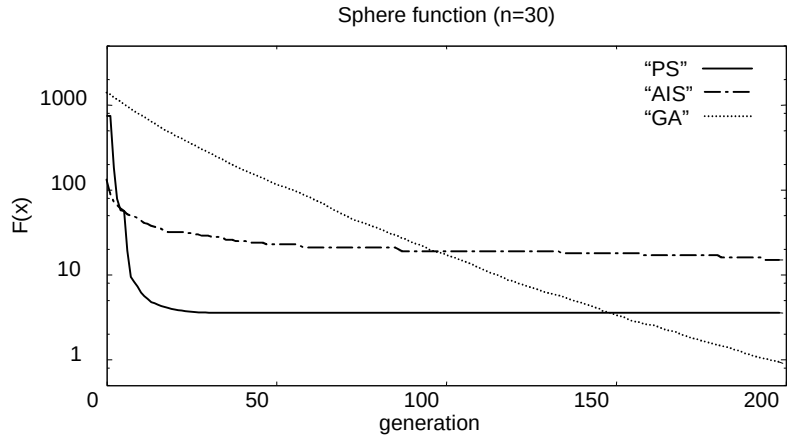

Figure 6.1. Minimization of Sphere model

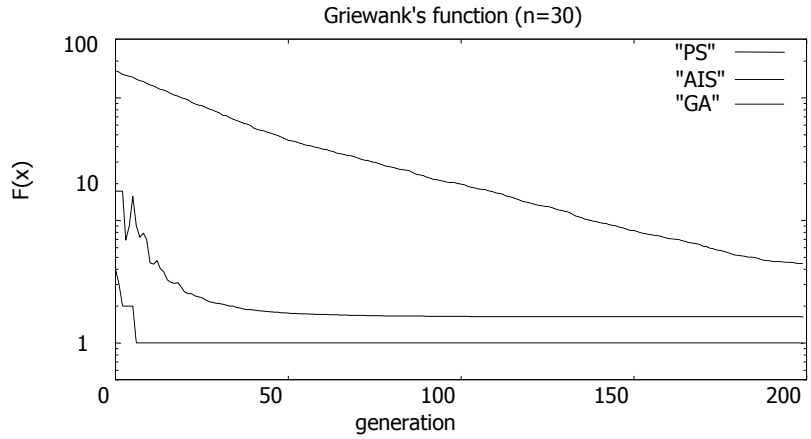

Figure 6.2. Minimization of Griewank's function

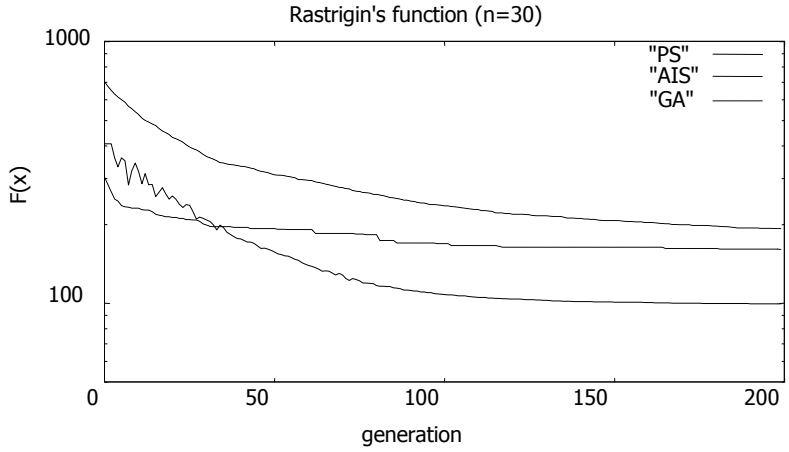

Figure 6.3. Minimization of Rastrigin's function

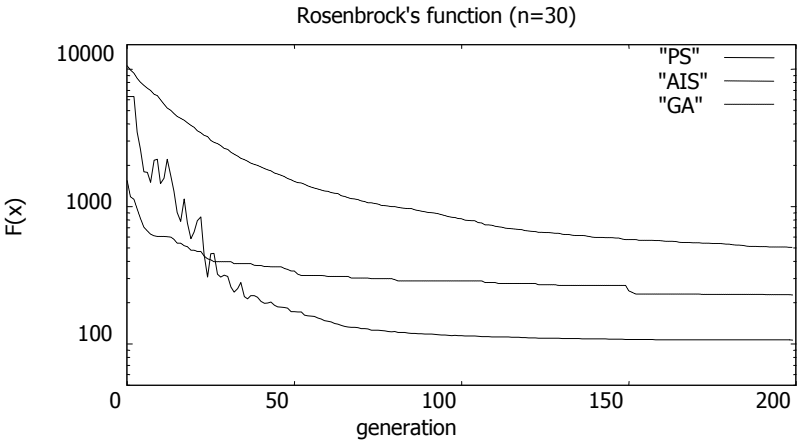

Figure 6.4. Minimization of Rosenbrock's function

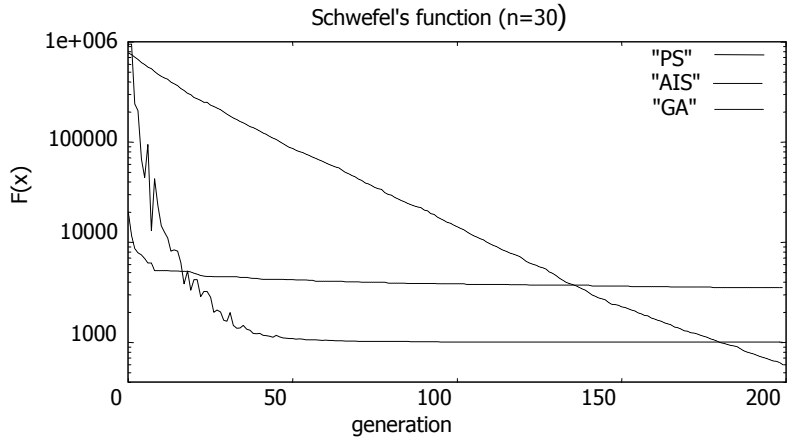

Figure 6.5. Minimization of Schwefel's function

It can easily be noticed that better results were obtained for PS than for AIS as far as regards finding the minimum for test functions. AIS only for one function (Griewank's function) showed worse than PS. In addition, it is worth noticing that AIS took about 130 times longer to execute than that of PS, which results from computational complexity of the compared algorithms. AIS algorithm performs multiple loops over the whole population of antibodies (*Ab* i *Abn*), and at the same time, it carries out costly operations of reproduction (cloning, hipermutation) and sorting. It is also worth noting that PS is the fastest heuristic, because it comprises very simple notions and mathematic paradigms. GA is also a fast algorithm.

Moreover, it is worth noticing, that finding an optimum of a function in PS algorithm after not more than 30-50 generations decreases rapidly and stays on a certain level of values of the functions tested and further populations do not bring a noticeable improvement of the result. AIS behaves similarly, the algorithm stabilizes after about 10-30 generations. GA behaves in a different way. As the number of generations increases, they gradually and linearly approach the optimum of the functions tested. The behaviour of heuristics does not depend on the type of the function (unimodal or multimodal function).

Chosen as a model GA, which proved the best using sphere model and Schwefel's function, turned out to be the least efficient search method of finding minima of the three remaining test functions. Next generations (more than 200) will probably give better results.

7 Discussion and Conclusion

The particle swarm algorithm is usually discussed together with evolutionary strategies. However, there are numerous differences between them. Evolution results in the improvement of the objective function by means of selection. In this way, only the fittest individuals survive, and the weakest are eliminated. In particle swarms, all individuals survive, and the improvement is achieved by means of social interactions. In this way, individuals learn from one another.

Experiments carried out in the same testing environment proved that PS and GA are better than AIS in finding the minimum of multivariable function. Experiments showed, that for two functions (Rastrigin's and Rosenbrock's function) the best algorithm is PSO, for sphere model and Schwefel's function the best heuristics is GA, for Griewank's function the best algorithm is AIS. It is also worth noting that PS is the fastest heuristic among those presented in the paper. It results from the fact that no genetic operators are used in this method and the whole population is used in the next generation. However, this comparison is not purely linear because we do not take into consideration an important parameter – time of execution of one generation of the algorithm. In order to define linear effectiveness of optimization methods compared in this paper, experiments assessing time of execution of the algorithms should be carried out. Furthermore, it is possible to find the best available modifications improving the efficiency of the algorithms discussed, and carry out experiments again in order to establish the best method of optimization of the functions, e.g. introducing new crossover operator in algorithm AIS [9], modifications in PSO – alternative methods of setting inertia weight by using a fuzzy variable [13][14] or for GA – coevelutionary algorithms LCGA (Loosely Coupled Genetic Algorithm) [1][11], CCGA (Cooperative Coevolutionary Genetic Algorithm) [9][11].

References

1. Bouvry P., Arbab F., Seredyński F., (2000). *Distributed Evolutionary Optimization in Mainfold; Rosenbrock's Function Case Study*, Information Sciences 122(2-4), pp. 141-159.
2. Goldberg D.E., (1995). *Algorytmy genetyczne i ich zastosowania*, WNT, Warszawa.
3. James Kennedy, Russell C. Eberhart (1999). *The Particle Swarm: Social Adaptation in Information – Processing Systems* in Corn D., Dorigo M., Glover F. (eds), New ideas in optimization, McGraw-Hill Publishing Company, London.
4. Leonardo N. de Castro, Fernando J. Von Zuben (2002). *Learning and Optimization Using the Clonal Selection Principle*, IEEE Transaction on Evolutionary Computation, vol. 6. no. 3.

5. Liping Zhang, Huanjun Yu, and Shangxu Hu, (2003). *A New Approach to Improve Particle Swarm Optimization*, GECCO 2003.
6. Michalewicz Z., (1996). *Genetic Algorithms + Data Structures = Evolution Programs*, Springer.
7. Millonas M.M., (1994). *Swarm, phase transitions, and collective intelligence*, Artificial Life III, Addison-Wesley, Reading MA.
8. Parsopoulos K.E. and Vrahatis M.N., (2002). *Recent approaches to global optimization problems through particle swarm optimisation*, Natural Computing 1.
9. Pilski M., Seredyński F., (2005). *Applying Modern Heuristics In Function Optimization.* Proceedings of Artificial Intelligence Studies, Vol. 2 (25)/2005 University of Podlasie, Siedlce.
10. Potter M.A., De Jong K.A., (1994). *A Cooperative Coevolutionary Aproach to Function Optimization*, LNCS 866, Springer.
11. Seredyński F., Zomaya A.Y., (2003). *Parallel and Distributed Computing with Coevolutionary Algorithms*, Springer.
12. Shi Y. and Eberhart R., (1997). *A modified particle swarm optimiser*, IEEE Int. Conf. on Evolutionary Computation.
13. Shi Y. and Eberhart R., (2000). *Experimental study of particle swarm optimization*, Proc. SCI2000 Conference, Orlando, FL.
14. Shi Y. and Eberhart R., (2001). *Fuzzy adaptive particle swarm optimization*, Proceedings of the 2001 Congress on Evolutionary Computation.
15. Wierzchoń S.T., (2002). *Algorytmy immunologiczne w działaniu: optymalizacja funkcji niestacjonarnych*, www.ipipan.waw.pl/~stw/ais/pubs/SzI2001.pdf
16. Wierzchoń, S.T., (2001). *Sztuczne systemy immunologiczne. Teoria i zastosowania*, Akademicka Oficyna Wydawnicza EXIT, Warszawa.