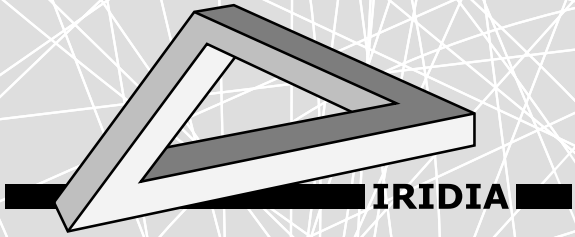


Université Libre de Bruxelles

Institut de Recherches Interdisciplinaires
et de Développements en Intelligence Artificielle

Frankenstein's PSO: An *Engineered* Composite Particle Swarm Optimization Algorithm

Marco A. Montes de Oca, Thomas Stützle,
Mauro Birattari and Marco Dorigo

IRIDIA – Technical Report Series

Technical Report No.
TR/IRIDIA/2007-006

March 2007

IRIDIA – Technical Report Series
ISSN 1781-3794

Published by:
IRIDIA, *Institut de Recherches Interdisciplinaires et de Développements en Intelligence Artificielle*
Université Libre de Bruxelles
Av F. D. Roosevelt 50, CP 194/6
1050 Bruxelles, Belgium

Technical report number TR/IRIDIA/2007-006

Revision history:
TR/IRIDIA/2007-006.001 March 2007

Frankenstein's PSO: An *Engineered* Composite Particle Swarm Optimization Algorithm

Marco A. Montes de Oca mmontes@ulb.ac.be
Thomas Stützle stuetzle@ulb.ac.be
Mauro Birattari mbiro@ulb.ac.be
Marco Dorigo mdorigo@ulb.ac.be
IRIDIA, Université Libre de Bruxelles, Brussels, Belgium

March 2007

Abstract

We introduce a high-performing composite particle swarm optimization (PSO) algorithm. In an analogy to the popular character of Mary Shelley's famous novel, we call our algorithm *Frankenstein's PSO*, as it consists of different algorithmic components drawn from other PSO variants. Frankenstein's PSO constituents were selected after careful evaluation of their impact on speed and reliability.

We present the process that guided us in selecting and adapting the algorithmic components included in the final version of the algorithm. The algorithm is validated through a comparison with other PSO variants on a number of well-known benchmark problems.

Frankenstein's PSO typically reaches high quality solutions faster and more frequently than the most commonly used PSO algorithms. We provide parameter selection guidelines for properly configuring Frankenstein's PSO taking into account the requirements of the optimization task at hand.

1 Introduction

In particle swarm optimization (PSO) algorithms [1, 2, 3], entities (called particles) move in a d-dimensional space by following a set of velocity- and position-update rules that use only local information. In the context of optimization, the desired behavior is a fast collective movement toward the tallest peak or the deepest valley of a landscape. Finding a set of rules that consistently leads the swarm toward these extremes across different landscapes is one of the most active research topics in the field.

In striving for the most effective PSO algorithm, many PSO variants have been proposed. Differences between variants range from added constants [4] to evolved particle-movement rules for specific problems [5, 6]. The underlying problem that justifies the existence of these completely different approaches is the lack of specific knowledge about which algorithmic components provide good performance on particular types of problems and under different operating conditions.

In this article, we experimentally compare some of the most influential and promising PSO variants on a number of benchmarks functions. The experimental setup and the choice of the PSO variants allow the identification of performance differences that can only be ascribed to specific algorithmic components. In particular, our comparison focuses on the differences between mechanisms for updating a particle's velocity. This includes the selection of the population topology, the number of particles and the strategies for updating various parameters that influence performance. Furthermore, we assume that no *a priori* knowledge about the structure of the problem is available so that we can compare different PSO algorithms with their most commonly used parameter settings.

The algorithmic components that, according to the results of the comparison, provide good performance are then "patched together" in a composite PSO algorithm that we call *Frankenstein's PSO*, as it consists of different algorithmic components drawn from other PSO variants. Frankenstein's PSO is an *engineered* algorithm in the sense that it is assembled based on the observed strengths and weaknesses of the studied algorithms.

The performance of Frankenstein's PSO is evaluated by comparing it with that of other PSO algorithms. The results show that Frankenstein's PSO typically reaches a specific required solution quality in fewer function evaluations and with a higher success probability than the most commonly used PSO algorithms. We provide parameter selection guidelines for configuring Frankenstein's PSO taking into account the requirements of the optimization task at hand.

2 Particle Swarm Optimization Algorithms

Given an unconstrained d-dimensional objective function $f : \mathbb{R}^d \to \mathbb{R}$, a population of particles $\mathcal{P} = \{p_1, \ldots, p_n\}$ (called *swarm*) is randomly placed in the solution space. The objective function determines the quality of the solution represented by a particle's position. (Without loss of generality, we restrict the following discussion to minimization problems.)

At time step t, a particle p_i has an associated position vector $\boldsymbol{x}_i^t$ and a velocity vector $\boldsymbol{v}_i^t$. A vector $\boldsymbol{pb}_i^t$ (known as *personal best*) stores the best position the particle has found until time step t. This vector is updated every time particle p_i finds a better position.

A particle p_i has a topological neighborhood $\mathcal{N}_i \subseteq \mathcal{P}$ of particles. The best *personal best* vector in a particle's neighborhood (called *local best*) is a vector $\boldsymbol{lb}_i^t$ such that $f(\boldsymbol{lb}_i^t) \leq f(\boldsymbol{pb}_j^t)\, \forall p_j \in \mathcal{N}_i$.

PSO algorithms iterate updating the particles' velocities and positions until a stopping criterion is met. The basic velocity- and position-update rules are:

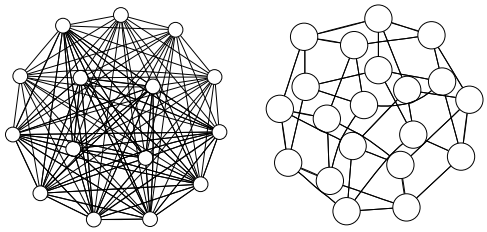
$$\boldsymbol{v}_i^{t+1} = \boldsymbol{v}_i^t + \varphi_1 \boldsymbol{U}_1^t (\boldsymbol{pb}_i^t - \boldsymbol{x}_i^t) + \varphi_2 \boldsymbol{U}_2^t (\boldsymbol{lb}_i^t - \boldsymbol{x}_i^t)\,, \tag{1}$$

and

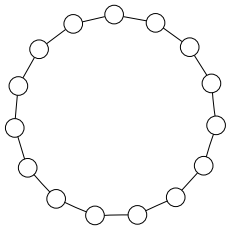
$$\boldsymbol{x}_i^{t+1} = \boldsymbol{x}_i^t + \boldsymbol{v}_i^{t+1}\,, \tag{2}$$

where φ_1 and φ_2 are two parameters called the *cognitive* and *social* acceleration coefficients respectively, $\boldsymbol{U}_1^t$ and $\boldsymbol{U}_2^t$ are two $d \times d$ diagonal matrices with in-diagonal elements uniformly distributed in the interval $[0, 1]$. (These matrices are generated at every iteration.) A particle's maximum velocity is a parameter that prevents velocities from growing to extremely large values [7, 8].

It is useful to think of the topology of a PSO algorithm as an undirected graph G, with a vertex set $V(G)$ having a one-to-one correspondence with the set of particles $\mathcal{P}$, and with an edge set $E(G)$ corresponding to the neighborhood relations between pairs of particles. Fig. 1 shows three example topologies.



(a) Fully connected (b) Square



(c) Ring

Figure 1: Some commonly used topologies in PSO algorithms. A particle's neighborhood is composed of all the particles directly connected to it. In (a), a fully connected topology is shown, in which each node is a neighbor of any other particle in the swarm. In (b), a square topology is shown, in which each particle is a neighbor of other 4 particles. In (c), a ring topology is shown, in which each particle is a neighbor of another 2 particles.

In the following, we describe the algorithmic variants that are part of our study. They are among the most influential and promising algorithmic variants. Moreover, they are representatives of the most common approaches taken to date for modifying the original PSO algorithm.

2.1 Canonical Particle Swarm Optimizer

Clerc and Kennedy [4] introduced a so-called *constriction factor* to avoid the unlimited growth of particles' velocities and to eliminate the maximum velocity parameter. The constriction factor χ modifies Eq. 1 to

$$\boldsymbol{v}_i^{t+1} = \chi \left(\boldsymbol{v}_i^t + \varphi_1 \boldsymbol{U}_1^t (\boldsymbol{pb}_i^t - \boldsymbol{x}_i^t) + \varphi_2 \boldsymbol{U}_2^t (\boldsymbol{lb}_i^t - \boldsymbol{x}_i^t) \right), \tag{3}$$

with

$$\chi = \frac{2k}{\left| 2 - \varphi - \sqrt{\varphi^2 - 4\varphi} \right|}, \tag{4}$$

where $k \in [0, 1]$, $\varphi = \sum_i \varphi_i$ and $\varphi > 4$. Usually, k is set to 1 and both φ_1 and φ_2 are set to 2.05, giving as a result χ equal to 0.729 [8, 9]. This way of setting the parameter values is now so widely used that it is known as the *canonical* PSO. We will refer to it simply as *canonical*.

2.2 Time-Varying Inertia Weight Particle Swarm Optimizer

Shi and Eberhart [10, 11] noticed that the first term of the right side of Eq. 1 plays the role of a particle's "inertia" and they introduced the idea of an *inertia weight*. The velocity-update rule was modified to

$$\boldsymbol{v}_i^{t+1} = w^t \boldsymbol{v}_i^t + \varphi_1 \boldsymbol{U}_1^t (\boldsymbol{pb}_i^t - \boldsymbol{x}_i^t) + \varphi_2 \boldsymbol{U}_2^t (\boldsymbol{lb}_i^t - \boldsymbol{x}_i^t)\,, \tag{5}$$

where w^t is the time-dependent inertia weight. Shi and Eberhart proposed to set the inertia weight according to a time-decreasing function; the idea is to have an algorithm that initially explores the search space and only later focuses on the most promising regions. Experimental results showed that this approach is effective [10, 7, 11]. The function used to schedule the inertia weight is defined as

$$w^t = \frac{wt_{max} - t}{wt_{max}}(w_{max} - w_{min}) + w_{min}\,, \tag{6}$$

where wt_{max} marks the time at which $w^t = w_{min}$; w_{max} and w_{min} are the maximum and minimum values the inertia weight can take, respectively. Usual settings are $w_{max} = 0.9$ and $w_{min} = 0.4$. We identify this variant as *decreasing-IW*.

Zheng et al. [12, 13] studied the effects of using a time-increasing inertia weight function obtaining, in some cases, better results than with the time-decreasing inertia weight variant. Concerning the schedule of the inertia weight, Zheng et al. used also Eq. 5, except that the values of w_{max} and w_{min} were interchanged. This variant is referred to as *increasing-IW*.

Eberhart and Shi [14] proposed another variant in which an inertia weight vector is randomly generated according to a uniform distribution in the range [0.5,1.0] (a different inertia weight per dimension). This range was inspired by Clerc and Kennedy's constriction factor because the expected value of the inertia weight in this case is $0.75 \approx 0.729$. Accordingly, in this *stochastic-IW* algorithm, acceleration coefficients are set to $\chi\varphi_i$ with $i \in \{1, 2\}$.

2.3 Fully Informed Particle Swarm Optimizer

In PSO, population topologies serve as a method to select the vector $\boldsymbol{lb}_i^t$ that will guide particle i in its search. To avoid relying only on the best neighbor, Mendes et al. [15] proposed the fully informed particle swarm (FIPS), in which a particle uses information from all its topological neighbors. Clerc and Kennedy's constriction factor is also adopted in FIPS; however, the value φ (i.e., the sum of the acceleration coefficients) is equally distributed among all the neighbors of a particle.

For a given particle p_i, we decompose, as suggested in [15], φ as $\varphi_k = \varphi/|\mathcal{N}_i| \;\; \forall p_k \in \mathcal{N}_i$. As a result, the velocity-update equation becomes

$$\boldsymbol{v}_i^{t+1} = \chi \left[\boldsymbol{v}_i^t + \sum_{p_k \in \mathcal{N}_i} \varphi_k \boldsymbol{U}_k^t (\boldsymbol{pb}_k^t - \boldsymbol{x}_i^t) \right]. \tag{7}$$

2.4 Self-Organizing Hierarchical Particle Swarm Optimizer with Time-varying Acceleration Coefficients

Ratnaweera et al. [16] proposed the self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients (HPSOTVAC), in which the inertia term in the velocity-update rule is eliminated. Additionally, if any component of a particle's velocity vector becomes zero (or very close to zero), it is reinitialized to a value proportional to the maximum velocity allowed V_{max}. This gives the algorithm a local search behavior which is amplified by linearly adapting the value of the acceleration coefficients φ_1 and φ_2. The cognitive coefficient φ_1 is decreased from 2.5 to 0.5 and the social coefficient φ_2 is increased from 0.5 to 2.5. In HPSOTVAC, V_{max} is linearly decreased during a run so as to reach the value $V_{max}/10$ at the end. A low reinitialization velocity near the end of the run allows particles to move slowly near the best region they have found. The resulting PSO variant is a kind of local search algorithm with occasional magnitude-decreasing unidimensional restarts.

2.5 Adaptive Hierarchical Particle Swarm Optimizer

Differently from the other variants, the adaptive hierarchical PSO (AHPSO) [17] modifies the neighborhood topology at run time. It uses a tree-like topology structure in which particles with better objective function evaluations are located in the upper nodes of the tree. At each iteration, a child particle updates its velocity considering its own previous best performance and the previous best performance of its parent. Before the velocity-update process takes place, the previous best fitness value of any particle is compared with that of its parent. If it is better, child and parent swap their positions in the hierarchy. Additionally, AHPSO adapts the branching degree of the tree while solving a problem to balance the diversification-intensification behavior of the algorithm: a hierarchy with a low branching degree has a more explorative behavior than a hierarchy with a high branching degree. In AHPSO, the branching degree is decreased by k_{adapt} degrees (one at a time) until a certain minimum degree d_{min} is reached. This process takes place every f_{adapt} number of iterations. For more details, see [17].

3 Experimental Setup

The focus of the comparison is on the performance effects of different mechanisms for updating a particle's velocity. However, other factors are also considered. The complete experimental design examines five factors:

1. **PSO algorithm**. This factor considers differences between PSO variants. Specifically, we focused on different strategies for updating inertia weights (fixed and time-varying inertia weight variants), dynamic population topologies (AHPSO) and different strategies for updating a particle's velocity (FIPS and HPSOTVAC).

2. **Problem**. We selected some of the most commonly used benchmark functions in experimental evolutionary computation. Most of these functions have their global optimum at the origin, but since it is known that some

Table 1: Benchmark Problems

Function Name	Search Range	Modality
Ackley	$[-32.0, 32.0]^n$	Multimodal
Griewank	$[-600.0, 600.0]^n$	Multimodal
Rastrigin	$[-5.12, 5.12]^n$	Multimodal
Salomon	$[-100.0, 100.0]^n$	Multimodal
Schwefel (sine root)	$[-512.0, 512.0]^n$	Multimodal
Step	$[-5.12, 5.12]^n$	Multimodal
Rosenbrock	$[-30.0, 30.0]^n$	Unimodal
Sphere	$[-100.0, 100.0]^n$	Unimodal

PSO variants have a tendency to search near this region [18], we shifted and biased their optimum value. In most cases, we used exactly the same values that were proposed in the set of benchmark functions used for the special session on real parameter optimization of the IEEE CEC 2005 [19]. Table 1 lists the benchmark functions used in our study. In all cases, we used their 30-dimensional versions. (Their mathematical definitions can be found in this paper's supplementary information web page [20][1].) All algorithms were run 100 times on each problem.

3. **Population topology**. We considered three population topologies: fully connected, square, and ring (see Fig. 1). These topologies provide different degrees of connectivity between particles. Different topologies favor exploration in different degrees: The less connected is a topology, the more it delays the propagation of the best-so-far solution. Thus, low connected topologies result in more explorative behavior than highly connected ones [21].

4. **Population size**. For this factor, we considered three levels: 20, 40 and 60 particles. Square topologies can have different configurations for the same number of particles. The configurations that we considered for 20, 40 and 60 particles were 5×4, 5×8 and 6×10 respectively. Large population sizes provide higher initial diversity.

5. **Maximum number of function evaluations**. This factor determined the stopping criterion. The limit was set to 10^6 function evaluations. However, data were collected during a run to determine relative performances for shorter runs.

We assume that no *a priori* knowledge about the structure of the problem is available. This situation is not uncommon when solving practical problems. If this is the case, our first approach would probably be to use a state-of-the-art algorithm with good parameter settings recommended by earlier studies. Our experimental setup reflects this reasoning by using, for each algorithm, the same parameterization across all benchmark problems. When possible, we use the most commonly used parameter settings found in the literature. These parameter settings are listed in Table 2.

[1] At this same address the reader can find all the supplementary information that, for the sake of conciseness, we do not present here.

Table 2: Parameter settings

Algorithm	Settings
Canonical	Acceleration coefficients $\varphi_1 = \varphi_2 = 2.05$. Constriction factor $\chi = 0.729$. Maximum velocity $V_{max} = \pm X_{max}$, where X_{max} is the maximum of the search range.
Decreasing-IW	Acceleration coefficients $\varphi_1 = \varphi_2 = 2.0$. Linearly-decreasing inertia weight from 0.9 to 0.4. The final value is reached at the end of the run. Maximum velocity $V_{max} = \pm X_{max}$.
Increasing-IW	Acceleration coefficients $\varphi_1 = \varphi_2 = 2.0$. Linearly-increasing inertia weight from 0.4 to 0.9. The final value is reached at the end of the run. Maximum velocity $V_{max} = \pm X_{max}$.
Stochastic-IW	Acceleration coefficients $\varphi_1 = \varphi_2 = 1.494$. Uniformly distributed random inertia weight in the range $[0.5, 1.0]$. Maximum velocity $V_{max} = \pm X_{max}$.
FIPS	Acceleration parameter $\varphi = 4.1$. Constriction factor $\chi = 0.729$. Maximum velocity $V_{max} = \pm X_{max}$.
HPSOTVAC	Linearly decreased φ_1 from 2.5 to 0.5. Linearly increased φ_2 from 0.5 to 2.5. Linearly decreased reinitialization velocity from V_{max} to $0.1 \cdot V_{max}$. Maximum velocity $V_{max} = \pm X_{max}$.
AHPSO	Acceleration coefficients $\varphi_1 = \varphi_2 = 2.05$. Constriction factor $\chi = 0.729$. Initial branching factor is set to 20, d_{min}, f_{adapt}, and k_{adapt} were set to 2, $1000 \cdot m$, and 3 respectively, where m is the number of particles.

In total, we have 6×9 plus 3 configurations (note that AHPSO adapts its topology automatically and, hence, only the population size is varied), that is, a total of 57 algorithm configurations.

In our experimental analysis, we examined the algorithms' performance at different levels of aggregation. At the lowest level, we used an analysis on individual instances based on run-time distributions (RTDs) [22], while for a more aggregate view, we ranked the algorithms according to the solution quality reached after different numbers of function evaluations. While the way of aggregating results is explained later, we describe here the most important elements of the RTD methodology; for more details see [22].

For our experimental analysis, we examined *qualified* RTDs, which give the empirical cumulative probability distribution of the time required by an algorithm to reach a specific bound on the solution quality. Rather than measuring computation time, we used the number of function evaluations as a surrogate measure. This is justified since in continuous optimization the evaluation of the objective function is typically the most expensive operation. Hence, in fact, we measured qualified *run-length distributions* (RLDs). Note that from the data collected for measuring RLDs, other typical performance characteristics can be derived easily. This also includes the so-called "success rate", that is, the ratio between the number of runs finding a solution of a certain quality and the total number of runs. The latter performance measure is actually only a "snapshot" of an RLD at a specific number of function evaluations.

Apart from the use of RLDs for the comparison of algorithms, another important aspect is that they give a clear indication on the convergence speed and stagnation tendency of an algorithm. In PSO literature, the term convergence has been given different meanings in different contexts [23]. Here, we use the term convergence to refer to the progress of an algorithm toward a definite solution quality value. We say that an algorithm stagnates if it shows a very slowly increasing or a non-increasing RLD toward the right tail of the distribution —see [22] for a more detailed discussion. In such situations, it is possible to improve an algorithm's performance by using occasional restarts and optimal restart policies can be derived *a posteriori*. If an algorithm restarts after l function evaluations and $RL_q(l)$ is the estimated probability of finding a solution of quality q after l function evaluations, one can easily estimate the number of function evaluations needed by the modified algorithm to find a required solution with a probability greater than or equal to z. This estimation is also called the *computational effort* in [24] and it is defined as

$$effort = \min_l l \cdot \frac{ln(1-z)}{ln(1-RL_q(l))} \,. \tag{8}$$

Fig. 2 shows the empirical RLDs of an algorithm with and without restarts together with an exponential distribution that the algorithm with restarts is ideally following. Both the cut-off time and the estimated effort for a probability of $z = 0.99$ are indicated with arrows.

A measure that will be used in the description of results is the *first hitting time* L_q for a specific solution quality q. L_q is the minimum number of function evaluations that an algorithm needs for finding a solution of a quality level q. It is defined as

$$L_q = \min\{l \geq 0; RL_q(l) > 0\} \,. \tag{9}$$

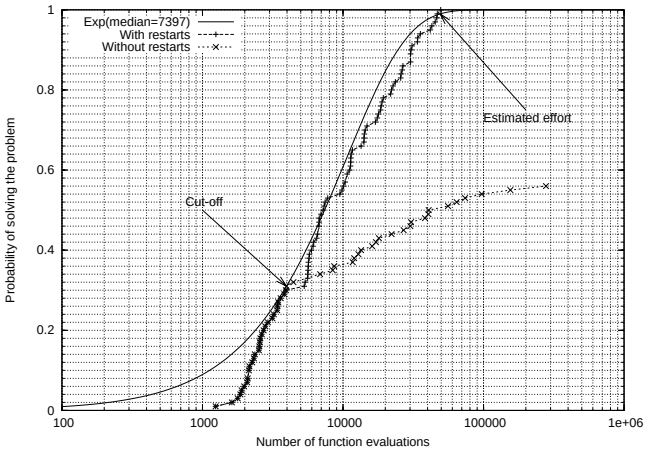


Figure 2: Example of empirical RLDs of an algorithm with and without restarts. See the text for more details.

4 Performance Comparison of Particle Swarm Optimization Algorithms

We carried out a two-phase experimental study. In the first phase, a problem-dependent run-time behavior comparison based on RLDs was performed (a preliminary series of results is published in [25]). In the second phase, aggregated information across all the problems of our benchmark suite was obtained. In this phase, the performance of the algorithms is analyzed as a function of the stopping criterion. Results that are valid for all tested problems are explicitly summarized in italics.

4.1 Results: Run-Length Distributions

The graphs presented below show a curve for each of the compared algorithms corresponding to a particular combination of a population topology and a population size. Since AHPSO does not use a fixed topology, its RLDs are the same across topologies and its results can be used as a fixed point to compare the influence of the neighborhood topology. The RLDs we present here were obtained using swarms of 20 and 60 particles.

As a first example, Fig. 3 shows the RLDs on the Griewank function. These plots are given with respect to a bound of 0.001% above the optimum value, corresponding to an absolute error of 0.0018. The fastest first hitting times for the same algorithm across different population size and topology settings are obtained with a population size of 20 and the fully connected topology. Conversely, the longest are obtained with a population size of 60 and the ring topology. The right tails of the distributions show a slowly-increasing or a non-increasing slope. This means that all PSO variants have a strong stagnation tendency, which is probably due to being trapped in a specific region of the search space. In fact, no variant is capable of finding a solution of the required quality with a probability of 1.0 with a population size of 20 particles. With 60 particles and a ring topology, only FIPS finds the required solution quality with

Table 3: Best performing configurations of each algorithm using independent restarts on the Griewank function[1,2]

Algorithm	Pop. Size	Topology	Cut-off	Effort	Restarts
FIPS	60	Ring	46440	**46440**	0
Canonical	60	Ring	71880	71880	0
Sto-IW	40	Ring	52160	131075	2
Inc-IW	20	Ring	24040	138644	5
HPSOTVAC	40	Ring	132080	155482	1
AHPSO	40	Adaptive	17360	207295	11
Dec-IW	60	Ring	663000	1326000	1

[1] Probabilities taken from the RLDs.
[2] Cut-off and effort measured in function evaluations. The effort is computed using Eq. 8.

a probability of 1.0, while the canonical PSO and HPSOTVAC reach a solution of the required quality with a probability of 0.99.

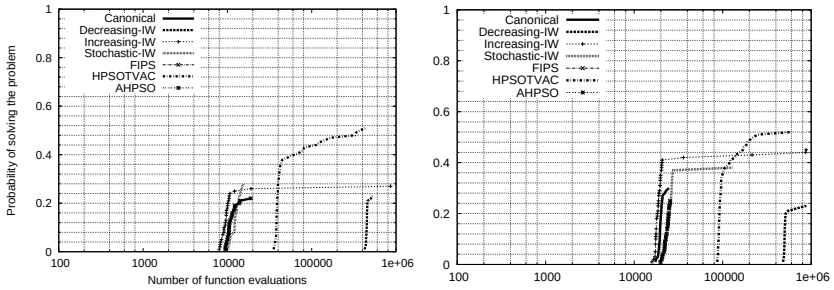
Result 1: *Depending on the required solution quality, PSO algorithms exhibit a stagnation tendency with different degrees of severity. This tendency is less strong with large population sizes and/or low connected topologies, that is, when a more explorative behavior is favored. However, even though the probability of solving the problem increases, first hitting times are delayed. In other words, there is no sign of a clear dominance of one configuration over the others across the whole range of allowed function evaluations.*

Another interesting observation is the strong influence of the topology on FIPS's results: while FIPS with a fully connected topology fails completely (i.e., it does not find a single solution of the required quality), with a ring topology, it is among the fastest algorithms with a high probability of success.

Result 2: *The algorithms are sensitive to a change in the population topology in different degrees. Among those tested, FIPS is the variant that shows the largest sensitivity to a change of this nature. On the contrary, HPSOTVAC and the decreasing inertia weight PSO algorithm are quite stable to topology changes.*

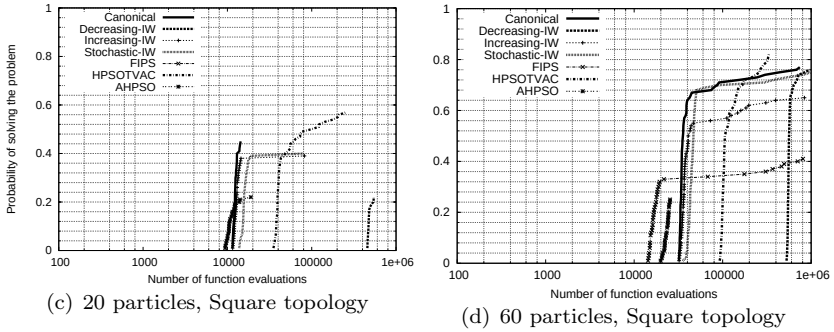
Clearly, the differences in the algorithms' relative performance may differ if the possibility of restarting them is considered. Assuming optimal restarts, in Table 3 we show for each algorithm the best configuration to solve this problem with a probability of 0.99 (at 0.001% above the global optimum). The best performing configurations of FIPS and the canonical PSO, both with 60 particles and the ring topology, do not profit from restarts under these conditions and they are overall also the two best variants for the considered goal. Hence, in this case the joint effect of choosing the right algorithm, a large population size and the right topology (in this case, ring), cannot be outperformed by strongly stagnating configurations and independent restarts.

As a second example, Fig. 4 shows the RLDs obtained by the compared PSO algorithms on the Rastrigin function. The solution quality bound is of 20.0% above the optimum, which corresponds to an absolute error of 66.0. While the overall shape of the RLDs is similar to that observed on Griewank, the stagnation tendency seems to be lower and the curves increase over a wider interval. Nevertheless, stagnation is still a major problem. On the Rastrigin function, the best performance is achieved when the algorithms use a square topology.



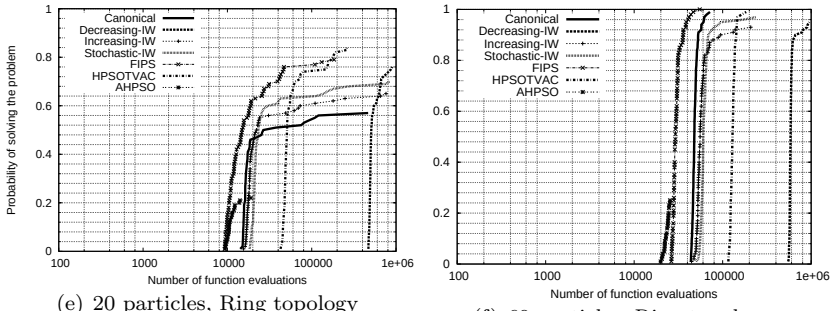
(a) 20 particles, Fully connected topology

(b) 60 particles, Fully connected topology



(c) 20 particles, Square topology

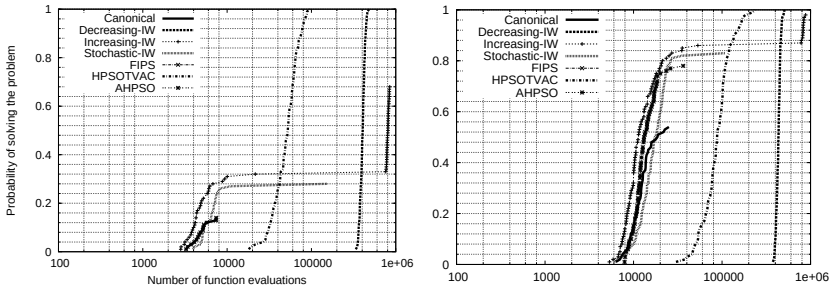
(d) 60 particles, Square topology



(e) 20 particles, Ring topology

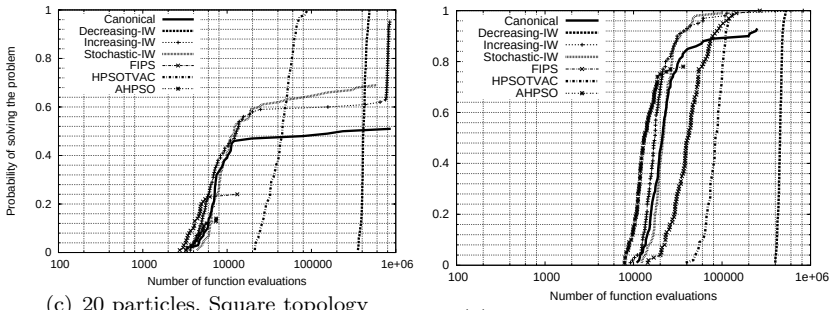
(f) 60 particles, Ring topology

Figure 3: RLDs on the Griewank function. The solution bound is set to 0.001% above the global optimum (equivalent to an absolute error of 0.0018). Plots (a), (c), and (e) in the left column show the RLDs obtained with 20 particles. Plots (b), (d), and (f) in the right column show the RLDs obtained with 60 particles. The effect of using different population topologies can be seen by comparing plots in different rows. The effect of using different number of particles can be seen by comparing columns.



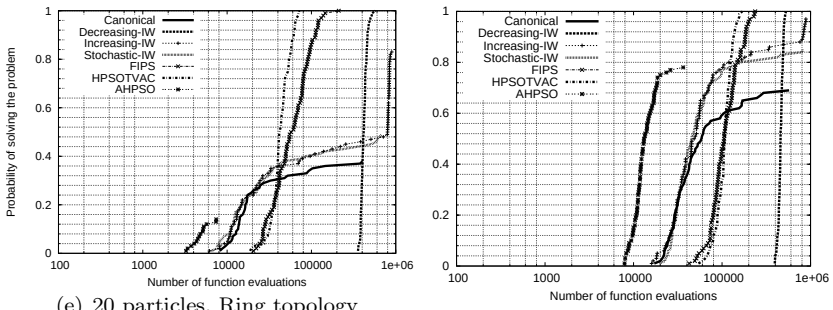
(a) 20 particles, Fully connected topology

(b) 60 particles, Fully connected topology



(c) 20 particles, Square topology

(d) 60 particles, Square topology



(e) 20 particles, Ring topology

(f) 60 particles, Ring topology

Figure 4: RLDs on the Rastrigin function. The solution bound is set to 20.0% above the global optimum (equivalent to an absolute error of 66.0). Plots (a), (c), and (e) in the left column show the RLDs obtained with 20 particles. Plots (b), (d), and (f) in the right column show the RLDs obtained with 60 particles. The effect of using different population topologies can be seen by comparing plots in different rows. The effect of using different number of particles can be seen by comparing columns.

Table 4: Best performing configurations of each algorithm using independent restarts on the Rastrigin function[1,2]

Algorithm	Pop. Size	Topology	Cut-off	Effort	Restarts
Inc-IW	40	Fully connected	12760	**41176**	3
Sto-IW	60	Square	50220	59119	1
Canonical	40	Square	17880	61126	3
AHPSO	60	Adaptive	18660	63792	3
HPSOTVAC	20	Ring	70220	70220	0
FIPS	40	Square	38640	93797	2
Dec-IW	20	Fully connected	460200	460200	0

[1] Probabilities taken from the RLDs.
[2] Cut-off and effort measured in function evaluations. The effort is computed using Eq. 8.

The only two variants that can find the required solution quality (regardless of the population topology used) with probability 1.0, are HPSOTVAC and decreasing-IW.

Assuming we optimally restart the algorithms, the best configurations to solve this problem with a probability of 0.99 (at 20.0% away from the global optimum) are shown in Table 4. Differently from what was observed on the Griewank function, on the Rastrigin function the best performing variants use a configuration with a strong stagnation tendency that benefits strongly from restarts.

Result 3: *Independent restarts can improve the performance of various PSO algorithms. In some cases, fast convergent configurations (that usually show a strong stagnation tendency) can outperform explorative ones. However, the optimal restart policy is problem and algorithm dependent.*

4.2 Results: Aggregated Data

Next, we analyze and present aggregated views of the results based on the median solution quality achieved by an algorithm after some specific number of function evaluations. This analysis considers only the 40 particles case which seemed a sensible choice to balance fast convergence and high population diversity. For each problem, we ranked 19 configurations (6 PSO algorithms $\times$ 3 topologies + AHPSO) and selected only those that were ranked in the first three places (what we call the top-three group). For this analysis, we do not assume that the algorithms are restarted in any way. Although restarting might improve the performance of some variants on some problems, the optimal cut-off time can only be determined *after* an estimate of success probabilities is available —i.e., when knowledge about the structure of the problem is available. This is in contradiction with the assumptions of our experimental setting.

Table 5 shows the distribution of appearances of the compared PSO algorithms in the top-three group. The table shows configurations ranked among the three best algorithms after 10^3, 10^4, 10^5, and 10^6 function evaluations (FES). The topology used by a particular configuration is shown in parenthesis. If two or more configurations found solutions with the same quality level (differences lower than 10^{-15} are not considered) and it was among the three best solution qualities, these configurations were considered to be part of the top-three group. In fact, we observed that, as the number of function evaluations increases, more

algorithms appear in the top-three group. This indicates that the difference in the solution quality achieved by different algorithms decreases and that many algorithms achieve the same quality level.

Table 5: Distribution of appearances of different PSO algorithms in the top-three group[1]

FES	Ackley	Griewank	Rastrigin	Salomon	Schwefel	Step	Rosenbrock	Sphere
10^3	FIPS (F,S) Inc-IW (F)	FIPS (F,S) Inc-IW (F)	FIPS (F,S) Inc-IW (F)	FIPS (F,S) HPSOTVAC	Inc-IW (F,S,R)	FIPS (F,S) Inc-IW (F)	AHPSO Canonical (F) Sto-IW (F)	FIPS (F,S) Inc-IW (F)
10^4	FIPS (S,R) Inc-IW (F)	Canonical (F) FIPS (S) Inc-IW (F)	AHPSO Canonical (F) Inc-IW (F)	Canonical (F) Inc-IW (F) Sto-IW (F)	AHPSO Inc-IW (F) Sto-IW (F)	AHPSO Canonical (F) Inc-IW (F) Sto-IW (F)	AHPSO Canonical (F) Sto-IW (F)	AHPSO Canonical (F) Inc-IW (F)
10^5	Canonical (S) FIPS (R) Inc-IW (F)	Canonical (S,R) FIPS (R) Inc-IW (S,R) Sto-IW (S,R)	FIPS (S) Inc-IW (S) Sto-IW (S)	Canonical (S,R) FIPS (R) Inc-IW (F,S) Sto-IW (F,S,R)	HPSOTVAC (F,S,R)	Canonical (S) Inc-IW (F) Sto-IW (F)	AHPSO Canonical (F) Sto-IW (F)	AHPSO Canonical (F,S,R) FIPS (R) Inc-IW (F,S,R) Sto-IW (F,S,R)
10^6	Canonical (S,R) Dec-IW (F,S,R) FIPS (R) Inc-IW (S,R) Sto-IW (S,R)	Canonical (S,R) Dec-IW (S,R) FIPS (R) HPSOTVAC (F,S,R) Inc-IW (S,R) Sto-IW (S,R)	HPSOTVAC (F,S,R)	Canonical (S,R) Dec-IW (F,S,R) FIPS (R) HPSOTVAC (F,S,R) Inc-IW (S,R) Sto-IW (S,R)	Dec-IW (S) FIPS (R) HPSOTVAC (R)	Canonical (S,R) Dec-IW (F,S,R) FIPS (R) HPSOTVAC (F,S,R) Inc-IW (F,S,R) Sto-IW (F,S,R)	AHPSO Canonical (F) Sto-IW (F)	AHPSO Canonical (F,S,R) Dec-IW (F,S,R) FIPS (R) HPSOTVAC (S) Inc-IW (F,S,R) Sto-IW (F,S,R)

[1] F, S and R stand for fully connected, square and ring respectively.

If we consider the maximum number of function evaluations as a termination criterion, we want to know which PSO variants are the best for different numbers of function evaluations. Table 6 shows the algorithms that most often appear in the top-three group in Table 5 for different termination criteria. The column Σ shows the total number of times each algorithm appeared in the top-three group. The rightmost column shows the distribution of appearances in the top-three group between multi- and unimodal functions.

Result 4: *If only a very limited number of function evaluations is allowed, quickly converging configurations (i.e., those with highly connected topologies and/or low inertia weights) obtain the best results. When solution quality is the most important aspect, algorithms with explorative properties perform best.*

4.3 Results: Different Inertia Weight Schedules

In the time-decreasing inertia weight PSO algorithm, the schedule of the inertia weight is decreased from a maximum to a minimum over the whole optimization process. We wanted to know whether by a faster decrease of the inertia weight and the resulting faster convergence, decreasing-IW would still be able to find high-quality solutions. To do so, we modified the inertia weight schedule, which is based on Eq. 6: whenever the inertia weight reaches its minimum value, it remains there. Five inertia weight schedules of $wt_{max} \in \{10^2, 10^3, 10^4, 10^5, 10^6\}$ function evaluations each, were tried. The remaining parameters were set as shown in Table 2.

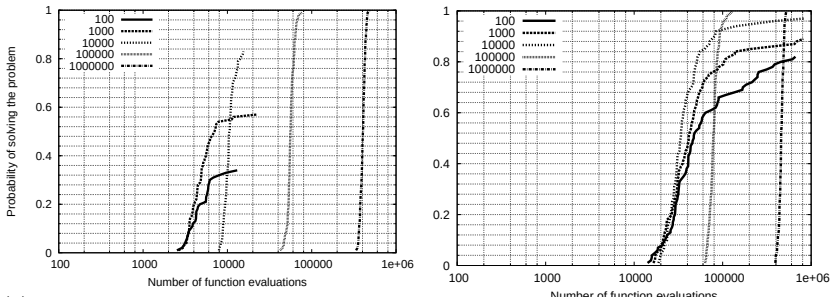
As an example of the effects of different inertia weight schedules, Fig. 5 shows the RLDs obtained by the decreasing inertia weight PSO algorithm on the Rastrigin function at a solution quality of 20.0% away from the global optimum. We only show the graphs corresponding to configurations using 20 and 60 particles with a fully connected topology and a ring topology, respectively. The results obtained with other configurations and other problems show a similar pattern.

Table 6: Best PSO variants for different termination criteria

Budget (in FES)	Algorithm(Topology)	Σ	multi/unimodal
10^3	Inc-IW(F), FIPS(F,S)	6	5/1
10^4	Inc-IW(F)	7	6/1
10^5	Canonical(S)	5	4/1
10^6	Dec-IW(S), FIPS(R)	6	5/1

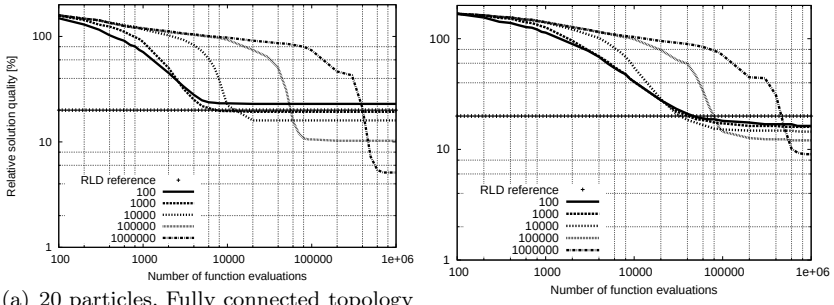
As expected, faster schedules make the algorithm behave more greedily. In some cases, this greediness results in a strong reduction of the first hitting time. How much it is reduced, depends also on the number of particles and the population topology. However, the more aggressive the schedules are, the stronger is the stagnation tendency. For a same schedule, the severity of the stagnation tendency is alleviated by both an increase in the number of particles and the use of a low connected topology.

Fig. 6 shows the development over time of the solution quality for different inertia weight schedules on the Rastrigin function. Even though slow schedules perform very poorly during the first phase in the optimization process, they are the ones that are capable of finding the best quality solutions. On the other hand, for short run-times the best solutions are found with fast schedules. The trade-off between fast convergence and high quality reaching capabilities is clear.



(a) 20 particles, Fully connected topology (b) 60 particles, Ring topology

Figure 5: RLDs obtained by the decreasing inertia weight PSO with different inertia weight schedules on the Rastrigin function. The solution quality demanded is of 20.0% away from the global optimum.



(a) 20 particles, Fully connected topology (b) 60 particles, Ring topology

Figure 6: Solution quality development over time for different inertia weight schedules on Rastrigin function. These plots are based on the medians of the algorithms' RLDs. The horizontal line represents the solution quality requirement used to obtain the RLDs shown in Fig. 5.

Table 7: Best performing decreasing inertia weight configurations using independent restarts on the Rastrigin function[1,2]

Schedule	Pop. Size	Topology	Cut-off	Effort	Restarts
10^2	20	Square	10120	44393	4
10^3	40	Fully connected	12440	**34496**	2
10^4	40	Fully connected	20960	38217	1
10^5	40	Fully connected	73800	73800	0
10^6	20	Fully connected	460200	460200	0

[1] Probabilities taken from the RLDs.
[2] Cut-off and effort measured in function evaluations. The effort is computed using Eq. 8.

Table 7 shows the results of using independent restarts together with different inertia weight schedules. The best configuration is the one with the computational effort highlighted in boldface. The results show that, when solving the Rastrigin function, it is possible to outperform configurations with relatively slow schedules through the use of independent restarts. Additionally, the best decreasing inertia weight PSO configuration would outperform the best PSO algorithm of the first comparison if both were optimally restarted (compare Tables 7 and 4).

Result 5: *Faster inertia weight schedules induce a faster convergence of decreasing-IW. At the same time, they increase the algorithm's stagnation tendencies. The inertia weight schedule can be used as a tunable parameter to control the algorithm's convergence speed and, consequently, its ability to find high-quality solutions. The application scenario defines the compromise to be taken: speed or quality.*

We carried out an analysis similar to the one just presented with the increasing inertia weight PSO algorithm. In this case, only with schedules of 10^5 and 10^6 function evaluations meaningful results are obtained.

4.4 Summary of Results

The results of our experimental study can be summarized as follows.

Stagnation tendencies of PSO algorithms can be alleviated by using a large population and/or low connected topologies. However, this comes at the price of a slower convergence speed. Another approach to reduce stagnation is to use restarts. Fast convergent variants can outperform explorative ones if independent restarts are used. However, optimal restart schedules are problem-dependent and determining them requires previous experimentation [22].

Some PSO algorithms use time-varying parameters and scheduling them in different ways has a major impact on their performance. Slow inertia weight schedules favor exploration in the decreasing inertia weight PSO while fast ones favor fast convergence. An appropriate schedule can be selected to meet the requirements of an application scenario.

Finally, the usage of different population topologies can have a tremendous impact on the performance of a PSO algorithm. FIPS, for example, is among the best algorithms for short runs when using a fully connected topology. With a ring topology, it is among the best ones for long runs.

5 Frankenstein's Particle Swarm Optimization Algorithm

Insights on experimental results ideally also guide towards the definition of new, better performing algorithms. In fact, here we introduce a new PSO algorithm that is assembled from algorithmic components that are taken from the other PSO algorithms we have examined. The algorithmic components included in our *Frankenstein's PSO* algorithm contribute in their original context to either find solutions of very good quality in the long run or find good quality solutions in few function evaluations.

5.1 Constituent Algorithmic Components

Frankenstein's PSO is composed of three main algorithmic components, namely (i) an adaptive population topology, (ii) a mechanism for updating a particle's velocity and position that provides fast convergence, and (iii) a decreasing inertia weight.

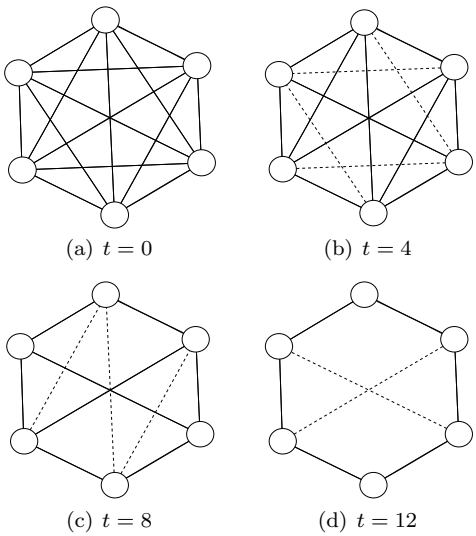
The adaptive population topology starts as a fully connected one that, as the optimization process evolves, decreases its connectivity until it ends up being a ring topology. Interestingly enough, it is exactly the opposite approach than the one taken by Suganthan [26]. However, note that our approach is entirely based on the results of the empirical analysis presented in the previous section. Specifically, our choice is based on the fact that a highly connected topology during the first iterations gives an algorithm a fast convergent behavior that allows it to find good quality solutions early in a run (see Results 1 and 4). The topology connectivity is then decreased, so that the risk of getting trapped somewhere in the search space is lowered and, hence, exploration is enhanced. Including this component into the algorithm permits the achievement of good performance across a wider range of run lengths as is shown later.

The connectivity of the population topology is also decreased in AHPSO. However, we do not use a hierarchical topology as it is not clear from our results what its contribution to a good performance is.

The topology adaptation scheme works as follows. Suppose we have a particle swarm composed of n particles. We schedule the adaptation of the topology so that in k iterations (with $k \geq n$), we transform a fully connected topology with $n(n-1)/2$ edges into a ring topology that has only n edges. The total number of edges that have to be eliminated is $n(n-3)/2$. Every $\lceil k/(n-3) \rceil$ iterations we remove m edges. The number m of edges to remove follows an arithmetic regression pattern of the form $n-2, n-3, \ldots, 2$. We sweep m nodes removing one edge per node chosen uniformly at random from the edges that do not belong to the exterior ring which is predefined in advance (just as it is done when using the normal ring topology). In this way, in $n-3$ elimination steps we transform the population topology. Fig. 7 shows a graphical example of how the process just described is carried out.

The adaptation of the population topology must be exploited by the underlying particles' velocity- and position-update mechanism. If we refer to Table 6, we see that the only velocity- and position-update mechanism that is ranked among the best and that uses different topologies is FIPS: For short runs, FIPS's best performance is obtained with the fully connected topology; for longer runs, this algorithm reaches very high performance with a low connected topology. Hence, the second main component of Frankenstein's PSO is FIPS's velocity and position-update mechanism, since it allows for a fast convergence during the very first iterations and it effectively uses the ring topology for long runs. The only modification to this component is that Clerc and Kennedy's constriction factor is not used. We employ a decreasing inertia weight since it allows the user to tune the algorithm's exploration/exploitation capabilities. In Section 4.3, we saw how a proper selection of the schedule of the inertia weight can dramatically change the performance of a PSO algorithm. We use a decreasing inertia weight so that the exploration behavior, which is favored by a low connected topology, can be controlled.

The pseudocode of Frankenstein's PSO is shown in Algorithm 1. The main



(a) $t = 0$ (b) $t = 4$

(c) $t = 8$ (d) $t = 12$

Figure 7: Topology adaptation process. Suppose $n = 6$ and $k = 12$. Then, every $\lceil 12/(6-3) \rceil = 4$ iterations we remove some edges from the graph. In $6 - 3 = 3$ steps, the elimination process will be finished. (a), at $t = 0$ a fully connected topology is used. (b), at $t = 4$ the $6 - 2 = 4$ edges to be removed are shown in dashed lines. (c), at $t = 8$ the $6 - 3 = 3$ edges to be removed are shown in dashed lines. (d), at $t = 12$ the remaining $6 - 4 = 2$ edges to be removed are shown in dashed lines. From $t = 12$ on, the algorithm uses a ring topology.

loop cycles through the three algorithmic components: topology adaptation, inertia weight, and the particles' velocity and position updates. The topology update mechanism is only executed while the algorithm's current number of iterations is lower than or equal to a parameter k, which specifies the topology update schedule. Since it is guaranteed that the ring topology is reached after iteration k, there is no need to call this procedure thereafter. In Algorithm 1, a variable *esteps* is used to ensure that the number of eliminated edges in the topology follows an arithmetic regression pattern. Note that the elimination of neighborhood relations is symmetrical, that is, if particle r is removed from the neighborhood of particle i, particle i is also removed from the neighborhood of particle r. The inertia weight is updated as in the decreasing inertia weight PSO with different schedules. Finally, the velocity- and position-update mechanism is applied in the same way as in FIPS. The user needs to specify the value of the parameter φ.

5.2 Parameterization Effects

We studied the impact on the algorithm's performance of different parameter settings. In particular, on the effects of the topology adaptation and the inertia weight schedules. The remaining parameters were the same as those used in the original context of the different algorithmic components as is shown in Table 8.

The experimental conditions under which we evaluated the performance of the algorithm were the same that we used in the comparison of the different

Algorithm 1: Frankenstein's PSO algorithm

/* Initialization */
for $i = 1$ to n **do**
  Create particle p_i and add it to the set of particles $\mathcal{P}$
  Initialize its vectors $\boldsymbol{x}_i$ and $\boldsymbol{v}_i$ to random values within the search range and maximum allowed velocities
  Set $\boldsymbol{pb}_i = \boldsymbol{x}_i$
  Set $\mathcal{N}_i = \mathcal{P}$
end for

/* Main Loop */
Set $t = 0$
Set $esteps = 0$
repeat
  /* Evaluation Loop */
  for $i = 1$ to n **do**
    if $f(\boldsymbol{x}_i)$ is better than $f(\boldsymbol{pb}_i)$ **then**
      Set $\boldsymbol{pb}_i = \boldsymbol{x}_i$
    end if
  end for
  /* Topology Update */
  if $t > 0 \wedge t <= k \wedge t \bmod \lceil k/(n-3) \rceil = 0$ **then**
    /* $t > 0$ ensures that a fully connected topology is used first */
    /* $t <= k$ ensures that the topology adaptation process is not called after iteration k */
    /* $t \bmod \lceil k/(n-3) \rceil = 0$ ensures the correct scheduling of the topology adaptation process */
    for $i = 1$ to $n - (2 + esteps)$ **do**
      /* $n - (2 + esteps)$ ensures the arithmetic regression pattern */
      if $|\mathcal{N}_i| > 2$ **then**
        /* $|\mathcal{N}_i| > 2$ ensures proper node selection */
        Select at random particle p_r from $\mathcal{N}_i$ such that p_r is not adjacent to p_i
        Eliminate particle p_r from $\mathcal{N}_i$
        Eliminate particle p_i from $\mathcal{N}_r$
      end if
    end for
    Set $esteps = esteps + 1$
  end if
  /* Inertia Weight Update */
  if $t \leq iwt_{max}$ **then**
    Set $w(t) = \frac{wt_{max} - t}{wt_{max}}(w_{max} - w_{min}) + w_{min}$
  else
    Set $w(t) = w_{min}$
  end if
  /* Velocity and Position Update */
  for $i = 1$ to n **do**
    Generate $\boldsymbol{U}_m^t \ \forall p_m \in \mathcal{N}_i$
    Set $\varphi_m = \varphi / |\mathcal{N}_i| \ \forall p_m \in \mathcal{N}_i$
    Set $\boldsymbol{v}_i^{t+1} = w^t \boldsymbol{v}_i^t + \sum_{p_m \in \mathcal{N}_i} \varphi_k \boldsymbol{U}_k^t (\boldsymbol{pb}_k^t - \boldsymbol{x}_i^t)$
    Set $\boldsymbol{x}_i = \boldsymbol{x}_i + \boldsymbol{v}_i$
  end for
  Set $t = t + 1$
  Set $\boldsymbol{sol} = \underset{p_i \in \mathcal{P}}{\operatorname{argmin}} f(\boldsymbol{pb}_i^t)$
until $f(\boldsymbol{sol})$ value is good enough or $t = t_{max}$

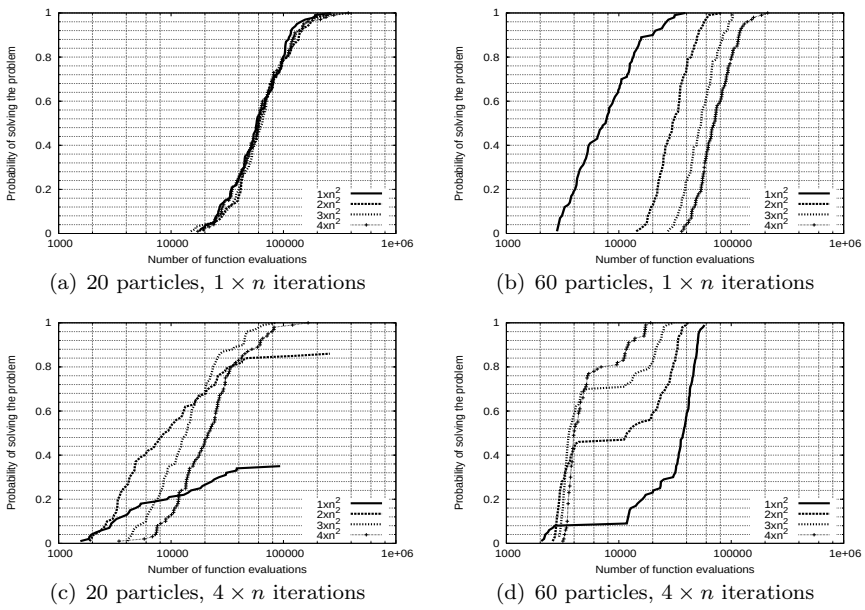
PSO algorithms. Three swarm sizes ($n = 20,\ 40,\ 60$), four schedules of the

Table 8: Frankenstein's PSO Parameter Settings

Algorithm	Settings
Frankenstein's PSO	Maximum velocity $V_{max} = \pm X_{max}$. Linearly-decreasing inertia weight from 0.9 to 0.4. The sum of the acceleration coefficients, φ, is equal to 4.0.

topology adaptation (measured in iterations) ($k = n,\ 2n,\ 3n,\ 4n$) and four schedules of the inertia weight (measured in function evaluations) ($wt_{max} = n^2$, $2n^2,\ 3n^2,\ 4n^2$) were tried.

As an illustrative example of the results, consider Fig. 8. It shows the RLDs obtained by Frankenstein's PSO algorithm on the Rastrigin function. These distributions correspond, as before, to a solution quality 20.0% above the optimum value. Only the results obtained with the extreme configurations are shown.



(a) 20 particles, $1 \times n$ iterations (b) 60 particles, $1 \times n$ iterations

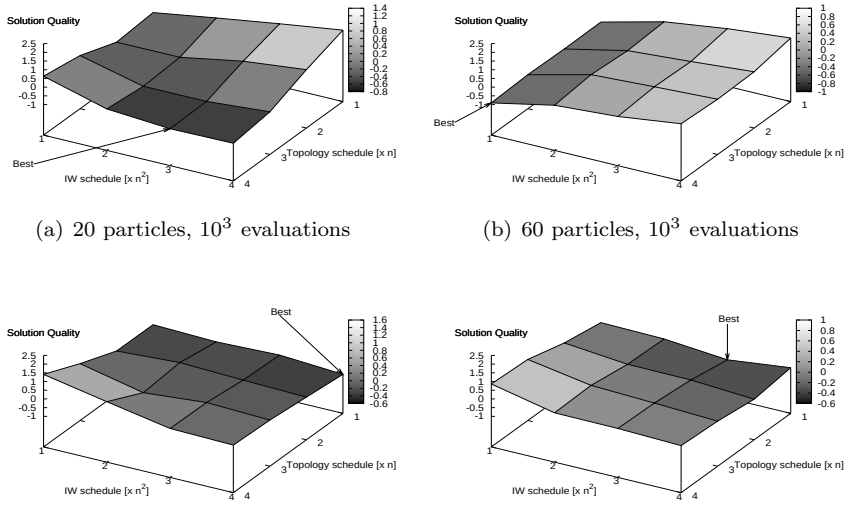
(c) 20 particles, $4 \times n$ iterations (d) 60 particles, $4 \times n$ iterations

Figure 8: RLDs obtained by Frankenstein's PSO algorithm on Rastrigin function. The solution quality demanded is of 20.0% above the global optimum. Columns show the RLDs obtained with different number of particles. Rows show the RLDs obtained with different topology adaptation schedules. Each graph shows four RLDs that correspond to different inertia weight schedules.

A combination of a slow topology adaptation schedule ($3n$ or $4n$) and a fast inertia weight schedule (n^2 or $2n^2$) promotes the stagnation of the algorithm. This can be explained if we recall that FIPS has a strong stagnation tendency when using a highly connected topology: A slow topology adaptation schedule maintains for more iterations a high topology connectivity and a fast inertia weight schedule reduces quickly the exploration capabilities of the particle swarm. These two effects also increase the algorithm's stagnation tendency.

However, such a configuration also reduces the first hitting time, as can be observed in the graphs. To counteract a fast convergence/stagnation tendency, the two possibilities are to slow down the inertia weight schedule or to fasten the adaptation of the topology. This again comes with a delay of the first hitting times.

Increasing the number of particles increases initial diversity during the algorithm's first iterations. The exploitation of this initial diversity depends on the topology adaptation and inertia weight schedules. The configurations that appear to better exploit initial diversity are those in which these two schedules are slow.



(a) 20 particles, 10^3 evaluations

(b) 60 particles, 10^3 evaluations

(c) 20 particles, 10^6 evaluations

(d) 60 particles, 10^6 evaluations

Figure 9: Average standard solution quality as a function of the topology adaptation and the inertia weight schedules for different termination criteria. Columns show the results obtained with different number of particles. Rows show the results obtained for different termination criteria. In each case, the best configuration is pointed by an arrow.

Fig. 9 shows the average (over the 8 benchmark problems of the experimental setup) standard solution quality (i.e., for each group, the mean is equal to zero and the standard deviation is one) as a function of the topology adaptation and the inertia weight schedules for different termination criteria. Since we work with minimization, a more negative average standard solution quality means that the specific configuration found better solutions.

According to Fig. 9, the algorithm needs more explorative configurations (i.e., fast topology adaptation schedules and slow inertia weight schedules) for long runs. For short runs, greedy configurations (i.e., slow topology adaptation schedules and fast inertia weight schedules) yield the best results. For runs of 10^4 and 10^5 function evaluations, the best configurations are intermediate ones (i.e., either fast or slow topology adaptation or inertia weight schedules).

The extra diversity that a large population provides, needs to be counter-

balanced by the chosen configuration. For example, at 10^3 function evaluations, the best configuration tends to have faster inertia weight schedules for larger swarms. With 20 particles, the best configuration is at point $(4, 3)$ while with 40 and 60 particles, the best configurations are at $(4, 2)$ and $(4, 1)$, respectively. These results are consistent with those of the experimental comparison.

6 Performance Validation

Finally, we evaluate the performance of Frankenstein's PSO by comparing its best configurations with those of the PSO algorithms described in Section 4.

To do so, we first identify the best configurations of every PSO variant as a function of the maximum number of function evaluations in an analogous way to Section 4.2. (We select candidate configurations for each PSO algorithm separately to avoid artificially favoring some PSO variants because of having more representatives in the ranking.) For each PSO algorithm we consider all its configurations resulting from our experimental setup and we choose, for each termination criterion, the best performing ones. Table 9 shows these best configurations for each PSO algorithm and termination criterion.

In a second step, we compared the algorithms' configurations shown in Table 9. For this purpose, and to be able to compare their relative effectiveness, we standardized the median solution qualities achieved by each one of them. Table 10 shows the standardized median solution quality obtained by each configuration (identified only by the algorithm's name) for each termination criterion. The best values for each individual problem and stopping criterion are highlighted in boldface.

Table 9: Best configurations of different PSO variants for different termination criteria

FES	Algorithm	Configuration			
		Particles (n)	Topology[1]	Topology Schedule	Inertia Weight Schedule
10^3	Canonical	20	F	-	-
	Decreasing-IW	20	F	-	10^2
	Increasing-IW	20	F	-	10^5
	Stochastic-IW	20	F	-	-
	FIPS	20	S	-	-
	HPSOTVAC	20	S	-	-
	AHPSO	20	T	-	-
	Frankenstein's PSO	20	A	$4n$	$3n^2$
10^4	Canonical	20	S	-	-
	Decreasing-IW	20	F	-	10^3
	Increasing-IW	20	S	-	10^6
	Stochastic-IW	20	S	-	-
	FIPS	20	R	-	-
	HPSOTVAC	20	S	-	-
	AHPSO	40	T	-	-
	Frankenstein's PSO	20	A	$4n$	$4n^2$
10^5	Canonical	40	S	-	-
	Decreasing-IW	60	F	-	10^4
	Increasing-IW	60	F	-	10^6
	Stochastic-IW	20	S	-	-
	FIPS	40	R	-	-
	HPSOTVAC	20	R	-	-
	AHPSO	60	T	-	-
	Frankenstein's PSO	20	A	$4n$	$3n^2$
10^6	Canonical	60	S	-	-
	Decreasing-IW	40	R	-	10^5
	Increasing-IW	60	S	-	10^6
	Stochastic-IW	60	S	-	-
	FIPS	40	R	-	-
	HPSOTVAC	40	S	-	-
	AHPSO	60	T	-	-
	Frankenstein's PSO	60	A	$2n$	$4n^2$

[1] F, S, R, T and A stand for fully connected, square, ring, tree-like and adaptive, respectively.

Table 10: Best overall configurations of different PSO variants for different termination criteria. Each group is sorted by the average standard solution quality in ascending order, so the best overall configuration is listed first.

FES	Algorithm	Ackley	Griewank	Rastrigin	Salomon	Schwefel	Step	Rosenbrock	Sphere	Average
10^3	Frankenstein's PSO	**-2.024**	**-0.955**	-0.975	**-0.517**	1.378	**-1.315**	-0.302	**-1.108**	-0.727
	Increasing-IW	-0.013	-0.393	-0.950	-0.323	**-1.229**	-0.645	-0.367	-0.371	-0.536
	Decreasing-IW	-0.002	-0.386	**-1.067**	-0.316	-1.199	-0.359	-0.474	-0.425	-0.528
	FIPS	-0.765	-0.430	-0.080	-0.457	1.432	-0.932	0.206	-0.538	-0.195
	Canonical	0.476	-0.156	0.287	-0.276	-0.213	0.406	**-0.491**	-0.057	-0.003
	Stochastic-IW	0.656	0.124	0.652	-0.237	-0.046	0.693	-0.488	0.304	0.207
	AHPSO	0.476	-0.156	0.287	2.464	-0.213	0.406	**-0.491**	-0.057	0.340
	HPSOTVAC	1.198	2.353	1.847	-0.338	0.090	1.745	2.406	2.251	1.444
10^4	Increasing-IW	-0.129	-0.564	-0.593	-0.349	**-0.797**	-0.539	-0.348	-0.359	-0.460
	Canonical	-0.212	-0.616	-0.591	-0.373	-0.459	-0.539	-0.376	-0.359	-0.441
	Decreasing-IW	-0.065	-0.518	**-0.962**	-0.341	-0.754	-0.085	-0.370	-0.358	-0.431
	Frankenstein's PSO	**-1.061**	**-0.761**	0.056	**-0.386**	1.332	**-0.993**	**-0.414**	**-0.361**	-0.324
	Stochastic-IW	-0.131	0.443	-0.512	-0.361	-0.541	-0.085	-0.290	-0.359	-0.230
	FIPS	-1.056	-0.718	1.567	-0.378	1.760	-0.539	-0.364	**-0.361**	-0.011
	AHPSO	0.569	0.656	-0.512	2.474	-0.641	0.596	-0.312	-0.316	0.314
	HPSOTVAC	2.086	2.077	1.546	-0.287	0.101	2.185	2.473	2.475	1.582
10^5	Frankenstein's PSO	**-0.354**	**-0.883**	-1.192	**-0.359**	**-1.548**	-0.487	0.782	**-0.354**	-0.549
	Decreasing-IW	**-0.354**	0.631	-0.709	-0.355	-0.311	**-0.787**	-0.983	**-0.354**	-0.402
	Increasing-IW	**-0.354**	0.631	0.108	-0.355	-0.271	**-0.787**	-0.441	**-0.354**	-0.228
	Canonical	**-0.354**	**-0.883**	0.313	**-0.359**	0.729	-0.487	0.216	**-0.354**	-0.147
	Stochastic-IW	**-0.354**	0.631	1.130	**-0.359**	0.649	**-0.787**	-1.013	**-0.354**	-0.057
	FIPS	**-0.354**	**-0.883**	1.060	-0.355	1.372	0.712	1.008	**-0.354**	0.276
	AHPSO	**-0.354**	1.639	0.721	2.475	0.529	0.712	**-1.019**	**-0.354**	0.544
	HPSOTVAC	2.475	**-0.883**	**-1.431**	-0.334	-1.149	1.911	1.449	2.475	0.564
10^6	Frankenstein's PSO	**-0.354**	**-0.354**	-0.787	**-0.358**	-1.257	**-0.661**	-0.058	**-0.504**	-0.542
	Increasing-IW	**-0.354**	**-0.354**	0.002	-0.354	0.019	**-0.661**	0.039	**-0.504**	-0.271
	Decreasing-IW	**-0.354**	**-0.354**	0.472	-0.354	0.367	**-0.661**	**-0.778**	**-0.504**	-0.271
	FIPS	**-0.354**	**-0.354**	-0.546	-0.354	**-1.349**	0.661	0.685	**-0.504**	-0.264
	Stochastic-IW	**-0.354**	**-0.354**	0.415	**-0.358**	0.705	**-0.661**	-0.529	**-0.504**	-0.205
	Canonical	**-0.354**	**-0.354**	0.815	**-0.358**	1.072	**-0.661**	-0.717	**-0.504**	-0.132
	HPSOTVAC	2.475	**-0.354**	**-1.760**	-0.341	-0.705	0.661	2.129	2.184	0.536
	AHPSO	**-0.354**	2.475	1.388	2.475	1.149	1.984	-0.771	0.840	1.148

For runs of 10^3, 10^5 and 10^6 function evaluations, the best overall configuration is the one of Frankenstein's PSO. For runs of 10^4 function evaluations, the configuration of Frankenstein's PSO is ranked in the fourth place. However, with this same number of function evaluations, the configuration of Frankenstein's PSO is the best configuration in 6 of the 8 benchmark problems. The average rank of Frankenstein's PSO after 10^4 function evaluations can be explained with the results on Schwefel's function: FIPS (whose particles' velocity- and position-update mechanism is the same of Frankenstein's PSO) is the worst algorithm for this termination criterion (and also for the one of 10^3 function evaluations) on Schwefel's function.

Overall, we conclude that Frankenstein's PSO algorithm is a high-performing PSO variant that, if properly parameterized, is faster and more reliable than the most commonly used PSO algorithms. The good performance of Frankenstein's PSO algorithm is the result of the synergistic effects of its algorithmic components.

7 Conclusions and Future Work

There is a large number of PSO variants proposed in current literature. This is a sign of the great interest that PSO has received since its introduction, and to some extent, its success. However, it is also a sign of the generalized lack of knowledge about which algorithmic components provide good performance on particular types of problems and under different operating conditions.

In an attempt to gain insight into the performance advantages of different algorithmic components, we compared the most influential and some of the most promising PSO variants. The results of the comparison itself revealed that no variant dominates all the others on all benchmark problems and under all tested circumstances. This means that some variants are able to find a solution of a certain quality faster than others, or given the possibility of using the same number of function evaluations, they are able to find solutions of better quality. Since variants differ only on some specific algorithmic components, differences in performance must come from these components and/or the way they interact with others. The question then becomes: Is it possible to combine different algorithmic components that seem to provide good performance into a single PSO variant capable of performing better than the variants from which these components were taken?

In this paper, we provide an answer to that question. The algorithm is called *Frankenstein's PSO*. It is an algorithm with three main algorithmic components: (i) an adaptive population topology that decreases its connectivity as the optimization process evolves; (ii) a particles' velocity- and position-update mechanism that exploits every stage of the topology adaptation process. The choice was to include the mechanism used in FIPS; and (iii) a time-decreasing inertia weight that allows the user to tune the algorithm's exploration/exploitation capabilities.

The results of our performance validation show that Frankenstein's PSO typically finds high quality solutions using fewer function evaluations and often also with a higher frequency than the most commonly used PSO algorithms. Some parameter tuning guidelines that take into account the requirements of the optimization task were provided.

One algorithmic component that could further improve Frankenstein's PSO performance, is the use of restarts. In our empirical evaluation, we showed that even simple independent restarts can dramatically improve the performance of many PSO variants. Unfortunately, there is no fixed restart policy that would work equally well for all problems. A restarting mechanism that uses information of the development of the optimization run is then one promising research direction that deserves further investigation.

Acknowledgment

This work was supported by the ANTS project, an *Action de Recherche Concertée* funded by the Scientific Research Directorate of the French Community of Belgium. Marco A. Montes de Oca acknowledges support from Programme Alβan, the European Union Programme of High Level Scholarships for Latin America, scholarship No. E05D054889MX. Thomas Stützle and Marco Dorigo acknowledge support from the Belgian National Fund for Scientific Research (FNRS), of which they are a Research Associate and a Research Director, respectively.

References

[1] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of IEEE International Conference on Neural Networks*. Piscataway, NJ, USA: IEEE Press, 1995, pp. 1942–1948.

[2] R. Eberhart and J. Kennedy, "A new optimizer using particle swarm theory," in *Proceedings of the 6th International Symposium on Micro Machine and Human Science*. Piscataway, NJ, USA: IEEE Press, 1995, pp. 39–43.

[3] J. Kennedy, R. Eberhart, and Y. Shi, *Swarm Intelligence*. San Francisco, CA, USA: Morgan Kaufmann, 2001.

[4] 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.

[5] R. Poli, C. D. Chio, and W. B. Langdon, "Exploring extended particle swarms: A genetic programming approach," in *Proceedings of the 2005 conference on Genetic and Evolutionary Computation*. New York, NY, USA: ACM Press, 2005, pp. 169–176.

[6] R. Poli, W. B. Langdon, and O. Holland, "Extending particle swarm optimisation via genetic programming," in *LNCS 3447. Genetic Programming: 8th European Conference, EuroGP 2005*, M. Keijzer, A. Tettamanzi, P. Collet, J. van Hemert, and M. Tomassini, Eds. Berlin, Germany: Springer-Verlag, 2005, pp. 291–300.

[7] Y. Shi and R. Eberhart, "Parameter selection in particle swarm optimization," in *LNCS 1447. Evolutionary Programming VII: 7th International Conference, Ep98*. Berlin, Germany: Springer-Verlag, 1998, pp. 591–600.

[8] R. Eberhart and Y. Shi, "Comparing inertia weights and constriction factors in particle swarm optimization," in *Proceedings of the 2000 IEEE Congress on Evolutionary Computation*. Piscataway, NJ, USA: IEEE Press, 2000, pp. 84–88.

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

[10] Y. Shi and R. Eberhart, "A modified particle swarm optimizer," in *Proceedings of the IEEE International Conference on Evolutionary Computation*. Piscataway, NJ, USA: IEEE Press, 1998, pp. 69–73.

[11] ——, "Empirical study of particle swarm optimization," in *Proceedings of the 1999 IEEE Congress on Evolutionary Computation*. Piscataway, NJ, USA: IEEE Press, 1999, pp. 1945–1950.

[12] Y.-L. Zheng, L.-H. Ma, L.-Y. Zhang, and J.-X. Qian, "On the convergence analysis and parameter selection in particle swarm optimization," in *Proceedings of the 2003 IEEE International Conference on Machine Learning and Cybernetics*. Piscataway, NJ, USA: IEEE Press, 2003, pp. 1802–1807.

[13] ——, "Empirical study of particle swarm optimizer with an increasing inertia weight," in *Proceedings of the 2003 IEEE Congress on Evolutionary Computation*. Piscataway, NJ, USA: IEEE Press, 2003, pp. 221–226.

[14] R. Eberhart and Y. Shi, "Tracking and optimizing dynamic systems with particle swarms," in *Proceedings of the 2001 IEEE Congress on Evolutionary Computation*. Piscataway, NJ, USA: IEEE Press, 2001, pp. 94–100.

[15] R. Mendes, J. Kennedy, and J. Neves, "The fully informed particle swarm: Simpler, maybe better," *IEEE Transactions on Evolutionary Computation*, vol. 8, no. 3, pp. 204–210, 2004.

[16] A. Ratnaweera, S. K. Halgamuge, and H. C. Watson, "Self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients," *IEEE Transactions on Evolutionary Computation*, vol. 8, no. 3, pp. 240–255, 2004.

[17] S. Janson and M. Middendorf, "A hierarchical particle swarm optimizer and its adaptive variant," *IEEE Transactions on Systems, Man and Cybernetics–Part B*, vol. 35, no. 6, pp. 1272–1282, 2005.

[18] C. K. Monson and K. D. Seppi, "Exposing origin-seeking bias in PSO," in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, H. G. Beyer *et al.*, Eds. New York, NY, USA: ACM Press, 2005, pp. 241–248.

[19] P. N. Suganthan, N. Hansen, J. J. Liang, K. Deb, Y.-P. Chen, A. Auger, and S. Tiwari, "Problem definitions and evaluation criteria for the CEC 2005 special session on real-parameter optimization," Nanyang Technological University, Singapore and IIT Kanpur, India, Tech. Rep. 2005005, 2005.

[20] M. A. Montes de Oca, T. Stützle, M. Birattari, and M. Dorigo, "Frankenstein's PSO: Complete data," 2007, Supplementary information page at http://iridia.ulb.ac.be/supp/IridiaSupp2007-002/.

[21] R. Mendes, "Population topologies and their influence in particle swarm performance," Ph.D. dissertation, Escola de Engenharia, Universidade do Minho, 2004.

[22] H. H. Hoos and T. Stützle, *Stochastic Local Search: Foundations and Applications*. San Francisco, CA, USA: Morgan Kaufmann, 2004.

[23] A. P. Engelbrecht, *Fundamentals of Computational Swarm Intelligence*. Chichester, West Sussex, England: John Wiley & Sons, 2005.

[24] J. Niehaus and W. Banzhaf, "More on computational effort statistics for genetic programming," in *LNCS 2610. Genetic Programming: 6th European Conference, EuroGP 2003*, C. Ryan, T. Soule, M. Keijzer, E. Tsang, R. Poli, and E. Costa, Eds. Berlin, Germany: Springer-Verlag, 2003, pp. 164–172.

[25] M. A. Montes de Oca, T. Stützle, M. Birattari, and M. Dorigo, "A comparison of particle swarm optimization algorithms based on run-length distributions," in *LNCS 4150. Ant Colony Optimization and Swarm Intelligence. 5th International Workshop, ANTS 2006*, M. Dorigo, L. M. Gambardella, M. Birattari, A. Martinoli, R. Poli, and T. Stützle, Eds. Berlin, Germany: Springer-Verlag, 2006, pp. 1–12.

[26] P. N. Suganthan, "Particle swarm optimiser with neighbourhood operator," in *Proceedings of the 1999 IEEE Congress on Evolutionary Computation*. Piscataway, NJ, USA: IEEE Press, 1999, pp. 1958–1962.