Controlling the Parameters of the Particle Swarm Optimization with a Self-Organized Criticality Model

Carlos M. Fernandes[1,2], Juan J. Merelo[2], and Agostinho C. Rosa[1]

[1] Technical University of Lisbon
c.m.fernandes.photo@gmail.com, acrosa@laseeb.org
[2] University of Granada
jjmerelo@gmail.com

Abstract. This paper investigates a Particle Swarm Optimization (PSO) with a Self-Organized Criticality (SOC) strategy that controls the parameter values and perturbs the position of the particles. The algorithm uses a SOC system known as *Bak-Sneppen* for establishing the inertia weight and acceleration coefficients for each particle in each time-step. Besides adjusting the parameters, the SOC model may be also used to perturb the particles' positions, thus increasing exploration and preventing premature convergence. The implementation of both schemes is straightforward and does not require hand-tuning. An empirical study compares the *Bak-Sneppen PSO* (BS-PSO) with other PSOs, including a state-of-the-art algorithm with dynamic variation of the weight and perturbation of the particles. The results demonstrate the validity of the algorithm.

1 Introduction

Inspired by the swarm and social behavior of bird flocks and fish schools, Kennedy and Eberhart proposed in [6] the Particle Swarm Optimization (PSO) algorithm for binary and real-valued function optimization. Since its inception, PSO has been applied with success to a number of problems and motivated several lines of research that investigate its main working mechanisms. One of these research trends deals with PSO's parameters and aims at devising methods for controlling those parameters and improve the algorithms' performance and robustness. Self-Organized Criticality (SOC), proposed in [2], provides interesting schemes for controlling PSO's working mechanisms. In fact, SOC has been used in the past in population-based metaheuristics, like Evolutionary Algorithms ([5] and [7]) and even PSO [8].

This paper proposes a versatile method, inspired by the SOC theory [2], for controlling the parameters of PSO, and demonstrates that it is a viable and effective method. The algorithm is based on a SOC system known as the *Bak-Sneppen model of co-evolution between interacting species* (or simply *Bak-Sneppen*), proposed by Bak and Sneppen in [3]. The *Bak-Sneppen PSO* (BS-PSO) uses the fitness values of the population of co-evolving species for regulating the parameters of the algorithm. Furthermore, the exact same values are used for perturbing the particle's position, thus introducing a kind of mutation in the PSO equations. The potentiality of the

C.A. Coello Coello et al. (Eds.): PPSN 2012, Part II, LNCS 7492, pp. 153–163, 2012.

proposed method as a stochastic (although with predictable global behavior) seed for varying the parameters is investigated here, postponing a study of a stronger hybridization of the SOC model and the PSO for a future work.

A simple experimental setup was designed as a proof-of-concept. BS-PSO is compared with methods for controlling the inertia weight, as well as with a state-of-the-art PSO that also combines dynamic control of the parameters with perturbations of the particles' positions. The tests are conducted in a way such that each new component of BS-PSO is examined separately in order to investigate its effects on the performance of the algorithm. The results demonstrate the validity of the approach and show that BS-PSO, without requiring the hand-tuning of the traditional parameters or any additional one, is competitive with other PSOs. Furthermore, the base-model is simple and well-studied by the SOC theory, and may be treated as a black-box system that outputs batches of values for the parameters.

2 Particle Swarm Optimization

PSO is a population-based algorithm in which a group of solutions travels through the search space according to a set of rules that favor their movement towards optimal regions of the space. PSO is described by a simple set of equations that define the velocity and position of each particle. The position vector of the *i-th* particle is given by $\vec{X}_i = (x_{i,1}, x_{i,2}, \dots x_{i,D})$, where D is the dimension of the search space. The velocity is given by $\vec{V}_i = (v_{i,1}, v_{i,2}, \dots v_{i,D})$. The particles are evaluated with a fitness function $f(\vec{X}_i)$ in each time step and then their positions and velocities are updated by:

$$v_{i,d}(t) = v_{i,d}(t-1) + c_1 r_1 \big(p_{i,d} - x_{i,d}(t-1)\big) + c_2 r_2 \big(p_{g,d} - x_{i,d}(t-1)\big) \quad (1)$$

$$x_{i,d}(t) = x_{i,d}(t-1) + v_{i,d}(t) \quad (2)$$

were p_i is the best solution found so far by particle i and p_g is the best solution found so far by the neighborhood. Parameters $r_1$and r_2 are random numbers uniformly distributed in the range [0,1.0] and $c_1$and c_2 are acceleration coefficients that tune the relative influence of each term of the formula (usually set within the range [1.0,2.0]). The first, influenced by the particles best solution, is known as the *cognitive part*, since it relies on the particle's own experience. The last term is the *social part*, since it describes the influence of the community in the velocity of the particle.

The neighborhood of the particle may be defined by a great number of schemes but most PSOs use one of two simple sociometric principles. The first connects all the members of the swarm to one another, and it is called $gbest$, were g stands for *global*. The second, called $lbest$ (l stands for local), creates a neighborhood that comprises the particle itself and its k nearest neighbors. In order to prevent particles from stepping out of the limits of the search space, the positions $x_{i,d}(t)$ of the particles are limited by constants that, in general, correspond to the domain of the problem: $x_{i,d}(t) \in [-Xmax, Xmax]$. Velocity may also be limited within a range in order to prevent the *explosion* of the velocity vector: $v_{i,d}(t) \in [-Vmax, Vmax]$.

Although the basic PSO may be very efficient on numerical optimization, it requires a proper balance between local and global search. If we look at equation 1, we see that the first term on the right-hand side of the formula provides the particle with global search abilities. On the other hand, the second and third terms act as a local search mechanism and it is trivial to demonstrate that without the first term the swarm shrinks around the best position found so far. Therefore, by weighting these two parts of the formula it is possible to balance local and global search. In order to achieve a balancing mechanism, Shi an Eberhart [10] introduced the inertia weight ω, which is adjusted — usually within the range [0, 1.0] — together with the constants c_1 and c_2, in order to achieve the desired balance. The modified velocity equation is:

$$v_{i,d}(t) = \omega . v_{i,d}(t-1) + c_1 r_1 \left(p_{i,d} - x_{i,d}(t-1)\right) + c_2 r_2 \left(p_{g,d} - x_{i,d}(t-1)\right) \quad (3)$$

The parameter may be used as a constant that is defined after an empirical investigation of the algorithm's behavior. Another possible strategy is to use *time-varying inertia weights* (TVIW-PSO) [11]: starting with an initial and pre-defined value, the parameter value decreases linearly with time, until it reaches the minimum value. Later, Eberhart and Shi [4] found that the TVIW-PSO is not very effective on dynamic environments and proposed a random inertia weight for tracking dynamic systems. In the remainder of this paper, this method is referred to as RANDIW-PSO.

An adaptive approach is proposed in [1]. The authors describe the *global local best inertia weight PSO* (GLbestIW-PSO), an on-line variation strategy that depends on the $p_i$and p_g values. The strategy is defined in a way that better solutions use lower inertia weight values, thus increasing their local search abilities. The worst particles are modified with higher ω values and therefore they tend to explore the search space.

In [9], Ratnaweera *et al.* describe new parameter automation strategies that act upon several working mechanisms of the algorithm. The authors propose the concept of time-varying acceleration coefficients. They also introduce the concept of mutation, by adding perturbations to randomly selected modulus of the velocity vector. Finally, the authors describe a *self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients* (HPSO-TVAC), which restricts the velocity update policy to the influence of the cognitive and social part, reinitializing the particles whenever they are stagnated in the search space.

Another method for controlling ω is given by Suresh *et al.* in [12]. The authors use the Euclidean distance between the particle and $gbest$ for computing ω in each time-step for each particle. Particles closer to the best global solution tend to have higher ω values, while particles far from $gbest$ are modified with lower inertia. The algorithm introduces a parameter ω_0 that restricts the inertia weight to working values. In addition, Suresh *et al.* also uses a perturbation mechanism of the particles' positions that introduces a random value in the range $[1, \rho]$, where ρ is a new parameter for the algorithm (see equation 4, which replaces equation 2). The authors report that the *Inertia-Adaptive PSO* (IA-PSO) outperforms several other methods in a 12-function benchmark, including the abovereferred state-of-the-art HPSO-TVAC.

$$x_{i,d}(t) = (1+\rho) . x_{i,d}(t-1) + v_{i,d}(t) \quad (4)$$

Like HPSO-.TVAC and IA-PSO, the method proposed in this paper also aims at controlling the balance between local and global search by dynamically varying the inertia weight and/or the acceleration coefficients, while introducing perturbations in the particles' positions (like IA-PSO, but with ρ controlled by the SOC model). The main objective is to construct a simple scheme that does not require complex parameter tuning or pre-established strategies.

3 SOC and the Bak-Sneppen Particle Swarm

In complex adaptive systems, complexity and self-organization usually arise in the transition region between order and chaos. SOC systems are dynamical system with a critical point in that region as an attractor. However, and unlike many physical systems which have a parameter that needs to be tuned for criticality, a SOC model is able to self-tune to that critical state.

One of the properties of SOC systems is that small disturbances can lead to *avalanches*, i.e., events that are spatially or temporally spread through the system. Moreover, the same perturbation may lead to small or large avalanches, which in the end show a power-law proportion between the size of the events and its abundance.

The Bak-Sneppen is a model of co-evolution that displays SOC properties. Different species in the same ecosystem are related trough several features; they co-evolve, and the extinction of one species affects the species that are related to it, in a chain reaction that can affect large segments of the population. In the model, each species has a fitness value assigned to it and it is connected to other species in a ring topology (i.e., each one has two neighbors). Every time step, the species with the worst fitness and its neighbors are replaced by individuals with random fitness. When plotting the size of extinctions over their frequency in a local segment of the population and below a certain threshold close to a critical value, a power-law relationship is observed.

This description may be translated to a mathematical model. The system is defined by n^d fitness values f_i arranged on a d-dimensional lattice with n cells. At each time step, the smallest f value and its neighbors are replaced by uncorrelated random values drawn from a uniform distribution. The system is thus driven to a critical state where most species reach a fitness value above a certain threshold, with avalanches producing non-equilibrium fluctuations in the configuration of the fitness values.

The behavior of the numerical values of the Bak-Sneppen model — power-law relationships, increasing average fitness of the population, periods of stasis in segments of the population punctuated by intense activity — are the motivation behind this study. By linking a Bak-Sneppen model to the particles and then using the species' fitness values as input for adjusting the algorithm's parameters, it is expected that the resulting strategy is able to control PSO. To the extent of our knowledge, this is the first proposal of a scheme linking the Bak-Sneppen model and PSO in such a way. However, SOC has been applied to this field of research in the past.

In [7], Krink *et al.* proposed SOC-based mass extinction and mutation operator schemes for Evolutionary Algorithms. A *sandpile model* [2] is used here and its

equations are previously computed in order to obtain a record of values with a power-law relationship. Those values are then used during the run to control the number of individuals that will be replaced by randomly generated solutions or the mutation probability of the Evolutionary Algorithm. Tinós and Yang [13] were also inspired by the Bak-Sneppen model to create the *Self-Organized Random Immigrants Genetic Algorithm* (SORIGA). The authors apply the algorithm to time-varying fitness landscapes and claim that SORIGA is able to outperform other algorithms in the proposed test set. In [5], Fernandes *et al.* describe an Evolutionary Algorithm attached to a sandpile model. The avalanches dynamically control the algorithm's mutation operator. The authors use the proposed scheme in time-varying fitness functions and claim that the algorithm is able to outperform other state-of-the-art methods in a wide range of dynamic problems. Finally, Løvbjerg and Krink [8] apply SOC to PSO in order to control the diversity to the population. The authors introduce a *critical value* associated to each particle and define a rule that increments that value when two particles are closer than a *threshold distance*. When the critical value of a particle exceeds a globally set *criticality limit*, the algorithm disperses the criticality of the particle within a certain neighborhood and mutates it. The algorithm also uses the particle's critical value to control the inertia. The authors claim that the method attains better solutions than the basic PSO. However, it introduces working mechanisms that can complicate its design. Overall, there are five parameters that must be tuned or set to *ad hoc* values.

The proposed BS-PSO uses the Bak-Sneppen model without introducing complicated control mechanisms. The only exception is an upper limit for the number of mutations in each time-step, a practical limitation due to the nature of the model and the requirements of numerical optimization. Besides that, the model is run in its original form, feeding the PSO with values in the range [0,1.0] (species' fitness values) that are used by the algorithm to control the parameters. Please note that if the PSO does not interact directly with the model (which is the case in this paper), the model can be executed prior to the optimization process and its fitness values stored in order to be used later in any kind of problem (meaning also that the running times are exactly the same of a basic PSO). However, in order to generalize the system and describe a framework that can easily be adapted to another level of hybridization of the SOC model and PSO, it is assumed that the model evolves on-line with the swarm.

In the Bak-Sneppen model, a population of species is placed in a ring topology and a random value between 0 and 1.0 is assigned to each individual. In BS-PSO, the number of species is equal to the size of the swarm. Therefore, the algorithm may be implemented just by assigning a secondary fitness, called *bak-sneppen fitness* (*bs_fitness*), to each individual in the swarm. This way, each individual is both the particle of the PSO and the species of the co-evolutionary model, with two independent fitness values: a quality measure fitness $f(\vec{X})$, computed by the objective function, and the *bs_fitness* $f_{bs}(\vec{X})$, modified according to Algorithm 1.

The main body of the BS-PSO is very similar to the basic PSO. The differences are: Algorithm 1 is called in each time-step, and modifies three or more *bs_fitness* values; the inertia weigh of each particle is defined in each time-step and for each

particle i with equation 5, where $\overrightarrow{X_i}$ is the position of particle i; the acceleration coefficients c_1 and c_2 are defined in each time-step by equation 6; the position's update is done using equation 7, where $\rho_i(t) = random[0,1 - bs\_fitness(\overrightarrow{X_i})]$.

$$\omega_i(t) = 1 - bs\_fitness(\overrightarrow{X_i}) \tag{5}$$

$$c_1(t) = c_2(t) = 1 + bs\_fitness(\overrightarrow{X_i}) \tag{6}$$

$$x_{i,d}(t) = (1 + \rho_i(t)).x_{i,d}(t-1) + v_{i,d}(t) \tag{7}$$

Algorithm 1 is executed in each time-step. At $t = 0$, the *bs_fitness* values are randomly drawn from a uniform distribution in the range $[0, 1.0]$. Then, the algorithm searches for the worst species in the population (lowest *bs_fitness*), stores its fitness value (*minFit*) and mutates it by replacing the fitness with a random value in $[0, 1.0]$. In addition, the neighbors of the worst are also mutated (remember that a ring topology connects each species with index *j* to its two neighbors with indexes $j + 1$ and $j - 1$). Then, the algorithm searches again for the worst. If the fitness of that species is lower than *minFit*, the process repeats: species and its neighbors are mutated. This cycle proceeds while the worst fitness in the population is bellow *minFit* (and the number of mutations is below the limit). When the worst is found to be above *minFit*, the algorithm proceeds to PSO's standard procedures (see Algorithms 1 and 2).

Algorithm 1 .(Bak-Sneppen Model)

1. Set $mutations = 0$; set $max\_mutation = 2 \times swarm\_size$
2. Find the index *j* of the species with lowest bak-sneppen *fitness*
3. Set $minFit = bs\_fitness(\overrightarrow{X_j})$
4. Replace the fitness of individuals with indices j, $j - 1$, and $j + 1$ by random values
5. Increment mutations: $++mutations$
6. Find the index *j* of the species with lowest fitness
7. If $bs\_fitness(\overrightarrow{X_j}) < minFit$ or $mutations = max\_mutation$, return to 4; else, end

Algorithm 2. (BS-PSO)

1. Initialize velocity and position of each particle.
2. Evaluate each particle i: $fitness\ (\overrightarrow{X_i}) = f(\overrightarrow{X_i})$
3. Initialize bak-sneppen fitness values: $bs\_fitness(\overrightarrow{X_i}) = random[0, 1.0]$
4. Update Bak-Sneppen Model (Algorithm 1).
5. For each particle i:
    6. Set $\omega = \rho = 1 - bs\_fitness(\overrightarrow{X_i})$
    7. Set $c_1 = c_2 = 1 + bs\_fitness(\overrightarrow{X_i})$
    8. Update velocity (equation 3) and position (equation 7); evaluate $fitness\ (\overrightarrow{X_i}) = f(\overrightarrow{X_i})$
9. If (stop criteria not met) return to 4; else, end.

As stated above, a stop criterion is introduced in Algorithm 1 in order to avoid long mutation cycles that would slow down BS-PSO after a certain number of iterations. If the number of mutations reaches a maximum pre-defined value, Algorithm 1 ends. In

this paper, the critical value is set to twice the swarm's size. This value was intuitively fixed, not tuned for optimization of the performance. It is treated as a constant and its effects on the algorithm are beyond the scope of this paper. It is even possible that other strategies for avoiding long intra-time-steps mutation cycles that do not require a constant can be devised. However, such a study is left for future work. The main objective here is to demonstrate that controlling the inertia, acceleration coefficients and particles' positions with values given by a SOC model is viable and effective.

4 Testbed Set and Results

The experiments were designed with five benchmark functions (see Table 1). The minimum of all functions is in the origin with fitness 0. The dimension of the search space is set to $D = 30$ (except f_6). TVIW-, RANDIW-, GLbestIW- and IA-PSO were included in the tests in order to evaluate the efficiency of the method. (It is not our intention to prove that BS-PSO is better than the best PSOs in a wide range of functions. This simple experiment is mainly a proof-of-concept, and the peer-algorithms were chosen so that the different mechanism of BS-PSO can be properly evaluated.)

The population size n is set to 20 for all algorithms; $lbest$ topology is used. The acceleration coefficients were set to 1.494, a value suggested in [4] for RANDIW-PSO. However, and since we are using algorithms with varying parameters, it is expected that other PSOs require different c values. Therefore, the coefficients c were also set to 1.2 and 2.0 (as in the studies that introduce GLbestIW-PSO and IA-PSO). $Xmax$ is defined as usual by the domain's upper limit and $Vmax = Xmax$. TVIW-PSO uses linearly decreasing inertia weight, from 0.9 to 0.4. The maximum number of generations is 3000 (except f_6, for which the limit is 1000); 50 runs for each experiment are conducted. Since PSO takes advantage of the fact that the optima are located in the centre of search space, asymmetrical initialization is often used for testing PSO. The initialization range for each function is given in Table 1.

Table 1. Benchmarks for the experiments. Dynamic and initialization range.

function	*mathematical representation*	*Range of search*	*Range of initialization*
Sphere f_1	$f_1(\vec{X}) = \sum_{i=}^{D} {x_i}^2$	$(-100, 100)^D$	$(50, 100)^D$
Rosenbrock f_2	$f_2(\vec{x}) = \sum_{i=1}^{D-1} (100(x_{i+1} - {x_i}^2)^2 + (x_i - 1)^2$	$(-100, 100)^D$	$(15, 30)^D$
Rastrigin f_3	$f_3(\vec{x}) = \sum_{i=1}^{D} ({x_i}^2 - 10\cos(2\pi x_i) + 10)$	$(-10, 10)^D$	$(2.56, 5.12)^D$
Griewank f_4	$f_4(\vec{x}) = 1 + \frac{1}{4000}\sum_{i=1}^{D} {x_i}^2 - \prod_{i=1}^{D} \cos\left(\frac{x_i}{\sqrt{i}}\right)$	$(-600, 600)^D$	$(300, 600)^D$
Schaffer f_6	$f_6(\vec{x}) = 0.5 + \frac{\left(\sin\sqrt{x^2 + y^2}\right)^2 - 0.5}{\left(1.0 + 0.001(x^2 + y^2)\right)^2}$	$(-100, 100)^2$	$(15, 30)^2$

The first test compares BS-PSO with different degrees of parameter control (i.e., the control of the acceleration coefficients and the perturbation of position were disabled in order to evaluate the effects of introducing the schemes). Table 2 summarizes the results. In the table's header, bs means that ω, c or ρ are controlled by *bs_fitness* values; otherwise, the control is disabled and the parameter is set to the corresponding value. As seen in Table 2, BS-PSO was tested with inertia control enabled and different c values (with $\rho = 0$); in general, higher c lead to a better performance. When the dynamic control of c is enabled (i.e, $(\omega, c, \rho) = (bs, bs, 0)$) the performance on f_1 and f_2 is improved, while for the other functions the fitness value decreases when compared to the best configuration with fixed c. However, the results are better than those attained by suboptimal configurations, which means that it may be an alternative to fine-tuning the parameter. Introducing a perturbation of the positions with the ρ parameter clearly improves the results, especially when ρ is controlled by the model. (Please note that ρ is set to 0.25, as in [12], in order to compare not only fixed and SOC-based perturbation, but also BS-PSO and IA-PSO later in in this section).

Table 2. BS-PSO: average and standard deviation of the optimal value for 50 trials

$(\omega, c, \rho)\rightarrow$	$(bs, 1.2, 0)$	$(bs, 1.49, 0)$	$(bs, 2.0, 0)$	$(bs, bs, 0)$	$(bs, bs, 0.25)$	(bs, bs, bs)
f_1	2.21e+04 (7.72e+03)	3.35e+01 (1.90e+02)	1.38e-15 (3.21e-15)	8.30e-32 (3.47e-31)	**0.00e+00 (0.00e+00)**	**0.00e+00 (0.00e+00)**
f_2	9.76e+07 (2.83e+07)	1.67e+05 (1.17e+06)	1.88e+02 (2.53e+02)	8.56e+01 (7.98e+01)	2.61e+01 (2.66e-01)	**2.60e+01 (1.58e-01)**
f_3	3.57e+02 (4.91e+01)	2.82e+02 (4.44e+01)	1.11e+02 (2.75e+01)	2.02e+02 (4.16e+01)	4.88e+00 (7.73e+00)	**3.32e+00 (7.09e+00)**
f_4	1.53e+02 (7.30e+01)	1.63e+00 (5.93e+00)	1.25e-02 (1.26e-02)	1.65e-02 (2.24e-02)	**3.79e-03 (2.29e-03)**	4.51e-03 (4.00e-03)
f_6	9.63e-02 (9.43e-02)	3.31e-02 (4.08e-02)	4.05e-03 (4.72 e-03)	5.55e-03 (4.80e-03)	1.55e-03 (3.60e-03)	**3.89e-04 (1.92e-04)**

Table 3. TVIW-PSO, RANDIW-PSO and GLbestIW-PSO

	TVIW $c = 1.2$	TVIW $c = 1.49$	TVIW $c = 2.0$	RANDIW $c = 1.2$	RANDIW $c = 1.49$	RANDIW $c = 2.0$	GLbestIW $c = 1.2$	GLbestIW $c = 1.49$	GLbestIW $c = 2.0$
f_1	1.22e-23 (5.81e-23)	**8.64e-29 (1.75e-28)**	2.81e-06 (2.77e-06)	**1.12e-33 (1.90e-33)**	1.22e-18 (1.26E-18)	6.68e+02 (2.60e+02)	1.14e+05 (6.47e+03)	3.67e+04 (8.25e+03)	**2.83e+03 (1.92e+03)**
f_2	1.24e+02 (1.66e+02)	**1.03e+02 (9.31e+01)**	5.96e+02 (1.72e+03)	7.53e+01 (7.24e+01)	**7.28e+01 (6.69e+01)**	2.07e+07 (1.26e+07)	2.95e+08 (4.80e+07)	**9.10e+07 (3.41e+07)**	3.46e+08 (9.03e+07)
f_3	9.82e+01 (2.44e+01)	7.85e+01 (2.01e+01)	**5.84e+01 (1.39e+01)**	1.80e+02 (3.01e+01)	**1.11e+02 (2.51e+01)**	1.94e+02 (2.77e+01)	4.37e+02 (3.24e+01)	3.56e+02 (3.56e+01)	**1.68e+02 (2.79e+01)**
f_4	8.71e-03 (1.06Ee-02)	**8.66e-03 (1.14e-02)**	1.22e-02 (1.26e-02)	1.25e-02 (1.64e-02)	**1.04e-02 (1.50e-02)**	5.96e+00 (1.62e+00)	9.77e+02 (5.30e+01)	3.08e+02 (6.63e+01)	**2.34e+01 (1.53e+01)**
f_6	2.34e-03 (4.19e-03)	**2.18e-03 (3.94e-03)**	2.34e-03 (4.07e-03)	**4.14e-03 (4.80e-03)**	4.48e-03 (4.88e-03)	2.60e-03 (4.18e-03)	6.99e-02 (1.10e-01)	8.12e-03 (4.05e-02)	**0.00e+00 (0.00e+00)**

Table 4. Kolmogorov-Smirnov statistical tests comparing the best configurations of each algorithm. '+' sign means that PSO 1 is statistically better than PSO 2, '~' means that the PSOs are equivalent, and '–' means that PSO 1 is worse.

PSO 1 *vs.* PSO 2	f_1	f_2	f_3	f_4	f_6
BS-PSO (bs, bs, bs) *vs* TVIW-PSO	+	+	+	+	+
BS-PSO (bs, bs, bs) *vs* RANDIW-PSO	+	+	+	+	+
BS-PSO (bs, bs, bs)*vs* GLbestIW-PSO	+	+	+	+	–
BS-PSO (bs, bs, 0) *vs* TVIW-PSO	+	+	–	–	–
BS-PSO (bs, bs, 0) *vs* RANDIW-PSO	–	~	–	~	~
BS-PSO (bs, bs, 0) *vs* GLbestIW-PSO	+	+	~	+	–

Table 5. IA-PSO: average and standard deviation of the optimal value for 50 trials

	$c = 1.2$ $\rho = 0$	$c = 1.2$ $\rho = 0.25$	$c = 1.2$ $\rho = bs$	$c = 1.49$ $\rho = 0$	$c = 1.49$ $\rho = 0.25$	$c = 1.49$ $\rho = bs$	$c = 2.0$ $\rho = 0$	$c = 2.0$ $\rho = 0.25$	$c = 2.0$ $\rho = bs$
f_1	1.16e+04	**0.00e+00**	**0.00e+00**	2.42e+02	**0.00e+00**	**0.00e+00**	5.19e-02	6.56e-03	2.60e-02
	(8.32e+03)	**(0.00e+00)**	**(0.00e+00)**	(1.43e+03)	**(0.00e+00)**	**(0.00e+00)**	(2.61e-02)	(5.34e-03)	(1.70e-02)
f_2	2.05e+07	2.71e+01	2.63e+01	7.45e+04	2.62e+01	**2.60e+01**	4.26e+02	3.97e+01	7.21e+01
	(1.43e+07)	(4.44e+00)	(1.29e+00)	(5.26e+05)	(3.71e-01)	**(1.84e-01)**	(8.30e+02)	(2.14e+01)	(8.25e+01)
f_3	3.74e+02	1.16e+02	7,79e+01	2.82e+02	5.26e+01	3.96e+01	8.87e+01	1.81e+00	**1.12e+01**
	(3.27e+01)	(2.10e+01)	(2.17e+01)	(3.47e+01)	(3.08e+01)	(2.02e+01)	(2.66e+01)	(3.12e+00)	**(1.42e+01)**
f_4	1.21e+02	4.02e-03	3,95e-03	2,62e+00	**3.72e-03**	4.71e-03	1.84e+00	1.11e-02	1.30e-02
	(7.57e+01)	(2.80e-03)	(2.22e-03)	(1.30e+01)	**(2.23e-03)**	(3.12e-03)	(1.27e+01)	(7.74e-03)	(7.08e-03)
f_6	2.45e-01	9.52e-03	5.26e-03	8.90e-02	5.44e-03	7.77e-04	3.75e-03	**3.89e-04**	4.89e-04
	(7.60e-02)	(1.37e-03)	(4.87e-03)	(8.39e-02)	(4.87e-03)	(2.66e-03)	(4.73e-03)	**(1.92e-03)**	(2.03e-03)

In order to assure fair comparisons, Table 3 shows the complete set of results attained by TVIW-, RANDIW- and GLbestIW-PSO. Apparently, BS-PSO outperforms the other algorithms in most of the scenarios. However, PSOs in Table 3 do not include perturbation of the particle's position and therefore they should be also compared to a BS-PSO with that scheme disabled ($(bs, bs, 0)$ in Table 2): Table 4 compares BS-PSO (with and without perturbation of the particles) to the other PSOs using statistical non-parametric tests (best configurations in Table 3 were chosen). It is confirmed that the fully enabled BS-PSO outperforms the other algorithms. As for the version restricted to parameter control, it is in general better than GLbestIW, while being competitive with the other methods. These are interesting results, since the performance of BS-PSO is attained without fine-tuning the parameters.

Table 6. Kolmogorov-Smirnov statistical tests comparing IA-PSO and BS-PSO

PSO 1 *vs.* PSO 2	f_1	f_2	f_3	f_4	f_6
BS-PSO (bs, 2.0, 0) *vs* IA-PSO ($\rho = 0$)	+	+	~	+	~
BS-PSO (bs, bs, bs)*vs* IA-PSO (*bs* controlled ρ)	~	~	+	~	+

A final test compares BS-PSO with IA-PSO. The later was tested with different acceleration coefficients and three perturbation strategies: disabled ($\rho = 0$), set to 0.25 (as in [12]) and controlled by the Bak-Sneppen model (using a Bak-Sneppen controlled IA-PSO permits to compare only the parameter control scheme of both algorithms). The results are in Table 5 and the statistical tests in Table 6. BS-PSO is, in general, more efficient than IA-PSO, whether the schemes are fully enabled or not. At this point, a question arises: what are the mechanisms behind the control scheme that make BS-PSO efficient in adjusting the parameters? Figure 1 gives some hints. The plot in the figure represents the distribution of ω_i during a typical run of BS-PSO, and, although it is not the definitive answer, helps to clarify this issue. The values seem to keep within a range that is not only suited for ω but also appropriate to model a mutation scheme. If the system had higher values with more frequency, the effect would be destructive, since it would increase exploration beyond a reasonable point.

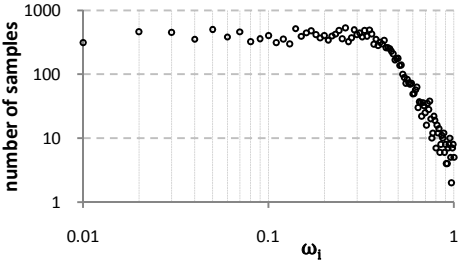

Fig. 1. Distribution of the ω_i values of all particles in a typical run

5 Conclusions and Future Work

The Bak-Sneppen Particle Swarm Optimization (BS-PSO) is a variation of the basic PSO that uses a Self-Organized Criticality (SOC) model to control the inertia weight and the acceleration coefficients, as well as the perturbation factor of the particles' positions. A single scheme controls three parameters making hand-tuning of the basis PSO unnecessary. An experimental setup demonstrates the validity of the algorithm and shows that the incorporation of each control mechanism may improve the performance or at least reduce the tuning effort. The BS-PSO is compared with other methods. In particular, the algorithm is able to attain better results than a recently proposed inertia weight PSO (IA-PSO) in most of the experimental scenarios. In a future work, a scalability analysis will be conducted, as well as study on the effects of the limit imposed to mutation events by the current algorithm, and possible alternatives to this *ad hoc* solution. The test set will also be extended and BS-PSO compared with the algorithms proposed in [8] and [9]. Finally, different levels of hybridization between the Bak-Sneppen model and PSO will be tested, in order to introduce information from the search into the variation scheme of the parameter values.

Acknowledgement. The first author wishes to thank FCT, *Ministério da Ciência e Tecnologia*, his Research Fellowship SFRH/BPD/66876/2009. This work is supported by project TIN2011-28627-C04-02 awarded by the Spanish Ministry of Science and Innovation and P08-TIC-03903 awarded by the Andalusian Regional Government.

References

1. Arumugam, M.S., Rao, M.V.C.: On the Performance of the Particle Swarm Optimization Algorithm with Various Inertia Weight Variants for Computing Optimal Control of a Class of Hybrid Systems. Discrete Dynamics in Nature and Society (2006), Article ID 79295, 17 pages (2006)
2. Bak, P., Tang, C., Wiesenfeld, K.: Self-organized Criticality: an Explanation of 1/f Noise. Physical Review Letters 59(4), 381–384 (1987)
3. Bak, P., Sneppen, K.: Punctuated Equilibrium and Criticality in a Simple Model of Evolution. Physical Review Letters 71(24), 4083–4086 (1993)
4. Eberhart, R.C., Shi, Y.: Tracking and optimizing dynamic systems with particle swarms. In: Proc. IEEE of the Congress on Evolutionary Computation 2001, pp. 94–97. IEEE Press (2001)
5. Fernandes, C.M., Merelo, J.J., Ramos, V., Rosa, A.C.: A Self-Organized Criticality Mutation Operator for Dynamic Optimization Problems. In: Proc. of the 2008 Genetic and Evolutionary Computation Conference, pp. 937–944. ACM (2008)
6. Kennedy, J., Eberhart, R.: Particle Swarm Optimization. In: Proceedings of IEEE International Conference on Neural Networks, vol. 4, pp. 1942–1948 (1995)
7. Krink, T., Rickers, P., René, T.: Applying Self-organized Criticality to Evolutionary Algorithms. In: Deb, K., Rudolph, G., Lutton, E., Merelo, J.J., Schoenauer, M., Schwefel, H.-P., Yao, X. (eds.) PPSN VI. LNCS, vol. 1917, pp. 375–384. Springer, Heidelberg (2000)
8. Løvbjerg, M., Krink, T.: Extending particle swarm optimizers with self-organized criticality. In: Proceedings of the 2002 IEEE Congress on Evolutionary Computation, vol. 2, pp. 1588–1593. IEEE Computer Society (2002)
9. Ratnaweera, A., Halgamuge, K.S., Watson, H.C.: Self-organizing Hierarchical Particle Swarm Optimizer with Time-varying Acceleration Coefficients. IEEE Transactions on Evolutionary Computation 8(3), 240–254 (2004)
10. Shi, Y., Eberhart, R.C.: A Modified Particle Swarm Optimizer. In: Proc. of IEEE 1998 Congress on Evolutionary Computation, pp. 69–73. IEEE Press (1998)
11. Shi, Y., Eberhart, R.C.: Empirical Study of Particle Swarm Optimization. In: Proc. of the IEEE. Congress on Evolutionary Computation, pp. 101–106 (1999)
12. Suresh, K., Ghosh, S., Kundu, D., Sen, A., Das, S., Abraham, A.: Inertia-Adaptive Particle Swarm Optimizer for Improved Global Search. In: Proceedings of the 8th International Conference on Intelligent Systems Design and Applications, vol. 2, pp. 253–258. IEEE, Washington, DC (2008)
13. Tinós, R., Yang, S.: A self-organizing Random Immigrants Genetic Algorithm for Dynamic Optimization Problems. In: Genetic Programming and Evolvable Machines, vol. 8(3), pp. 255–286 (2007)