Hindawi Publishing Corporation
Journal of Artificial Evolution and Applications
Volume 2008, Article ID 289564, 12 pages
doi:10.1155/2008/289564

Research Article

What Else Is the Evolution of PSO Telling Us?

Laura Dioşan and Mihai Oltean

Department of Computer Science, Faculty of Mathematics and Computer Science, Babeş-Bolyai University, Kogălniceanu 1, 400084 Cluj-Napoca, Romania

Correspondence should be addressed to Laura Dioşan, lauras@cs.ubbcluj.ro

Received 20 July 2007; Revised 29 November 2007; Accepted 29 November 2007

Recommended by Alex Freitas

Evolutionary algorithms (EAs) can be used in order to design particle swarm optimization (PSO) algorithms that work, in some cases, considerably better than the human-designed ones. By analyzing the evolutionary process of designing PSO algorithms, we can identify different swarm phenomena (such as patterns or rules) that can give us deep insights about the swarm behavior. The rules that have been observed can help us design better PSO algorithms for optimization. We investigate and analyze swarm phenomena by looking into the process of evolving PSO algorithms. Several test problems have been analyzed in the experiments and interesting facts can be inferred from the strategy evolution process (the particle quality could influence the update order, some particles are updated more frequently than others, the initial swarm size is not always optimal).

1. INTRODUCTION

Various evolutionary and nonevolutionary methods have been proposed in order to solve complex search and optimization problems. Among these methods, a special place is occupied by evolutionary algorithms (EAs) [1, 2] and by the techniques based on swarm intelligence (such as particle swarm optimization (PSO) [3] and ant colony optimization [4]). The main advantage of these methods is given by the possibility to use them for searching in various spaces without performing big changes in the structure of the algorithm. They can be easily adapted (by the human being or by themselves) to the peculiarities of the problem which is being solved.

PSO is a population-based stochastic optimization technique proposed by Kennedy and Eberhart [5–7]. The standard PSO algorithm randomly initializes a group of particles (solutions) and then searches for optima by updating all the particles along a number of iterations. In any iteration, each particle is updated by following a few simple rules [8, 9].

The standard model implies that particles are updated synchronously [6]. This means that the current position and speed of a particle are computed by taking into account only the information from the previous iteration of particles. The model investigated in this paper is a more general one and it was proposed in [10]. In this paper, we are taking further steps into our research by exploring the phenomena that arise inside a swarm during the evolutionary design.

Our analysis focuses on an asynchronous version of the PSO algorithm. This variant has the following characteristics.

(i) When a particle is updated, the current state of the swarm (the position and the velocity of all the particles) is taken into account. 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 (or synchronous PSO algorithm [6]) where the particles were updated taking into account only the information from the previous iteration (the modifications performed so far by a standard PSO in the current iteration had no influence over the modifications performed further in the current iteration) and it is closer to the asynchronous PSO algorithm [11, 12] that updates particle positions and velocities continuously, based on currently available information.

(ii) In our model, the particles are updated based on their quality. This fitness-based update is important because it could be better to firstly modify the best particles of the swarm and secondly the worst particles (or vice versa). This is again different from the standard asynchronous PSO algorithm [11, 12], which updates the particle positions and velocities by always using the same predefined order: first particle, second particle, and so on (there are no relationships between the update order and the quality of the particles).

(iii) Some particles may be updated more often than other particles. For instance, in some cases, it is more important to update the best particles several times per iteration than to update the worst particles.

(iv) During the evolution, the swarm size can be modified for at least two reasons: some particles perform more moves (in order to improve their quality), while other particles are never updated. This is why the weakest particles are eliminated from the swarm. Unlike the standard PSO algorithm, which works with a pre-established swarm size, the current model finds by itself the optimal size of the swarm along the evolution.

We intend to study how we can obtain better PSO algorithms. In order to achieve this goal we employ an evolutionary approach: we start with a population of randomly generated PSO algorithms and we try to improve them along a fixed number of generations. During the evolution, we try to discover new swarm phenomena such as the special relationships between particles, their quality, their update order, and the optimal swarm size or some rules in the update strategy of the swarm during the evolution process. These rules can be repeatedly applied in order to obtain better approximations of the solution. The process of particle update is influenced by these rules.

Because a PSO is a complex algorithm, we cannot evolve all its aspects. We are taking into account only an important one, namely the order in which the particles are updated. Other aspects, such as the equation used for updating a particle, have been analyzed in [13]. The new swarm phenomena investigated in this paper are

(i) the update frequencies, how many times a particle is updated;
(ii) the order of the updates, the order of particles that is taken into account in order to modify their position and velocity;
(iii) the optimal swarm size.

Such information can help us design better PSO algorithms for optimization. These phenomena have been examined for different test problems.

The paper is structured as follows: Section 2 provides a brief review of the work on PSO parameter optimization. Section 3 describes the model for evolving the PSO update strategy. In Section 4.1, an analysis of the optimal swarm size detected during the evolution is presented. In addition, the frequency of the updates performed in the swarm is investigated. Several rules identified in the PSO update strategy are presented in Section 4.3. Section 4.4 summarizes the most important ideas of this analysis and the main features of the developed model. Conclusions and further work directions are suggested in Section 5.

2. RELATED WORK

Many improvements of the basic form of PSO have been proposed and tested in the literature [14–19]. Also, several analyses of the PSO behavior have been performed [13, 20–22]. Much of this work is focused on the convergence of the PSO algorithm.

Ozcan and Mohan [22] have analyzed the trajectory of a particle in the “original” PSO algorithm (without an inertia weight or a constrict coefficient) and van der Bergh has carried out the first PSO convergence study [23]. Later, Clerc and Kennedy [20] have proposed the model based on the constrict coefficient.

Langdon et al. [21] have evolved kernel functions, which describe the average behavior of a swarm of particles as if it were responding as a single point moving on a landscape transformed by the kernel. The evolved functions (obtained with the genetic programming technique) give another landscape, which is “perceived” by a simple hill climber. The goal for the genetic programming is to evolve a kernel, which causes the movement of the hill climber to resemble the movement of the whole PSO swarm.

Several approaches [13, 19, 24–28] have proposed various hybrid evolutionary algorithms that combine the concepts of EA and PSO.

The aim of the model from [19] has been to extend the PSO algorithm so that it could effectively search into multi-constrained solution spaces (whose constraints are imposed by some real-world problems, such as scheduling), due to the constraints rigidly imposed by the PSO equations. In order to overcome these constraints, this algorithm completely replaces the PSO equations with a self-updating mechanism, which emulates the workings of the equations and which allows flexible incorporation of the real-world heuristics into the algorithm.

Kwong and Jacob [25] have proved the way in which EAs can be used to explore complex pattern formations of swarm systems in 3D space. It is noteworthy that these patterns exhibit a high level of self-organization.

The particle evolutionary swarm optimization algorithm [27] has introduced two new perturbation operators: “c-perturbation” and “m-perturbation.” The goal of these operators is to fight premature convergence and poor diversity issues observed in PSO implementations.

Hendtlass [24], Parsopoulos and Vrahatis [26], and Zhang and Xie [28] have used the differential evolution algorithm (suggested by Storn and Price [29]) for the “on the fly” adaptation of the PSO parameters. Using genetic programming, Poli et al. [13] have studied the possibility of evolving the optimal force generating equations, which control the particles in a PSO (forces that stimulate each particle to fly towards the best point sampled by it and towards the swarm best and back).

Several attempts at evolving evolutionary algorithms (EAs) using similar techniques have been performed in the past. A nongenerational EA has been evolved [30] by using the multiexpression programming technique [31]. A generational EA has been evolved [32] by using the linear genetic programming technique [33–35]. Numerical experiments have shown [30, 32] that the evolved EAs perform similarly and sometimes even better than the standard evolutionary approaches with which they have been compared.

3. THE MODEL FOR EVOLVING THE UPDATE STRATEGY OF PARTICLES

The main idea of the model proposed in [10] is to evolve arrays of integers, which provide a meaning for updating the individuals within a PSO algorithm during iteration. The model is a hybrid technique that works at two levels: the first (macro) level consists in a steady-state genetic algorithm (GA) whose chromosomes encode the update strategy of PSO algorithms. In order to compute the quality of a GA chromosome, a PSO algorithm is run (its update order being encoded into that chromosome). Thus, the second (micro) level consists in a modified PSO algorithm that computes the quality of a GA chromosome.

3.1. Representation

The standard PSO algorithm works with a group of particles (solutions) and then searches for the optima by updating them during iteration. Each particle is updated following two "best" values. The first one is the location of the best solution that a particle has achieved so far. This value is called p Best. Another "best" value is the location of the best solution that any neighbor of a particle has achieved so far. This best value is a neighbourhood best and called n Best.

In a standard PSO algorithm, all the particles will be updated once during the course of the iteration. In a real world swarm (such as a flock of birds), not all the birds update their position and velocity at the same time. Some of them update these values more often and others update theirs later or never. By tacking into account these frequencies, it is interesting to discover (evolve) a model that can tell us which particles/birds must be updated and which is the optimal order for updating them.

A GA [1] is used (in [10]) for evolving the update strategy of a PSO algorithm. Each GA individual is a fixed-length string of genes and each gene is an integer number, from the $\{0, 1, \ldots, \text{Swarm Size} - 1\}$ set. These values represent the indexes of the particles that will be updated during PSO iterations. It is possible that some particles should be updated more often, while others should not be updated at all. Therefore, a GA chromosome must be transformed so that it should contain only the values from 0 to *Max*, where *Max* represents the number of different genes within the current array.

Suppose that we want to evolve the update strategy of a PSO algorithm with eight particles. This means that the Swarm Size = 8 and all the chromosomes of the macrolevel algorithm will have eight genes whose values are in the $\{1, 2, \ldots, 8\}$ set. A GA individual with eight genes can be

$$C_1 = (3, 1, 5, 2, 8, 6, 7, 4). \tag{1}$$

In order to compute the fitness of this chromosome, a swarm with eight individuals is used and the following updates are performed during iteration:

$$\begin{aligned}
&\text{update (Swarm [3])},\\
&\text{update (Swarm [1])},\\
&\text{update (Swarm [5])},\\
&\text{update (Swarm [2])},\\
&\text{update (Swarm [8])},\\
&\text{update (Swarm [6])},\\
&\text{update (Swarm [7])},\\
&\text{update (Swarm [4])}.
\end{aligned} \tag{2}$$

In this example, all the eight particles have been updated once per iteration.

Let us consider another example, which consists of a chromosome C_2 with 8 genes that contain only 5 different values

$$C_2 = (6, 2, 1, 4, 7, 1, 6, 2). \tag{3}$$

In this case, particles 1, 2, and 6 are updated two times each, while the particles 3, 5, and 8 are not updated at all. Because of this, it is necessary to remove the useless particles and to scale the genes of the GA chromosome to the set $\{1, 2, \ldots, 5\}$. The obtained chromosome is

$$C_2' = (4, 2, 1, 3, 5, 1, 4, 2). \tag{4}$$

The quality of this chromosome will be computed by using a swarm of size five (five swarm particles), performing the following eight updates:

$$\begin{aligned}
&\text{update (Swarm [4])},\\
&\text{update (Swarm [2])},\\
&\text{update (Swarm [1])},\\
&\text{update (Swarm [3])},\\
&\text{update (Swarm [5])},\\
&\text{update (Swarm [1])},\\
&\text{update (Swarm [4])},\\
&\text{update (Swarm [2])}.
\end{aligned} \tag{5}$$

Performing this transformation, we can obtain another swarm which has a new size. The model described evolves only the update strategy for a PSO algorithm, but it can also find the optimal swarm size in the same time, even if this parameter is not directly evolved.

We evolve an array of indexes based on the information taken from the function to be optimized. In other words, we evolve an array that contains the update order for the PSO algorithm. This algorithm is used in order to find the optimal value(s) of a function. The quality of the update strategy is given by the performance of the PSO algorithm.

Note that the mentioned mechanism should not be only based on the index of the particles in the Swarm array. This means that it would not be interested in updating a particular position, since the same position can contain a very good individual in one run and a very poor individual in another.

For instance, it is easy to see that all the GA chromosomes, encoding permutations, perform similarly when averaged.

In order to avoid this problem, the Swarm array is sorted in an ascending way (after each iteration), based on the fitness value. The first position will always hold the best particle at the beginning of the iteration. The last particle in this array will always hold the worst particle found at the beginning of the iteration. In this way, it is known that update (Swarm [1]) will mean that the respective particle is not updated, but the best particle at the beginning of the current iteration will be.

3.2. Fitness assignment

The model for evolving the PSO update strategy is structured on two levels: a macrolevel and a microlevel. The macro-level is a GA that evolves the update strategy of a PSO algorithm. For this purpose, a particular function is used as a training problem. The microlevel is a PSO algorithm used for computing the quality of a GA chromosome from the macrolevel.

The array of integers encoded into a GA chromosome represents the update order for the particles used by a PSO algorithm in order to solve a particular problem. The evolved order is embedded in a modified particle swarm optimization algorithm, as described in Section 3.3.

Roughly speaking, the fitness of a GA individual is equal to the fitness of the best solution generated by the PSO algorithm encoded into that GA chromosome. However, since the PSO algorithm uses pseudorandom numbers, it is very likely that successive runs of the same algorithm should generate completely different solutions. This problem can be handled in a standard manner: the PSO algorithm encoded by the GA individual is run multiple times (50 runs, in fact) and the fitness of the GA chromosome is averaged over all the runs.

3.3. The algorithms

The algorithms used in order to evolve the PSO update strategy are described in this section. Because the hybrid technique from [10] combines a GA and a PSO algorithm within a two-level model, two algorithms are described: one for the macrolevel (GA) and another one for the microlevel (PSO algorithm).

3.3.1. The macrolevel algorithm

The macrolevel algorithm is a standard GA [1] used in order to evolve the update order of the particles. We use the steady-state evolutionary model [36] as the underlying mechanism for our GA implementation. The GA starts by creating a random population of individuals. Each individual is a fixed-length array of integer numbers. The following steps are repeated until a given number of generations is reached: two parents are selected using a standard selection procedure [37, 38]. In order to perform selection, which is one step of the algorithm, we run twice a "tournament" [38] between two individuals randomly chosen from the population and select the winner (the one with better fitness). The parents are recombined (using one-cutting point crossover [2, 39]) in order to obtain two offspring O_1 and $O_{2;}$ a crossover point is selected in each parent. The genes after the cutting point are swapped between the parents. The offspring are then considered for weak mutation [40] (the values of one or more genes of each chromosome are replaced with other randomly generated numbers from the $\{1, 2, \dots,$ Swarm_Size $\}$ set), obtaining O_1' and O_2'. The best offspring O^* (out of O_1' and O_2') replaces the worst individual W in the current population only if O^* is better than W [36]. This fact is similar to what happens in nature for longer-lived species where the offspring and parents are alive concurrently and have to compete.

3.3.2. The microlevel algorithm

The microlevel algorithm is a modified PSO algorithm [8, 41] used for computing the fitness of a GA individual from the macrolevel. The algorithm is quite different from the standard synchronous PSO algorithm [8, 41] and from the asynchronous PSO algorithm [11, 12].

The standard PSO algorithm works on two stages: one stage establishes the fitness, the p Best and the n Best values for each particle, and the other stage determines the velocity and makes the updates for each particle. The standard PSO usually works with two populations/swarms. The individuals are updated by computing the p Best and n Best values using the information from the previous population. The newly obtained individuals are added to the current population.

The asynchronous PSO algorithm works only in one stage: the fitness, p Best, n Best, velocity, and position of each particle are continuously updated. The particles are considered one by one for these modifications, following a pre-established order: the first particle, the second particle, and so on.

The developed algorithm performs all the operations in one stage only: it determines the fitness, p Best, n Best, 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 iteration. This PSO algorithm uses only one population/swarm. Each updated particle will automatically replace its parent. Moreover, the genes of the GA chromosome indicate the update order of the particles. The PSO individuals are not modified one by one following the initial order $(1, 2, 3, \dots)$, but following the sequence encoded into the GA chromosome.

Reference [10] presents some numerical experiments for evolving the PSO update strategies. The results obtained have proved the effectiveness of this approach. In this paper, we just remember that the evolved PSO performs better than the standard PSO (see Figure 1).

4. LESSONS LEARNT DURING THE EVOLUTION

We will try to identify some rules in the update strategy of the swarm during the evolving process. We want to identify these rules because they can help us design better PSO algorithms. Before we detail our analysis, we will give a brief definition of the swarm rule.

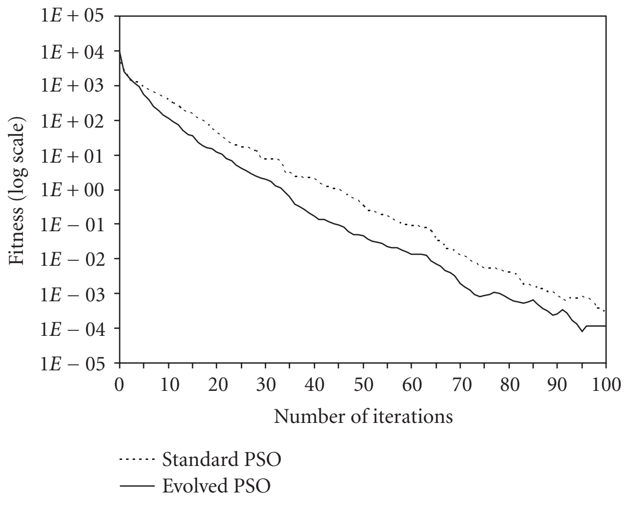

Figure 1: The evolution of the fitness of the best particle along the number of iterations for standard PSO (with a fixed order and equal frequency) and evolved PSO (with the optimized update order/frequency). These results have been obtained for the f_1 test function with 5 dimensions. The results are averaged over 30 runs.

A swarm rule is a sequence of operations, which can be repeatedly used in order to generate a new swarm or some subswarms. The set of rules regards the order of the updates (e.g., this order could be correlated with the particle quality: better particles of the swarm are updated and only after that, the weaker particles are), the frequency of the updates (some particles are updated more frequently than others during the search process), and the optimal swarm size. These rules could emphasize some lessons learnt by the particles during the evolutionary process.

All the numerical experiments performed in this paper use a PSO algorithm in order to optimize several well-known functions. Some of them are unimodal test functions, while others are highly multimodal problems (the number of the local minimum increases exponentially with the dimension of the problem [42]). These functions are described in Table 1.

4.1. Determining the optimal swarm size

First we analyze the evolution of the swarm size along the number of GA generations for an unimodal problem. For GA we use a population of 20 individuals which are evolved during 100 generations. Because we fix the problem size to 5 ($n = 5$), the maximal number of particles for each swarm is 14 (according to Clerc suggestions [41], a good value for the swarm size is $10 + 2 \times \sqrt{n}$). Therefore, each GA individual has 14 genes. We perform binary tournament selection, one cutting point recombination (applied with a probability of 0.8) and weak mutation (applied with a probability of 0.1). The parameters of the PSO algorithm (microlevel) are given in Table 2. The real Swarm Size is not included in this table because different PSOs may have different number of particles. However, the number of function evaluations/iteration is 14 for all the evolved PSO.

In Figure 2 we have depicted the evolution of the swarm size (a swarm whose update strategy is encoded into the GA chromosome) for several particular GA generations in the case of De Jong function 1. The function is smooth, unimodal, strongly convex, symmetric, and continuous.

We can observe that the smallest swarm from the first GA generation contains only seven particles and the largest swarm contains eleven particles. These statistics are repeated during the next generation. Starting with the third generation, the smallest swarm and the largest one tend to decrease their sizes. In the last GA generation, the majority of swarms contain only five particles, this size seems to represent the optimal swarm size for the current problem.

In Figure 3 we have depicted the evolution of the minimum, maximum, and average of the swarm size along the number of GA generations. The test problem was, again, function f_1. The average of the swarm size in a particular generation is computed as the sum of swarm sizes (the swarms encoded into all the GA chromosomes) over the number of individuals from the GA population.

We can observe (in Figure 3) that the average of the swarm size decrease from 9 in the first generation to 6 in the 25th generation and to 5.05 in the 85th GA generation. Starting with the 85th generation, the mean of the swarm size is stabilized at level 5; this value seems to be the optimal size of the swarm for the considered problem.

The same experiment was repeated for each test function presented in Table 1 (for $n = 5$). The results are depicted in Figure 4. We can observe that our model is capable of identifying the optimal swarm size. Even if during the first GA generations there are some fluctuations in the swarm sizes, these sizes tend to be more and more stable, eventually freezing at a particular level to the end of the evolutionary process.

Moreover, we can observe in Figure 4 that the optimal evolved swarm sizes for unimodal test functions have a descendent trend, while the optimal evolved swarm sizes for the multimodal test functions tend to increase along the number of GA generations.

In addition to these remarks, we want to verify our model for more difficult problems. For this purpose, we have chosen to evolve a PSO update order for three functions: Griewangk's function 8, Rosenbrock's valley, and Rastrigin's function 6, for each of them being considered 30 dimensions. Rosenbrock's function is unimodal, but it is considered to be difficult as it has a very narrow ridge (the tip of ridge is very sharp and it runs around a parabola), while the other two functions are highly multimodal and nonlinear. Rastringin's function contains millions of local optima in the interval of consideration, making it a fairly difficult problem. In the Griewangk case, the terms of the summation produce a parabola, while the local optima are above parabola level. The location of the minima is regularly distributed.

We analyze the evolution of the swarm size along the number of GA generations for these difficult problems as well. Actually, the same methodology as that from the previous experiment is applied, but 30 dimensions are considered for each function. Each swarm can contain no more than 20 particles (according to the Clerc's suggestions [41]), and a population of 100 individuals (each individual having 20

Table 1: Test functions used in our experimental study: the parameter n represents the problem size and $f_{\min}$ is the minimum value of the function. All the functions should be minimized.

	Test function	Domain	$f_{\min}$
De Jong's function 1	$f_1(x) = \sum_{i=1}^{n} x_i^2$	$[-100, 100]^n$	0
Axis parallel hyperellipsoid function	$f_2(x) = \sum_{i=1}^{n} (i \cdot x_i^2)$	$[-10, 10]^n$	0
Rosenbrock's valley function	$f_3(x) = \sum_{i=1}^{n-1} 100 \cdot (x_{i+1} - x_i^2)^2 + (1 - x_i)^2$	$[-10, 10]^n$	0
Rastrigin's function 6	$f_4(x) = 10 \cdot n + \sum_{i=1}^{n} (x_i^2 - 10 \cdot \cos(2 \cdot \pi \cdot x_i))$	$[-10, 10]^n$	0
Griewangk's function 8	$f_5(x) = \frac{1}{4000} \cdot \sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	$[-500, 500]^n$	0
Shifted parabola/sphere[a]	$f_6(x) = \sum_{i=1}^{n} (x_i - g_i)$	$[-100, 100]^n$	0

[a]CEC 2005 benchmark.

Table 2: The parameters of the PSO algorithm (the microlevel algorithm) used in order to compute the fitness of a GA chromosome.

Parameters		$n = 5$	$n = 30$
Number of function evaluations/iteration		14	20
Maximal number of evaluations for each PSO run	f_1	1000	7000
	f_2	1000	7000
	f_3	4000	40000
	f_4	4000	40000
	f_5	1000	9000
	f_6	1000	7000
Learning factor c_1		1.193	
Learning factor c_2		1.193	
Inertia weight		0.721	

genes) is evolved during 100 generations. The parameters of the PSO algorithm are those presented in Table 2.

In Figure 5 we have depicted the evolution of the average swarm size along the number of GA generations. We can observe that the average of the swarm size increases during the first 30 GA generations and it tends to stabilize during the rest of generations for the majority of problems.

4.2. *Which particles are updated more often?*

The evolution of update frequencies for each particle along the number of generations is presented in Figure 6. In order to compute this statistic, we calculate the average number of updates for each particle in all the GA chromosomes. For instance, it is possible to update the best particle (which is the first particle from the swarm because they are sorted based on their quality) two times in a GA chromosome and three times in other GA chromosomes. In other words, the first particle is updated for an average of 2.5 times (supposing that we have only two GA chromosomes for this example). In order to obtain a synthesis over all the evolution process, this average is computed for each particle that can be in a swarm (in our case it is possible to have a maximum of ten particles) over all the individuals from the GA population (20 chromosomes in this experiment).

In Figure 6 we have depicted the evolution of the update frequencies only for the f_1 test function (with five dimensions). Therefore, in what follows, we will detail an analysis for this case.

Taking into account the frequency of the updates performed for a particle, we can observe (Figure 6) that the most frequently updated particle is the first one. The best particle is updated for an average of 1.38 times in the first GA generation. This average rises up to 4.5 during the last generation. Note that in the first GA generation, other particles (p_2 to p_8) are updated more frequently than the first one, but taking into account the frequencies from all generations, the first particle is the most updated.

Although the swarm particles are sorted based on their fitness, the next frequently updated particle is the third particle. The average of the update frequency for this particle starts from 1.5 times in the first GA generation and increases to 3, but this value is obtained only in the 75th generation.

A similar trend can be observed for the second and the fourth particles as well. Unlike these particles, the rest

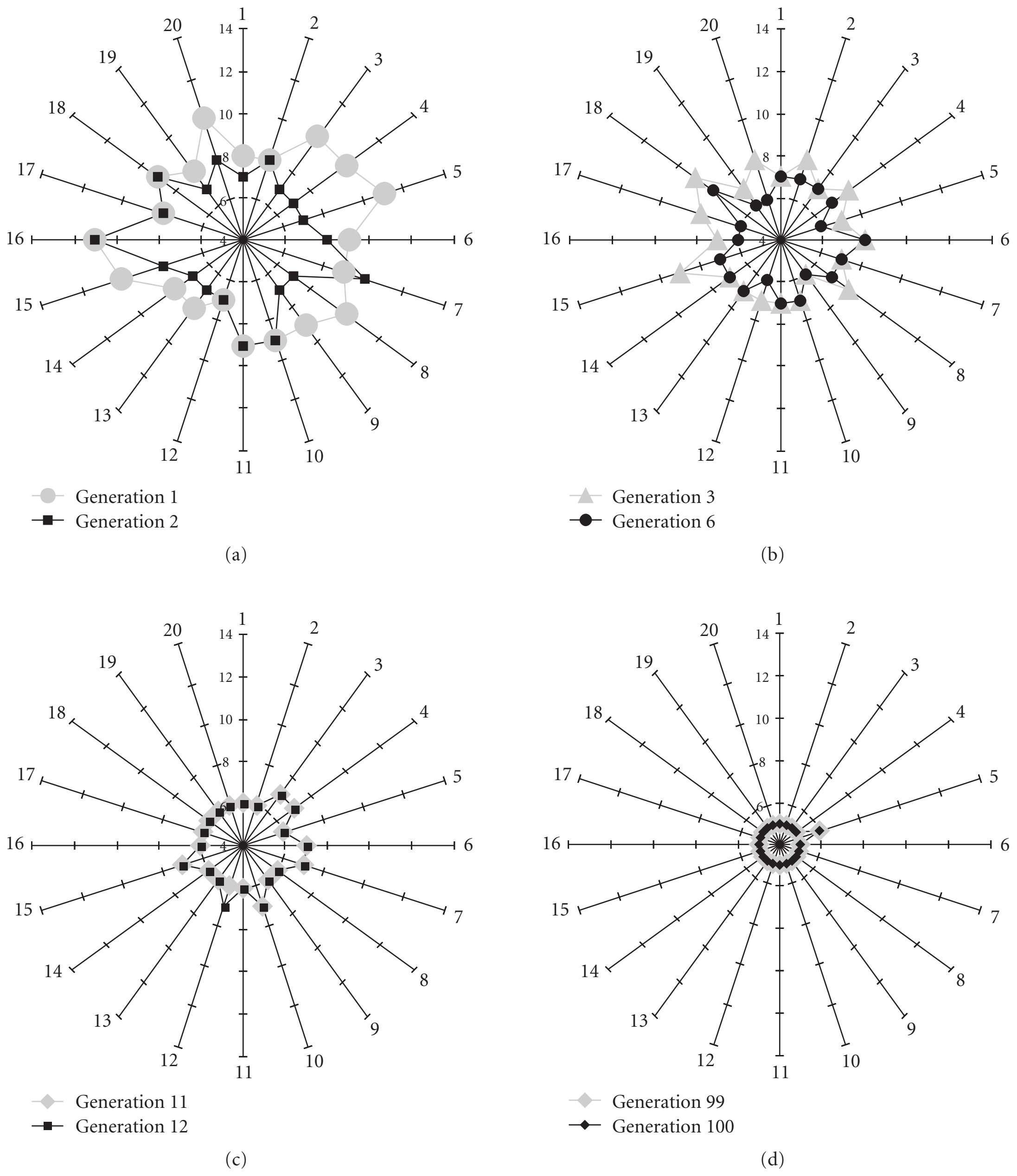

FIGURE 2: The evolution of the swarm size during the GA generations. Each ray depicts the size of a swarm whose update order is encoded into a GA chromosome (we have 20 rays because we work with a GA population made up of 20 chromosomes). These results were obtained for the f_1 test function with 5 dimensions.

of them are updated fewer times as the generation number increases. Moreover, starting with the fourth generation, the eleventh and twelfth particles are no longer updated.

Another remark regards the last two particles from the swarm: p_{13} and p_{14} (the "worst birds" of the swarm). These particles are never updated. In other words, a smaller swarm (with only twelve particles) can solve our problem. This phenomenon can also be observed if we analyze the results obtained for the other test functions: better particles are more frequently updated than the worst particles.

4.3. Analyzing the swarm update order

Several rules identified during the evolution of the PSO update strategy are investigated in this section.

For instance, the qualities of the particles from the PSO algorithm whose update order is encoded in the best GA chromosome for the f_1 test problem with 5 dimensions (obtained in a particular run) are presented in Figure 7. This chromosome is $c = (5\ 6\ 3\ 13\ 6\ 3\ 4\ 3\ 4\ 4\ 5\ 3\ 3\ 5)$ and it encodes the update order for a swarm with only 5 particles (the scaled order of updates being $(3\ 4\ 1\ 5\ 4\ 1\ 2\ 1\ 2\ 2\ 3\ 1\ 1\ 3)$).

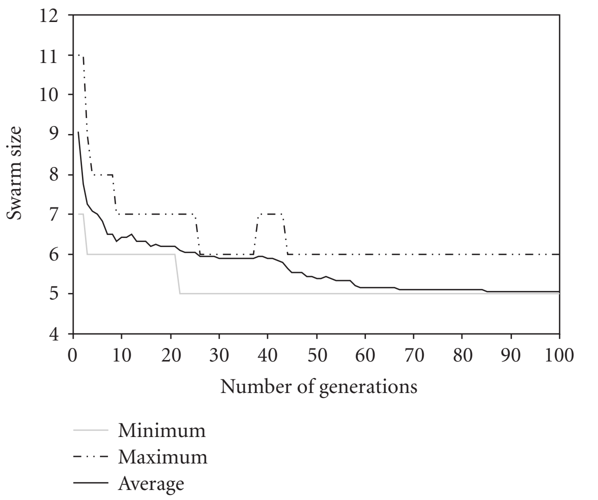

Figure 3: The evolution of the minimum, maximum, and average of the swarm size along the number of generations; the function f_1 with 5 dimensions was used as a test problem.

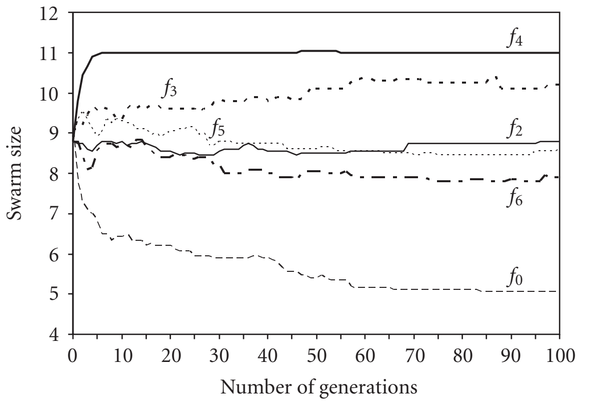

Figure 4: The evolution of the average swarm size along the number of generations for all the test functions ($n = 5$).

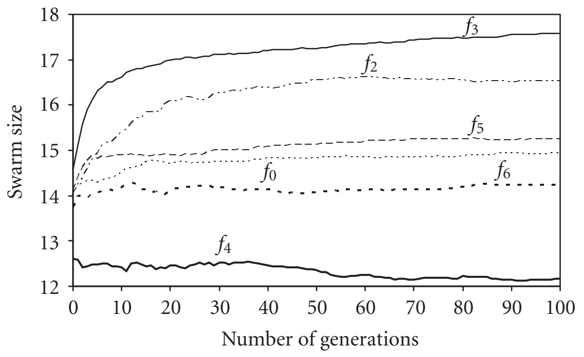

Figure 5: The evolution of the average swarm size along the number of generations for different problems. For all the problems, 30 dimensions have been considered.

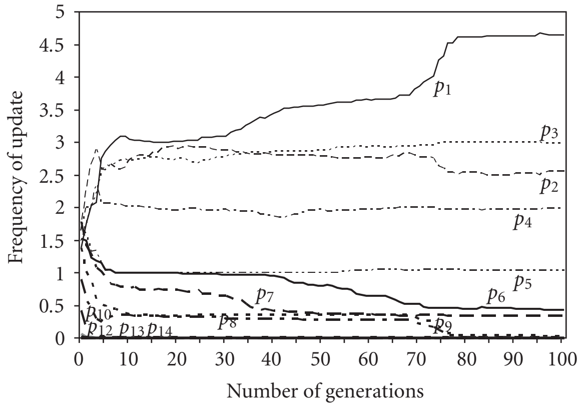

Figure 6: The evolution of the update frequency for each particle during the GA generations. By p_i we have denoted a particle, $i = \overline{1, 14}$.

The values on the Ox scale indicate the update order of the particles. The Oy values indicate the quality of the particles: the lower-sized bars indicate better particles.

The first graphic (top-left corner image) presents the initial swarm with five different particles. The order of the initialization of the particles is as follows: third, fourth, first, fifth, fourth again, first, and so on.

In the second graphic (the top-right image), the first particle is updated at the 3rd step and again, at the 6th step. After this modification, the quality of the first particle (given by the p Best value) is improved. After the 8th step and the 12th step, respectively, when the first particle is updated again, a better fitness is obtained also. Even if at the 13th step the first particle is updated another once, its quality is not improved. An quality improvement is obtained also for the second particle after the 10th step (when this particle is updated for the third time).

A more clear example can be observed in the forth graphic (the middle-right image). In this case, the particles that are modified more times during a PSO iteration (actually all the particles, except the weakest one p_5) improve their quality after each update:

(i) p_3 improves its quality after the 11th and the 14th steps;
(ii) p_4 improves its quality after the 5th step;
(iii) p_1 (the best particle) improves its quality after the 6th, the 8th, the 12th, and the 13th steps;
(iv) p_2 improves its quality after the 9th and the 10th steps.

Table 3 presents the average number of updates performed for each swarm particle of the proposed algorithm (function f_1 with 5 dimensions was considered).

As already stated, we have performed 14 updates inside a swarm even though the swarm size is smaller than 14 (in this case, some particles are updated more than once). For each particle the number of updates performed at each moment is quantified (because we have performed 14 evaluations for each swarm, we obviously have 14 time moments). The particles are indexed based on the p Best value.

The most frequently updated particle is the best one (p_1); it was updated in average for 148 times, while the less frequent updated particle is the worst one. Moreover, we can

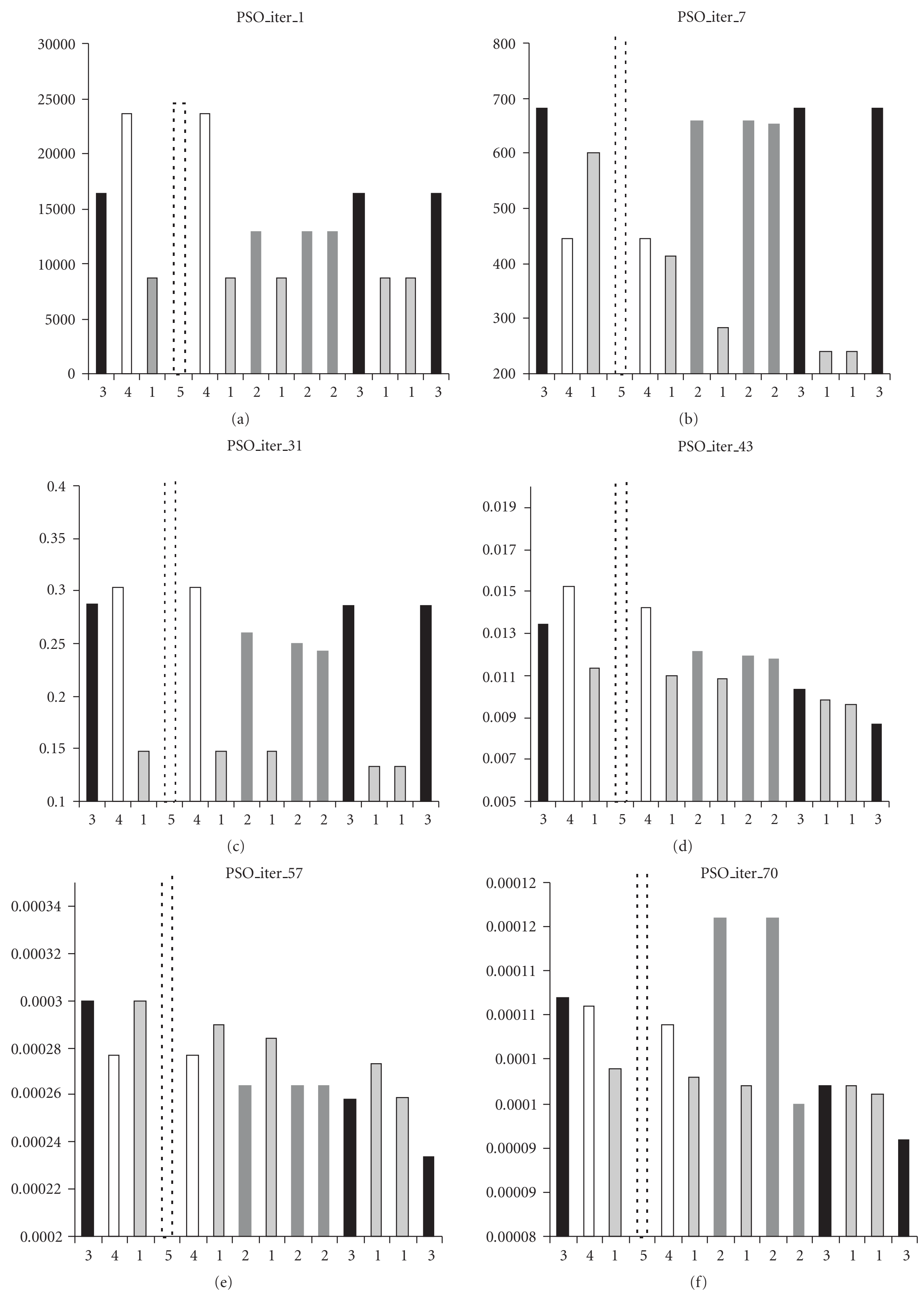

Figure 7: The update order encoded into the best GA chromosome. We have depicted different iterations of the PSO algorithm that updates the particles based on the order encoded into the GA chromosome. Different particles are represented with different colors and the height of the columns represents the quality of a particle; the lower-sized bars indicate better particles.

Table 3: The average number of updates performed for every swarm particle (P) from an iteration to another one or from an update step to another one (during the same PSO iteration).

Particle	p_3	p_4	p_1	p_5	p_4	p_1	p_2	p_1	p_2	p_2	p_3	p_1	p_1	p_3
Number of updates	32	28.6	43.6	18.8	24.8	25.4	41.8	27.6	26.4	25	27	26.4	25.4	24.8

observe that there are more improvements for a particle from a PSO iteration to another PSO iteration than from a step to another one (during the same PSO iteration).

The same conclusion can be drawn if we analyze the results obtained for the other test problems: better particles are more frequently updated than the worst particles are.

4.4. Summarizing the analysis

Analyzing the evolved update order for particles and the rules that appear during the evolution, we can conclude that

(i) the proposed model is able to determine the optimal swarm size at the end of the evolutionary process;
(ii) the evolved update order is based on the particle quality; the best particles are updated, in general, before the worst particles;
(iii) the frequency of the updates: the best particles are more frequently updated than the worst particles; even if the best particle from a swarm is updated more than once, these updates are not consecutive. There are some updates, for other particles than the best, which are performed between two successive updates for the best particle.

5. CONCLUSION AND FURTHER WORK

In this paper, we have performed an analysis of the evolution of PSO algorithms. The model of the PSO algorithm involved in this analysis has several particular properties.

(i) It is different from the standard synchronous PSO where all the particles are simultaneously updated.
(ii) It is similar to the asynchronous PSO where the particles are continuously updated, but it is more general because it considers a dynamical update order and not a predefined one (as in the asynchronous model). Moreover, the update order takes into account the particle quality.

Note that, according to the no-free-lunch theorems [43], we cannot expect to design a perfect PSO which performs the best for all optimization problems. This is why any claim about the generalization ability of the evolved PSO should be made only based on the results provided by numerical experiments.

In addition to this, we have tried to analyze the data generated during the evolution of the update strategy. This will help us understand the nature of the PSO algorithm and design PSO algorithms that use larger swarms. Based on the analysis of the evolved update strategy, we can draw some conclusions. The first one is that the best particles from the swarm are updated most frequently. Moreover, the evolution process can determine the optimal size of the swarm.

Further work will be focused on the following:

(i) several other kinds of function optimization problems will be considered, in an attempt to get more generic results, or to evolve PSOs able to solve a much more challenging and a more interesting kind of problems,
(ii) evolving more parameters of the PSO algorithm (independently, one by one, or synchronously),
(iii) using larger swarms,
(iv) using larger GA population, this will prevent the premature convergence of the macrolevel algorithm,
(v) studying the generalization ability of the evolved PSO algorithm (how well it will perform on some new and difficult problems),
(vi) designing an evolutionary algorithm able to identify by itself the rules analyzed in this paper and studying if these rules will actually improve the quality of a PSO algorithm in terms of convergence speed or accuracy of the solution.

ACKNOWLEDGMENTS

This research was supported by the National Research Grant “Natural computing of excellence,” PNCDI 2007, CNCSIS, Romania. PSO source code was taken from http://www.particleswarm.info.

REFERENCES

[1] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, Boston, Mass, USA, 1989.
[2] J. H. Holland, *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, Mich, USA, 1975.
[3] R. C. Eberhart and Y. Shi, “Particle swarm optimization: developments, applications, and resources,” in *Proceedings of the IEEE Congress on Evolutionary Computation (CEC ’01)*, vol. 1, pp. 81–86, IEEE Press, Seoul, Korea, May 2001.
[4] M. Dorigo, V. Maniezzo, and A. Colorni, “Ant system: optimization by a colony of cooperating agents,” *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, vol. 26, no. 1, pp. 29–41, 1996.
[5] J. Kennedy, “The behavior of particles,” in *Proceedings of the 7th International Conference on Evolutionary Programming VII (EP ’98)*, V. W. Porto, N. Saravanan, D. Waagen, and A. E. Eiben, Eds., vol. 1447 of *Lecture Notes in Computer Science*, pp. 581–589, Springer, San Diego, Calif, USA, 1998.
[6] J. Kennedy and R. C. Eberhart, “Particle swarm optimization,” in *Proceedings of the IEEE International Conference on Neural Networks (ICNN ’95)*, pp. 1942–1948, IEEE Service Center, Perth, Australia, November-December 1995.
[7] J. Kennedy and R. C. Eberhart, “The particle swarm: social adaptation in information-processing systems,” in *New Ideas in Optimization*, D. Corne, M. Dorigo, and F. Glaover, Eds., pp. 379–387, McGraw-Hill, London, UK, 1999.

[8] X. Hu, Y. Shi, and R. Eberhart, "Recent advances in particle swarm," in *Proceedings of the IEEE Congress on Evolutionary Computation (CEC '04)*, vol. 1, pp. 90–97, IEEE Press, Portland, Ore, USA, June 2004.

[9] Y. Shi and R. C. Eberhart, "Empirical study of particle swarm optimization," in *Proceedings of the IEEE Congress on Evolutionary Computation (CEC '99)*, P. J. Angeline, Z. Michalewicz, M. Schoenauer, X. Yao, and A. Zalzala, Eds., vol. 3, pp. 1945–1950, IEEE Service Center, Washington, DC, USA, July 1999.

[10] L. Dioşan and M. Oltean, "Evolving the structure of the particle swarm optimization algorithms," in *Proceedings of the 6th European Conference on Evolutionary Computation in Combinatorial Optimization (EvoCOP '06)*, vol. 3906 of *Lecture Notes in Computer Science*, pp. 25–36, Budapest, Hungary, April 2006.

[11] A. Carlisle and G. Dozier, "An off-the-shelf pso," in *Proceedings of the Particle Swarm Optimization Workshop*, pp. 1–6, Indianapolis, Ind, USA, April 2001.

[12] B.-I. Koh, A. D. George, R. T. Haftka, and B. J. Fregly, "Parallel asynchronous particle swarm optimization," *International Journal for Numerical Methods in Engineering*, vol. 67, no. 4, pp. 578–595, 2006.

[13] R. Poli, W. B. Langdon, and O. Holland, "Extending particle swarm optimisation via genetic programming," in *Proceedings of the 8th European Conference on Genetic Programming (EuroGP '05)*, vol. 3447 of *Lecture Notes in Computer Science*, pp. 291–300, Lausanne, Switzerland, March-April 2005.

[14] C. A. Coello Coello and M. Salazar Lechuga, "MOPSO: a proposal for multiple objective particle swarm optimization," in *Proceedings of the IEEE Congress on Evolutionary Computation (CEC '02)*, vol. 2, pp. 1051–1056, IEEE Service Center, Honolulu, Hawaii, USA, May 2002.

[15] X. Hu and R. Eberhart, "Multi-objective optimization using dynamic neighborhood particle swarm optimization," in *Proceedings of the IEEE Congress on Evolutionary Computation (CEC '02)*, vol. 2, pp. 1677–1681, IEEE Service Center, Honolulu, Hawaii, USA, May 2002.

[16] X. Hu, R. C. Eberhart, and Y. Shi, "Swarm intelligence for permutation optimization: case study of n-queens problem," in *Proceedings of the IEEE Swarm Intelligence Symposium (SIS '03)*, Y. Com, Ed., pp. 243–246, Indianapolis, Ind, USA, April 2003.

[17] J. Kennedy and R. Eberhart, "A discrete binary version of the particle swarm algorithm," in *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, vol. 5, pp. 4104–4108, IEEE Service Center, Orlando, Fla, USA, October 1997.

[18] C. K. Mohan and B. Al-kazemi, "Discrete particle swarm optimization," in *Proceedings of the Particle Swarm Optimization Workshop*, Indianapolis, Ind, USA, April 2001.

[19] D. Srinivasan and T. H. Seow, "Particle swarm inspired evolutionary algorithm (PS-EA) for multi-objective optimization problem," in *Proceedings of the IEEE Congress on Evolutionary Computation (CEC '03)*, R. Sarker, R. Reynolds, H. Abbass, et al., Eds., pp. 2292–2297, IEEE Press, Canbella, Australia, December 2003.

[20] M. Clerc and J. Kennedy, "The particle swarm—explosion, stability, and convergence in a multidimensional complex space," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 1, pp. 58–73, 2002.

[21] W. B. Langdon, R. Poli, and C. R. Stephens, "Kernel methods for PSOs," Tech. Rep., Computer Science, University of Essex, UK, 2005.

[22] E. Ozcan and C. Mohan, "Particle swarm optimization: surfing the waves," in *Proceedings of the IEEE Congress on Evolutionary Computation (CEC '99)*, pp. 1939–1944, IEEE Service Center, Washington, DC, USA, July 1999.

[23] F. van den Bergh, *An Analysis of Particle Swarm Optimizers*, Ph.D. thesis, Department of Computer Science, University of Pretoria, Pretoria, South Africa, 2002.

[24] T. Hendtlass, "A combined swarm differential evolution algorithm for optimization problems," in *Proceedings of the 14th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems (IEA/AIE '01)*, L. Monostori, J. Váncza, and M. Ali, Eds., vol. 2070 of *Lecture Notes in Computer Science*, pp. 11–18, Budapest, Hungary, June 2001.

[25] H. Kwong and C. Jacob, "Evolutionary exploration of dynamic swarm behaviour," in *Proceedings of the IEEE Congress on Evolutionary Computation (CEC '03)*, R. Sarker, R. Reynolds, H. Abbass, et al., Eds., pp. 367–374, IEEE Press, Canbella, Australia, December 2003.

[26] K. E. Parsopoulos and M. N. Vrahatis, "Recent approaches to global optimization problems through particle swarm optimization," *Natural Computing*, vol. 1, no. 2–3, pp. 235–306, 2002.

[27] A. E. M. Zavala, A. H. Aguirre, and E. R. V. Diharce, "Particle evolutionary swarm optimization algorithm (PESO)," in *Proceedings of the 6th Mexican International Conference on Computer Science (ENC '05)*, vol. 2005, pp. 282–289, IEEE Computer Society, Guanajuato, Mexico, September 2005.

[28] W.-J. Zhang and X.-F. Xie, "DEPSO: hybrid particle swarm with differential evolution operator," in *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, vol. 4, pp. 3816–3821, IEEE Press, Washington, DC, USA, October 2003.

[29] R. Storn and K. Price, "Differential evolution—a simple and effcient adaptive scheme for global optimization over continuous spaces," Tech. Rep., International Computer Science Institute, Berkeley, Calif, USA, 1995.

[30] M. Oltean, "Solving even-parity problems using multi expression programming," in *Proceedings of the 7th Joint Conference on Information Sciences*, K. Chen, Ed., vol. 1, pp. 315–318, Association for Intelligent Machinery, Cary, NC, USA, September 2003.

[31] M. Oltean and C. Grosan, "Evolving evolutionary algorithms using multi expression programming," in *Proceedings of the 7th European Conference on Artificial Life (ECAL '03)*, W. Banzhaf, T. Christaller, P. Dittrich, J. T. Kim, and J. Ziegler, Eds., vol. 2801 of *Lecture Notes in Computer Science*, pp. 651–658, Springer, Dortmund, Germany, September 2003.

[32] M. Oltean, "Evolving evolutionary algorithms using linear genetic programming," *Evolutionary Computation*, vol. 13, no. 3, pp. 387–410, 2005.

[33] M. Brameier and W. Banzhaf, "A comparison of linear genetic programming and neural networks in medical data mining," *IEEE Transactions on Evolutionary Computation*, vol. 5, no. 1, pp. 17–26, 2001.

[34] M. Brameier, W. Banzhaf, et al., "Evolving teams of predictors with linear genetic programming," *Genetic Programming and Evolvable Machines*, vol. 2, no. 4, pp. 381–407, 2001.

[35] M. Brameier and W. Banzhaf, "Explicit control of diversity and effective variation distance in linear genetic programming," in *Proceedings of the 5th European Conference on Genetic Programming (EuroGP '02)*, vol. 2278 of *Lecture Notes in Computer Science*, pp. 37–49, Kinsale, Ireland, April 2002.

[36] G. Syswerda, "A study of reproduction in generational and steady state genetic algorithms," in *Foundations of Genetic Algorithms*, G. J. E. Rawlins, Ed., pp. 94–101, Morgan Kaufmann, San Francisco, Calif, USA, 1991.

[37] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolutionprograms*, Springer, NewYork, NY, USA, 1996.

[38] B. L. Miller and D. E. Goldberg, "Genetic algorithms, tournament selection, and the effects of noise," *Complex Systems*, vol. 9, pp. 193–212, 1995.

[39] L. J. Eshelman, "The CHC adaptive search algorithm: how to have safe search when engaging in non-traditional genetic recombination," in *Foundations of Genetic Algorithms*, G. J. E. Rawlins, Ed., pp. 265–283, Morgan Kaufmann, San Francisco, Calif, USA, 1991.

[40] W. E. Howden, "Weak mutation testing and completeness of test sets," *IEEE Transactions on Software Engineering*, vol. 8, no. 4, pp. 371–379, 1982.

[41] M. Clerc, *Particle Swarm Optimization*, ISTE, London, UK, 2006.

[42] X. Yao, Y. Liu, and G. Lin, "Evolutionary programming made faster," *IEEE Transactions on Evolutionary Computation*, vol. 3, no. 2, pp. 82–102, 1999.

[43] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 67–82, 1997.