Stretching technique for obtaining global minimizers through Particle Swarm Optimization

K.E. Parsopoulos*
Department of Mathematics
University of Patras
GR–26110, Patras, Greece

V.P. Plagianakos†
Department of Mathematics
University of Patras
GR–26110, Patras, Greece

G.D. Magoulas‡
Department of Information
Systems and Computing
Brunel University
Uxbridge UB8 3PH
United Kingdom

M.N. Vrahatis§
Department of Mathematics
University of Patras
GR–26110, Patras, Greece

Abstract

The Particle Swarm Optimizer, like many other evolutionary and classical minimization methods, suffers the problem of occasional convergence to local minima, especially in multimodal and scattered landscapes. In this work we propose a modification of the Particle Swarm Optimizer that makes use of a new technique, named Function "Stretching", to alleviate the local minima problem. Function "Stretching" consists of a two–stage transformation of the objective function that eliminates local minima, while preserving global ones. Experiments indicate that the Particle Swarm Optimizer equipped with the "Stretching" technique exhibits good performance and results in finding global minima reliably and predictably.

1 Introduction

In many practical optimization problems the search is usually focused on locating the global minimizer of a real valued objective function $f : \mathcal{E} \to \mathbb{R}$, i.e. finding a point $x^* \in \mathcal{E}$ such that

$$f(x^*) \leq f(x), \qquad \forall\, x \in \mathcal{E}, \tag{1}$$

where the compact set $\mathcal{E} \subset \mathbb{R}^D$ is a D–dimensional parallelepiped.

There are many *Global Optimization* (GO) methods developed so far to deal with the above problem, see [9] for details. In general, GO methods possess strong theoretical convergence properties, and, at least in principle, are straightforward to implement and apply. Issues related to their numerical efficiency are considered by equipping GO algorithms with a "traditional" local optimization phase. Global convergence, however, needs to be guaranteed by the global–scope algorithm component which, theoretically, should be used in a complete, "exhaustive" fashion. This behavior indicates the inherent computational demand of the GO algorithms, which increases non–polynomially, as a function of problem–size, even in the simplest cases.

Evolutionary computation techniques belong to a special category of GO methods. Evolutionary computation methods work on a set of potential solutions, which is called *population*, and find the optimal problem solution through cooperation and competition among the potential solutions [7]. The most commonly used

*kostasp@math.upatras.gr
†vpp@math.upatras.gr
‡George.Magoulas@brunel.ac.uk
§vrahatis@math.upatras.gr

population–based evolutionary computation techniques, such as Genetic Algorithms and Artificial Life methods, are motivated from the evolution of nature and the social behavior. These methods can often find optima in complicated optimization problems more quickly than traditional optimization methods.

Recently, Eberhart and Kennedy [3, 5, 11, 12] proposed the Particle Swarm Optimization (*PSO*) algorithm: a new, simple evolutionary algorithm, which differs from other evolution–motivated evolutionary computation techniques in that it is motivated from the simulation of social behavior. Indeed, in PSO the population dynamics simulates a bird flock's behavior where social sharing of information takes place and individuals can profit from the discoveries and previous experience of all other companions during the search for food. Thus, each companion, called particle, in the population, which is now called swarm, is assumed to "fly" over the search space in order to find promising regions of the landscape. For example, in the minimization case, such regions possess lower functional values than other visited previously. In this context, each particle is treated as a point in a D–dimensional space which adjusts its own "flying" according to its flying experience as well as the flying experience of other particles (companions).

PSO results, in general, good solutions. However, there are cases where PSO, as many other GO methods, can detect just *sub–optimal solutions* of the function f. These sub–optimal solutions can be stated as follows. Let a point $\bar{x}$ such that there exists a neighborhood $\mathcal{B}$ of $\bar{x}$ with

$$f(\bar{x}) \leq f(x), \qquad \forall\, x \in \mathcal{B}. \tag{2}$$

This point is a local minimizer of the objective function and in many cases this sub–optimal solution is acceptable. However, there are applications where the optimal solution, i.e. the global minimizer is not only desirable but also indispensable. Therefore, the development of robust and efficient techniques for alleviating this local minima problem is a subject of considerable ongoing research. To this end, the paper introduces a new technique, named *Function "Stretching"*, and shows through simulation experiments that Function "Stretching" provides a way of escape from the local minima when PSO's convergence stalls.

The paper is organized as follows. The background of the PSO and the PSO version used in our experiments are presented in Section 2. After introducing, in Section 3, the function "Stretching" technique, we propose and test, in Section 4, a modification of the PSO algorithm, named "Stretched" PSO. Finally, discussion and conclusions are given in Section 5.

2 Background of the PSO method

There are many variants of the PSO proposed so far, after Eberhart and Kennedy introduced this method [4, 5]. In our experiments we have used a new version of this algorithm, which is derived by adding a new inertia weight to the original PSO dynamics [3]. But before presenting the method, let us introduce the notation adopted in this paper: the i-th particle of the swarm is represented by the D–dimensional vector $X_i = (x_{i1}, x_{i2}, \ldots, x_{iD})$ and the best particle in the swarm, i.e. the particle with the smallest function value, is denoted by the index g. The best previous position (the position giving the best function value) of the i-th particle is recorded and represented as $P_i = (p_{i1}, p_{i2}, \ldots, p_{iD})$, and the position change (velocity) of the i-th particle is $V_i = (v_{i1}, v_{i2}, \ldots, v_{iD})$.

The particles are manipulated according to the equations

$$v_{id} = w v_{id} + c_1 r_1 (p_{id} - x_{id}) + c_2 r_2 (p_{gd} - x_{id}), \tag{3}$$

$$x_{id} = x_{id} + v_{id}, \tag{4}$$

where $d = 1, 2, \ldots, D$; $i = 1, 2, \ldots, N$ and N is the size of population; w is the *inertia weight*; c_1 and c_2 are two positive constants; r_1 and r_2 are two random values in the range $[0, 1]$.

The first equation is used to calculate i-th particle's new velocity by taking into consideration three terms: the particle's previous velocity, the distance between the particle's best previous and current position, and, finally, the distance between swarm's best experience (the position of the best particle in the swarm) and i-th particle's current position. Then, following the second equation, the i-th particle flies toward a new position. In general, the performance of each particle is measured according to a predefined fitness function, which is problem–dependent. The inertia weight is employed to control the impact of the previous history of velocities on the current velocity. In this way, the parameter w regulates the trade–off between the global

(wide–range) and local (nearby) exploration abilities of the swarm and influences PSO convergence behavior. A large inertia weight facilitates global exploration (searching new areas), while a small one tends to facilitate local exploration, i.e. fine–tuning the current search area. A suitable value for the inertia weight w usually provides balance between global and local exploration abilities and consequently a reduction on the number of iterations required to locate the optimum solution. As a general rule of thumb, it is better to initially set the inertia weight to a large value, in order to make better global exploration of the search space, and gradually decrease it to get more refined solutions; thus, a time decreasing inertia weight value has been used in our experiments presented in the next section.

It is becoming obvious from the above discussion that PSO, to some extent, resembles evolutionary computing. However, in PSO, instead of using genetic operators, each individual (particle) updates its own position based on its own search experience and other individuals (companions) experience and discoveries. Adding the velocity term to the current position, in order to generate the next position, resembles the mutation operation in evolutionary computing. Note that in PSO, however, the "mutation" operator is guided by particle's own "flying" experience and benefits by the swarm's "flying" experience. In another words, PSO is considered as performing mutation with a "conscience", as pointed out by Eberhart and Shi [3].

3 Equipping PSO with function "stretching"

In this section we describe the function "stretching" technique and we propose an algorithm model for equipping PSO with function "stretching"; this modified PSO is named "Stretched" PSO (SPSO).

The basic idea behind function "stretching" is to perform a two–stage transformation on the form of the original function $f(x)$. This can be applied soon after a local minimum $\bar{x}$ of the function f has been detected:

$$G(x) = f(x) + \gamma_1 \frac{\|x - \bar{x}\| \cdot (\text{sign}(f(x) - f(\bar{x})) + 1)}{2}, \tag{5}$$

$$H(x) = G(x) + \gamma_2 \frac{\text{sign}\,(f(x) - f(\bar{x})) + 1}{2\,\tanh\,(\mu(G(x) - G(\bar{x})))}, \tag{6}$$

where γ_1, γ_2 and μ are arbitrary chosen positive constants, and sign$(\cdot)$ defines the well known three valued sign function.

The first transformation stage, Eq. 5, elevates the function $f(x)$ and makes disappear all the local minima which are located above $\bar{x}$. The second stage, Eq. 6, stretches the neighborhood of $\bar{x}$ upwards, since it assigns higher function values to those points. Both stages do not alter the local minima located below $\bar{x}$; thus, the location of the global minimum is left unchanged. Note that the sign function, which is used in the above transformation, can be approximated by the well known logistic function:

$$\text{sign}(x) \approx \text{logsig}(x) = \frac{2}{1 + \exp(-\lambda_1 x)} - 1 \simeq \tanh(\lambda_2 x),$$

for large values of λ_1 and λ_2. This sigmoid function is continuously differentiable and is widely used as a transfer function in artificial neurons.

At this point it is useful to provide an application example of the technique so as to illustrate its effect. The problem considered is a notorious two dimensional test function, called the *Levy No. 5* [6]:

$$\begin{aligned} f(x) \quad = \quad & \sum_{i=1}^{5} i \cos[(i+1)x_1 + i] \times \sum_{j=1}^{5} j \cos[(j+1)x_2 + j] + \\ & + (x_1 + 1.42513)^2 + (x_2 + 0.80032)^2, \end{aligned}$$

where $-10 \leq x_i \leq 10, i = 1, 2$. There are about 760 local minima and one global minimum with function value $f^* = -176.1375$ located at $x^* = (-1.3068, -1.4248)$. The large number of local minimizer makes extremely difficult for any method to locate the global minimizer. In Fig. 1, the original plot of the *Levy No. 5* into the cube $[-2, 2]^2$ is shown.

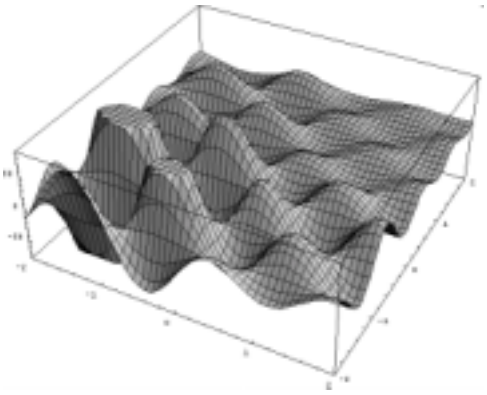

Figure 1: Plot of the original *Levy No. 5* function.

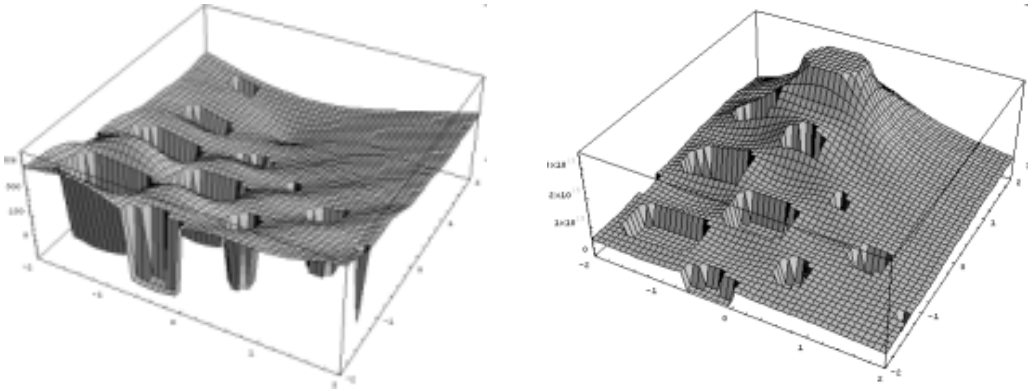

Figure 2: Plot of the *Levy No. 5* after the first stage (left) and the second stage (right) of the "Stretching" transformations.

After applying the transformation of Eq. 5 (first stage of function "Stretching") to the *Levy No. 5*, the new form of the function is shown on the left side of Fig. 2. As one can observe, local minima with higher functional values than the "stretched" local minimum (which looks as if a pin is positioned over it and the rest of the function is stretched around it) disappeared, while lower minima as well as the global one have been left unaffected. The final shape of the landscape is shown on the right side of Fig. 2; that is a result of applying the second transformation stage to the *Levy No. 5*. It is clearly shown how the whole neighborhood of the local minimum has been elevated; thus, the former local minimum has now turned to be a local maximum of the function.

Table 1, below, provides an algorithm model for a modified PSO method, named 'Stretched" PSO. SPSO is initialized with the PSO method, as presented in Section 2, for minimizing the fitness function. When PSO stumbles upon a local minimum, the function "Stretching" technique is applied to the original fitness function and SPSO is re–initialized with the PSO for the minimization of the stretched function.

Details on the performance of the SPSO in some well known test problems, as well as suggestions for selecting parameter values for the function "Stretching" technique are presented in the next section.

4 Experimental results

In this section, we present results from testing the classical PSO method and the "Stretched" PSO. We have used some hard and well known optimization problems, such as the minimization of several of the *Levy* test functions [6], the 2–dimensional and the 4–dimensional *Corana* functions [2], the *Freudenstein–Roth*

"Stretched" Particle Swarm Optimizer	
0:	Set iteration counter $it \leftarrow 0$ and local minima counter $LMC \leftarrow 0$.
1:	Initialize randomly population and velocities, as two matrices of dimension $D \times PS$, where D is the dimension of the problem and PS is the population size.
2:	Set as best position for each particle the initial position assigned in Step 1.
3:	Set inertia w to its initial value w_0.
4:	Find the index g of the best particle of the population.
5:	**While** Error Goal not met, do
6:	Set $it \leftarrow it + 1$.
7:	Set $w \leftarrow (w - w_d)$, where w_d is a predefined value.
8:	Update population and velocities according to Eq. 3 and 4.
9:	Update best positions and the index g.
10:	Check if a local minimum has been detected. If yes, set $LMC \leftarrow LMC + 1$.
11:	if $LMC \neq 0$ then set $f(x) \leftarrow H(x)$, where $f(x)$ is the objective function and $H(x)$ is defined by Eq. 6.
12:	**End While**
13:	Return Results

Table 1: Algorithm model of the SPSO.

and *Goldstein–Price* functions [8], and a classical Artificial Neural Networks (ANNs) pattern classification problem, i.e. the classification of the eXclusive–OR (XOR) patterns.

In all the simulations reported, the values of γ_1, γ_2 and μ were fixed: $\gamma_1 = 10000, \gamma_2 = 1$ and $\mu = 10^{-10}$. Default values for the parameters c_1 and c_2 have been used: $c_1 = c_2 = 0.5$. Although the choice of the parameter values seems to be not critical for the success of the methods, faster convergence can be obtained by proper fine–tuning. The balance between the global and local exploration abilities of the SPSO is mainly controlled by the inertia weights, since the particles' positions are updated according to the classical PSO strategy. A time decreasing inertia weight value, i.e. start from 1 and gradually decrease towards 0.4, has been found to work better than using a constant value. This is because large inertia weights help to find good seeds at the beginning of the search, while, later, small inertia weights facilitate a finer search.

For each problem 100 runs were performed using the SPSO and the average performance is exhibited in terms of the mean value and standard deviation of the number of function evaluations, and the percentage of SPSO success. Results for all of the aforementioned problems can be seen in Table 2, while the size of the population used for each problem as well as the initial hypercube into which the initial population was randomly taken, are presented in Table 3.

		"Stretching" applied			PSO			SPSO		
Test Problem	Dim.	Mean	St.D.	Succ.	Mean	St.D.	Succ.	Mean	St.D.	Succ.
Levy No.3	2	15246.6	6027.3	15%	5530.5	6748.0	85%	6988.0	7405.5	100%
Levy No.5	2	3854.2	1630.1	7%	1049.4	235.1	93%	1245.8	854.2	100%
Levy No.8	3	0	0	0%	509.6	253.2	100%	509.6	253.2	100%
Freud.-Roth	2	3615.0	156.1	40%	1543.3	268.1	60%	2372.0	1092.1	100%
Goldst.-Price	2	17420.0	3236.56	5%	1080.0	225.6	95%	1897.0	3660.3	100%
Corana 2D	2	0	0	0%	1409.6	392.8	100%	1409.6	392.8	100%
Corana 4D	4	13704.6	7433.5	26%	2563.2	677.5	74%	5460.0	6183.8	100%
XOR	9	29328.6	15504.2	23%	1459.7	1143.1	77%	7869.6	13905.4	100%

Table 2: Analysis of the results for the minimization of several test problems.

Test Problem	Dim.	Popul. Size	Initial Hypercube
Levy No.3	2	20	$[-10,10]^2$
Levy No.5	2	20	$[-2,2]^2$
Levy No.8	3	20	$[-5,5]^3$
Freud.-Roth	2	20	$[-5,5]^2$
Goldst.-Price	2	20	$[-2,2]^2$
Corana 2D	2	20	$[-5,5]^2$
Corana 4D	4	80	$[-1,1]^4$
XOR	9	80	$[-1,1]^9$

Table 3: Population size and initial hypercube for each test problem.

The *Levy No. 3* function is given by the formula:

$$f(x) \quad = \quad \sum_{i=1}^{5} i\cos[(i+1)x_1+i] \times \sum_{j=1}^{5} j\cos[(j+1)x_2+j]$$

where $-10 \le x_i \le 10, i = 1,2$. There are about 760 local minima for this function and 18 global minima with function value $f^* = -176.542$. As can be seen from Table 2, in 85 out of 100 cases PSO found the global minimum without any help, while in 15 cases it got stuck in a local minimum and function "Stretching" has been successfully applied. Thus the success rate of PSO increased by 15%.

An increase of the success rate, from 93% to 100%, was observed for the *Levy No. 5* function, where "Stretching" has been applied in 7 cases, while for the *Levy No. 8*, PSO was able to detect the global minimum without any help in all 100 runs. The *Levy No. 8* function is given by the equation:

$$f(x) \quad = \quad \sin^2(\pi y_1) + \sum_{i=1}^{n-1}(y_i-1)^2[1+10\sin^2(\pi y_{i+1})] + (y_n-1)^2$$

where $y_i = 1+(x_i-1)/4$, $i = 1,\ldots,n$ and $x_i \in [-10,10]$ for $i = 1,2,3$ and has one global minimum at the point $x^* = (1,1,1)$ with function value $f^* = 0$, and, approximately, 125 local minima.

The *Corana* function in 4 dimensions is defined by the equation:

$$f(x) = \sum_{j=1}^{4} \begin{cases} 0.15 \times \left(z_j - 0.05 \times \mathrm{sgn}(z_j)\right)^2 \times d_j, & \text{if } |x_j - z_j| < 0.05, \\ d_j \times x_j^2, & \text{otherwise,} \end{cases}$$

where $x_j \in [-1000,1000]$, $d_j = 1,1000,10,100$ and

$$z_j = \left\lfloor \left| \frac{x_j}{0.2} \right| + 0.49999 \right\rfloor \times \mathrm{sgn}(x_j) \times 0.2.$$

and is a very difficult minimization problem for many methods. As can be seen from Table 2, plain PSO has only 74% success rate, but using SPSO the success rate increases to 100%.

Similar results can be seen for the *Freudenstein–Roth* and *Goldstein–Price* functions, where the PSO success rate is increased by 40% and 5% respectively. The *Freudenstein–Roth* function is given by the formula:

$$f(x) \quad = \quad [-13+x_1+((5-x_2)x_2-2)x_2]^2 + [-29+x_1+((x_2+1)x_2-14)x_2]^2$$

and has a global minimizer $x^* = (5,4)$ with function value $f(x^*) = 0$, while the *Goldstein–Price* function is given by the formula:

$$\begin{aligned} f(x) \quad = \quad & (1+(x_1+x_2+1)^2(19-14x_1+3x_1^2-14x_2+6x_1x_2+3x_2^2)) \times \\ & \times(30+(2x_1-3x_2)^2(18-32x_1+12x_1^2+48x_2-36x_1x_2+27x_2^2)) \end{aligned}$$

and has a global minimum with function value $f^* = 3$ at the point $x^* = (0, -1)$.

Several other optimization methods, such as Steepest Descent with adaptive stepsize and backtracking (SD), Fletcher–Reeves (FR), Polak–Ribiere (PR), Davidon–Fletcher–Powell (DFP) and Broyden–Fletcher–Goldfarb–Shanno (BFGS) [10], were tested too in the *Freudenstein–Roth* and the *Goldstein–Price*functions. None of these methods was able to outperform the SPSO, as can be seen from the results exhibited in Table 4.

	Freudenstein–Roth			Goldstein–Price		
Algorithm	Mean	St.D.	Succ.	Mean	St.D.	Succ.
SPSO	2372	1092.1	100%	1897	3660.3	100%
SD	41651	6153.4	77.3%	700	363.9	84.3%
FR	245	599.8	44.3%	1475	3097.1	71.4%
PR	120	13.2	43.8%	238	41.3	79.9%
DFP	91	9.7	43.7%	162	28.6	59.3%
BFGS	91	9.8	43.8%	161	27.0	59.9%

Table 4: Comparative results for the Freudenstein–Roth and Goldstein–Price test functions.

In another experiment, an ANN has been trained using the SPSO to learn the XOR Boolean classification problem. The XOR function maps two binary inputs to a single binary output and the ANN that was trained to solve the problem had 2 linear input nodes, two hidden nodes and one output node, all with logistic activations. This task corresponds to the minimization of the following objective function [13]:

$$
\begin{aligned}
f(x) \;=\; & \left[1+\exp\left(-\frac{x_7}{1+\exp(-x_1-x_2-x_5)}-\frac{x_8}{1+\exp(-x_3-x_4-x_6)}-x_9\right)\right]^{-2} \\
& +\left[1+\exp\left(-\frac{x_7}{1+\exp(-x_5)}-\frac{x_8}{1+\exp(-x_6)}-x_9\right)\right]^{-2} \\
& +\left[1-\left\{1+\exp\left(-\frac{x_7}{1+\exp(-x_1-x_5)}-\frac{x_8}{1+\exp(-x_3-x_6)}-x_9\right)\right\}^{-1}\right]^2 \\
& +\left[1-\left\{1+\exp\left(-\frac{x_7}{1+\exp(-x_2-x_5)}-\frac{x_8}{1+\exp(-x_4-x_6)}-x_9\right)\right\}^{-1}\right]^2 .
\end{aligned}
$$

In the context of ANNs, the parameters $x_1, x_2, \ldots, x_9$ are called weights and are usually initialized in the interval $[-1, 1]$. It is well known from the neural networks literature that successful training in this case, i.e. reaching a global minimizer, strongly depends on the initial weight values and that the above–mentioned function presents a multitude of local minima [1]. It is obvious from the results reported in Table 2 that the function "Stretching" technique helped to increase significantly the success percentage of the PSO, i.e. the success rate has been increased from 77% to 100%.

5 Conclusions

Locating global minimizers is a very challenging task for any minimization method. In this paper we use a new technique, named function "Stretching", for the alleviation of the local minima problem of the Particle Swarm Optimizer. This technique applies a two–stage transformation to the shape of the fitness function that eliminates undesired local minima, but preserves the global ones.

Experiments from many hard optimization test problems indicate that the PSO method when equipped with the proposed technique (SPSO) is capable of escaping from neighborhoods of local minima and locate the global minimizer effectively. The function "Stretching" technique provides stable convergence and thus a better probability of success for the PSO. The price we pay for the increased success rates of SPSO is an increase in the total number of function evaluations performed by the algorithm, but we are almost sure that the final result is the global minimum of the objective function.

Future work is focused on optimizing the performance of the proposed modification of the PSO algorithm. In addition, extensive testing on more complicated real–life optimization tasks is necessary to fully investigate the properties and evaluate the performance of the function "Stretching" technique.

References

[1] E.K. Blum (1989). Approximation of Boolean functions by sigmoidal networks: Part I: XOR and other two–variable functions, *Neural Computation*, 1, 532–540.

[2] A. Corana, M. Marchesi, C. Martini and S. Ridella (1987). Minimizing multimodal functions of continuous variables with the "Simulated Annealing Algorithm", *ACM Trans. Math. Soft.*, 13(3), 262–280.

[3] R.C. Eberhart and Y.H. Shi (1998). Evolving Artificial Neural Networks. *Proc. International Conference on Neural Networks and Brain*, Beijing, P.R. China.

[4] R.C. Eberhart, P.K. Simpson and R.W. Dobbins (1996). *Computational Intelligence PC Tools*, Academic Press Professional, Boston, MA.

[5] J. Kennedy and R.C. Eberhart (1995). Particle Swarm Optimization. *Proc. IEEE International Conference on Neural Networks*, Piscataway, NJ, IV:1942–1948.

[6] A. Levy, A. Montalvo, S. Gomez and A. Galderon (1981). Topics in Global Optimization. Lecture Notes in Mathematics No. 909. Springer–Verlag, New York.

[7] Z. Michalewicz (1996). Genetic Algorithms + Data Structures = Evolution Programs. Springer, New York.

[8] J.J. More, B.S. Garbow and K.E. Hillstrom (1981). Testing Unconstrained Optimization Software, *ACM Trans. Math. Soft.*, 7(1), 17–41.

[9] A. Neumaier (2000). On the server of the Computational Mathematics group at the University of Vienna, Austria, http://solon.cma.univie.ac.at/~neum/glopt.html, accessed 26/06/2000.

[10] E. Polak (1997). Optimization: Algorithms and Consistent Approximations. Springer–Verlag, New York.

[11] Y.H. Shi and R.C. Eberhart (1998). Parameter Selection in Particle Swarm Optimization. *Proc. Annual Conference on Evolutionary Programming*, San Diego.

[12] Y.H. Shi and R.C. Eberhart (1998). A Modified Particle Swarm Optimizer, *Proc. IEEE International Conference on Evolutionary Computation*, Anchorage, Alaska.

[13] M.N. Vrahatis, G.S. Androulakis, J.N. Lambrinos and G.D. Magoulas (2000), A class of gradient unconstrained minimization algorithms with adaptive stepsize. *Journal of Computational and Applied Mathematics*, 114, 367–386.