Proceedings of the *International Conference on Theory and Application of Mathematics and Informatics ICTAMI 2005* - Alba Iulia, Romania

FAR FROM EQUILIBRIUM COMPUTATION AND PARTICLE SWARM OPTIMIZATION

Laura Dioşan, Dumitru Dumitrescu, Delia David

Abstract. A new model for far from equilibrium computation is proposed in this paper. The model is based on classic Particle Swarm Optimization paradigm, but it is developed according to self-organization of far from equilibrium systems. A stimulus is introduced and particles movement is influenced both by neighboring particles and by external factors. Numerical experiments show that the proposed model performs similarly or sometimes even better than standard PSO approaches for several well-known problems.

2000 *Mathematics Subject Classification*: 68T05, 37F05.

1. Introduction

The theory of self-organization has been approached in a variety of disciplines: thermodynamics, cybernetics and computer modeling. In [1, 5] self-organization is defined as the spontaneous creation of a globally coherent pattern out of local interactions. Because of its distributive character, this organization tends to be robust and resisting against perturbations. The dynamics of a self-organizing system is typically non-linear, because of circular (or feedback) relations between the components. Positive feedback leads to an explosive growth, which ends when all components have been absorbed into the new configuration, leaving the system in a stable, negative feedback state. Non-linear systems have in general several stable states, and this number tends to increase (bifurcate) as an increasing input of energy pushes the system further from its thermodynamic equilibrium.

2. Self organization using PSO model

Particle Swarm Optimization (PSO) is a population based stochastic optimization technique developed by Kennedy and Eberhart in 1995 [6]. The PSO consists of a set of particles that represent candidate solutions for the problem. Each particle consists of a point in the search space (x_i), an associated velocity along each dimension (v_i), a memory of the best solution encountered by the particle so far (*pbest*), and knowledge of the best performing particle in the neighborhood during the current iteration (*nbest* or *gbest*). In each iteration a particle flies (or hops) through the search space, its trajectory is determined by its velocity, the attraction to *pbest*, and the attraction to *nbest*.

Standard model of PSO implies that particles are updated synchronously [6]. This means that the current position and speed for a particle is computed taken into account only information from the previous generation of particles. A more general model allows updating any particle anytime. This basically means two things:

- The current state of the swarm is taken into account when a particle is updated. The best global and local values are computed for each particle which is about to be updated, because the previous modifications could affect these two values. This is different from the standard PSO algorithm where the particles were updated taken into account only the information from the previous generation. Modifications performed so far (by a standard PSO) in the current generation had no influence over the modifications performed further in the current generation.

- We will work with only one swarm. Any updated particle will replace its parent. Note that two populations/swarms are used in the standard PSO and the current swarm is filled taken into account the information from previous generation.

Our purpose is to establish a parallel between a PSO system and a FFE system. This means that some parameters of PSO algorithm can play the role of an external factor which holds (keeps) the system out-of-equilibrium. Unfortunately, unlike the case for equilibrium systems, there is no well-developed formalism for studying out-of-equilibrium systems. Nevertheless one can identify some principles and resort to computer simulations of models to test ideas.

The paper is structured as follows: section 3 discusses work related to the PSO. Section 4 describes, in detail, the proposed model. Several numerical

experiments are performed in section 5. Conclusions and further work are given in section 6.

3. Related Work

Several attempts for PSO using similar techniques were made in the past.

A. J. Carlisle, in [3] presents two update methods: synchronous and asynchronous update. There is a difference in the two methods beyond the obvious. In the synchronous update, all particles have moved in parallel before the best selection is made, but in the asynchronous update, the neighbors on one side of the particle to be adjusted have already been updated, whereas the neighbors on the other side have not. His conclusion was that asynchronous updates are generally less costly.

In [12] a dissipative particle swarm optimization is developed according to the self-organization of dissipative structure. The negative entropy is introduced to construct an opening dissipative system that is far-from-equilibrium so as to drive the irreversible evolution process with better fitness.

Note that this approach implies that particles are updated synchronously: the current position and speed for a particle is computed taken into account only information from the previous generation. Our approach is quite different. The model proposed in this paper allows updating any particle anytime. The current state of the swarm is taken into account when a particle is updated.

Our approach is a new technique that proposes a new type of PSO algorithm and that improves the similarities between PSO and FFE systems. Numerical experiments show that the FFEPSO technique performs similarly and sometimes even better than dissipative approaches for several well-known benchmarking problems.

4. Proposed model

The proposed model for FFECPSO algorithm is described in this section. We study the possibility of developing far-from-equilibrium computational systems, in analogy with corresponding self-organizing systems. The evolution towards equilibrium of some systems can be interpreted as a type of self-organization, as regular structures may result from an initially unstructured state, through interactions between the components of the systems [5].

In real word swarm (such as flock of birds) can be seen like a self-organizing system. The simple social model in standard PSO has some characteristics for

self-organization of dissipative structure. All system's particles "follow" the best particle (local or global). Each particle evolution depends by it and by all neighborhood particles history and by "relations" established between it and other system components. At each step the size and the direction of each particle's move is a function of its own history (experience) and the social influence of its peer group. This dependence function can express the feedback relations that exist between the system components.

In this paper we try to identify some factors that keeps a system in an equilibrium state or perturbs the system using PSO technique. The factors are: inertia weight (w) and learning factors (or acceleration constants - c_1, c_2). Several numerical experiments performed in section 4 will prove that a linearly decreasing inertia weight (at each generation) performs better that a constant value for this parameter. Moreover, if we stop the process of decreasing at some level we can obtain much better results.

Concerning learning factors (c_1, c_2), numerical experiments prove that lower values determine equilibrium states and upper values determine perturbations.

The Algorithm. PSO is initialized with a group of random particles (solutions) and then searches for optima by updating each generation. Each particle i has an associated current position in search space x_i, a current velocity v_i and a personal best position in search space y_i. For each dimension of search space we have two limits: $-x_{max}$ and x_{max}. In every iteration, each particle is updated by following two *best* values. Our model uses a modified Particle Swarm Optimization algorithm [11].

S_1 Initialize the swarm of particles randomly
S_2 **While** not stop_condition **do**
S_3 **For** each particle **do**
  S_{31} Compute fitness of the particle
  S_{32} **If** the current fitness value is better than *pbest*, **then** update *pbest*
  S_{33} Determine *nbest* for the current particle: choose the particle with the best fitness value of all the neighbors as the *nbest*
  S_{34} Update particle's velocity according to eq. (1)
  S_{35} **If** (rand() < velocity chaos factor) **then** perturb particle's velocity according to eq. (3)
  S_{36} Update particle's position according to eq. (2)
  S_{37} **If** (rand() < position chaos factor) **then** perturb particle's position according to eq. (4)
S_4 **EndFor**

The formula for particle velocity update:

$$v_{id} = w * v_{id} + c_1 * rand() * (p_{id} - x_{id}) + +c_2 * rand() * (p_{nd} - x_{id}) \quad (1)$$

Each component of the velocity vector is restricted to a range $[-v_{max}, v_{max}]$ to ensure that individual particles do not leave the search space. We have chosen that v_{max} to be x_{max}.

The formula for particle location update:

$$x_{id} = x_{id} + v_{id} \quad (2)$$

The chaos for velocity of particle is represented as:

$$v_{id} = rand() * v_{max,d} \quad (3)$$

The chaos for location of particle is represented as:

$$x_{id} = Random(x_{min,d}, x_{max,d}) \quad (4)$$

where:

- $rand()$ is a function which generates a random real value between 0 and 1;
- $Random(a, b)$ is a random value between a and b;
- c_1 and c_2 are two learning factors, which control the influence of *pbest* and *nbest* on the search process. The learning factors c_1 and c_2 represent the weighting of the stochastic acceleration terms that pull each particle toward *pbest* and *nbest* positions;
- w is the inertia weight. Inertia weight was first introduced by Shi and Eberhart [9]. The function of inertia weight is to balance global exploration and local exploitation.

The above algorithm is quite different from the standard PSO algorithm [11]. Standard PSO algorithm works on two stages: one stage that establishes the fitness, *pbest* and *nbest* values for each particle and another stage that determines the velocity (according to equation (1)) and makes updates according to equation (1) for each particle. Standard PSO usually works with

two populations/swarms. Individuals are updated by computing the *pbest* and *nbest* value using the information from the previous population. The newly obtained individuals are added to the current population.

Our algorithm performs all operations in one stage only: determines the fitness, *pbest*, *nbest* and velocity values only when a particle is about to be updated. In this manner, the update of the current particle takes into account the previous updates in the current generation. Our PSO algorithm uses only one population/swarm. Each updated particle will automatically replace its parent.

Note that similar perturbation factors can be found in PSO algorithm proposed in [12], but this model is quite different: in [12] both chaos factors are applied after that the velocity and position of particle are computed. In our model we determine particle's velocity, perturb this (first chaos factor), compute particle's position and the also perturb this.

Algorithm's parameters
Algorithm's parameters are presented in table 1.

Table 1: PSO Parameters

Parameters	Value
Particles Number	20
Dimensions Number	5
Generations Number	1000
Neighborhood	Euclidean distance
Inertia weight	Linearly decreasing with threshold
Social factor	variable, between 0.2 and 2.6
Cognitive factor	variable, between 0.2 and 2.6
Position chaos factor	variable, between 0.1 and 1
Velocity chaos factor	variable, between 0.1 and 1

5. Experiments

Test Functions In order to test the capability of the PSO to "sustainable development", ten functions that are commonly used in the evolutionary

computation literature [8, 10] are used. All functions are designed to have minimum at the origin. These functions are presented in table 2.

Experiment 1. For assessing the performance of the model proposed in this paper we will compare three algorithms: two that use a linear decreasing inertia weight during generational updating (one that stops at some level - *Decr w 1* - and one that does not stop - *Decr w 2*) and another algorithm that use a fix weight for all generations. For first two algorithms inertia weight starts to decrease from 1.5 value to 0.1 (the threshold), respectively to 0, and the third algorithm uses the same randomized inertia weight in all generations (the inertia weight is set to [0.5 + (rand()/2.0)], which is selected according with Clerc's constriction factor [4]).

Table 2: Test functions used in our experimental study. The parameter n is the space dimension ($n = 5$ in our numerical experiments) and f_{min} is the minimum value of the function. All functions should be minimized.

Test function	Domain	f_{min}
$f_1(x) = \sum_{i=1}^{n} (i \cdot x_i^2)$.	[-10, 10]^n	0
$f_2(x) = \sum_{i=1}^{n} x_i^2$.	[-100, 100]^n	0
$f_3(x) = \sum_{i=1}^{n} \lvert x_i \rvert + \prod_{i=1}^{n} \lvert x_i \rvert$.	[-10, 10]^n	0
$f_4(x) = \sum_{i=1}^{n} \left(\sum_{j=1}^{i} x_j \right)^2$.	[-100, 100]^n	0
$f_5(x) = \max_i \{x_i, 1 \leq i \leq n\}$.	[-100, 100]^n	0
$f_6(x) = \sum_{i=1}^{n-1} 100 \cdot (x_{i+1} - x_i^2)^2 + (1 - x_i)^2$.	[-30, 30]^n	0
$f_7(x) = 10 \cdot n + \sum_{i=1}^{n} (x_i^2 - 10 \cdot \cos(2 \cdot \pi \cdot x_i))$	[-5, 5]^n	0
$f_8(x) = -a \cdot e^{-b\sqrt{\frac{\sum_{i=1}^{n} x_i^2}{n}}} - e^{\frac{\sum \cos(c \cdot x_i)}{n}} + a + e$.	[-32, 32]^n , $a = 20$, $b = 0.2$, $c = 2\pi$.	0
$f_9(x) = \frac{1}{4000} \cdot \sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos(\frac{x_i}{\sqrt{i}}) + 1$.	[-500, 500]^n	0
$f_{10}(x) = \sum_{i=1}^{n} (-x_i \cdot \sin(\sqrt{\lvert x_i \rvert}))$	[-500, 500]^n	-n* 418.98

For this experiment we have used in all iterations constant learning factor ($c_1 = c_2 = 0.8$) and chaos factors: $c_v = 0.2$ and $c_p = 0.3$. We have chosen to perform this experiment because the inertia weight controls the impact of particle is prior-period velocity on its current velocity [2].

In figure 1 and 2 we have represented the results (*pbest* value) obtained during 100 runs for all functions. We can observe that the algorithm with linearly decreasing inertia weight has the best performance.

Experiment 2. For assessing the performance of the proposed model, we will run the algorithm for different values of learning factors from [0.2, 2] range (using for inertia weight the linearly decreasing idea because we saw in previous experiment that this works better than the other models). We suppose that both coefficients have same value (social behavior and cognitive behavior have same influence). We have performed this experiment because of the importance of learning factors: in every iteration each particle's velocity is accelerated towards previous best position and towards a neighborhood (global) best position [2]. Chaos factors are similar with those used in first experiment.

By looking at the shape of curves in the Figure 3, it is easy to see a "balance point" for our algorithm. When the coefficient c is larger than the balance value, the system has a chaotic state.

Experiment 3. This experiment serves our purpose to compare the performance of the two algorithms proposed in this paper. First algorithm uses a linearly decreasing inertia weight and second algorithm uses a fix inertia weight. Both algorithms are run for different values of learning factors.

Experiment 4. Another two factors that can perturb or not our system are chaos coefficients. In figure 4 are depicted the results obtained running the PSO algorithm that uses linearly decreasing weight (from 1.5 to 0.1) for different values of chaos coefficients (in [0, 1] range). We can see that for all functions a value for chaos factors around 0.5 value assures a steady state for the system. Little values determine the system to entry in a stable state and big values for chaos factor determine large perturbations.

6. Conclusion and Further Work

In this paper a new far from equilibrium computational model based on Particle Swarm Optimization technique has been proposed. The new elements introduced by this model are: the similarities between far from equilibrium systems and particle swarm. Several numerical experiments have been performed

Figure 1: Best value for first 6 functions during 100 runs

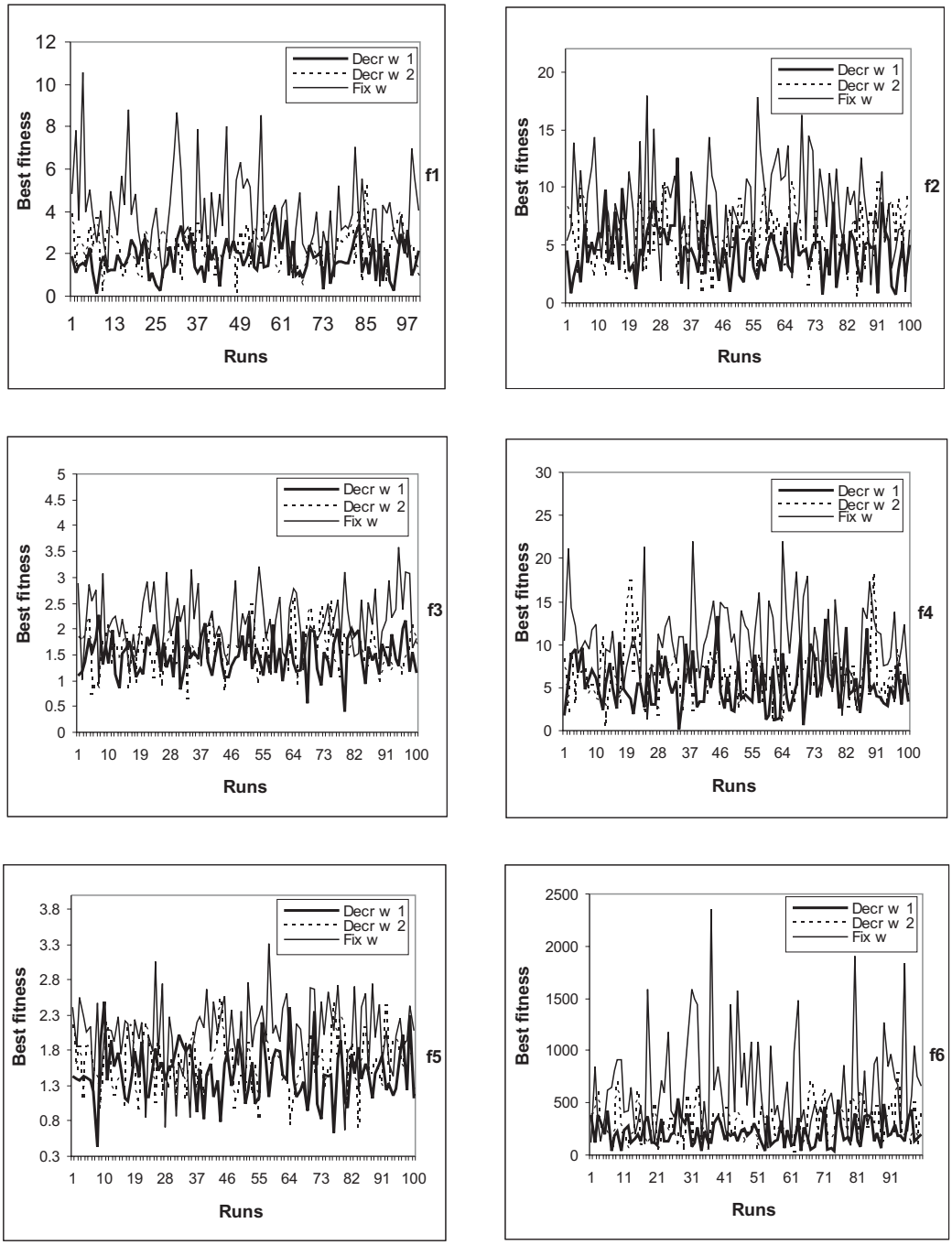


Figure 2: Best value for last 4 functions during 100 runs

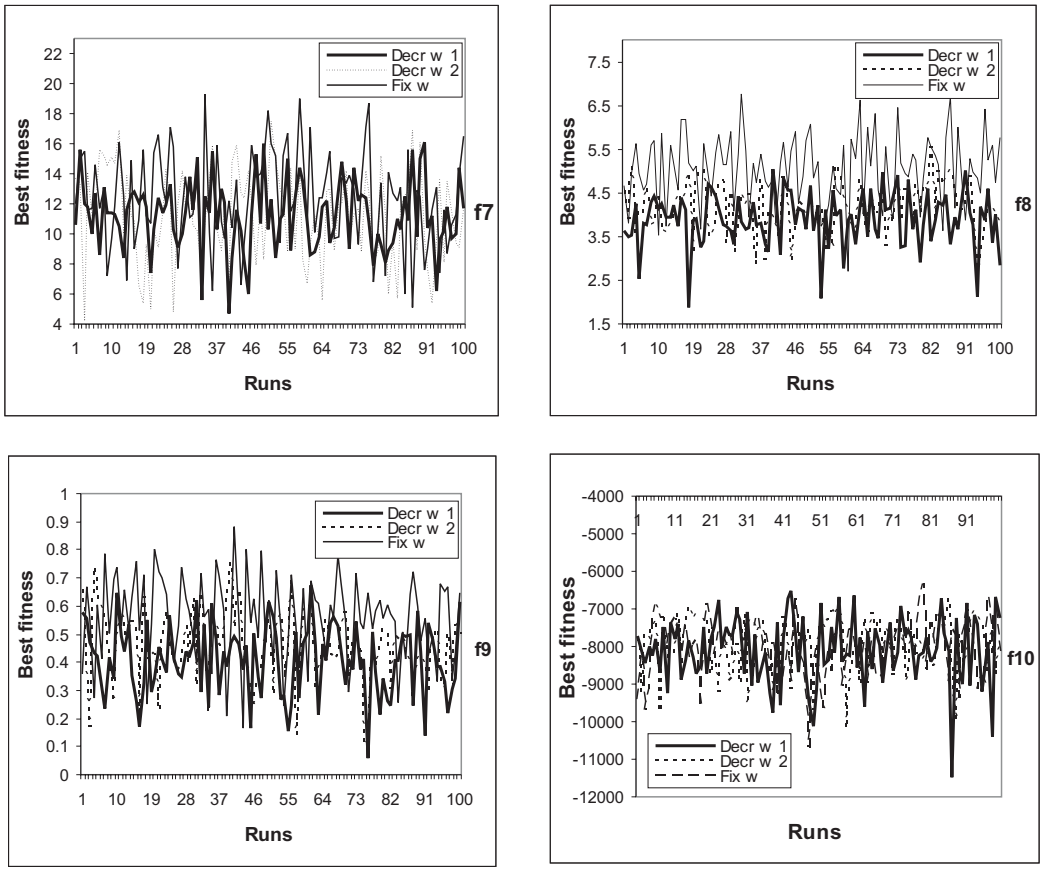


Figure 3: Evolution of best values for different learning factors

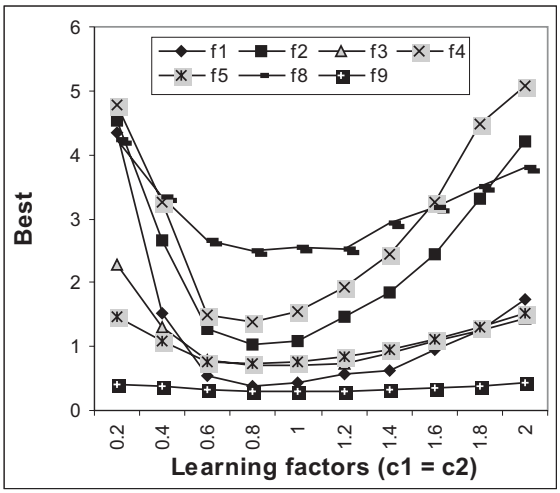
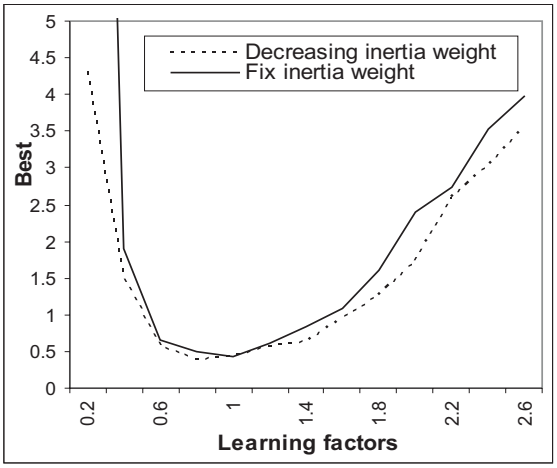


Figure 4: Evolution of best values for different learning factors - comparison for f_1



using different control parameters values. Further numerical experiments will analyze the relationship between the model parameters and it performance.

References

[1] P. Bak, *How Nature Works: The Science of Self-Organized Criticality, Springer, Berlin, 1996.*

[2] A. Brabazon, A. Silva, T. F. Sousa, M. O'Neill, R. Matthews, E. Costa, A Particle Swarm Model of Organizational Adaption, Proceedings of Genetic and Evolutionary Computation Conference, Seattle, USA, 2004.

[3] A. J. Carlisle, *Applying the Particle Swarm Optimizer to Non-Stationary Environments, Ph.D. Thesis, Auburn University, USA, 2002.*

[4] R. C. Eberhart and Y. Shi, Particle swarm optimization: developments, applications and resources, Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2001), Seoul, Korea. 2001

[5] F. Heylighen, *The Science of Self-organization and Adaptivity, in: L. D. Kiel, (ed.) Knowledge Management, Organizational Intelligence and Learning, and Complexity, in: The Encyclopedia of Life Support Systems (EOLSS), Eolss Publishers, Oxford.*

[6] J. Kennedy, R. C. Eberhart, Particle Swarm Optimization, Proceedings of the 1995 IEEE International Conference on Neural Networks, pp. 1942-1948, 1995.

[7] G. Nicolis, I. Prigogine, *I. Self-Organization in Non-Equilibrium Systems, Wiley, New York, 1977.*

[8] Y. Shi, R. Eberhart, Empirical study of particle swarm optimization, Proceedings of the Congress on Evolutionary Computation, pp. 1945-1950, 1999.

[9] Shi, Y. and Eberhart, R. C. *A modified particle swarm optimizer, Proceedings of the IEEE Congress on Evolutionary Computation (CEC 1998), Piscataway, NJ. pp. 69-73, 1998*

[10] P. N. Suganthan, Particle swarm optimiser with neighbourhood operator, Proc. Congress on Evolutionary Computation, pp. 1958- 1962, 1999.

[11] H. Xiaohui, S. Yuhui, R. Eberhart, *Recent Advances in Particle Swarm, Proceedings of the IEEE Congress on Evolutionary Computation, pp. 90 - 97, 2004.*

[12] X. F. Xie, W. J. Zhang, Z. L. Yang, A Dissipative Particle Swarm Optimization, Proceedings of the Congress on Evolutionary Computation (CEC),

Hawaii, USA, 2002, pp. 1456 - 1461.

Laura Dioşan, D. Dmitrescu, Delia David
Department of Computer Science
Babeş Bolyai University of Cluj-Napoca
Address: Kogălniceanu, 1
email: *lauras, ddumitr@cs.ubbcluj.ro, delia_deea@yahoo.com*