10th World Congress on Structural and Multidisciplinary Optimization
May 19 -24, 2013, Orlando, Florida, USA

Benchmark for testing evolutionary algorithms

Csaba Barcsák[1], **Károly Jármai**[2]

[1] BSc. student, University of Miskolc, Egyetemváros, Hungary, csaba.barcsak@gmail.com
[2] Professor, University of Miskolc, Egyetemváros, Hungary, altjar@uni-miskolc.hu

1. Abstract
Evolutionary algorithms have become popular for finding approximate solutions of optimization problems. We have created a benchmark to compare the efficiency of these algorithms. Also we have modified the original particle swarm optimization algorithm with gradient information to improve its efficiency.
2. Keywords: Evolutionary algorithms, Benchmark for testing, Particle swarm optimization.

3. Introduction
In the literature a lot of evolutionary algorithms can be found for example the ant colony algorithm [1] which simulates the behaviour of ants, genetic algorithms [2] which solves the optimization problem by simulating the process of evolution, Particle Swarm Optimization (PSO) algorithm [3,4], and hybrid techniques which are created as a mixture of algorithms to compound their beneficial features. The researchers of this area usually use test problems to compare these algorithms, however, the efficiency of an algorithm against another algorithm cannot be measured by the number of problems that it solves better, but we can state that which algorithm is better for a given test problem. Because of this, the benchmark should contain a wide variety of test functions with different properties.

4. Advantages and disadvantages of evolutionary algorithms
Evolutionary algorithms are popular for their efficiency, and easy implementation, but these algorithms have drawbacks which we have to consider in order to create a good benchmark. We should find test problems which can bring up these disadvantages. The largest disadvantage is that these algorithms select the better solutions in every iteration step knowing only the solutions in the previous iteration steps. Because of this it is hard to find the optimum for a noisy test function. The other disadvantage is that the algorithms cannot test whether the solution is optimal or not? In some cases the algorithms find the local optimum instead of the global. We can put various exit criteria in the algorithms which can help achieving better results, but it doesn't solve the problem.

5. A modified PSO algorithm
As previously mentioned, several variants of the PSO algorithm have been developed to improve the effectiveness of the technique. One of the solutions is to establish multiple groups of particles instead of one group. Then the local best results in each group compared and the best result of the best group gives the solution. At this case the communication is interpreted not only between the individual particles, but between the groups, so that for individual particles in the speed and position changes not only the position of the local best of the group, but the best results of all the groups taken into consideration.
Another modification is known as crazy bird. This variant is uses randomly selected particles and these particles are flying into a random direction, so that the group does not tear out particles towards the hoped *gbest* position, which differs from current *gbest* direction. It helps finding global minimum if the number of crazy bird is kept small.
The aforementioned procedures effectiveness has a random nature. We do not know that how many groups for the particles to send into random direction to get a better result than using the standard algorithm.
At the standard algorithm there is no other information about the objective function which is computed. But in many cases, depending on the individual characteristics of the problem, it would be useful to have local knowledge, since such information may make the procedure to be more efficient. One such information is the local gradient, which as we have only discrete points of samples, can be estimated. There are some methods in the

literature which use gradient information to increase the performance of the PSO algorithm [11, 12, 13]. A lot of these methods use the gradient information to move the particle in the direction of the gradient, achieving better local search results. We use this information to set a velocity multiplier coefficient to increase convergence speed. The finite-difference-based solution is a fast and efficient solution for the gradient estimation for discrete data. Each finite difference scheme is based on the Taylor-row. At the differentiable function it is assumed that the one-dimensional function *f*(***x***) can be written in the following way:

$$f(x_0+h) \approx f(x_0) + \frac{f'(x_0)}{1!}h + \frac{f''(x_0)}{2!}h^2 + ... = \sum_{n=0}^{\infty} \frac{f^{(n)}(x_0)}{n!}h^n \tag{1}$$

In this case we can stop at the second member of the formula as follows:

$$f(x_0+h) \approx f(x_0) + \frac{f'(x_0)}{1!}h \tag{2}$$

Expressed as the derivative of the following formula applies:

$$f'(x_0) \approx \frac{f(x_0+h) - f(x_0)}{h} \tag{3}$$

This formula is called in the literature as forward difference estimate.
Then, if instead of $f(x_0+h)$ we use $f(x_0-h)$ in the equation, we get the following result:

$$f'(x_0) \approx \frac{f(x_0) - f(x_0-h)}{h} \tag{4}$$

This formula in the literature is called as backward difference. As mentioned these estimates are simple and easily calculated, but its drawback that it is less accurate. More complex gradient estimations are available in the literature, but their calculation is more time consuming than the above described procedures and require more than two sampling points.
During the movement of a particle up to a given moment, the points of the earlier function values can be used to estimate gradients in this point. We only use one dimensional gradient estimation formula, because the trajectory of a given particle is always one dimensional and does not depend from the dimension of the objective function. We have implemented the algorithm using backward difference method since it was easier to implement. The gradients of the completed algorithm is used to adjust the speed of the particles, thus the particles move faster in one interval the feasible domain and move slower on the other interval. Each particle position and velocity data are stored, also the number of consecutive points where positive gradients have been found. In case when this value exceeds a pre-defined constant, than the velocity of the particle is increased. In case a negative gradient is found, or it is not interpreted by the gradient, the speed is reset to the default value.
If a particle goes through on high number consecutive sample points, where the gradient is positive, it means that the particle during this time did not pass through the local extreme values, or the global ones, so we can accelerate the speed of the particle to have less iterations to reach the extremum. The efficiency of the method is shown at several test examples.

6. Benchmark problems

In this section we present the test problems of our benchmark. A lot of test problems and test results can be found in the literature [5, 6, 7], but the authors does not mention why they include a given test problem to their tests. The complexity of the test problems depends on the number and distribution of local optimums, and the number of variables. In the design of our benchmark we have taken into account the disadvantages of evolutionary algorithms. In order to create a useful benchmark we have to include some not too complex problems (Table 1).

Table 1: Benchmark problems

name	definition	optimum (maximum)
De Jong	$f(\mathbf{x}) = -\sum_{i=1}^{n} x_i^2$	$\mathbf{x}^*=(0,0,0,...,0)$ $f(\mathbf{x}^*)=0$
Generalized Easom	$f(\mathbf{x}) = \cos(x_1)\cos(x_2)\cos(x_3)...\cos(x_n)\cdot$ $\cdot e^{-(x_1-\pi)^2-(x_2-\pi)^2-(x_3-\pi)^2-...-(x_n-\pi)^2}$	$-100 \le x_i \le 100$ $\mathbf{x}^* = (\pi,\pi,\pi,...,\pi)$ $f(\mathbf{x}^*) = 1$
Modified Generalized Easom	$f(\mathbf{x}) = \left[abs\left(\prod_{i=1}^{n} [\cos(x_i)] e^{-\sum_{i=1}^{n} \frac{1}{5} x_i^2} \right) \right]^{0.1}$	$-10 \le x_i \le 10$ $\mathbf{x}^* = (0,0,0,...,0)$ $f(\mathbf{x}^*) = 1$
Rosenbrock	$f(\mathbf{x}) = -\sum_{i=1}^{n-1} \left[100(x_{i+1}-x_i^2)^2 + (x_i-1)^2 \right]$	$-2.048 \le x_i \le 2.048$ $\mathbf{x}^*=(1,1,1,...,1)$ $f(\mathbf{x}^*)=0$
GeneralizedDrop Wave	$f(\mathbf{x}) = \frac{1+\cos(12\sqrt{x_1^2+x_2^2+x_3^2+...+x_n^2})}{\frac{1}{2}(x_1^2+x_2^2+x_3^2+...+x_n^2)+2}$	$-5.12 \le x_i \le 5.12$ $\mathbf{x}^*=(0,0,0,...,0)$ $f(\mathbf{x}^*)=1$
Rastrigin	$f(\mathbf{x}) = -(10n + \sum_{i=1}^{n} (x_i^2 - 10\cos(2\pi x_i)))$	$-5.12 \le x_i \le 5.12$ $\mathbf{x}^*=(0,0,0,...,0)$ $f(\mathbf{x}^*)=0$
Two Towers	$f(\mathbf{x}) = \prod_{i=1}^{n} [\cos(3(x_i-5))] e^{-\left(\sum_{i=1}^{n} 3(x_i-5)^2\right)^{0.2}} +$ $+1.2\prod_{i=1}^{n} [\cos(3(x_i+5))] e^{-\left(\sum_{i=1}^{n} 3(x_i+5)^2\right)^{0.5}}$	$-10 \le x_i \le 10$ $\mathbf{x}^*=(-5,-5,-5,...,-5)$ $f(\mathbf{x}^*) \approx 1.2$
Four Drop Waves	$f(\mathbf{x}) = \sin\left(\frac{1}{5}\prod_{i=1}^{n} x_i\right)^2 \frac{1}{10^n} abs\left(\prod_{i=1}^{n} x_i\right) +$ $+1.1\prod_{i=1}^{n} [\cos(x_i)] e^{-\left(\sum_{i=1}^{n} x_i^2\right)}$	$-10 \le x_i \le 10$ $\mathbf{x}^*=(0,0,0,...,0)$ $f(\mathbf{x}^*)=1.1$

The DeJong [8], Easom and Rosenbrock functions are not complex problems, because they don't have a lot of local optima. These are useful because we can see that our algorithm works in the right way, and we haven't made coding mistakes. The Drop Wave and Rastrigin functions are more complex ones, because they have a lot of local optima and for the algorithms it is harder to find the global one on a given dimension.

7. Composition of the test functions

In [9] the author has proposed a method that can generate more complex functions from given basis functions. In this section we propose a general framework how to create your own method for function generation.
The input data:

$[X_{\min}, X_{\max}]^D$	the search range of the function which we want to generate
$f_i(\theta)$	basic functions
$\left[x_{\min_i}, x_{\max_i}\right]^D$	search range of the basic functions
o_i	The position of the global optimum (maximum) for the *i*-th basic function
bias	A vector which contains the bias values for each basic function. We can define the global optimum with this. The function with the biggest bias value will have the global optimum (The optimum value is the biggest bias value plus one).

The output data:

$F(\theta)$

The steps of the algorithm without shifting the optimums:

In the first step we have to shrink the basic functions to the search range of the output function.

let
$$\kappa_i = \frac{X_{\max} - X_{\min}}{x_{\max_i} - x_{\min_i}} \tag{5}$$

the transformed basic function:
$$f_i\left(\frac{\theta}{\kappa_i}\right), \tag{6}$$

After shifting we have:
$$f_i(\varphi_i), \varphi_i = \frac{\theta - \left(X_{\min} - x_{\min_i}\kappa_i\right)}{\kappa} \tag{7}$$

After that we have to normalize our function:
$$\frac{f_i(\varphi_i)}{f_i(o_i)} \tag{8}$$

If the function value is 0 at the optimum point we should modify the basic function.

At present we have a basic function shrinked and shifted to our new function's search range. The output function is the following:

$$F(\theta) = \sum\left[w_i\left(\frac{f_i(\varphi_i)}{f_i(o_i)} + bias_i\right)\right] \tag{9}$$

The steps of the algorithm with shifting the optimums:

If we would like to define not only the global optimum, but the positions of the optimums too, we should shift the optimum points. In order to do this, the basic functions needs to be evaluated outside the defined search rage, and the given optimum for the function have to be the global optimum outside the search range too. If we would like to keep the original search range, we should use different type of shrinking which have mentioned before. With shifting the optimums the output function:

$$F(\theta) = \sum \left[w_i \left(\frac{f_i(\vartheta_i)}{f_i(o_i)} + bias_i \right) \right] \tag{10}$$

where
$$\vartheta_i = \frac{\theta - \left(X_{\min} - x_{\min_i} \kappa_i + \Psi_i \right)}{\kappa_i} \tag{11}$$

Where Ψ is the shifting vector.
W_i is the weighting function. We use this because we would like to keep the predefined optimum positions and values. This function should give a bigger coefficient to a function when we are "near" the given function's optimum, and small coefficient for the other functions. We can use different type of functions to do this. We have used the following functions:

Distance

$$w_i = euclidean\_dist\left(\vartheta_i, o_i\right), \tag{12}$$

Normalize the distances:

$$w_i = \frac{w_i}{\sum w} \tag{13}$$

$$w_i = 1 - w_i \tag{14}$$

Search for the biggest complement distance
$if\ \ w_i \neq biggestComplementDist\ \ w_i = (1 - biggestComplementDist) / (n-1)$
where n is the number of basic functions.

Gaussians

The use Gaussian functions result smoother edges.

Some other type of functions which can generate noise to our problem:

We can generate more difficult problems if we give some noise to our weighting function. During the design of the weighting function we have to keep in mind that we want to keep the original optimums. In order to do this, our new weighting function should have a value set on [0,1], and it's global maximum should be at the optimum point (Table 2).

A Gabor function like weighting function:

$w_i = \left| \prod_k \cos(\vartheta_i(k)\tau_1) e^{-\frac{\sum_k \vartheta_i(k)^2}{\tau_2}} \right|$ where $\vartheta_i(k)$ means the k-th element of the ϑ_i vector, τ_1 is the noisiness parameter. the grater this value the greater the noise, τ_2 is the convergence range (Table 3).
After we have calculated the values, we have to search for the biggest value and update the vector:
$if\ \ w_i \neq biggestWeightValue\ \ w_i = (1 - biggestWeightValue) / (n-1)$

An example:

We generated a 2D function with a search range [-10,10] in x and y direction. We have used 7 basis functions, every function is the same:

$f_i(\theta) = 1 - \sum_k \theta(k)^2$ where is the k-th $\theta(k)$ element of the variable vector. The x coordinates of the optimum points: [-8,6,2,7,2,9,-2], and in the y coordinates: [8,-2,4,7,2,3,-8], the bias vector: [100,25,3,4,70,2,1]

Table 2: Example function with distance based and Gaussian weighting

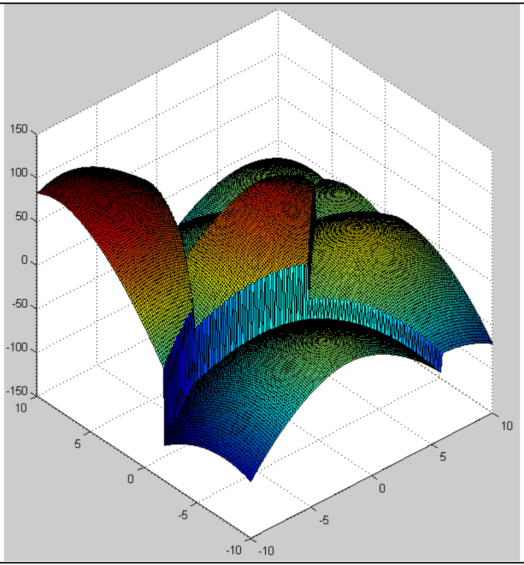
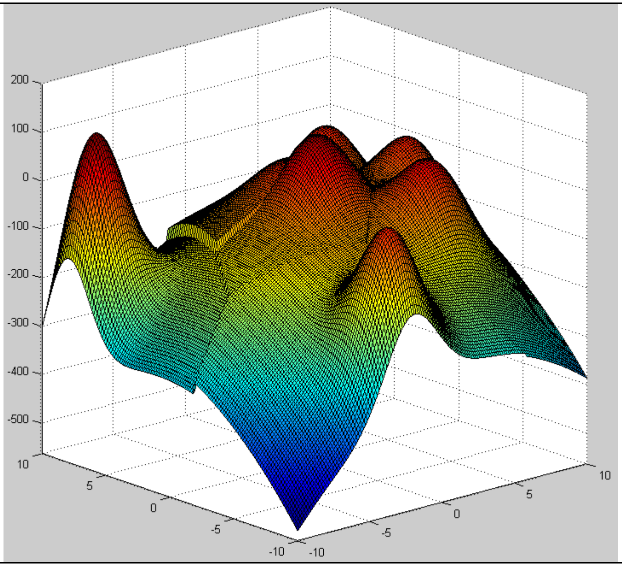
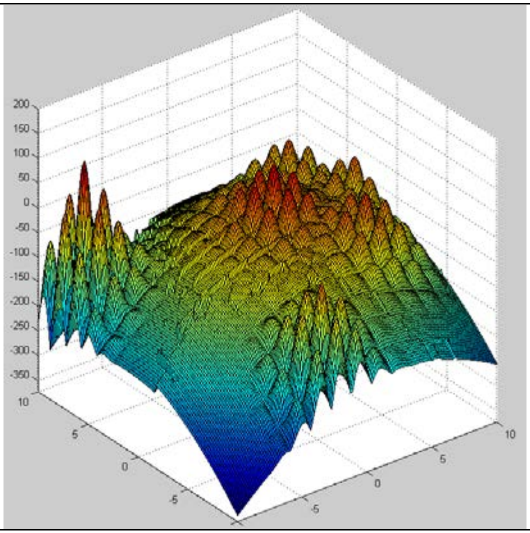
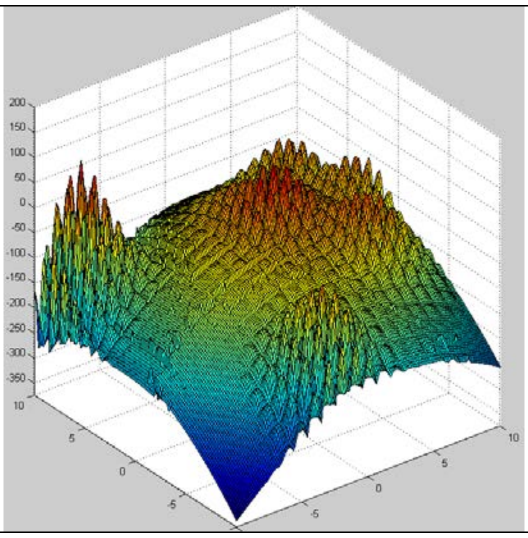
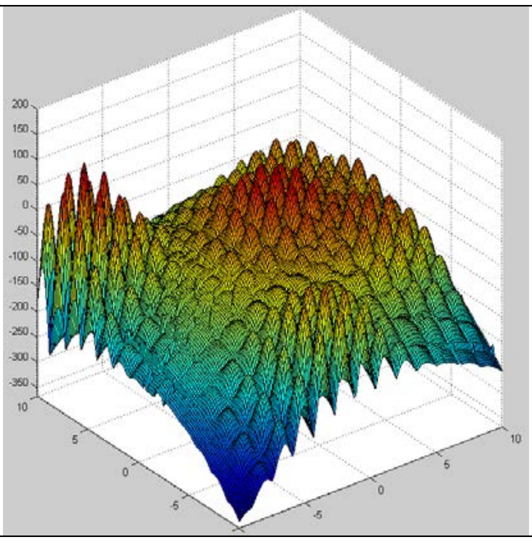
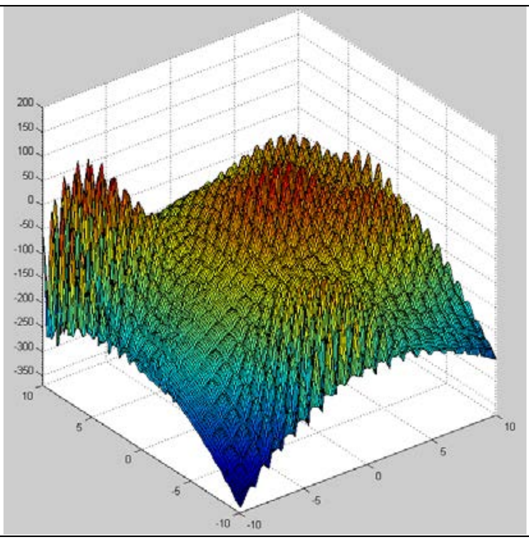
The function with distance weighting function	With Gaussian weighting
	

Table 3: Example function with Gabor function like weighting with different parameters

Gabor function like weighting		
	$\tau_1 = 2$	$\tau_1 = 3$
$\tau_2 = 8$		

$\tau_2 = 20$		

9. Comparing the standard PSO and the modified PSO (GPSO)

We have compared these two methods using our previously proposed set of functions (the definitions of the used composition test functions are in the above example). It is hard to create a summary which contains a lot of information and can be easily read and understood in the same time. There are a lot of articles which use different statistical indexes to propose their results.

In our test we run the two algorithms one hundred times for each 2D test function with two hundred particles and we calculate the average number of iterations (in one iteration there are two hundred function evaluations), the average of the distances between the numerical and theoretical solutions, the best solution (the distance is minimal), the worst solution, and the standard deviation of the distances between the numerical and theoretical solutions.

From the Table 4 we can conclude that the GPSO algorithm have found solutions within less iterations in the most cases than the standard algorithm. The comparisons show, that the application of gradient information makes the procedure more efficient, without great additional time consuming calculations.

Table 4: Statistical indexes

Function name	Method	Avg of iterations	Avg of distances	Best	Worst	Std dev.
De Jong	PSO	271.44	0.00012	241	307	0.00015
	GPSO	169.66	0.00019	148	184	0.00027
Generalized Easom	PSO	269.58	0.0008	229	298	0.00526
	GPSO	191.68	0.00004	149	277	0.00032
Modified Generalized Easom	PSO	271.48	0.0000003	241	321	0.0000007
	GPSO	222.24	0.000002	96	316	0.000008
Rosenbrock	PSO	256.13	0.00122	134	303	0.00605
	GPSO	171.62	0.00032	124	230	0.00284
GeneralizedDrop Wave	PSO	153.89	0.03443	54	285	0.02979
	GPSO	157.50	0.01647	56	331	0.02452

Rastrigin	PSO	157.85	0.31868	86	295	0.30559
	GPSO	122.96	0.33681	70	309	0.22001
Two Towers	PSO	167.94	0.15069	87	279	0.17236
	GPSO	142.16	0.19134	56	350	0.18367
Four Drop Waves	PSO	240.60	0.00128	105	300	0.00549
	GPSO	169.32	0.00298	85	289	0.01245
Composition Test function with distance weighting	PSO	271.17	0.01369	232	299	0.01412
	GPSO	180.07	0.01698	148	274	0.01318
Composition Test function with Gaussian weighting	PSO	270.89	0.00112	240	303	0.00231
	GPSO	183.65	0.00795	104	301	0.03328
Composition Test function with Gabor like weighting	PSO	146.00	12.0394	71	274	13.5161
	GPSO	129.21	14.6124	74	281	13.4657

10. Visualizing the optimization process

From the table above we can conclude that the GPSO algorithm have found solutions within less iterations in the most cases than the Standard algorithm, but the "Avg of distances" are better in the standard PSO. And we can see that the numerical results generated using the composition functions are the expected results according to the theoretical ones. These kind of statistical indexes can be used for summarize the results, but they do not inform us about the reasons of the results.

To generate the Figure 1-3 we have run the algorithms one thousand times on the 2D Rastrigin, 2D Drop Wave, 2D De Jong test functions respectively with one thousand particles. We can see on Figure 1-3 that the GPSO algorithm gets near the optimum with less iterations than the standard algorithm, but when we recall the results in Table 4 in many cases it cannot get as close to the optimum as the standard algorithm.

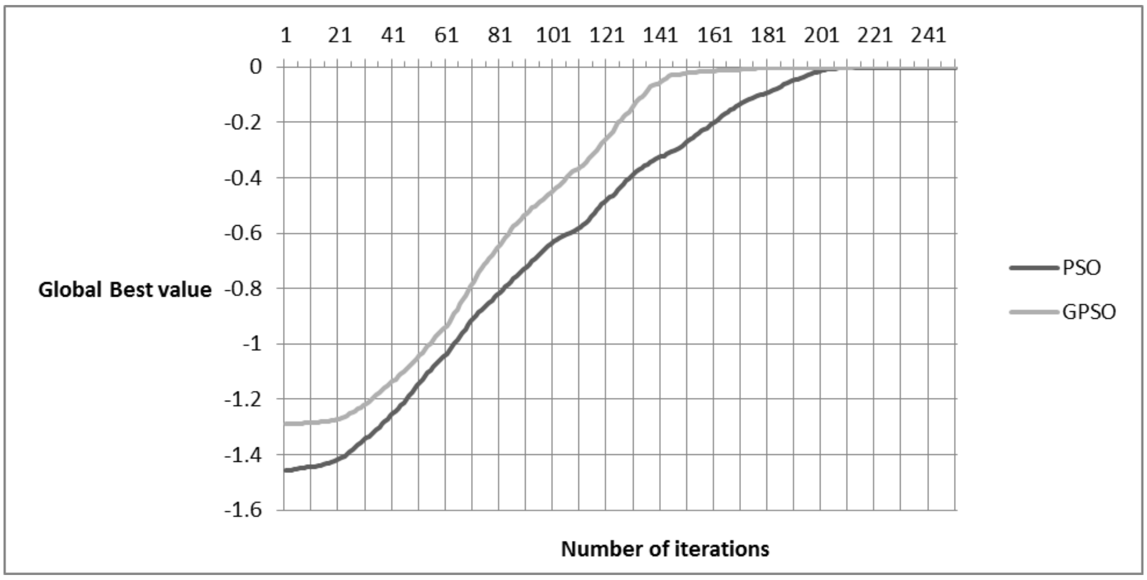


Figure 1: History of PSO and GPSO, 2D Rastrigin test function

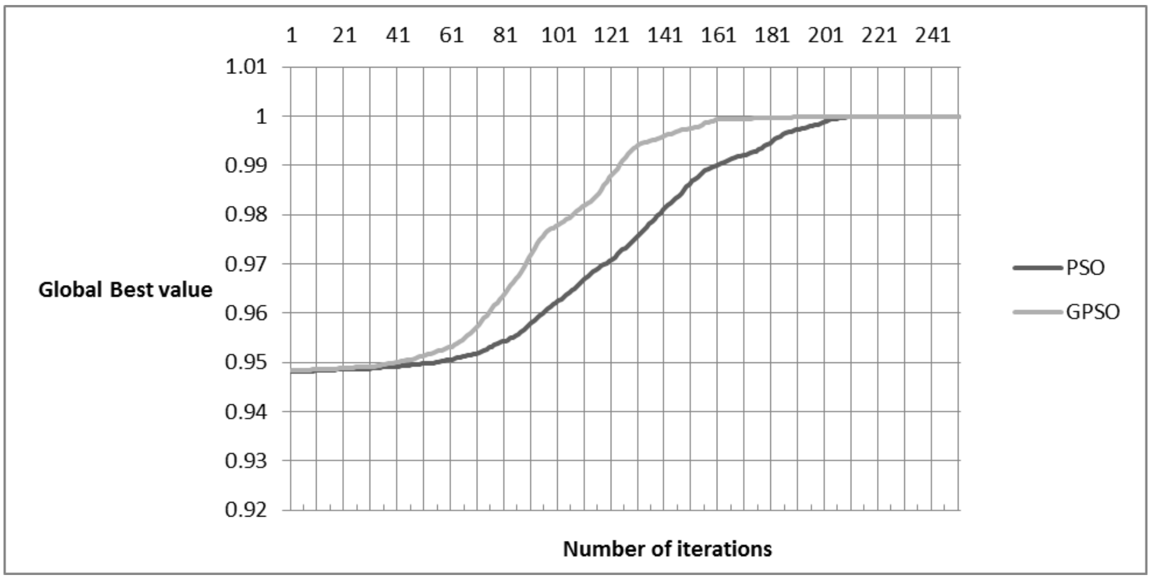


Figure 2: History of PSO and GPSO, 2D Drop Wave test function

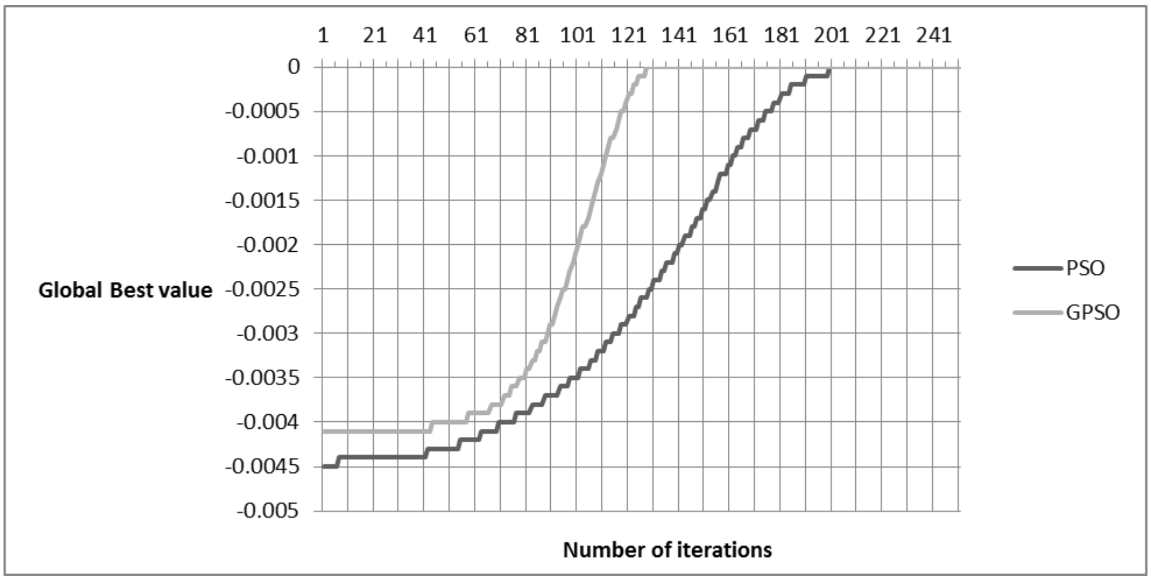


Figure 3: History of PSO and GPSO, 2D De Jong test function

11. Conclusion and further work

We have proposed an extensible framework which can be used for testing evolutionary algorithms. This framework contains the generalized form of some well-known test functions and some new ones as well in order to cover a lot of function properties. We have proposed a simple method for generating optimization test functions from a basic set of functions and a weighting function to control the noisiness parameter. We have used the test framework to compare the standard and our modified form of the PSO algorithm, which uses gradient information. We can conclude that our modified algorithm will find the neighbourhood of the optimum faster than the standard algorithm in most cases, but the distance between the found solution and the theoretical one is usually larger. One direction in our further research is to find a technique which can reduce this distance and to create some interesting weighting functions which can produce different types of noise to the generated function.

12. Acknowledgements

The authors gratefully acknowledge the support of the Hungarian Scientific Research Fund under the OTKA 75678 project number. The project was also supported by the TÁMOP 4.2.1.B-10/1/KONV-2010-0001 Increasing the quality of higher education through the development of research-development and innovation program at the University of Miskolc.

13. References

[1] A. Ostfeld, *Ant colony optimization methods and applications*, InTech Publishers 2011. ISBN 978-953-307-157-2

[2] M. Mitchell, *An introduction to genetic algorithms*, The MIT Press, 1998.

[3] J. Kennedy, R. Eberhart, *Particle swarm optimization*, IEEE International Conference on Neural Networks, Vol. 4, pp. 1942–1948, 1995.

[4] R. Eberhart, J. Kennedy, *A new optimizer using particle swarm theory*, Proceedings of the Sixth International Symposium on Micro Machine and Human Science, pp. 39–43, 1995.

[5] D. Swagatam, P. N. Suganthan, *Criteria for CEC 2011 Competition on testing evolutionary algorithms on real world optimization problems*, Technical Report, December, 2010

[6] R. Storn, K. Price, Differential Evolution – *A Simple and efficient heuristic for global optimization over continuous spaces*, Journal of Global Optimization, 11: 341–359, 1997.

[7] M. Mologa, C. Smutnicki, *Test functions for optimization needs*, 2005. pp. 1-10. www.bioinformaticslaboratory.nl

[8] K. De Jong, *An analysis of the behavior of a class of genetic adaptive systems*, Doctoral dissertation, University of Michigan, 1975.

[9] J. Liang, N. Suganthan, K. Deb, *Novel composition test functions for numerical global optimization*, Swarm Intelligence Symposium, 2005. SIS 2005. Proceedings 2005 IEEE, pp 68-75, 2005.

[10] L.C. Cagnina, S.C. Esquivel and C.A.C. Coello, *Solving constrained optimization problems with a hybrid particle swarm optimization algorithm*, Engineering Optimization, Vol. 43, No. 8, August 2011, 843–866.

[11] J. Farkas, K. Jármai: *Optimum design of steel structures*, Springer Verlag, Heidelberg, 2013. 288 p. ISBN 978-3-642-36867-7

[12] D. Barla-Szabó, *A study of gradient based particle swarm optimisers*, University of Pretoria, Msc Thesis, 2010.

[13] M.M. Noel, T.C. Jannett, *Simulation of a new hybrid particle swarm optimization algorithm*, Proceedings of Thirty-sixth Southeastern Symposium on System Theory, pages 150-153, 2004.