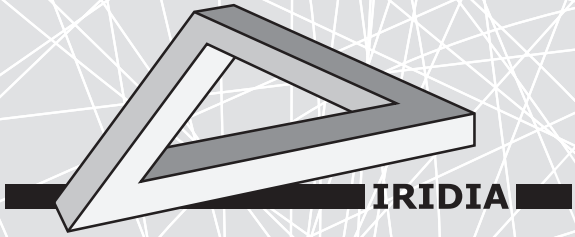


Université Libre de Bruxelles

Institut de Recherches Interdisciplinaires
et de Développements en Intelligence Artificielle

Incremental Social Learning in Particle Swarms

Marco A. Montes de Oca, Thomas Stützle,
Ken Van den Enden, and Marco Dorigo

IRIDIA – Technical Report Series

Technical Report No.
TR/IRIDIA/2009-002

January 2009

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/2009-002

Revision history:
TR/IRIDIA/2009-002.001 January 2009
TR/IRIDIA/2009-002.002 October 2009

Incremental Social Learning in Particle Swarms

Marco A. Montes de Oca[a] mmontes@ulb.ac.be
Thomas Stützle[a] stuetzle@ulb.ac.be
Ken Van den Enden[b] kvdenden@vub.ac.be
Marco Dorigo[a] mdorigo@ulb.ac.be

[a]IRIDIA, CoDE, Université Libre de Bruxelles, Brussels, Belgium
[b]Vrije Universiteit Brussel

October 2009

Abstract

Inspired by the phenomenon of social learning in animal societies, we have recently proposed an approach for facilitating the scalability of systems composed of multiple learning agents. The approach is called *incremental social learning* (ISL). In this paper, we show how ISL can be used in the context of population-based optimization algorithms, and in particular, in the context of particle swarm optimization (PSO). One of the algorithms that are the focus of this paper, called incremental particle swarm optimizer (IPSO), is a PSO algorithm with a growing population size in which new particles are initialized using a "social learning" rule that biases their initial position towards the best so far solution. The other algorithm, called incremental particle swarm optimizer with local search (IPSOLS), is an extension of IPSO in which particles also "learn individually", that is, they move in the search space by means of a local search procedure.

The goals of the work presented in this paper are three: *a)* to determine the probability density function induced by the "social learning" initialization rule in order to understand how and under which conditions that rule works, *b)* to empirically evaluate the performance of both IPSO and IPSOLS by comparing it with that of constant and variable population size PSO algorithms with and without local search, as well as with a random restart local search algorithm, and *c)* to empirically measure the extent to which the "social learning" initialization rule affects the performance of IPSO and IPSOLS on problems with different fitness distance correlations.

The results of the conducted experiments and of the analysis of the initialization rule show that IPSOLS is a competitive algorithm that is capable of finding good solutions very rapidly without compromising its global search capabilities.

1 Introduction

In a system composed of numerous learning agents, each agent not only must adapt to the features of the environment, but it also has to cope with behavioral

changes of other agents. This is an important problem in swarm intelligence research [4, 14] because learning is especially challenging when the number of agents involved is large [40, 49]. To tackle this problem, we have recently proposed an increasing population size approach that in some cases facilitates the scalability of systems composed of multiple learning agents [33]. This approach, which we call *incremental social learning* (ISL), is inspired by the phenomenon of social learning in animal societies [29].

In this paper, we show how ISL can be used in the context of population-based optimization algorithms, and in particular, in the context of particle swarm optimization (PSO) algorithms [27, 28, 18, 9, 42, 15]. Two facts justify the application of ISL to population-based optimization algorithms: First, just as the performance of a multiagent system is affected by the size of the population, population-based optimization algorithms usually exhibit a solution quality vs. speed tradeoff that depends, among other things, on the population size chosen [24, 47, 50, 32, 52]. Second, in the same way as learning agents in a multiagent system change often their behavior in order to test whether the last change is useful, individuals in a population-based optimization algorithm can be seen as changing position in an optimization problem's search space to test whether the new position is better. We decided to use PSO algorithms because they exhibit a solution quality vs. speed tradeoff that is amenable to the application of ISL: When a limited number of function evaluations are allowed, small populations obtain the best results. In contrast, when solution quality is the most important aspect, large populations work better [51, 34].

We propose two algorithms that result from the application of ISL to PSO algorithms. The first one is an incremental PSO algorithm (IPSO) [33], in which the population size grows over time. In this algorithm, when a new particle is added to the population, its position is initialized using a "social learning" rule that induces a bias toward the best particle. The second algorithm is an extension of IPSO (called IPSOLS) [36] in which "individual learning" is simulated by using a local search procedure.

The goals of the work presented in this paper are the following:

1. To determine the probability density function induced by the "social learning" initialization rule in order to understand how and under which conditions the initialization of new particles works (Section 4).

2. To empirically evaluate the performance of both IPSO and IPSOLS. The performance of the algorithms is compared with that of constant and variable population size PSO algorithms with and without local search, as well as with a random restart local search algorithm (Section 5).

3. To empirically measure the effect that the new particles initialization rule has on the performance of IPSO and IPSOLS on problems with different fitness distance correlations (Section 6).

We begin describing in detail the ISL framework (Section 2) and the ISL-based PSO algorithms (Section 3). Preliminary results on these topics can be found in [33] and [36]. After presenting the results for the three goals mentioned above, we conclude in Section 7.

Algorithm 1 Incremental social learning

```
/* Initialization */
t ← 0
Initialize environment E^t
Initialize population of agents X^t

/* Main loop */
while Stopping criteria not met do
  /* Agents are added according to a schedule */
  if Agent addition criterion is not met then
    X^{t+1} ← ilearn(X^t, E^t) /* Individual or default learning mechanism */
  else
    Create new agent a_new
    slearn(a_new, X^t) /* Social learning mechanism */
    X^{t+1} ← X^t ∪ {a_new}
  end if
  E^{t+1} ← update(E^t) /* Update environment */
  t ← t + 1
end while
```

2 Incremental Social Learning

Designing systems composed of numerous autonomous agents that at a collective level exhibit some desired behaviors is a very active research area. One approach consists in letting the agents that comprise the multiagent system learn by themselves the necessary individual behaviors. However, the interference caused by the coexistence of multiple simultaneously adapting agents, whose rewards depend on the group's performance, makes learning a very difficult task, especially when a large agent population is involved [40, 49].

The incremental social learning (ISL) [33] framework tackles the problem described above by adding to the population one agent at a time according to a schedule. The population is initially composed of a small number of agents in order to allow a faster learning than what would be possible with a larger population. Then, agents are incrementally added to the population, which makes it possible, in some cases, to allocate the optimal number of agents needed for solving a particular task. An agent that is added to the population learns socially from those that have been in the population for some time. This element of ISL is attractive because through social learning new agents acquire knowledge from more experienced ones, without incurring the costs of acquiring that knowledge individually [29]. Thus, ISL makes new agents save time that they can use to perform their tasks or to learn new things. After the inclusion of a new agent, the population needs to readapt to the new conditions, but the agents that are part of it do not need to learn everything from scratch. The algorithmic structure of the incremental social learning framework is outlined in Algorithm 1.

The environment and the population of agents are initialized before the main loop begins. If, according to the agent addition schedule, no agents are to be added, the agents in the population learn either individually or using another default learning mechanism, which could include elements of social or centralized

learning. The agent addition schedule controls the rate at which agents are added to the population. It also creates time delays that allow the agents in the population to learn from the interaction with the environment and with other agents. Before becoming part of the population, new agents learn socially from a subset of the already experienced agents. In Algorithm 1, the environment is updated explicitly in order to stress the fact that the environment might be dynamic (although it does not need to be so). In a real implementation, the environment can change at any time and not necessarily at the end of a training round.

The actual implementations of the individual (or default) and social learning mechanisms are independent of the incremental social learning framework outlined above. Both generic or application-specific mechanisms may be used.

Studies in simulation about the effects of different social learning mechanisms, like imitation, emulation, or stimulus enhancement, have been undertaken before [39]; in our work, however, we focus on the potential of learning socially in an incremental way to speed up learning in a large population.

3 Incremental Social Learning in Particle Swarm Optimization Algorithms

The framework described in Section 2 can be used not only in the context of multiagent systems, but also in the context of population-based optimization algorithms. This is possible if one interprets the individuals of a population-based optimization algorithm as the learning agents of a multiagent system. In this context, "learning", understood as a trial and error process, corresponds to the search mechanism used by the underlying optimization algorithm whereby new candidate solutions are tried and, eventually, better solutions are found. In this section, we describe two instantiations of the ISL framework using a PSO algorithm as the underlying search mechanism. We start by presenting a basic PSO algorithm, we continue with the description of the two ISL-based algorithms that we propose in this paper, and we finish by briefly reviewing related work. Preliminary studies and results on this topic can be found in [33, 36].

3.1 Particle Swarm Optimization

PSO is a population-based stochastic optimization technique used primarily to tackle continuous optimization problems. In PSO jargon, a *swarm* is a group of *particles* that move in the search space $\Theta \subseteq \mathbb{R}^n$ of an optimization problem $f : \Theta \rightarrow \mathbb{R}$ with the goal of finding an optimal solution.[1] The position of a particle represents a candidate solution to the problem under consideration. At each iteration, each particle is attracted toward its own previous best position and toward the best position found by the particles in its neighborhood, which is a subset of the swarm that is usually defined before the algorithm is run. Neighborhood relations define what is called a *population topology*, which can be seen as a graph $G = \{V, E\}$, where each vertex in V corresponds to a particle in the swarm and each edge in E establishes a neighbor relation between a pair

[1] Without loss of generality, in this paper we focus on the minimization case.

of particles. The rules that govern the movement of a particle i along the j-th dimension of the problem's search space are the following:

$$v_{i,j}^{t+1} = \chi \left[v_{i,j}^{t} + \varphi_1 U_1 \left(p_{i,j}^{t} - x_{i,j}^{t} \right) + \varphi_2 U_2 \left(l_{i,j}^{t} - x_{i,j}^{t} \right) \right] , \tag{1}$$

and

$$x_{i,j}^{t+1} = x_{i,j}^{t} + v_{i,j}^{t+1}, \tag{2}$$

where $v_{i,j}^{t}$ and $x_{i,j}^{t}$ are respectively the particle's velocity and position at time step t, $p_{i,j}^{t}$ is the particle's best position so far, $l_{i,j}^{t}$ is the best position found by the particle's neighbors, φ_1 and φ_2 are two parameters (called *acceleration coefficients*), U_1 and U_2 are two uniformly distributed random numbers in the range $[0, 1)$ that are generated at every iteration for each dimension, and χ is a parameter called *constriction factor* [10] that, if properly set, guarantees convergence in the search space (not necessarily to a local or global optimum).

3.2 Incremental Particle Swarm Optimizer

The first instantiation of the ISL framework in the context of PSO algorithms is an algorithm with a growing population that we call incremental particle swarm optimizer (IPSO) [33].

According to the ISL framework, every time a new agent is added to the population, it should learn socially from a subset of the more experienced agents. In the case of IPSO, this means that every time a new particle is added, it is initialized using information from particles that are already part of the population. This mechanism is implemented as an initialization rule that moves the new particle from an initial randomly generated position in the problem's search space to one that is closer to the position of a particle that serves as a "model" to imitate (hereafter referred to as model particle). The initialization rule used in IPSO, as applied to a new particle's j-th dimension, is the following:

$$x'_{new,j} = x_{new,j} + U \cdot (p_{model,j} - x_{new,j}), \tag{3}$$

where $x'_{new,j}$ is the new particle's updated position, $x_{new,j}$ is the new particle's original random position, $p_{model,j}$ is the model particle's previous best position and U is a uniformly distributed random number in the range $[0, 1)$. Once the rule is applied for each dimension, the new particle's previous best position is initialized to the point $\boldsymbol{x}'_{new}$ and its velocity is set to zero. The random number U is the same for all dimensions in order to ensure that the new particle's updated previous best position will lie somewhere along the direct attraction vector $\boldsymbol{p}_{model} - \boldsymbol{x}_{new}$. Using independent random numbers for each dimension would reduce the strength of the bias induced by the initialization rule because the resulting attraction vector would be rotated and scaled with respect to the direct attraction vector. Finally, the new particle's neighborhood, that is, the set of particles from which it will receive information in subsequent iterations, is generated at random, respecting the connectivity degree of the swarm's population topology.

The selection of the model particle can be done in several ways. In this paper, we focus on the behavior of the algorithm when the best particle is used as model. Preliminary results indicate that choosing the model particle at random does not produce significantly different results than using the best particle as model [33].

3.3 Incremental Particle Swarm Optimizer with Local Search

The incremental particle swarm optimizer with local search (IPSOLS) algorithm works in the same way as IPSO with the difference that in IPSOLS, particles not only move using the traditional PSO rules, but also by invoking a local search procedure [36]. In the context of the ISL framework, the local search procedure can be interpreted as a particle's "individual learning" ability because it allows a particle to improve its solution in the absence of any social influence. In this paper, the local search procedure employed in IPSOLS is the well-known Powell's direction set method [43] using Brent's technique [5] as the auxiliary line minimization algorithm, although other algorithms could be used. The quadratic convergence of Powell's direction set method can be very advantageous if the objective function is locally quadratic, which is not uncommon around a local optimum [46].

In IPSOLS, the local search procedure is called only when it is expected to be beneficial; that is, the local search procedure is invoked only when a particle's previous best position is not considered to be already in a local optimum. In [36], a particle invoked the local search procedure at every iteration, which could result in a waste of function evaluations. Checking whether running again the local search procedure may be beneficial or not is achieved by letting it return a value indicating whether it finished because of a very small difference between two solutions, or because the maximum number of iterations was reached. In the first case, it is assumed that the local search has converged to a local optimum and the particle does not invoke the procedure again because no further significant improvements are expected in that situation. In the second case, the particle may call the local search procedure again because further significant improvements can still be achieved. The two parameters of the local search procedure that control these exit criteria are the tolerance and the maximum number of iterations respectively. IPSO and IPSOLS are sketched in Algorithm 2. The differences between these two algorithms are indicated with boldface comments.

In the main loop, IPSO and IPSOLS share two common structures: the "horizontal social learning" and the "population growth and vertical social learning" parts. They are labeled in this way in order to distinguish between the two types of social learning simulated in these algorithms. Horizontal social learning occurs between agents of a same generation while vertical social learning occurs between agents of different generations [6].

3.4 Related Work

IPSO and IPSOLS are two PSO-based algorithms in which the population size changes during a run. IPSOLS is additionally a PSO–local search hybrid. In the remaining of this section, we briefly review related work on both topics.

3.4.1 Particle Swarm Optimization Algorithms with Dynamic Population Size

Population sizing has been studied within the field of evolutionary computation for many years. From that experience, it is now usually accepted that the population size in evolutionary algorithms should be proportional to the problem's difficulty [31]. The issue is that it is not uncommon to know little about

a problem's difficulty *a priori*. As a result, algorithms with dynamic population size have been proposed (see e.g. [1, 23, 3, 16, 2, 17, 19]). This research issue has just been recently addressed by the PSO community. For example, in [11], Coelho and de Oliveira adapt the population resizing mechanisms used in APGA [3] and PRoFIGA [16] for their use in PSO algorithms. Lanzarini *et al.* [30] proposed a method for varying the size of the population by assigning a maximum lifetime to groups of particles based on their performance and spatial distribution. In [7], the optimization process is divided into a number of periods at the end of which the population size changes. The decision of whether the population size should increase or decrease depends on a diversity measure. Finally, in [25], the authors adapt the swarm size based on the ability of the particles to improve their personal best solutions and the best-so-far solution.

Practically all strategies found in the literature consider the possibility of reducing the size of the population during an algorithm's run. The rationale is that large populations mean more function evaluations per iteration and thus, if the particles have converged, they can result in a waste of function evaluations. However, there are algorithms in which the population size is not decreased. In addition to our work, we can find the work of Auger and Hansen [2], in which the population size of a CMA-ES algorithm is doubled each time it is restarted.

3.4.2 Particle Swarm Optimization Algorithms Hybridized with Local Search Procedures

The idea of combining local search techniques with PSO algorithms comes partly from the observation that particles are attracted to their own and their neighbors' previous best positions. The underlying idea is that the better the attractors of a particle are, the higher the chances that a particle finds even better solutions. The goal of most hybrid algorithms is thus to accelerate the placement of the particles' previous best positions in good spots. For example, Chen *et al.* [8] combined a particle swarm algorithm with a hill-climbing local search procedure; Gimmler *et al.* [20] experiment with PSO-based hybrids using Nelder and Mead's simplex method as well as with Powell's direction set method, finding better results with Powell's method. Das *et al.* [13] also use Nelder and Mead's simplex method and propose the inclusion of an estimate of the local gradient into the particles' velocity update rule. Petalas *et al.* [41] report experiments with several local search-particle swarm combination schemes. In [37], Müller *et al.* describe a hybrid PSO–CMA-ES algorithm in which a full-fledged population-based algorithm (CMA-ES [22, 21]) is used as a local search procedure. The main difference between these approaches and our work is that ours is the first to explore the possible benefits of combining a variable population size with local search procedures in the context of PSO algorithms.

4 Analysis of the initialization rule: Probability density function

In IPSO and IPSOLS, the initial position of a new particle is generated through an initialization rule whose goal is to simulate the phenomenon of vertical social learning. This rule biases the position of the newly created particle towards the position of a particle that serves as an attractor or "model". In this section,

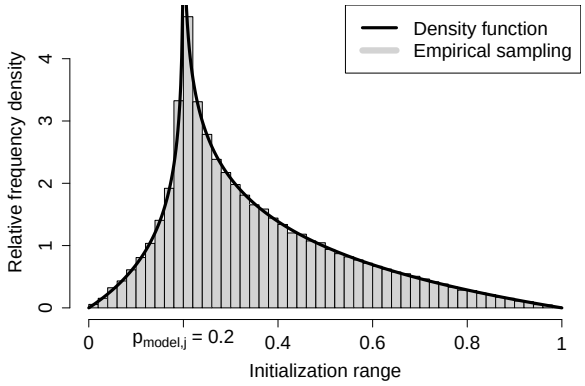


Figure 1: Probability density function induced by the initialization rule of new particles. In the figure, the attractor $p_{model,j} = 0.2$. The initialization range is $[0, 1)$. The figure shows both the analytical density function and the histogram obtained using Monte Carlo simulation (10^5 samples).

we present the exact probability density function induced by the initialization rule over the initialization range and describe some of its properties. This is an important step for understanding how and under which circumstances the initialization rule applied to new particles works, and what its effects on IPSO and IPSOLS are.

In IPSO and IPSOLS, the initialization rule applied to new particles (Eq. 3) is a function of two random variables: the uniformly distributed original position and a uniformly distributed random number in the range $[0, 1)$, which determines the strength of the attraction towards the position of the particle used as a model (the best particle in the swarm in our case). The model's position is, strictly speaking, also a random variable due to the fact that it is the result of a number of iterations of the PSO position-update mechanism. However, when the initialization rule is invoked, it can be taken as a constant.

The probability density function induced by the initialization rule for dimension j is the following (its derivation is shown in Appendix A):

$$f_{X_j}(x_j) = \frac{1}{x_{max,j} - x_{min,j}} \cdot \begin{cases} \ln \left| \frac{p_{model,j} - x_{min,j}}{p_{model,j} - x_j} \right| & \text{if } x_j < p_{model,j} \\ 0 & \text{if } x_j = p_{model,j} \\ \ln \left| \frac{p_{model,j} - x_{max,j}}{p_{model,j} - x_j} \right| & \text{if } x_j > p_{model,j} \,, \end{cases} \tag{4}$$

where $x_{min,j}$ and $x_{max,j}$ are the minimum and maximum limits of the initialization range over the problem's jth dimension and $x_{min,j} \leq x_j < x_{max,j}$. Figure 1 shows the exact density function and a histogram obtained using Monte Carlo simulation when the initialization range is $[0, 1)$ and $p_{model,j} = 0.2$.

Most of the samples concentrate around the model's position as desired. Note, however, that there is a nonzero probability of sampling regions far away from the model. This probability distribution offers a certain level of exploration-by-initialization which would be difficult to obtain with a normally distributed initialization around the model particle's position. The problem

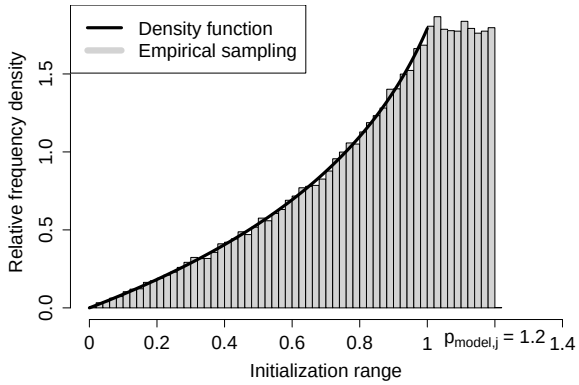


Figure 2: Probability density function induced by the initialization rule of new particles when the attractor lies outside the original initialization range. In the figure, the attractor $p_{model,j} = 1.2$. The initialization range is $[0, 1)$. The figure shows that the density function follows the analytical density function up to the limit of the original initialization range. The histogram obtained using Monte Carlo simulation (10^5 samples) shows the actual density function.

would be that setting the right value for the standard deviation would depend on the model particle's position. The probability density function induced by the new particles initialization rule is not symmetric except in the case $p_{model,j} = (x_{max,j} + x_{min,j})/2$. The expected value of a new particle's position is the following:

$$\begin{aligned} E(x'_{new,j}) &= E(x_{new,j}) + E(U)\,(p_{model,j} - E(x_{new,j})) \\ &= E(x_{new,j}) + \frac{1}{2}\,(p_{model,j} - E(x_{new,j})) \\ &= \frac{x_{max,j} + x_{min,j}}{4} + \frac{p_{model,j}}{2}\,. \end{aligned} \tag{5}$$

The analysis presented above is valid only if the attractor particle's position is within the range $[x_{min,j}, x_{max,j})$. If the attractor is outside the initialization range, the probability density function remains the same within the initialization range but it becomes a uniform distribution outside this range (see Figure 2).

Under these conditions, a new particle will follow the model only from one of its sides. The initialization rule is not able to position a new particle beyond the location of the attractor particle if this particle is outside the original initialization range. This is not a drawback because the initialization region is usually the region within which the optimum is expected to be.

5 Experimental Evaluation

In this section, we first describe the setup used to carry out our experiments. Next, we present and discuss the results of the empirical performance evaluation of the ISL-based PSO algorithms presented in Section 3.

5.1 Experimental Setup

The performance of IPSO and IPSOLS was compared to that of the following algorithms:

1. A traditional particle swarm optimization algorithm with constant population size (labeled PSO). This algorithm was included in the evaluation in order to measure the contribution of the incremental population component included in IPSO. Three population sizes of 10, 100 and 1000 particles were used.

2. A recently proposed PSO algorithm with dynamic population size (labeled EPUS) [25]. This algorithm increases the population size by one if the best-so-far solution is not improved during k consecutive iterations and if the current population size is not larger than a maximum limit. The new particle's position is equal to the result of a crossover operation on the personal best positions of two randomly selected particles. If the best-so-far solution is improved during k consecutive iterations, the worst particle of the swarm is removed from the swarm if the population size does not fall below a minimum limit after the operation. Finally, if the population size is equal to the maximum limit but the swarm is unable to improve the best-so-far solution during k consecutive iterations, the worst particle is replaced by a new one. In this paper, we do not use the mutation and solution sharing mechanisms described in [25] in order not to confound the effects of the variable population size with those of these operators. As suggested by the authors in [25], the parameter k was set to two. The minimum size of the swarm was set to one particle while the maximum was set to 1000 particles.

3. A hybrid particle swarm optimization algorithm with local search (labeled PSOLS). It is a constant population size particle swarm algorithm in which the particles' previous best positions undergo an improvement phase (via Powell's method) before the velocity update rule is applied. The local search is only applied when a particle's previous best position is not located in a local optimum, just as is done in the new version of IPSOLS (described in Section 3). Three population sizes of 5, 10, and 20 particles were used. With larger populations fewer iterations would have been executed, which would have prevented us from observing the effects that are due to the interaction of the PSO and local search components given the stopping criteria used. The reason is that given the number of function evaluations required by each invocation of the local search procedure and the maximum number of function evaluations allocated for each experiment (see below), a large population would essentially behave as a random restart local search (a method that was included in the comparison). PSOLS was included in the evaluation because, by comparing its performance to that of IPSOLS, we can measure the contribution of the incremental population component in combination with a local search procedure.

4. A hybrid algorithm (labeled EPUSLS) that combines EPUS with local search (Powell's method) in the same way as it is used by IPSOLS and PSOLS. This algorithm allows us to measure the relative performance

differences that may exist between a purely increasing and a variable population size approaches in combination with a local search procedure. The same parameter settings used for EPUS were used for EPUSLS.

5. A random restart local search algorithm (labeled RLS). Every time the local search procedure (Powell's method) converges, it is restarted from a newly generated random solution. The best solution found so far is considered to be the output of the algorithm. This algorithm was considered as a baseline for the evaluation of the effectiveness of the PSO component in PSOLS and IPSOLS.

In [33], IPSO proved to work well with a fast agent addition schedule. Accordingly, we used a fast particle addition schedule in which a new particle was added every iteration (this schedule was also used in IPSOLS). The maximum size of the swarm was set to 1000 particles. In all experiments, the best particle of the swarm served as a model to imitate.

All particle swarm-based algorithms (PSO, EPUS, IPSO, PSOLS, EPUSLS, and IPSOLS) were run with two population topologies: a fully connected topology, in which each particle is a neighbor to all others including itself, and the so-called ring topology, in which each particle has two neighbors apart from itself. In the incremental algorithms, the new particle is randomly placed within the topological structure. The parameter settings used for the PSO rules were: $\varphi_1 = \varphi_2 = 2.05$ and $\chi = 0.7298$.

Powell's method also has a number of parameters to be defined. The tolerance, which is used to stop the procedure once a very small difference between two solutions is detected, was set to 0.01. In case such a difference is not reached, the procedure is stopped after a maximum number of iterations. In our experiments, this parameter was set to 10. Other settings were explored but no significant differences in the results were found. Finally, the step size (used for the line minimization procedure) was set to 20% of the length of the corresponding search range. This value was found to give good results in [36].

All algorithms were run on a set of ten commonly used benchmark functions whose mathematical definition is shown in Table 1.

The results reported in this paper are based on statistics taken from 100 independent runs each of which was stopped whenever one of the following criteria was met: (i) 10^6 function evaluations have been performed, or (ii) the solution value is less than or equal to 10^{-15}. In each run, benchmark functions were randomly shifted within the specified range. The exceptions are Schwefel's and Step problems because their optima are not at the origin of the coordinate system. In all cases, we used the functions 100-dimensional versions, that is, $n = 100$. Bound constraints were enforced by putting variable values of candidate solutions on the corresponding bounds. This mechanism proved to be effective and easily applicable to both PSO and local search components.

5.2 Performance Evaluation Results

We look at the distribution of objective function values obtained after 10^4 and 10^6 function evaluations.[2] Figures 3 and 4 show a series of box plots on top

[2] In this paper's supplementary information web page [35], the reader can find the complete set of plots and data for different run durations that, for the sake of conciseness, are not included in the paper. Nevertheless, the main results are discussed in the text.

Table 1: Benchmark optimization problems

Name	Definition	Range
Ackley	$-20e^{-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}} - e^{\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)} + 20 + e$	$[-32,32]^n$
Griewank	$\frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n}\cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	$[-600,600]^n$
Expanded Schaffer	$ES(\boldsymbol{x}) = \sum_{i=1}^{n-1} S(x_i, x_{i+1}) + S(x_n, x_1)$, where $S(x,y) = 0.5 + \frac{\sin^2(\sqrt{x^2+y^2}) - 0.5}{(1+0.001(x^2+y^2))^2}$	$[-100,100]^n$
Rastrigin	$10n + \sum_{i=1}^{n}(x_i^2 - 10\cos(2\pi x_i))$	$[-5.12,5.12]^n$
Rosenbrock	$\sum_{i=1}^{n-1}[100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2]$	$[-30,30]^n$
Salomon	$1 - \cos\left(2\pi\sqrt{\sum_{i=1}^{n} x_i^2}\right) + 0.1\sqrt{\sum_{i=1}^{n} x_i^2}$	$[-100,100]^n$
Schwefel	$418.9829n + \sum_{i=1}^{n} -x_i \sin\left(\sqrt{\lvert x_i \rvert}\right)$	$[-500,500]^n$
Sphere	$\sum_{i=1}^{n} x_i^2$	$[-100,100]^n$
Step	$6n + \sum_{i=1}^{n} \lfloor x_i \rfloor$	$[-5.12,5.12]^n$
Weierstrass	$\sum_{i=1}^{n}\left(\sum_{k=1}^{k_{max}} a^k \cos\left(2\pi b^k (x_i + 0.5)\right)\right) - n\sum_{k=1}^{k_{max}}\left[a^k \cos\left(\pi b^k\right)\right]$ where a=0.5, b=3, k_{max}=20	$[-0.5,0.5]^n$

of which there are two rows of symbols. The lower row, made of + symbols, indicates in which cases a statistically significant difference exists between the marked algorithm and IPSO (in favor of IPSO). The upper row, made of × symbols, indicates in which cases a statistically significant difference exists between the marked algorithm and IPSOLS (in favor of IPSOLS). Statistically significant differences between all pairs of algorithms can be found in [35]. Significance was determined at the 5% level using a Wilcoxon test using Holm's correction method for multiple comparisons. Results are discussed paying particular attention to the two main components of the ISL-based algorithms: a variable population size and the use of a local search procedure.

5.2.1 Constant vs. variable population size

The performance of constant population size PSO algorithms without local search greatly depends on the value assigned to the population size parameter. The results obtained with the traditional PSO algorithm confirm the trade-off between solution quality and speed that we mentioned in the introduction. Swarms of 10 particles usually find better solutions than larger swarms up to around 10^4 function evaluations. Then, the swarms of 100 particles are typically the best performing after 10^5 function evaluations, and finally, after 10^6 function evaluations, the swarms with 1000 particles often return the best solutions. In contrast, the performance of EPUS and IPSO does not depend so much on the duration of a run. Both EPUS and IPSO compete with the best constant population size PSO algorithm at different run durations. This is a strong point in favor of PSO algorithms that vary the population size over time. However, the population size variation mechanism has an impact on performance. This can be seen in Table 2, which shows the number of times IPSO performs at least as well (in a statistical sense) than other PSO-based algorithms at different run durations. Twenty cases in total are considered, which are the result of summa-

Table 2: Number of Times IPSO Performs Either Better or no Worse[1] than Other PSO-based Algorithms at Different Run Lengths[2]

Evaluations	PSO-10^1	PSO-10^2	PSO-10^3	EPUS
10^2	13 (5)	18 (2)	18 (2)	18 (2)
10^3	1 (5)	19 (1)	20 (0)	17 (1)
10^4	8 (2)	13 (7)	20 (0)	10 (2)
10^5	12 (0)	2 (7)	18 (2)	8 (2)
10^6	19 (0)	8 (5)	15 (5)	9 (3)

[1] No worse cases shown in parenthesis.
[2] 10 problems × 2 topologies = 20 cases.

Table 3: Number of Times IPSOLS Performs Either Better or no Worse[1] than Other PSO-Local Search Hybrids at Different Run Lengths[2]

Evaluations	PSOLS-5	PSOLS-10	PSOLS-20	EPUSLS	RLS
10^2	0 (3)	0 (2)	0 (2)	0 (20)	0 (19)
10^3	13 (4)	13 (4)	13 (4)	0 (19)	0 (19)
10^4	14 (5)	15 (4)	15 (4)	8 (11)	8 (11)
10^5	15 (4)	14 (5)	14 (5)	15 (4)	14 (5)
10^6	12 (5)	10 (7)	12 (5)	14 (2)	14 (5)

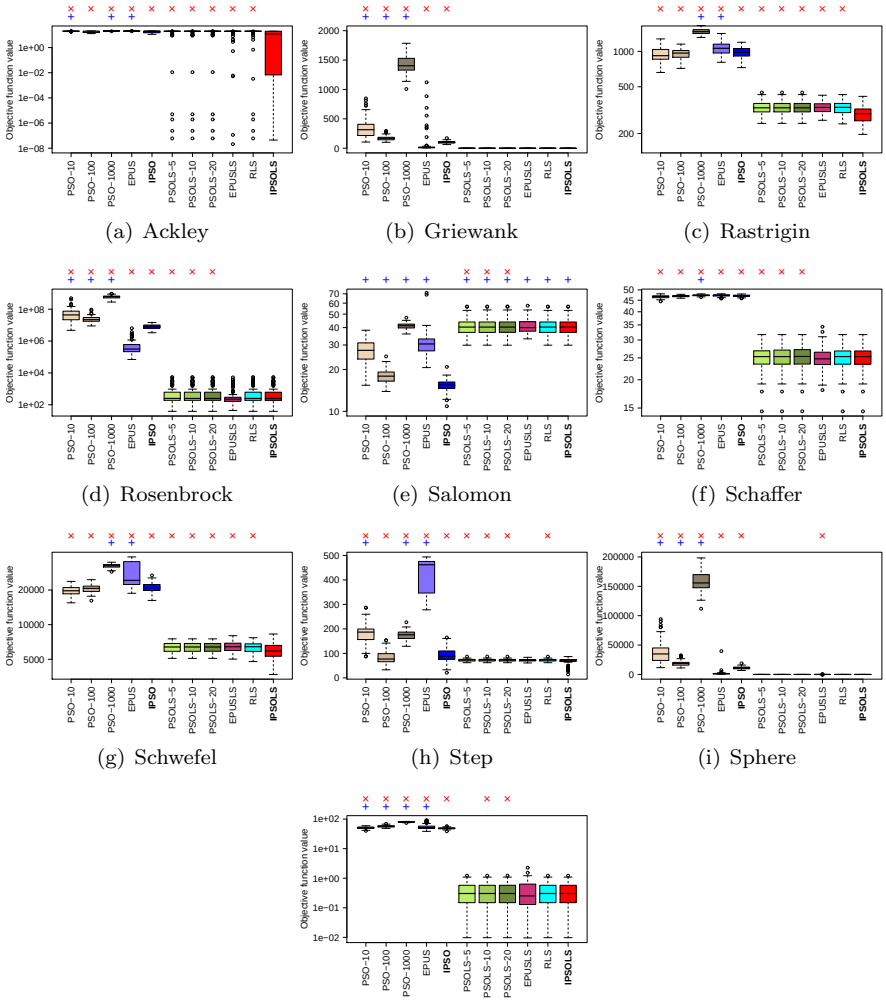
[1] No worse cases shown in parenthesis.
[2] 10 problems × 2 topologies = 20 cases.

rizing the results obtained on 10 benchmark functions using the fully connected and ring topologies. Figures 3 and 4 show the distribution of objective function values obtained with the fully connected topology after 10^4 and 10^6 function evaluations respectively.

Table 2 shows that IPSO dominates at least two of the constant population size PSO algorithms at different run durations. For runs of up to 10^5 function evaluations, the constant population size PSO algorithms with 100 and 1000 particles are dominated. For longer runs, the dominated algorithms are those with 10 and 1000 particles. This means that the performance of IPSO follows closely the performance of the best constant population size PSO algorithm. It should be noticed, however, that with a ring topology, the configuration with 100 particles is the best performing among the constant population size PSO algorithms (see [35]). Regarding the difference in performance due to differences in the mechanism for varying the population size, IPSO dominates EPUS for short runs. For long runs, IPSO performs better or not worse than EPUS in half of the cases.

5.2.2 Use vs. no use of local search

The local search component plays a major role on the performance of the algorithms that include it. Figures 3 and 4 show that the quality of the solutions obtained with the algorithms that include a local search procedure is typically higher than the one of the solutions obtained with the algorithms that do not. The only case in which the algorithms without a local search component dominate is when solving the Salomon function. Table 3 shows the number of times IPSOLS performs either better or no worse (in a statistical sense) than other PSO-local search hybrids at different run durations.



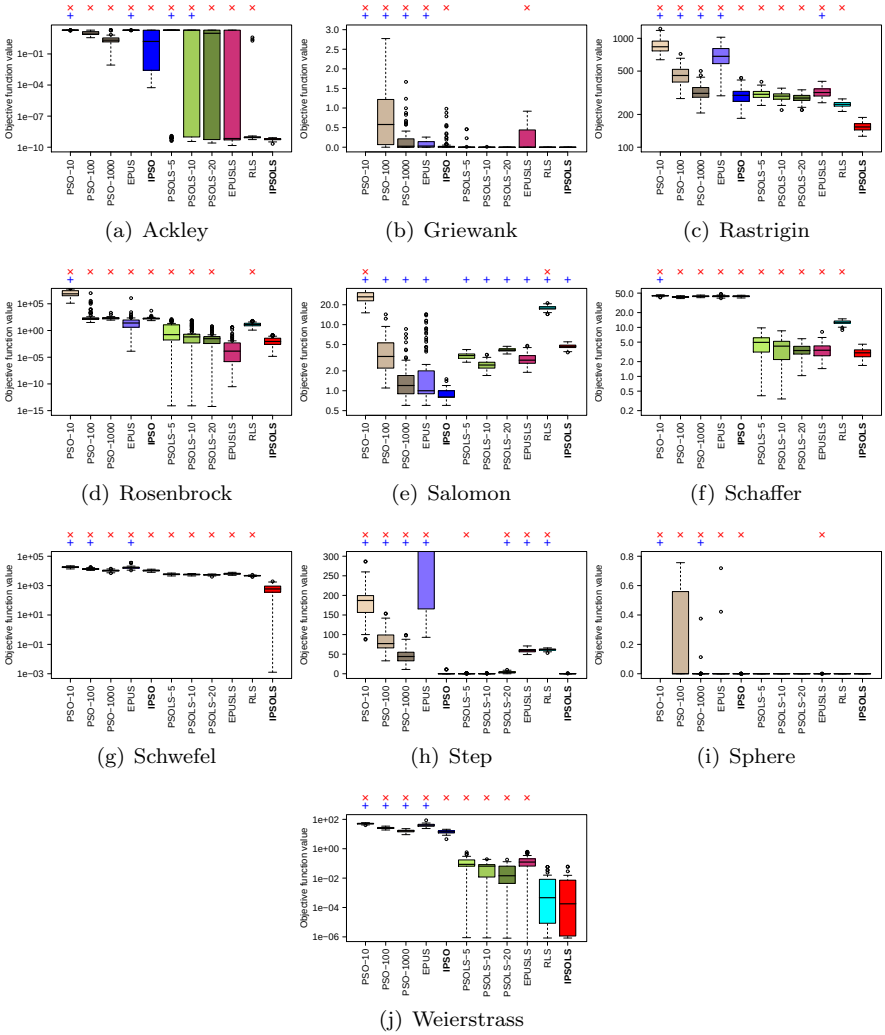
(a) Ackley (b) Griewank (c) Rastrigin

(d) Rosenbrock (e) Salomon (f) Schaffer

(g) Schwefel (h) Step (i) Sphere

(j) Weierstrass

Figure 3: The box plots show the distribution of the solution quality obtained with the compared algorithms after 10^4 function evaluations. These results correspond to the case in which a fully-connected topology is used with all particle swarm-based algorithms. A symbol on top of a box plot denotes a statistically significant difference at the 5% level between the results obtained with the indicated algorithm and those obtained with IPSO (in favor of IPSO, marked with a + symbol) or with IPSOLS (in favor of IPSOLS, marked with a $\times$ symbol).



(a) Ackley (b) Griewank (c) Rastrigin

(d) Rosenbrock (e) Salomon (f) Schaffer

(g) Schwefel (h) Step (i) Sphere

(j) Weierstrass

Figure 4: The box plots show the distribution of the solution quality obtained with the compared algorithms after 10^6 function evaluations. These results correspond to the case in which a fully-connected topology is used with all particle swarm-based algorithms. In the Griewank and Sphere functions, the solution values obtained with the traditional PSO algorithm with 10 particles are so much higher than those obtained with the other algorithms that its box plot does not appear. A symbol on top of a box plot denotes a statistically significant difference at the 5% level between the results obtained with the indicated algorithm and those obtained with IPSO (in favor of IPSO, marked with a + symbol) or with IPSOLS (in favor of IPSOLS, marked with a × symbol).

The difference in performance between the constant population size algorithms that we observed in the case when they do not use local search, almost disappears when local search is used. For runs of some hundreds of function evaluations, IPSOLS performs no better than any other hybrid PSO–local search algorithms (see first row in Table 3). This is because Powell's method has to perform at least n line searches (n is the number of dimensions of the problem) before making any significant improvement and because PSOLS first explores and then invokes the local search component. However, for longer runs, IPSOLS clearly dominates all other hybrid algorithms, including EPUSLS.

IPSOLS is an algorithm that calls repeatedly a local search procedure from different initial solutions. In this respect, IPSOLS works in the same way as a random restart local search algorithm (RLS). However, the main difference between RLS and IPSOLS resides in the way the new initial solutions are chosen. In RLS this choice is made at random; in IPSOLS it is the result of the application of the PSO rules. Thus, a comparison of the results obtained with these two algorithms can give us an indication of the impact of the PSO component in IPSOLS. The results presented in Figure 4 indicate that IPSOLS outperforms RLS in all problems except on Griewank, Sphere and Weierstrass. In the two first cases, the local search procedure alone is able to make such a significant progress that indeed it is able to find the minimum of the Sphere function (with a solution value that is less than or equal to 10^{-15}, one of our stopping criteria). In the case of the Griewank function, IPSOLS solves the problem with a population of around 3 particles (data shown in [35]). Thus, IPSOLS's behavior is essentially equivalent to that of RLS when its population does not grow significantly (see, for example, Figure 6).

As examples of the behavior of the algorithms over time, consider the results shown in Figures 5 and 6, which correspond to the solution development over the number of function evaluations obtained by a selection of the compared algorithms on the Rastrigin and Sphere functions. In these figures, we also show the average population size growth over time in IPSO, EPUS, EPUSLS and IPSOLS.

In some cases, as was noted before, IPSOLS is outperformed by other algorithms for short runs (in our case, runs between 10^2 and 10^3 function evaluations). However, IPSOLS improves dramatically once the population size starts growing, as exemplified by the plots in Figure 5 in which IPSOLS starts differentiating from RLS, EPUSLS and PSOLS after approximately 5,000 function evaluations. IPSOLS improves rapidly once the local search procedure begins to make progress, as seen in Figure 6. In this last figure, it can be seen how the populations in IPSOLS and EPUSLS, when they are applied to the Sphere function, do not grow. This explains the equivalence of IPSOLS, EPUSLS and RLS on problems solvable by local search alone. In most cases, the population growth in IPSOLS is independent of the population topology used (data shown in [35]).

From a practitioner's point of view, there are at least two advantages of using IPSOLS over a hybrid PSO algorithm: (i) IPSOLS does not require the practitioner to fix the population size in advance hoping to have chosen the right size for his/her problem, and (ii) IPSOLS is more robust to the choice of the population's topology. The difference between the results obtained through IPSOLS with a fully-connected and with a ring topology are smaller than the differences observed in the results obtained through the hybrid algorithms (data

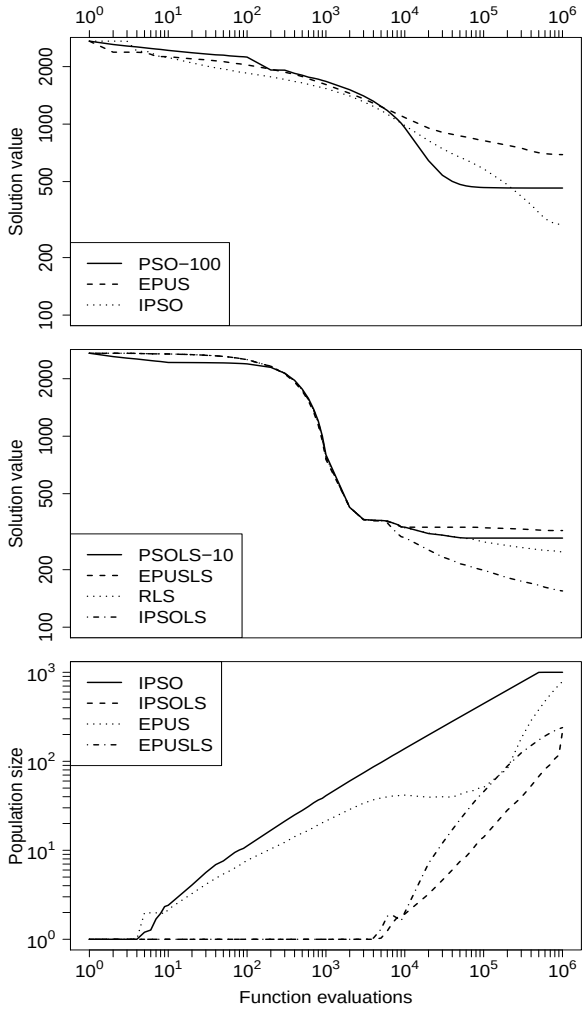


Figure 5: Solution development over time obtained by a selection of the compared algorithms (PSO-based algorithms using a fully-connected topology) on the Rastrigin function. Results without local search (upper plot). Results with local search (middle plot). Average population size growth in IPSO, EPUS, EPUSLS and IPSOLS (lower plot).

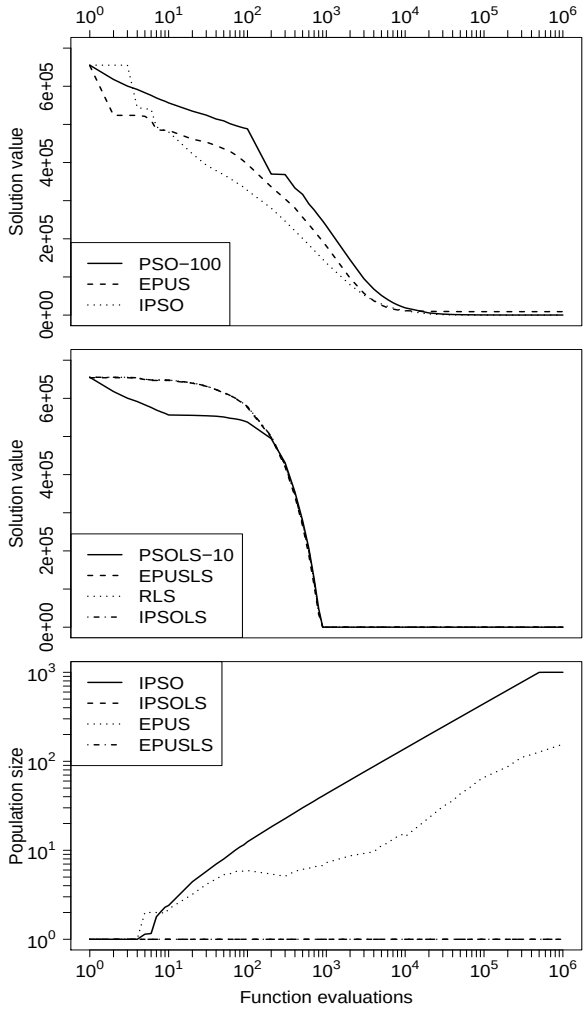


Figure 6: Solution development over time obtained by a selection of the compared algorithms (PSO-based algorithms using a fully-connected topology) on the Sphere function. Results without local search (upper plot). Results with local search (middle plot). Average population size growth in IPSO, EPUS, EPUSLS and IPSOLS (lower plot).

Table 4: Amplitudes used in the Rastrigin function to obtain specific fitness distance correlations (FDCs).

Amplitude	FDC	Amplitude	FDC
155.9111	≈ 0.001	13.56625	≈ 0.6
64.56054	≈ 0.1	10.60171	≈ 0.7
40.09456	≈ 0.2	7.938842	≈ 0.8
28.56419	≈ 0.3	5.21887	≈ 0.9
21.67512	≈ 0.4	0.0	≈ 0.999
16.95023	≈ 0.5	-	-

shown in [35]).

6 How useful is learning socially on initialization?

Finally, we present the results of an experiment aimed at measuring the effects of using the "vertical social learning" rule (Eq. 3) in IPSO as well as in IPSOLS.

In this section, we measure the extent to which the initialization rule applied to new particles affects the quality of the solution obtained after a certain number of function evaluations with respect to a random initialization. For this purpose, IPSO and IPSOLS are run with initialization mechanisms that induce a bias of different strength towards the best particle of the swarm. These mechanisms are (in increasing order of bias strength): (i) random initialization, (ii) the initialization rule as defined in Eq. 3 (labeled as "weak bias"), and (iii) the same rule as defined in Eq. 3, but with the random number U drawn from a uniform distribution in the range $[0.95, 1)$ (labeled as "strong bias").

The experiments are carried out on problems derived from the Rastrigin function, each of which has different fitness distance correlation (FDC) [26]. Since the initialization rule used in IPSO and IPSOLS implicitly assumes that good solutions are close to each other, the hypothesis is that the performance of the algorithms degrades as the problem's FDC approaches zero and that the rate of performance degradation is faster with stronger initialization bias.

The Rastrigin function, whose n-dimensional formulation is $nA+\sum_{i=1}^{n}(x_i^2 - A\cos(\omega x_i))$, can be thought of as a parabola with a superimposed (co)sine wave with an amplitude and frequency controlled by parameters A and ω respectively. By changing the values of A and ω one can obtain a whole family of problems. In our experiments, we set $\omega = 2\pi$ as is usually done, and tuned the amplitude A to obtain functions with specific FDCs. Other settings are the search range and the dimensionality of the problem, which we set to $[-5.12, 5.12]^n$ and $n = 100$, respectively. The amplitude and the resulting FDCs (estimated using 10^4 uniformly distributed random samples over the search range) are shown in Table 4.

IPSO and IPSOLS with the three initialization rules described above were run 100 times on each problem for up to 10^6 function evaluations. To measure the magnitude of the effect of using one or another initialization scheme, we use

Cohen's d statistic [12], which for the case of two samples is defined as follows:

$$d = \frac{\hat{\mu_1} - \hat{\mu_2}}{\sigma_{pooled}}, \tag{6}$$

with

$$\sigma_{pooled} = \sqrt{\frac{(n_1 - 1)\hat{\sigma_1}^2 + (n_2 - 1)\hat{\sigma_2}^2}{n_1 + n_2 - 2}}, \tag{7}$$

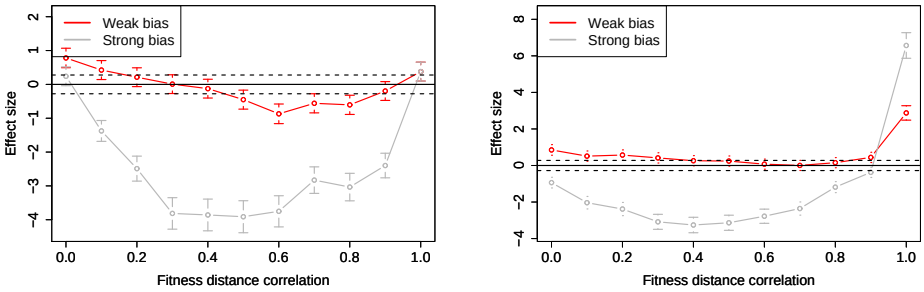
where $\hat{\mu_i}$ and $\hat{\sigma_i}$ are the mean and standard deviation of sample i, respectively [38].

As an effect size index, Cohen's d statistic measures the difference between the mean responses of a treatment and control groups expressed in standard deviation units [48]. The treatment group is, in our case, the set of solutions obtained with IPSO and IPSOLS using the initialization rule that biases the position of a new particle toward the best particle of the swarm. The control group is the set of solutions obtained with IPSO and IPSOLS when new particles are initialized completely at random. (Since in our case the lower the solution value the better, the order of the operands in the subtraction is reversed.) An effect size value of 0.8, for example, means that the average solution found using the particles' initialization rule is better than 79% of the solutions found without using it. The practical significance that the value associated to an effect has depends, of course, on the situation under consideration; however, a value of 0.8 can already be considered a large effect [12].

The observed effect sizes with 95% confidence intervals on the solution quality obtained with IPSO and IPSOLS after 10^6 function evaluations are shown in Figure 7.

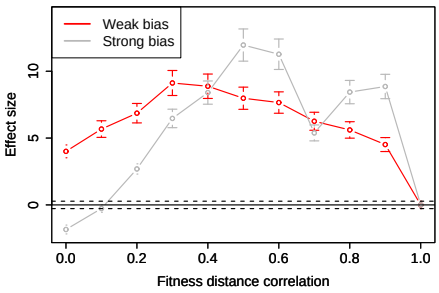
In IPSO, the effects of using the new particles initialization rule are very different from the ones in IPSOLS. In IPSO, the weak bias initialization rule produces better results than random initialization only in two cases: (i) when the problem's FDC is almost equal to one and the algorithm is run with a ring topology, and (ii) when the problem's FDC is close to zero irrespective of the population topology used. In all other cases, the weak bias initialization rule produces results similar to those obtained with a random initialization. The strong bias initialization rule reports benefits only in the case of a high FDC and a ring topology. In all other cases, it results in significantly worse solutions than the ones obtained with a random initialization. The worst performance of IPSO with the strong bias initialization rule is obtained when the problem's FDC is in the range (0.3,0.6). This behavior is a consequence of the new particle's velocity being equal to zero, which effectively reduces the particle's initial exploratory behavior. Setting the new particle's initial velocity to a value different from zero reduces the effect of the initialization bias because it would immediately make the particle move to a quasi-random position right after the first iteration of the algorithm's PSO component. The performance observed when the problem's FDC is close to zero is the result of the fact that with a fixed search range and a high amplitude, the parabolic component of the Rastrigin function has a much lower influence and many of the locally optimal solutions are of the same quality, thus moving close to or away from already good solutions has no major impact on the solution quality.

While the effect in IPSO is positive only in a few cases, in IPSOLS the effect size is not only positive in almost all cases but it is also large. IPSOLS with the

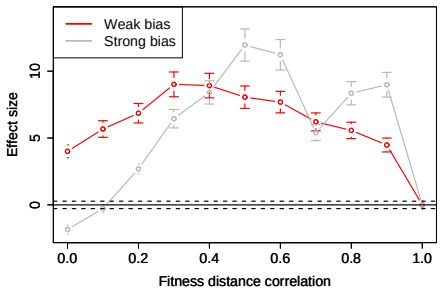


(a) IPSO, Fully-connected topology

(b) IPSO, Ring topology



(c) IPSOLS, Fully-connected topology



(d) IPSOLS, Ring topology

Figure 7: Effect size of the new particles initialization rule, as measured using Cohen's d statistic with 95% confidence intervals (indicated by either error bars or dashed lines), on the solution quality obtained with IPSO and IPSOLS after 10^6 function evaluations. Two bias strengths are tested: (i) weak bias and (ii) strong bias. The reference results (line at zero) are obtained with a random initialization.

weak bias initialization rule produces significantly better solutions than with a random initialization in all but one case, which corresponds to the situation where the problem's FDC is close to one. When the strong bias initialization rule is used, IPSOLS produces better solutions than with random initialization when the problem's FDC is in the range (0.1, 1.0). In the range (0.4,1.0), the solutions obtained with a strong bias initialization rule are better than or equal to those obtained with a weak bias initialization rule.

The fact that in IPSOLS using a random initialization of new particles is effectively the same as initializing them with a bias toward the location of the best particle of the swarm when the problem's FDC is almost one can be easily explained: under these circumstances IPSOLS is a local search algorithm that starts from a single solution. Since a local search algorithm alone can solve a problem with an FDC close to one, no population growth occurs and the initialization rule is never used. The degradation of the effect size as the problem's FDC decreases can be observed in the range (0.0,0.5) for the strong bias initialization rule, and in the range (0.0, 0.3) for the weak bias initialization rule. As hypothesized, the rate of degradation is faster when using a strong bias.

In summary, the use of the weak bias initialization rule in IPSOLS, which is the originally proposed vertical social learning rule, provides significant benefits over random initialization on the family of problems we examined with a fitness distance correlation in the range (0,1).

7 Conclusions

In this paper, we have shown how the incremental social learning (ISL) framework, originally designed for facilitating the scalability of systems composed of multiple learning agents, can be used for enhancing the performance of population-based optimization algorithms. In particular, we focused our attention on particle swarm optimization (PSO) algorithms because of the solution quality vs. speed tradeoff that they exhibit. We analyzed and empirically evaluated two algorithms that are the result of combining ISL and PSO ideas. The first one, called incremental particle swarm optimizer (IPSO), is a PSO algorithm with a growing population size, in which new particles are initialized biasing their initial position towards the best so far solution. The second algorithm, called incremental particle swarm optimizer with local search (IPSOLS), is an extension of IPSO which implements "individual learning" through a local search procedure.

In IPSOLS, which is the most competitive of these two algorithms, the population size is increased if the optimization problem at hand cannot be solved satisfactorily by local search alone. This can happen either because the local search converges to a local optimum or because the maximum number of iterations allocated to the local search procedure is reached. When a new particle is added to the population, a sort of "vertical social learning" is simulated. This approach is effective because if the problem is not so difficult, it may be solved by local search alone. If the problem is difficult, a growing population size will offer a better tradeoff between solution quality and speed than a constant population size PSO-based algorithm. The result is that, in effect, IPSOLS can adapt to the features of the objective function. Extended experimentation showed that not only increasing the population size is beneficial but also that the

biased initialization of new particles results in improved performance. The results presented in the paper show that the effects of simulating the phenomenon of vertical social learning are positive on problems of positive fitness distance correlation.

Future work includes investigating effective methods for dealing with bound and other kind of constraints, testing whether invoking the local search procedure on the particles' current positions can improve the algorithms' performance, and experimenting with other local search procedures such as Powell's NEWUOA or BOBYQA algorithms [44, 45]. Lastly, more research is needed in order to know whether ISL can be successfully used with other population-based optimization techniques.

Acknowledgments

This work was supported by the META-X project, an *Action de Recherche Concertée* funded by the Scientific Research Directorate of the French Community of Belgium. M. A. Montes de Oca acknowledges partial support from the 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 F.R.S-FNRS of the French Community of Belgium of which they are a Research Associate and a Research Director, respectively.

Appendix A. New particles initialization rule exact probability density function

The position of a newly added particle in IPSO and IPSOLS can be seen as a random variable Z which is a function of two independent continuous random variables X and Y. X is a uniformly distributed random variable in the complete initialization range $[x_{min}, x_{max})$, while Y is a uniformly distributed random variable in the range $[0, 1)$. Z is defined as follows:

$$Z = X + Y(c - X)\,, \tag{A.1}$$

where $x_{min} \leq c < x_{max}$ is a constant representing the location of the attractor particle.

The distribution function F_Z of Z is given by

$$F_Z(a) = P(Z \leq a) = \iint\limits_{(x,y):x+y(c-x)\leq a} f(x,y)\,dx\,dy\,, \tag{A.2}$$

where $f(x, y)$ is the joint probability distribution of X and Y.

Since X and Y are independent, we have that

$$f(x,y) = f_X(x) f_Y(y) = \frac{1}{x_{max} - x_{min}}\,, \tag{A.3}$$

where f_X and f_Y are the marginal probability functions of X and Y respectively. This holds for $x_{min} \leq x < x_{max}$ and $0 \leq y < 1$.

Using A.3 and considering that $y = \frac{a-x}{c-x}$, we can rewrite A.2 as follows

$$\begin{aligned} F_Z(a) &= \frac{1}{x_{max} - x_{min}} \int_{x_{min}}^{x_{max}} y \, dx \\ &= \frac{1}{x_{max} - x_{min}} \int_{x_{min}}^{x_{max}} \frac{a-x}{c-x} \, dx \,. \end{aligned} \tag{A.4}$$

Eq. A.4 must be solved in two parts: when $x_{min} \leq x \leq a < c$ and when $c < a \leq x < x_{max}$. In the special case when $x = c$, $F_Z(a) = c/(x_{max} - x_{min})$ (see Eq. A.1).

When $x_{min} \leq x \leq a < c$, we obtain

$$\begin{aligned} F_Z(a) &= \frac{1}{x_{max} - x_{min}} \int_{x_{min}}^{a} \frac{a-x}{c-x} \, dx \\ &= \frac{1}{x_{max} - x_{min}} \left[a + (a-c) \ln \left| \frac{c - x_{min}}{c-a} \right| \right] . \end{aligned} \tag{A.5}$$

When $c < a \leq x < x_{max}$, we obtain

$$\begin{aligned} F_Z(a) &= \frac{1}{x_{max} - x_{min}} \left[1 - \int_{a}^{x_{max}} \frac{a-x}{c-x} \, dx \right] \\ &= \frac{1}{x_{max} - x_{min}} \left[a + (a-c) \ln \left| \frac{c - x_{max}}{c-a} \right| \right] . \end{aligned} \tag{A.6}$$

Hence the probability density function f_Z of Z is given by

$$f_Z(z) = \frac{d}{dz} F_Z(z) = \frac{1}{x_{max} - x_{min}} \begin{cases} \ln \left| \frac{c - x_{min}}{c-z} \right| & \text{if } z < c \\ 0 & \text{if } z = c \\ \ln \left| \frac{c - x_{max}}{c-z} \right| & \text{if } z > c \end{cases} . \tag{A.7}$$

References

[1] Jaroslaw Arabas, Zbigniew Michalewicz, and Jan J. Mulawka. GAVaPS – A genetic algorithm with varying population size. In *Proceedings of the IEEE Conference on Evolutionary Computation*, pages 73–78, Piscataway, NJ, 1994. IEEE Press.

[2] Anne Auger and Nikolaus Hansen. A restart CMA evolution strategy with increasing population size. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2005)*, pages 1769–1776, Piscataway, NJ, 2005. IEEE Press.

[3] Thomas Bäck, A. E. Eiben, and Nikolai A. L. van der Vaart. An empirical study on GAs "without parameters". In *LNCS 1917. Parallel Problem Solving from Nature - PPSN VI, 6th International Conference*, pages 315–324, Berlin, Germany, 2000. Springer-Verlag.

[4] Eric Bonabeau, Marco Dorigo, and Guy Theraulaz. *Swarm Intelligence: From Natural to Artificial Systems.* Oxford University Press, New York, NY, USA, 1999.

[5] Richard P. Brent. *Algorithms for Minimization Without Derivatives*. Prentice-Hall, Englewood Cliffs, NJ, 1973.

[6] Luigi Luca Cavalli-Sforza and Marcus W. Feldman. *Cultural Transmission and Evolution. A Quantitative Approach*. Princeton University Press, Princeton, NJ, 1981.

[7] DeBao Chen and ChunXia Zhao. Particle swarm optimization with adaptive population size and its application. *Applied Soft Computing*, 9(1):39–48, 2009.

[8] Junying Chen, Zheng Qin, Yu Liu, and Jiang Lu. Particle swarm optimization with local search. In *Proceedings of the International Conference on Neural Networks and Brain (ICNN&B 2005)*, pages 481–484, Piscataway, NJ, 2005. IEEE Press.

[9] Maurice Clerc. *Particle Swarm Optimization*. ISTE, London, UK, 2006.

[10] Maurice Clerc and James Kennedy. The particle swarm–explosion, stability, and convergence in a multidimensional complex space. *IEEE Transactions on Evolutionary Computation*, 6(1):58–73, 2002.

[11] André L. V. Coelho and Daniel G. de Oliveira. Dynamically tuning the population size in particle swarm optimization. In *Proceedings of the ACM Symposium on Applied Computing (SAC'08)*, pages 1782–1787, New York, NY, 2008. ACM Press.

[12] Jacob Cohen. *Statistical Power Analysis for the Behavioral Sciences*. Lawrence Earlbaum Associates, Hillsdale, NJ, 1988.

[13] Sanjoy Das, Praveen Koduru, Min Gui, Michael Cochran, Austin Wareing, Stephen M. Welch, and Bruce R. Babin. Adding local search to particle swarm optimization. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2005)*, pages 428–433, Piscataway, NJ, 2006. IEEE Press.

[14] Marco Dorigo and Mauro Birattari. Swarm intelligence. *Scholarpedia*, 2(9):1462, 2007.

[15] Marco Dorigo, Marco A. Montes de Oca, and Andries P. Engelbrecht. Particle swarm optimization. *Scholarpedia*, 3(11):1486, 2008.

[16] A. E. Eiben, E. Marchiori, and V. A. Valkó. Evolutionary algorithms with on-the-fly population size adjustment. In *LNCS 3242. Parallel Problem Solving from Nature - PPSN VIII, 8th International Conference*, pages 41–50, Berlin, Germany, 2004. Springer-Verlag.

[17] A.Ẽ. Eiben, M.C̃. Shut, and A.R̃. de Wilde. Is self-adaptation of selection pressure and population size possible? – A case study. In *LNCS 4193. Parallel Problem Solving from Nature - PPSN IX, 9th International Conference*, pages 900–909, Berlin, Germany, 2006. Springer-Verlag.

[18] Andries P. Engelbrecht. *Fundamentals of Computational Swarm Intelligence*. John Wiley & Sons, Chichester, UK, 2005.

[19] Carlos Fernandes and Agostinho Rosa. Self-regulated population size in evolutionary algorithms. In *LNCS 4193. Parallel Problem Solving from Nature - PPSN IX, 9th International Conference*, pages 920–929, Berlin, Germany, 2006. Springer-Verlag.

[20] Jens Gimmler, Thomas Stützle, and Thomas E. Exner. Hybrid particle swarm optimization: An examination of the influence of iterative improvement algorithms on performance. In Marco Dorigo et al., editors, *LNCS 4150. Ant Colony Optimization and Swarm Intelligence. 5th International Workshop, ANTS 2006*, pages 436–443, Berlin, Germany, 2006. Springer-Verlag.

[21] N. Hansen and S. Kern. Evaluating the CMA evolution strategy on multimodal test functions. In Xin Yao et al., editors, *LNCS 3242. Parallel Problem Solving from Nature - PPSN VIII*, pages 282–291, Berlin, Germany, 2004. Springer-Verlag.

[22] N. Hansen, S. D. Müller, and P. Koumoutsakos. Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES). *Evolutionary Computation*, 11(1):1–18, 2003.

[23] Georges R. Harik and Fernando G. Lobo. A parameter-less genetic algorithm. In Wolfgang Banzhaf et al., editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 1999)*, pages 258–265, San Francisco, CA, 1999. Morgan Kaufmann.

[24] Jun He and Xin Yao. From an invividual to a population: An analysis of the first hitting time of population-based evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 6(5):495–511, 2002.

[25] Sheng-Ta Hsieh, Tsung-Ying Sun, Chan-Cheng Liu, and Shang-Jeng Tsai. Efficient population utilization strategy for particle swarm optimizer. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 39(2):444–456, 2009.

[26] Terry Jones and Stephanie Forrest. Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In *Proceedings of the 6th International Conference on Genetic Algorithms*, pages 184–192, San Francisco, CA, 1995. Morgan Kaufmann.

[27] James Kennedy and Russell Eberhart. Particle swarm optimization. In *Proceedings of IEEE International Conference on Neural Networks*, pages 1942–1948, Piscataway, NJ, 1995. IEEE Press.

[28] James Kennedy and Russell Eberhart. *Swarm Intelligence*. Morgan Kaufmann, San Francisco, CA, 2001.

[29] Kevin N. Laland. Social learning strategies. *Learning & Behavior*, 32(1):4–14, 2004.

[30] Laura Lanzarini, Victoria Leza, and Armando De Giusti. Particle swarm optimization with variable population size. In R. Goebel et al., editors, *LNAI 5097. Proceedings of the International Conference on Artificial Intelligence and Soft Computing. ICAISC 2008*, pages 438–449, Berlin, Germany, 2008. Springer-Verlag.

[31] Fernando G. Lobo and Claudio F. Lima. *Parameter Setting in Evolutionary Algorithms*, volume 54/2007 of *Studies in Computational Intelligence*, chapter Adaptive Population Sizing Schemes in Genetic Algorithms, pages 185–204. Springer, Berlin, Germany, 2007.

[32] R. Mallipeddi and P.Ñ. Suganthan. Empirical study on the effect of population size on differential evolution. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 3663–3670, Piscataway, NJ, 2008. IEEE Press.

[33] Marco A. Montes de Oca and Thomas Stützle. Towards incremental social learning in optimization and multiagent systems. In W. Rand et al., editors, *Workshop on Evolutionary Computation and Multiagent Systems Simulation of the Genetic and Evolutionary Computation Conference (GECCO 2008)*, pages 1939–1944, New York, NY, 2008. ACM Press.

[34] Marco A. Montes de Oca, Thomas Stützle, Mauro Birattari, and Marco Dorigo. Frankenstein's PSO: A composite particle swarm optimization algorithm. *IEEE Transactions on Evolutionary Computation*, 13(5):1120–1132, 2009.

[35] Marco A. Montes de Oca, Thomas Stützle, Ken Van den Enden, and Marco Dorigo. Incremental social learning in particle swarms: Complete results, 2009. Supplementary information page at `http://iridia.ulb.ac.be/supp/IridiaSupp2009-001/`.

[36] Marco A. Montes de Oca, Ken Van den Enden, and Thomas Stützle. Incremental particle swarm-guided local search for continuous optimization. In M. J. Blesa et al., editors, *LNCS 5296. Proceedings of the International Workshop on Hybrid Metaheuristics. HM 2008*, pages 72–86, Berlin, Germany, 2008. Springer-Verlag.

[37] C.L̃. Müller, B. Baumgartner, and I.F̃. Sbalzarini. Particle swarm CMA evolution strategy for the optimization of multi-funnel landscapes. In P. Haddow et al., editors, *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2009)*, pages 2685–2692, Piscataway, NJ, 2009. IEEE Press.

[38] Shinichi Nakagawa and Innes C. Cuthill. Effect size, confidence interval and statistical significance: a practical guide for biologists. *Biological Reviews*, 82(4):591–605, 2007.

[39] Jason Noble and Daniel W. Franks. Social learning in a multi-agent system. *Computers and Informatics*, 22(6):561–574, 2003.

[40] Liviu Panait and Sean Luke. Cooperative multi-agent learning: The state of the art. *Autonomous Agents and Multi-Agent Systems*, 11:387–434, 2005.

[41] Y. G. Petalas, K. E. Parsopoulos, and M. N. Vrahatis. Memetic particle swarm optimization. *Annals of Operations Research*, 156(1):99–127, 2007.

[42] Riccardo Poli, James Kennedy, and Tim Blackwell. Particle swarm optimization. An overview. *Swarm Intelligence*, 1(1):33–57, 2007.

[43] M. J. D. Powell. An efficient method for finding the minimum of a function of several variables without calculating derivatives. *The Computer Journal*, 7(2):155–162, 1964.

[44] M. J. D. Powell. *Large-Scale Nonlinear Optimization*, volume 83 of *Nonconvex Optimization and Its Applications*, chapter The NEWUOA software for unconstrained optimization, pages 255–297. Springer-Verlag, Berlin, Germany, 2006.

[45] M. J. D. Powell. Developments of NEWUOA for unconstrained minimization without derivatives. Technical Report DAMTP 2007/NA05, Department of Applied Mathematics and Theoretical Physics, University of Cambridge, November 2007.

[46] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes in C. The Art of Scientific Computing*. Cambridge University Press, New York , NY, second edition, 1992.

[47] Ignacio Rojas, Jesus González, Hector Pomares, J.J̃. Merelo, P.Ã. Castillo, and G. Romero. Statistical analysis of the main parameters involved in the design of a genetic algorithm. *IEEE Transactions on Systems, Man and Cybernetics–Part C: Applications and Reviews*, 32(1):31–37, 2002.

[48] David J. Sheskin. *Handbook of parametric and nonparametric statistical procedures*. Chapman & Hall/CRC, Boca Raton, FL, second edition, 2000.

[49] Valerio Sperati, Vito Trianni, and Stefano Nolfi. Evolving coordinated group behaviours through maximisation of mean mutual information. *Swarm Intelligence*, 2(2–4):73–95, 2008.

[50] Yuri R. Tsoy. The influence of population size and search time limit on genetic algorithm. In *Proceedings of the Seventh Korea-Russia International Symposium on Science and Technology*, pages 181–187, Piscataway, NJ, 2003. IEEE Press.

[51] Frans van den Bergh and Andries P. Engelbrecht. Effects of swarm size on cooperative particle swarm optimisers. In Lee Spector et al., editors, *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2001)*, pages 469–476, San Francisco, CA, 2001. Morgan Kaufmann.

[52] Gu-Li Zhang, Xiao-Xia Liu, and Tong Zhang. The impact of population size on the performance of GA. In *Proceedings of the Eighth International Conference on Machine Learning and Cybernetics*, pages 1866–1870, Piscataway, NJ, 2009. IEEE Press.

Algorithm 2 ISL-based PSO algorithms

Input: Objective function $f : \Theta \subseteq \mathbb{R}^n \to \mathbb{R}$, the initialization domain $\Theta' \subseteq \Theta$, the agent addition criterion A, the local search procedure parameters (tolerance, maximum number of iterations, step size) and the parameters used by the PSO rules (φ_1, φ_2, and χ).
Output: The best found solution $\boldsymbol{sol} \in \Theta$

/* Initialization */
$t \leftarrow 0$ /* Iteration counter */
$k \leftarrow 1$ /* Population size */
Initialize position vector $\boldsymbol{x}_k^t$ to random values within Θ'
Initialize velocity vector $\boldsymbol{v}_k^t$ to zero
$\boldsymbol{p}_k^t \leftarrow \boldsymbol{x}_k^t$
$e_k \leftarrow$ **true** /* If $e_k =$ **true**, a local search should be invoked for particle k (**only in IPSOLS**) */

/* Main Loop */
repeat
  /* Individual learning (**only in IPSOLS**) */
  for $i = 1$ to k **do**
    if $e_k =$ **true then**
      $e_k \leftarrow$ localsearch$(f, \boldsymbol{p}_k^t)$ /* Returns **true** if exited without converging, else returns **false** */
    end if
  end for

  /* Horizontal social learning */
  for $i = 1$ to k **do**
    Generate $\boldsymbol{x}_k^{t+1}$ using Eqs. 1 and 2
    if $f(\boldsymbol{x}_k^{t+1}) < f(\boldsymbol{p}_k^t)$ **then**
      $\boldsymbol{p}_k^{t+1} \leftarrow \boldsymbol{x}_k^{t+1}$
      $e_k \leftarrow$ **true** /* **only in IPSOLS** */
    end if
  end for

  /* Population growth and vertical social learning */
  if Agent addition criterion A is met **then**
    Initialize vector $\boldsymbol{p}_{k+1}^{t+1}$ using Eq. 3 for each component
    Initialize velocity vector $\boldsymbol{v}_{k+1}^{t+1}$ to zero
    $\boldsymbol{x}_{k+1}^{t+1} \leftarrow \boldsymbol{p}_{k+1}^{t+1}$
    $e_{k+1} \leftarrow$ **true** /* **only in IPSOLS** */
    $k \leftarrow k + 1$
  end if
  $t \leftarrow t + 1$
  $\boldsymbol{sol} \leftarrow \underset{i \in \{1,\ldots,k\}}{\operatorname{argmin}} f(\boldsymbol{p}_i^t)$
until Stopping criterion is satisfied