Particle Swarm Optimization

Christopher Friedt
Email: cf@tf.uni-kiel.de

Kashif Shahab
Email: ks@tf.uni-kiel.de

Roi Rath
Email: ror@tf.uni-kiel.de

Abstract— **The development of Particle swarm optimization (PSO), an optimization algorithm based on a social behavior model is reviewed from its first appearance in the year 1995 until today. This paper will provide the reader with an overview of PSO - from its basic principles and its development challenges to the most recent version of the algorithm. The following versions of the PSO algorithm: the original, the revised version and the latest one - Adaptive PSO, will be explained in more details. In addition, an analysis of the PSO algorithm as a dynamic system is introduced.**

I. Introduction

Particle Swarm Optimization (PSO) is an optimization algorithm that originally evolved from the study of the unpredictable behavior of the bird flock in 1995 by James Kennedy and Russel Eberhart [1]. Their aim was to simulate the social behavior of species in nature, which feature a movement as part of a group (e.g. fish, birds or bees) [1]. Apart from the shape of movement itself, the fact that this group of organisms could move together in order to find an optimized location (e.g. Birds adjust their positions to seek food and mates, optimizing environmental parameters such as temperature) encouraged the authors of [1] to shift their research to another direction. They started to develop an algorithm with a simulation in which they considered the birds as non-colliding *particles*, ending up with a simplified mathematical version for optimization problems.

PSO belongs to a family of *population-based* iterative algorithms, which all work in the same way, i.e. updating a group of particles in a *search space*, in order to locate the optimum solution [2]. Yet, the fact that PSO is very easy to implement and can be easily applied for multi-dimensional problems, with a relatively small number of particles [1], led many researchers to use it for real-world applications, e.g. [3]–[6]. Still, the PSO algorithm has some disadvantages (which will be described later in this paper), and as an outcome there is an ongoing research in order to improve it.

This paper in arranged as follows: in section II the original PSO algorithm is explained along with its pros and cons. The evolution of weighting parameters to the original version is discussed in section III. In section IV an analysis of the PSO algorithm as a dynamic system is introduced. The latest version: Adaptive PSO (APSO) will be explained in details in section V. Finally, some real-world applications will be described shortly in section VI.

II. The Original PSO Algorithm

A. Basic Concepts

PSO applies the concept of social interaction to problem solving by performing a direct search method in order to find an optimal solution. The PSO uses a set of N particles that constitute a swarm moving around in a D-dimensional *search space* looking for the best solution. The term *swarm* is well defined in [1]. Each particle is treated as a point which changes its trajectory according to two concepts: the first is the ability of a particle to remember the coordinates in the search space which are associated with its best solution (fitness) from its entire flying history . This value is called personal best (pbest). The second is the ability of the particles to “communicate” with each other in order to obtain the best value obtained so far by any particle in the swarm - the global best (gbest). A local best version exists in the literature as well. This case however is out of the scope of this paper. Each particle is randomly accelerating toward its pbest and the gbest locations, i.e. its velocity is changing with each time step. The modification of the position of a particle can be mathematically modeled according to these following iterative equations:

$$\begin{aligned} v_i^d(k+1) = v_i^d(k) + c_1 rand_1^d(k)(pBest_i^d(k) - x_i^d(k)) \\ + c_2 rand_2^d(k)(gBest_i^d(k) - x_i^d(k)) \end{aligned} \tag{1}$$

$$x_i^d(k+1) = x_i^d(k) + v_i^d(k+1) \tag{2}$$

where v_i^d is the velocity component of the i-th particle in the d-th axis, x_i^d is the corresponding position of the particle, k is the current iteration step, c_1 and c_2 are acceleration factors and $rand_1$ and $rand_2$ are random weights uniformly distributed in the range $(0,1)$.

B. The Algorithm

For the following subsection the position and velocity of the $i-th$ particle is defined as $\mathbf{x_i} = [x_i^1, x_i^2, \ldots, x_i^D]$ and $\mathbf{v_i} = [v_i^1, v_i^2, \ldots, v_i^D]$, respectively, where the same notations in equations 1 and 2 are used. One can implement the PSO algorithm in the following steps [3]:

1) Define the limits of the search space.
2) Initialize a population of N particles. Randomly select the position and velocity for each particle in all dimensions.
3) Evaluate the fitness function for each particle.
4) Compare fitness value with pbest. If the fitness value is better than pbest, update pbest as the current fitness value.
5) Find the best pbest of the entire population. If it is better than gbest, update gbest as the current best pbest.
6) Update velocities and positions of particles according to equations 1 and 2.
7) Go back to step 3 until the maximum number of iterations or when an optimization criteria are met.

C. Advantages and Disadvangates

As was mentioned before in Sec. I, the PSO is very easy to implement requires a low computational effort due to the usage of simple operators [1]. Another major advantage is the ability to run the PSO for multi-dimensional problems, and yet using a small number of particles (typically 30 [2]). The Algorithm tends to converge in the majority of simulations [2]. Unfortunately, the PSO, like any other population-based iterative algorithms, was most likely to face with two major difficulties: the first is a premature convergence into a suboptimal solution (e.g. a local minima or maxima of a function). The second - a very slow convergence, consists of a large number of iterations which increases the run-time of the algorithm, which can problematic in real-time applications [7].Finally, one can easily understand from Sec. II-B that the original algorithm assumes that the problem to be optimized is *time-invariant*, i.e. the basic algorithm is unadaptive to a time-variant problem.

III. Exploration, Exploitation and Parameter Selection

The first two disadvantages are often explained as an imbalance between the global and the local search [2], or as was stated later on in many PSO papers - exploration and exploitation. Exploration can be defined as the "desire" of the swarm to explore as many regions as possible in the search space, while exploitation is a search being held in a smaller region of the search space, in order to pin-point the optimal solution [8].

New parameters such as the inertia weight (IW) and the constriction factor (CF) were suggested by [3] in order to create a smooth transition between the exploration and exploitation stages. The CF will not be presented in this paper. The IW is calculated as follows:

$$w = w_{max} - (w_{max} - w_{min})\frac{g}{G} \tag{3}$$

where w_{max} and w_{min} are the maximum and minimum values for the IW, respectively, g is the current iteration step and G is the maximum number of iterations. the IW factors the component $v_i^d(k)$ in Eq. 1. typical values for w_{max} and w_{min} are 0.9 and 0.4, respectively. As a linearly decreasing function, the IW gradually decreases the velocities of the particles. This comes from the assumption that in the beginning of the search the swarm is in the *exploration* stage, as there is not enough information upon the location of the optimal solution. After locating the general location of the optima, exploitation should be performed in order to find the exact location [3]. These parameters are however based on empirical assumptions. In the following section an analytic way to choose this and other parameters is explained.

IV. A State-Space Model for Parameter Selection

PSO is at its basis a stochastic algorithm. Its efficiency has been proven only empirically in the first years after its development, as researchers could not explain how the algorithm works [9]. The desire to improve the algorithm's parameter selection in order to control the convergence of the particles (not only the speed of convergence but the path of the particle to its target as well) led the authors of [10] and [9] to conduct a comprehensive theoretical analysis based on the well-known system and control theory [11]. In 2002, the author of [8] suggested a slightly simplified analysis based on the same theory, along with some interesting guidelines for the parameters selection. This section summarizes this paper, and the reader is encouraged to refer to it for further information.

A. A State-Space Model

The iterative equations (for an arbitrary dimension, as these calculations are performed in each dimension independently) of the PSO algorithm can be written as:

$$\begin{aligned} v(k+1) = av(k) + b_1 rand_1(pBest - x(k)) \\ + b_2 rand_2(gBest - x(k)) \end{aligned} \tag{4}$$

$$x(k+1) = cx(k) + dv(k+1) \tag{5}$$

where k is the iteration step, a is a momentum factor, b_1 and b_2 are the acceleration coefficients and c and d are control coefficients. In order to apply the dynamic system model, the random factors were removed. A qualitative discussion between the deterministic version analyzed here and the original stochastic one is given in [8]. After some mathematic manipulations, the equations can be rewritten as:

$$v(k+1) = av(k) + b(p(k) - x(k)) \tag{6}$$

$$x(k+1) = cx(k) + dv(k+1) \tag{7}$$

where b is the arithmetic mean between b_1 and b_2, and p is a weighted mean of $pBest$ and $gBest$. It can be proven that c and d can be fixed (e.g. $c = d = 1$ as in the original algorithm), without any negative impact on the algorithm. Another manipulation on (6) and (7) is needed in order to rewrite them in the following matrix form:

$$\begin{bmatrix} x(k+1) \\ v(k+1) \end{bmatrix} = \begin{bmatrix} 1-b & a \\ -b & a \end{bmatrix} \begin{bmatrix} x(k) \\ v(k) \end{bmatrix} + \begin{bmatrix} b \\ b \end{bmatrix} p(k) \tag{8}$$

or in the shorter matrix form as:

$$\mathbf{x}(k+1) = \mathbf{A}\mathbf{x}(k) + \mathbf{B}p(k) \tag{9}$$

This is the state equation from the well-known *state-space model representation* for linear systems, where $\mathbf{x}$ is the state vector, composed of the particle's current position and velocity, p is the system's external input, $\mathbf{A}$ is the system matrix and $\mathbf{B}$ is the input matrix. It is possible now to analyze the convergence of the algorithm with tools from system and control theory.

B. Convergence Analysis

In linear discrete systems, it is necessary and sufficient to investigate the so-called *Eigen* values of $\mathbf{A}$, in order to determine the system's stability, and form of convergence (direct, oscillating, zigzagging) when stability is indeed attained. The first step is to solve the equation:

$$det(\mathbf{A} - \mathbf{I}\lambda) = 0 \tag{10}$$

which yields the roots of the typical polynomial - the *Eigen* values of $\mathbf{A}$. In order to attain stability for a discrete system, all its *Eigen* values should be inside the unit circle of the complex Z plane. This yields the following conditions on a and b:

$$a < 1, \qquad b > 0, \qquad 2a - b + 2 > 0 \tag{11}$$

which create a convergence triangle in the (a,b) space. In a similar way, the conditions for harmonic oscillations (both *Eigen* values are complex) and zigzagging (one of the *Eigen* values has a real part) yield other conditions on parameters a and b. All conditions are summarized graphically in Fig. 1. Regarding the speed of convergence, the algorithm converges faster when choosing a point (a, b) closer to the middle of the convergence triangle.

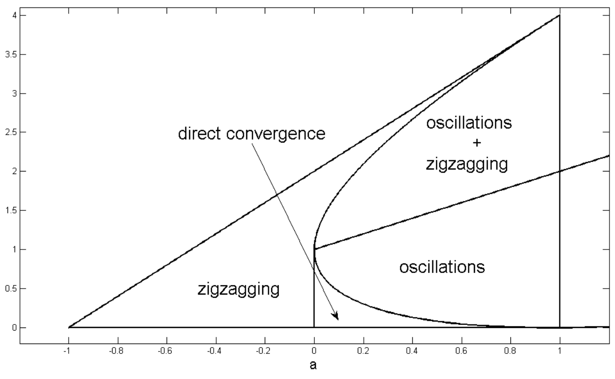

Fig. 1: The different fields of convergence according to the values of parameters a and b

This analysis was conducted under the assumption of a deterministic algorithm. The results for the original version would be similar but not the same, as the randomness supports the *exploration* feature of the algorithm, i.e. the convergence would be slower.

V. The Adaptive PSO Algorithm

A. Motivation

As a consiquence of the first two weaknesses of PSO mentioned in II-C, the most active current areas of PSO research aim to i) increase the rate of convergence, and ii) avoid local optimas [8], [12]. However, making up ground on one goal has historically led to losing ground on the other goal, leading to an *exploration-exploitation trade-off* [8], [12]. The strategy of Chung et al, the adaptive PSO algorithm, tries to empirically achieve a gain toward both goals simultaneously by introducing a new approach for parameter control [12].

B. Formulation

Through prior research findings and good observation, Chung et al took a different approach by basing parameter adaption on evolutionary state [12] rather than time [3], [8]. Using mean inter-particle distances the meaningful evolutionary states of the swarm were characterized and an evolutionary state estimation (ESE) strategy was devised [12].

Along with state-based parameter control, an *auxilliary search operator* was introduced specifically to improve the premature convergence to a local optima problem. The application of this search operator and its effect on the swarm as a whole led to the formulation of the *elitest learning strategy* (ELS) [12] which will be addressed in section V-D.

C. Evolutionary State Estimation (ESE)

In order to classify the swarm with one of several possible states, an *evolutionary factor*, f, and a custom-singleton defuzzification method were employed. The fuzzy states are described in table I and can be visualized in Fig. 2. To calculate the evolutionary factor the following procedure was used:

1) For each i particle, calculate the mean distance, d_i, to each of the other particles according to (12), where x_i^k is the position of the $i-th$ particle in the $k-th$ dimensionality.
2) Identify d_g, the globally best particle (with the fitness function), and determine the maximum and minimum distances, d_{max} and d_{min} from all d_i.
3) Calculate the evolutionary factor, f, according to (13).
4) Classify the state of the swarm.

$$d_i = \frac{1}{N-1} \sum_{j=1, j \neq i}^{N} \sqrt{\sum_{k=1}^{D} (x_i^k - x_j^k)^2} \tag{12}$$

$$f = \frac{d_g - d_{min}}{d_{max} - d_{min}} \tag{13}$$

Now, by inspection, the evolutionary factor is simply the normalized, mean inter-particle distance with respect to the globally best particle. Its mapping to the evolutionary states is well described in Fig. 2.

The custom defuzzification method uses a mixture of state-transition sequence (rule-base) in some cases and membership functions in other cases. The membership functions are defined in [12] and depicted in Fig. 2. The state-transition rules (e.g. $S_1 \rightarrow S_2 \rightarrow S_3 \rightarrow S_4 \rightarrow S_1 \ldots$) take precedence over the membership functions in the overlapping regions. Otherwise the singleton method is used to assign a state. Therefore, if f lies in an overlapping region, the previous state at iteration g will have precedence in determining the state at iteration $g+1$ [12].

In the ESE, the parameters c_1, c_2 and w, previously mentioned in sections II-B and III , are controlled dynamically.

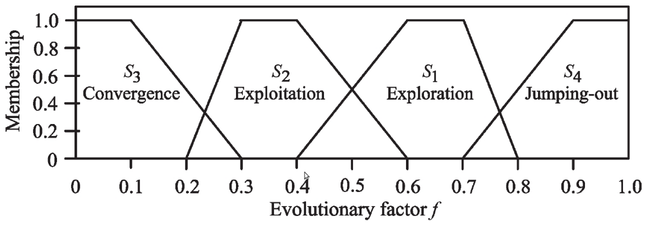

Fig. 2: Fuzzy states: states are mapped proportionally to mean inter-particle distance to g_{best}. Overlapping regions require defuzzification. [12]

State	Description	c_1	c_2
exploration (S_1)	Particles discover new optima and stray from the globally best location	++	--
exploitation (S_2)	Particles exploit their own history and move toward their previous best location	+	-
convergence (S_3)	Particles converge to the globally best location	+	+
jumping-out (S_4)	Globally best position is changed. Globally best particle 'jumps out' of the swarm in random directions, trying to improve global fitness	--	++

TABLE I: APSO state descriptions and associated acceleration parameters c_1 and c_2. The notation ++, --, +, - indicates an increase, decrease, slight increase or slight decrease, respectively.

The IW is controlled according to (14).

$$w(f) = \frac{1}{1 + 1.5e^{-2.6f}} \in [0.4, 0.9], \forall f \in [0, 1] \qquad (14)$$

The acceleration coefficients are increased or decreased based on state according to table I. For more information and for the full formulation, the reader should refer to [12].

D. Elitist Learning Strategy (ELS)

The ELS was designed to operate exclusively on gbest during S_4 in order to overcome the premature convergence to a local optima problem [12]. The ELS intentionally perturbs the globally best particle to jump-out of the swarm and probe toward other, potentially better optima [12]. To reduce the computational complexity of this process in higher dimensionalities, the probe performed in one dimension randomly chosen as [12]:

$$P^d = P^d + (x^d_{max} - x^d_{min})\mathcal{N}(\mu, \sigma^2) \qquad (15)$$

Here, $\mathcal{N}$ is the normalized Gaussian distribution, with mean $\mu = 0$ and a standard-deviation (SD) σ. The SD is termed the *elitist learning rate* [12] and is calculated according to:

$$\sigma = \sigma_{max} - (\sigma_{max} - \sigma_{min})\frac{g}{G} \qquad (16)$$

where g is the current iteration step and G is the maximum number of iterations. The values of σ_{min} and σ_{max} were empirically determined to be 0.1 and 1.0, respectively [12].

E. APSO Performance

The APSO achieves an increase in convergence speed with excellent local-optima avoidance. In nine of 12 test functions, each repeated 30 times, the APSO was shown to converge faster on average than seven contemporary PSO variants [12]. Overall the APSO converged to acceptable solutions 100% of the time (the exception being Griewank's function, 66%), again besting the seven other PSO variants [12].

VI. Applications

PSO can be used in numerous applications. This paper will focus on few examples related to topics from *Digital Communications*. In [4] PSO is used for an adaptive infinite impulse response (IIR) filter structures. A comparison is made between the PSO and the genetic algorithm (GA), both optimizers for a multimodal problem, in order to evaluate their performance in optimizing a non-linear filter. In [6] the use of PSO in virtual MIMO communication protocol is described. Another application is presented in [5], where PSO is used to improve the processing time of the Space Alternating Generalized Expectation (SAGE) maximization algorithm, which is used for channel parameter estimation. Many other applications can be found in [3].

VII. Summary

In this paper several milestones in the evolution of PSO were reviewed, from its basic concepts to the most recent version of the algorithm. The reader is encouraged to proceed to the cited articles and to implement and investigate different versions of the algorithm for deeper understanding of PSO.

References

[1] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Neural Networks, 1995. Proceedings., IEEE International Conference on*, vol. 4, Nov/Dec 1995, pp. 1942–1948 vol.4.

[2] Y. Shi and R. Eberhart, "Empirical study of particle swarm optimization," in *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, vol. 3, 1999, pp. –1950 Vol. 3.

[3] Eberhart and Y. Shi, "Particle swarm optimization: developments, applications and resources," in *Evolutionary Computation, 2001. Proceedings of the 2001 Congress on*, vol. 1, 2001, pp. 81–86 vol. 1.

[4] D. Krusienski and W. Jenkins, "Particle swarm optimization for adaptive iir filter structures," in *Evolutionary Computation, 2004. CEC2004. Congress on*, vol. 1, June 2004, pp. 965–970 Vol.1.

[5] H. Bodur, C. Tunc, D. Aktas, V. Erturk, and A. Altintas, "Particle swarm optimization for sage maximization step in channel parameter estimation," in *Antennas and Propagation, 2007. EuCAP 2007. The Second European Conference on*, Nov. 2007, pp. 1–4.

[6] Y. Yuan, Z. He, and M. Chen, "Virtual mimo-based cross-layer design for wireless sensor networks," *Vehicular Technology, IEEE Transactions on*, vol. 55, no. 3, pp. 856–864, May 2006.

[7] I. Supratid, "A multi-subpopulation particle swarm optimization: A hybrid intelligent computing for function optimization," in *Natural Computation, 2007. ICNC 2007. Third International Conference on*, vol. 5, Aug. 2007, pp. 679–684.

[8] I. C. Trelea, "The particle swarm optimization algorithm: convergence analysis and parameter selection," *Information Processing Letters*, vol. 85, no. 6, pp. 317–325, Mar. 2003.

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

[10] M. Clerc, "The swarm and the queen: towards a deterministic and adaptive particle swarm optimization," in *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, vol. 3, 1999, pp. –1957 Vol. 3.

[11] B. Girod, R. Rabenstein, and A. Stenger, *Signals and Systems*. John Wiley and Sons, LTD, 2001, ch. 2.

[12] Z.-H. Zhan, J. Zhang, Y. Li, and H.-H. Chung, "Adaptive particle swarm optimization," *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, vol. 39, no. 6, pp. 1362–1381, Dec. 2009.