A New Locally Convergent Particle Swarm Optimiser

F. van den Bergh, A. P.E ngelbrecht

Department of Computer Science
University of Pretoria
Pretoria, South Africa
fvdbergh@cs.up.ac.za, engel@driesie.cs.up.ac.za

Abstract—**This paper introduces a new Particle Swarm Optimisation (PSO) algorithm with strong local convergence properties. The new algorithm performs much better with a smaller number of particles, compared to the original PSO. This property is desirable when designing a niching PSO algorithm.**

Keywords—**Particle Swarm Optimiser, local convergence**

I. Introduction

The Particle Swarm Optimiser (PSO) is a population-based optimisation method first proposed by Kennedy and Eberhart [1]. Some of the attractive features of the PSO include ease of implementation and the fact that no gradient information is required. It can be used to solve a wide array of different optimisation problems; some example applications include neural network training [2][3][4][5] and function minimisation [6][7].

The original PSO algorithm does not have guaranteed convergence properties, as shown in [8]. A new PSO-based algorithm, called the Guaranteed Convergence Particle Swarm Optimiser (GCPSO), is introduced here. This new algorithm has provably guaranteed convergence onto local minima.

Section II provides a definition of the PSO algorithm, followed by a brief discussion of related work in Section III. The new algorithm is introduced in Section IV. Experimental results obtained with the new algorithm is presented in Section V, followed by a conclusion and some ideas for future research in Section VI.

II. PSO Definition

A formal definition of the PSO algorithm is presented in this section, defining the terminology used throughout this paper.

Each individual particle i has the following properties: A current position in search space, $\mathbf{x}_i$, a current velocity, $\mathbf{v}_i$, and a personal best position in search space, $\mathbf{y}_i$. The personal best position, $\mathbf{y}_i$, corresponds to the position in search space where particle i had the smallest error as determined by the objective function f, assuming a minimisation task. The position yielding the lowest error amongst all the $\mathbf{y}_i$ is called the *global best position*, and is denoted $\hat{\mathbf{y}}$. Equations (1) and (2) define how the personal and global best values are updated, respectively. It is assumed below that the swarm consists of s particles, thus $i \in [1..s]$.

$$\mathbf{y}_i(t+1) = \begin{cases} \mathbf{y}_i(t) & \text{if } f(\mathbf{x}_i(t+1)) \geq f(\mathbf{y}_i(t)) \\ \mathbf{x}_i(t+1) & \text{if } f(\mathbf{x}_i(t+1)) < f(\mathbf{y}_i(t)) \end{cases} \tag{1}$$

$$\hat{\mathbf{y}}(t) \in \{\mathbf{y}_0(t), \mathbf{y}_1(t), \ldots, \mathbf{y}_s(t)\} \mid f(\hat{\mathbf{y}}(t)) = \min\{f(\mathbf{y}_0(t)), f(\mathbf{y}_1(t)), \ldots, f(\mathbf{y}_s(t))\} \tag{2}$$

During each iteration each particle in the swarm is updated using (3) and (4). Two pseudo-random sequences, $r_1 \sim U(0,1)$ and $r_2 \sim U(0,1)$ are used to effect the stochastic nature of the algorithm. For all dimensions $j \in 1..n$, let $x_{i,j}$, $y_{i,j}$ and $v_{i,j}$ be the current position, current personal best position and velocity of the j^{th} dimension of the i^{th} particle. The purpose of the inertia weight w, and the constants c_1 and c_2, is discussed in more detail below. The velocity update step is

$$v_{i,j}(t+1) = wv_{i,j}(t) + c_1 r_{1,j}(t)[y_{i,j}(t) - x_{i,j}(t)] + c_2 r_{2,j}(t)[\hat{y}_j(t) - x_{i,j}(t)] \tag{3}$$

The new velocity is then added to the current position of the particle to obtain the next position of the particle:

$$\mathbf{x}_i(t+1) = \mathbf{x}_i(t) + \mathbf{v}_i(t+1) \tag{4}$$

The value of each dimension of every velocity vector $\mathbf{v}_i$ is clamped to the range $[-v_{max}, v_{max}]$ to reduce the likelihood of the particle leaving the search space. The value of v_{max} is usually chosen to be $k \times x_{max}$, with $0.1 \leq k \leq 1.0$ [9], where x_{max} denotes the domain of the search space. Note that this does not restrict the values of $\mathbf{x}_i$ to the range $[-v_{max}, v_{max}]$; it merely limits the maximum distance that a particle will move during one iteration.

The *inertia weight*, w, in equation (3) is used to control the convergence behaviour of the PSO. Small values of w result in more rapid convergence — usually on a suboptimal position, while a too large value may prevent convergence. Typical implementations of the PSO adapt the value of w during the training run, *e.g.* linearly decreasing it from 1 to near 0 over the run. Convergence can be obtained with fixed values as shown in [10][8].

The acceleration coefficients, c_1 and c_2, control how far a particle will move in a single iteration. Typically these are both set to a value of 2.0 [9], although it has been shown that setting $c_1 \neq c_2$ can lead to improved performance [11].

III. Related Work

The PSO is related to the Genetic Algorithm, another population based optimisation technique. Angeline exploited this relationship by adding fitness-based selection to the PSO update process [12], *i.e.* the properties of the better particles are shared with the worse particles through replication. Another Evolutionary Computation inspired idea was that of Løvbjerg *et al.* [13], who added crossover operators to combine two members in the population to form two new ones. They further partitioned the swarm into subpopulations — unfortunately this meant that the number of particles per subpopulation dwindled as the number of subpopulations was increased.

Not much research has been done to theoretically analyse the PSO algorithm. Clerc investigated a deterministic version of the algorithm [14], but did not extend his model to the general stochastic case. A later paper by Clerc *et al.* further investigated the PSO's convergence characteristics [10], however, they failed to prove that the algorithm will converge on a minimiser. Ozcan and Mohan [15] published explicit equations describing the particle trajectories for the original PSO algorithm. Their analysis did not discuss the purpose of the inertia weight in detail.

IV. A Modified Particle Swarm Optimiser (GCPSO)

The original PSO algorithm introduced in [1], including the later inertia weight and constriction factor versions, all have a potentially dangerous property: if $\mathbf{x}_i = \mathbf{y}_i = \hat{\mathbf{y}}$, then the velocity update will depend only on the value of $wv_{i,j}(t)$. In other words, if a particle's current position coincides with the global best position/particle, the particle will only move away from this point if its previous velocity and w are non-zero. If their previous velocities are very close to zero, then all the particles will stop moving once they catch up with the global best particle, which may lead to premature convergence of the algorithm. In fact, this does not even guarantee that the algorithm has converged on a local minimum — it merely means that all the particles have converged on the best position discovered so far by the swarm. This phenomenon will be referred to as *stagnation*. A more formal proof of this property can be found in [8].

To address this issue a new parameter is introduced to the PSO algorithm. Let τ be the index of the global best particle, so that

$$\mathbf{y}_\tau = \hat{\mathbf{y}}$$

In order to keep the global best particle moving until it has reached a local minimum, a new velocity update equation for the global best particle (*i.e.* particle τ) is suggested, so that

$$\begin{aligned} v_{\tau,j}(t+1) &= -x_{\tau,j}(t) + \hat{y}_j(t) + wv_{\tau,j}(t) + \\ &\quad \rho(t)(1 - 2r_{2,j}(t)) \end{aligned} \tag{5}$$

where ρ is a scaling factor defined below. The other particles in the swarm continue using the usual velocity update equation (*e.g.* equation 3). Briefly, the $-x_{\tau,j}(t)$ term "resets" the particle's position to the position $\hat{y}_j$. To this position a vector representing the current search direction, represented by the term $wv_{\tau,j}(t)$, is added. The $\rho(t)(1 - 2r_{2,j}(t))$ term generates a random sample from a sample space with side lengths $2\rho(t)$.

Combining the position update step (equation 4) and the new velocity update step (equation 5) for the global best particle τ results in the new position update equation

$$x_{\tau,j}(t+1) = \hat{y}_j(t) + wv_{\tau,j}(t) + \rho(t)(1 - 2r_2(t)) \tag{6}$$

The addition of the ρ term causes the PSO to perform a random search in an area surrounding the global best position $\hat{\mathbf{y}}$. The diameter of this search area is controlled by the parameter ρ. The value of $\rho(t)$ is adapted after each time step, using

$$\rho(t+1) = \begin{cases} 2\rho(t) & \text{if \#successes} > s_c \\ 0.5\rho(t) & \text{if \#failures} > f_c \\ \rho(t) & \text{otherwise} \end{cases} \tag{7}$$

where the terms *#failures* and *#successes* denote the number of consecutivef ailures or successes, respectively, where a failure is defined as $f(\hat{\mathbf{y}}(t)) = f(\hat{\mathbf{y}}(t-1))$. A default initial value of $\rho(0) = 1.0$ was found empirically to produce acceptable results. The values s_c and f_c are threshold parameters, discussed in more detail below. The following additional rules must also be implemented to ensure that equation (7) is well-defined:

$$\#successes(t+1) > \#successes(t) \Rightarrow \#failures(t+1) = 0$$

and

$$\#failures(t+1) > \#failures(t) \Rightarrow \#successes(t+1) = 0$$

Thus, on a success the failure count is set to zero, and likewise the success count is reset when a failure occurs.

The optimal choice of values for the parameters f_c and s_c depend on the objective function. In high-dimensional search spaces it is difficult to obtain better values using a random search in only a few iterations, so it is recommended to set $f_c = 5$, $s_c = 15$. These settings imply that the algorithm is quicker to punish a poor ρ setting than it is to reward a successful ρ value; a strategy found empirically (in Section V) to produce acceptable results.

Alternatively, the optimal values for f_c and $s_c$ca n be learnt dynamically. For example, the value of $s_c$ca n be

increased every time that $\#failures > f_c$, in other words, it becomes more difficult to reach the success state if failures occur frequently. This prevents the value of ρ from oscillating rapidly. Using this scheme the parameters can adapt to the local conditions of the error surface, with the ability to learn new settings when the error surface changes. A similar strategy can be designed for f_c.

The value of ρ is adapted in an attempt to learn the optimal size of the sampling volume given the current state of the algorithm. When a specific ρ value repeatedly results in a success, a larger sampling volume is selected to increase the maximum distance traveled in one step. Conversely, if ρ produces f_c consecutive failures, then the sampling volume is too large and must be reduced.

When ρ becomes sufficiently small (compared to the machine's precision, for example) the algorithm can either halt or keep ρ fixed at this lower bound until some other stopping criterion is met. Note that halting may not be the best option, as information regarding the position of the other particles must also be taken into account. A typical example might be where some of the particles are still exploring a distant region of the search space, while the global best particle has already converged on the local minimum closest to it. In this case the distant particles may still be able to discover a better minimum, so the algorithm should continue until the maximum allowed number of iterations have been reached.

The PSO algorithm using equation (5) to update the position of its global best particle is called the Guaranteed Convergence Particle Swarm Optimiser (GCPSO). A proof of guaranteed convergence onto local minima for this algorithm can be found in [8]. Note that neither the GCPSO nor the original PSO have guaranteed convergence onto global minima.

V. Results

This section compares the GCPSO algorithm to the original PSO algorithm, using two unimodal and two non-unimodal functions. The following functions were used to test the algorithms:

Spherical: A very simple, unimodal function with its global minimum located at $\mathbf{x}^* = \mathbf{0}$, with $f(\mathbf{x}^*) = 0$. This function has no interaction between its variables.

$$f_1(\mathbf{x}) = \sum_{i=1}^{n} x_i^2 \tag{8}$$

Note that this function is actually a quadratic function. The name 'Spherical' is used here since many other papers have used that name for this function, *e.g.* [16].

Quadric: A variation of the Spherical function, but with significant interaction between its variables. The global minimiser is located at $\mathbf{x}^* = \mathbf{0}$, so that $f(\mathbf{x}^*) = 0$.

$$f_2(\mathbf{x}) = \sum_{i=1}^{n} \left(\sum_{j=1}^{i} x_j \right)^2 \tag{9}$$

Ackley: A multi-modal function with deep local minima. The global minimiser is $\mathbf{x}^* = \mathbf{0}$, with $f(\mathbf{x}^*) = 0$. Note that the variables are independent.

$$\begin{aligned} f_3(\mathbf{x}) \quad = \quad & -20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) \\ & -\exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right) \\ & +20+e \end{aligned} \tag{10}$$

Rastrigin: A multi-modal version of the Spherical function, characterised by deep local minima arranged as sinusoidal bumps. The global minimum is $f(\mathbf{x}^*) = 0$, where $\mathbf{x}^* = \mathbf{0}$. The variables of this function are independent.

$$f_4(\mathbf{x}) = \sum_{i=1}^{n} \left(x_i^2 - 10\cos(2\pi x_i) + 10\right) \tag{11}$$

TABLE I
FUNCTION PARAMETERS

Function	n	Domain	Threshold
Ackley	30	30	5.00
Rastrigin	30	5.12	100
Spherical	30	100	0.01
Quadric	30	100	0.01

Table I lists the parameter settings for the functions in the benchmark suite. Note that all functions were tested using 30-dimensional search spaces.

All experiments consisted of 50 runs. The PSO parameters were set to the values $w = 0.72$, and $c_1 = c_2 = 1.49$ — these values lead to convergent behaviour [8].

Note that the GCPSO has some additional parameters that can be fine-tuned. The default parameters, specified in Section IV, were used throughout. Although these parameters may not be optimal, they have been found to produce acceptable results on a small set of test functions.

A. Fixed Number of Iterations

Table II presents the results of minimising various functions using both the PSO and GCPSO algorithms. The column labeled "s" lists the number of particles in the swarm for each row. When comparing the results in the table, keep in mind that both the Ackley and Rastrigin functions contain many local minima, and that both

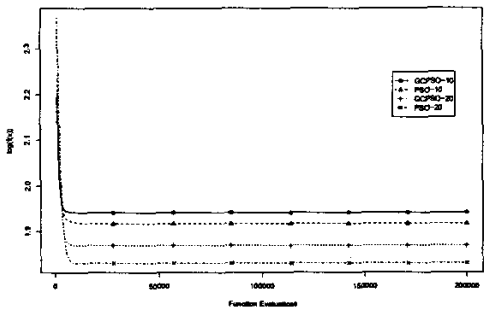

Fig. 1. Error profile for the Rastrigin function

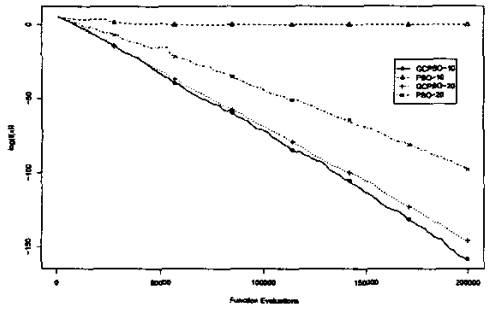

Fig. 2. Error profile for the Quadric function

the GCPSO and PSO algorithms are not explicitly designed to deal with this type of function without some mechanism to help them to locate the global minimiser. This implies that the quality of the solutions discovered by the algorithms are highly dependent on the initial positions of the particles, which were randomly chosen.

On the Ackley function, the GCPSO performed significantly better than the PSO when using only two particles, but there was no significant difference between the performance of the algorithms when 20 particles were used. The Rastrigin function produced similar results in the two-particle case, but the original PSO performed significantly better on the 20-particle experiment. Keep in mind that the results for both the Ackley and Rastrigin functions are presented only for completeness — neither algorithm is properly equiped to deal with multi-modal functions (see [8] for a formal proof).

The two unimodal functions in Table II clearly show the stronger local convergence property of the GCPSO. The Spherical function is exceedingly simple, having no interaction between the variables and only a single (thus global) minimum. The GCPSO is able to minimise this function to a very high degree when using only two particles; in contrast, the original PSO struggles with premature convergence if only two particles are used. This is a clear example of the stagnation mentioned in Section IV. Note that adding more particles allowed the standard PSO to perform significantly better; even the GCPSO benefitted from the increased diversity offered by the larger number of particles.

The last unimodal function, Quadric, illustrates the same concept, but to a less striking degree. This function has significant interaction between its variables, making this problem harder to solve than the Spherical function. Note that there was a large jump in performance between using 2 particles and 20 particles, even when using the GCPSO. This implies that the greater diversity provided by the additional particles helps the PSO to solve the problem more quickly.

Figure 1 plots the error profile of the Rastrigin function for PSO and GCPSO algorithms for swarm sizes of 10 and 20 particles. The most significant feature of this plot is the fact that all the algorithms become trapped in local minima, as indicated by their unchanging function values after the first 10000 function evaluations. Note that the original PSO fared slightly better on this function, and that both algorithms improved somewhat when larger swarms were used.

Figure 2 presents the error profile of the Quadric function. Note how the 10-particle PSO stagnates after about 50000 function evaluations. The 20-particle PSO, as well as both the GCPSOs, continue to improve during the whole run. Note how the 10-particle GCPSO gradually outperforms the 20-particle GCPSO — a property that will be discussed in more detail in the next section.

B. Rate of Convergence

Table III presents results obtained by running the various algorithms until they discovered an objective function value below the corresponding value listed in the 'threshold' column of Table I. These results give an indication of the robustness as well as the rate of convergence of the two algorithms on the various benchmark functions. The 's' column of Table III lists the number of particles, the 'N_s' column the number of runs (out of 50) that reached the threshold and the 'F' column lists the average number of objective function evaluations (only for the runs that reached the threshold).

The results for the non-unimodal functions, Ackley and Rastrigin, show that both the GCPSO and the PSO performed comparably. Both algorithms struggled to reach the threshold of the Ackley function using only 10 particles, but their performance improved as more particles were added. On Rastrigin's function, the original PSO seems to have fared slightly better than the GCPSO using 20 particles.

The algorithm's performance on the unimodal functions exhibited much clearer trends. The original PSO

TABLE II

COMPARING GCPSO AND PSO ON VARIOUS FUNCTIONS, AFTER 2×10^5 FUNCTION EVALUATIONS

Function	s	GCPSO	PSO
Ackley	2	1.85e+01 ± 1.99e−01	1.93e+01 ± 2.03e−01
	20	2.70e+00 ± 6.36e−01	3.40e+00 ± 4.58e−01
Rastrigin	2	1.81e+02 ± 2.13e+01	2.93e+02 ± 1.46e+01
	20	7.61e+01 ± 5.07e+00	6.52e+01 ± 4.84e+00
Spherical	2	6.54e−084 ± 1.44e−007	4.03e+004 ± 2.80e+003
	20	2.09e−201 ± 0.00e+00	1.15e−110 ± 1.37e−097
Quadric	2	2.87e+003 ± 1.90e+003	1.14e+007 ± 1.14e+006
	20	9.45e−152 ± 3.87e−146	3.65e−107 ± 4.20e−098

requires more function evaluations when fewer particles are used, which appears illogical given the fact that the objective function is unimodal, with no local minima to trap particles. The only explanation for this behaviour is that the PSO needs about 20 particles to prevent stagnation — even on a unimodal function. Using a swarm size of more than 30 particles causes the PSO to use more function evaluations to reach the threshold.

The GCPSO algorithm exhibits exactly the opposite behaviour, requiring fewer function evaluations when using fewer particles. This behaviour is expected: adding more particles incurs a greater overhead, since each particle consumes one function evaluation per iteration. This behaviour is clearly desirable when dealing with unimodal functions, since they do not have local minima that could trap some particles, thus necessitating the use of a large swarm.

These results show that the GCPSO is perfectly capable of locating the minimum of a unimodal function with only a small number of particles, implying that it does not need many particles to facilitate a local search. This property is desirable when the swarm is partitioned into sub-swarms, like in the breeding PSO proposed by Løvbjerg *et al.*[13]. To summarise, the GCPSO algorithm has significantly faster convergence on unimodal functions, especially when smaller swarm sizes are used. This improved performance is not visible on multi-modal functions, because the GCPSO can still become trapped in local minima, just like the original PSO. In fact, the GCPSO may be slightly more prone to becoming trapped because of its faster rate of convergence.

VI. Conclusion & Future Work

This paper introduced a new PSO-based algorithm, called the Guaranteed Convergence Particle Swarm Optimiser (GCPSO). This new algorithm has significantly faster convergence compared to the original PSO, especially when smaller swarm sizes are used. This property is desirable when the swarm is partitioned into smaller sub-swarms, as done by Løbjerg *et al.* The improved performance of the new algorithm is especially noticeable on unimodal functions; its performance on multi-modal functions remains comparable to the original PSO. A further benefit of the new algorithm is that it cannot converge prematurely on unimodal functions, since it has been proven to be a local optimisation algorithm with guaranteed convergence properties.

TABLE III

COMPUTATIONAL COMPLEXITY

		GCPSO		PSO	
Function	s	N_s	F	N_s	F
Ackley	10	12	1586	11	2099
	15	35	2018	32	3019
	20	46	2480	37	2986
Rastrigin	10	36	1636	45	2112
	15	45	1985	43	2525
	20	45	2326	49	3341
Spherical	10	50	4366	48	22851
	15	50	5201	50	11204
	20	50	6564	50	9775
	30	50	9138	50	10927
Quadric	10	50	9284	38	34838
	15	50	9599	50	16735
	20	50	11347	50	14574
	30	50	14317	50	15252

The GCPSO only affects the update equation of the global best particle, thus it can be used in conjunction with a large number of other enhancements made to the basic PSO algorithm.

The GCPSO introduces several new parameters. Future work will focus on designing better strategies to adapt these values dynamically as part of the optimisation process.

References

[1] J. Kennedy and R. C. Eberhart, "Particle Swarm Optimization," in *Proceedings of IEEE International Conference on Neural Networks*, vol. IV, (Perth, Australia), pp. 1942–1948, IEEE Service Center, Piscataway, NJ, 1995.

[2] A. P. Engelbrecht and A. Ismail, "Training product unit neural networks," *Stability and Control: Theory and Applications*, vol. 2, no. 1–2, pp. 59–74, 1999.

[3] F. van den Bergh, "Particle Swarm Weight Initialization in Multi-layer Perceptron Artificial Neural Networks," in *Development and Practice of Artificial Intelligence Techniques*,(Durba n, South Africa), pp. 41–45, Sept. 1999.

[4] F. van den Bergh and A. P. Engelbrecht, "Cooperative Learning in Neural Networks using Particle Swarm Optimizers," *South African Computer Journal*, pp. 84–90, Nov. 2000.

[5] R. C. Eberhart and X. Hu, "Human Tremor Analysis Using Particle Swarm Optimization," in *Proceedings of the Congress on Evolutionary Computation*, (Washington D.C, USA), pp. 1927–1930, IEEE Service Center, Piscataway, NJ, July 1999.

[6] Y. Shi and R. C. Eberhart, "Empirical Study of Particle Swarm Optimization," in *Proceedings of the Congress on Evolutionary Computation*, (Washington D.C, USA), pp. 1945–1949, IEEE Service Center, Piscataway, NJ, July 1999.

[7] Y. Shi and R. C. Eberhart, "A Modified Particle Swarm Optimizer," in *IEEE International Conference of Evolutionary Computation*, (Anchorage, Alaska), May 1998.

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

[9] R. C. Eberhart, P. Simpson, and R. Dobbins, *Computational Intelligence PC Tools*,c h. 6, pp. 212–226. Academic Press Professional, 1996.

[10] M. Clerc and J. Kennedy, "The Particle Swarm: Explosion, Stability and Convergence in a Multi-Dimensional Complex Space," *IEEE Transactions on Evolutionary Computation*, pp. 58–?, 2002.

[11] P. N. Suganthan, "Particle Swarm Optimizer with Neighbourhood Operator," in *Proceedings of the Congress on Evolutionary Computation*, (Washington DC, USA), pp. 1958–1961, IEEE Service Center, Piscataway, NJ, July 1999.

[12] P.J. Angeline, "Using Selection to Improve Particle Swarm Optimization," in *Proceedings of IJCNN'99*, (Washington, USA), pp. 84–89, July 1999.

[13] M. Løvbjerg, T. K. Rasmussen, and T. Krink, "Hybrid Particle Swarm Optimiser with Breeding and Subpopulations," in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*,(Sa n Francisco, USA), July 2001.

[14] M. Clerc, "The Swarm and the Queen: Towards a Deterministic and Adaptive Particle Swarm Optimization," in *Proceedings of the Congress on Evolutionary Computation*, (Washington DC, USA), pp. 1951–1957, IEEE Service Center, Piscataway, NJ, July 1999.

[15] E. Ozcan and C. K. Mohan, "Particle Swarm Optimization : Surfing the Waves," in *Proceedings of the International Congress on Evolutionary Computation*, (Washington, USA), pp. 1939–1944, 1999.

[16] R. C. Eberhart and Y. Shi, "Comparing Inertia Weights and Constriction Factors in Particle Swarm Optimization," in *Proceedings of the Congress on Evolutionary Computing*, (San Diego, USA), pp. 84–89, IEEE Service Center, Piscataway, NJ, 2000.