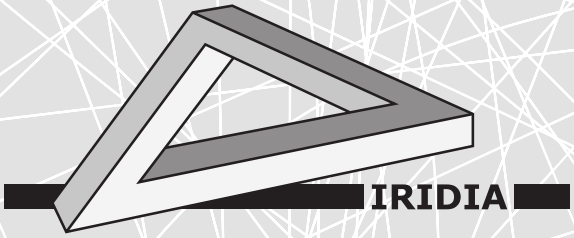


Université Libre de Bruxelles

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
Last revision: May 2010

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
TR/IRIDIA/2009-002.003	May 2010

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

May 28, 2010

Abstract

Incremental social learning (ISL) was proposed as a way to improve the scalability of systems composed of multiple learning agents. In this article, we show that ISL can be very useful to improve the performance of population-based optimization algorithms. Our study focuses on two particle swarm optimization (PSO) algorithms: *a)* the incremental particle swarm optimizer (IPSO), which is a PSO algorithm with a growing population size in which the initial position of new particles is biased toward the best-so-far solution, and *b)* the incremental particle swarm optimizer with local search (IPSOLS), in which solutions are further improved through a local search procedure.

We first derive analytically the probability density function induced by the proposed initialization rule applied to new particles. We then compare the performance of IPSO and IPSOLS on a set of benchmark functions with that of other PSO algorithms (with and without local search) and a random restart local search algorithm. Finally, we measure the benefits of using incremental social learning on PSO algorithms by running IPSO and IPSOLS on problems with different fitness distance correlations.

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, 17] because learning is especially challenging when the number of agents involved is large [49, 58]. 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 [42]. This approach,

which we call *incremental social learning* (ISL), is inspired by the phenomenon of social learning in animal societies [36].

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 [34, 35, 23, 10, 51, 18]. 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 [30, 56, 59, 41, 62]. 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 [60, 43].

We propose two algorithms that result from the application of ISL to PSO algorithms. The first one is an incremental PSO algorithm (IPSO) [42], 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 (IPSOLS) [45] in which particles go through a process of "individual learning", which is simulated through 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 biased initialization of new particles works (Section 5).

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 6).

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 7).

We start by reviewing related work in Section 2. We then describe in detail the ISL framework in Section 3, and the ISL-based PSO algorithms in Section 4. The core of the paper is developed in Sections 5, 6, and 7. Conclusions and future work are presented in Section 8.

2 Related Work

IPSO and IPSOLS are two PSO-based algorithms in which the population size changes during the optimization process. IPSOLS is additionally a PSO–local search hybrid. In this section, we briefly review related work on both topics. We highlight the differences that exist between the previous approaches, and both IPSO and IPSOLS.

2.1 Particle Swarm Optimization Algorithms with Time-Varying 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 [40]. The issue is that it is not uncommon to know little about a problem's difficulty *a priori*. As a result, evolutionary algorithms with *time-varying* population size have been proposed (see e.g. [1, 29, 3, 21, 2, 22, 24]). This research issue has just been recently addressed by the PSO community, and thus not many research contributions exist on this topic. Coelho and de Oliveira [12] adapt the population resizing mechanisms used in APGA [3] and PRoFIGA [21] for their use in PSO algorithms. Lanzarini *et al.* [37] 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. A time-varying population size approach has been adopted by Leong and Yen [38] for tackling multiobjective optimization problems with PSO algorithms. 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 [32], 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.

All these proposals share a common problem: they eliminate the population size parameter, but introduce many more. For example, they need the user to set a particle's maximum lifetime, to select the number of iterations without improvement so that a particle is added or removed, to choose particle recombination operators, and so on. In contrast, our approach introduces only two parameters: the rate at which the population size should grow and the way new particles should be initialized. The setting of the first parameter is based on the ISL framework and the rules for initializing new particles are analyzed in detail in Sections 5 and 7. Additionally, our approach is simple to understand and to implement, and provides advantages that will be described in the rest of the paper.

In contrast to practically all previously studied strategies, our approach, in its current form, does not consider the possibility of reducing the size of the population during an algorithm's run. The rationale behind previous approaches 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. As it will be seen later, not decreasing the population size does not affect negatively the performance of IPSO or of IPSOLS.

2.2 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, IPSOLS included, is thus to accelerate the placement of the particles' previous best positions in good locations. For example, Chen *et al.* [8] combined a particle swarm algorithm with a hill-climbing local search procedure. Liang and Suganthan [39] used a quasi-Newton method to improve a subset of the solutions found by a multi-swarm algorithm. Gimmler *et al.* [25] experimented 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.* [16] also used Nelder and Mead's simplex method and proposed the inclusion of an estimate of the local gradient into the particles' velocity update rule. In [13], a two-phase approach is described where a PSO algorithm is used first to find a good solution and, in a second phase, a quasi-Newton method is used to refine it. Petalas *et al.* [50] report experiments with several local search-particle swarm combination schemes. In [46], Müller *et al.* describe a hybrid PSO–CMA-ES algorithm in which a full-fledged population-based algorithm (CMA-ES [27, 26]) is used as a local search procedure. Other PSO-local search hybrids are reported in [28, 9]. Our proposal is not different from the above-mentioned approaches in the sense that it uses a local search procedure. In all cases, the goal is to accelerate the discovery of good solutions. However, our work is the first to explore the possible benefits of combining a variable population size with local search procedures in the context of PSO algorithms. We will see that this combination allows IPSOLS to adapt to the features of the objective function as discussed in Section 8.

3 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 [49, 58].

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
```

The incremental social learning framework [42] 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 [36]. Thus, ISL allows the new agents to 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 [48]; in our work, however, we focus on the potential of learning socially in an incremental way to speed up learning in a large population.

4 Incremental Social Learning in Particle Swarm Optimization Algorithms

The framework described in Section 3 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. Preliminary studies and results on this topic can be found in [42] and [45].

4.1 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 of particles. The rules that govern the movement of a particle i along the j-th

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

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* [11] that, if properly set, guarantees convergence in the search space (although not necessarily to a local or global optimum [23]). A particle's velocity is usually constrained within the range $[-V_{\max,j}, V_{\max,j}]$ as this can improve the algorithm's performance [20].

4.2 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 [42].

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 [42]. We conjecture that this is due to the tendency that particles have to cluster in the search space. In such a case, the distance between the best particle and a random one would not be large enough to produce significantly different results.

4.3 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 [45]. 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 [52] 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 [55].

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 [45], 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.

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 maximum population size K, 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 **and** $k < K$ **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

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]. When the maximum population size is reached, IPSO and IPSOLS become traditional constant population size algorithms.

5 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 toward the position of a particle that serves as an attractor or "model". In this section, we present the exact probability density function induced by the initialization rule 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 toward 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 density histogram obtained using Monte Carlo simulation when the initialization range is $[0, 1)$ and $p_{model,j} = 0.2$. In a density histogram, the height of each rectangle is equal to $\frac{k_i}{w_i N}$, where k_i is the number of observations of class i in an experiment of N samples. The value w_i is known as class i's width and it is the length of the range that defines class i. In our case, we set the class width to $w_i = 0.02$.

Most of the samples concentrate around the model's position as desired. Note, however, that there is a nonzero probability of sampling regions far

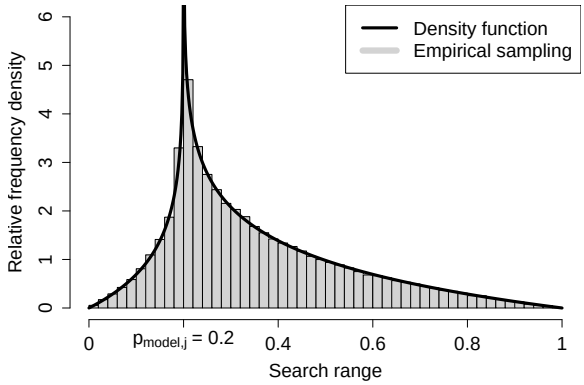


Figure 1: Probability density function induced by the initialization rule of new particles. In the figure, the attractor $p_{model,j} = 0.2$. The initialization range in this example is $[0, 1)$. The figure shows both the analytical density function and the density histogram obtained using Monte Carlo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

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 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 necessarily a drawback because one would usually expect the sought global optimum to lie within the chosen initialization region.

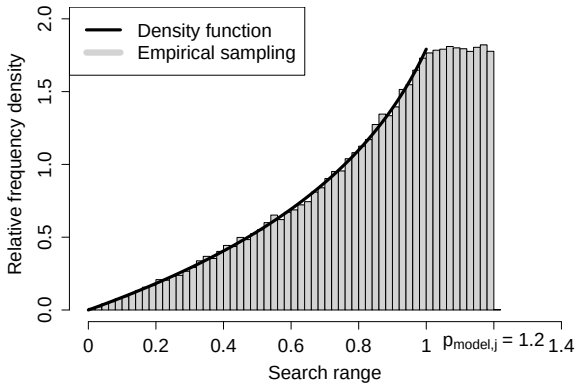


Figure 2: Probability density function induced by the initialization rule of new particles when the attractor lies outside the original initialization range. In the figure, the attractor $p_{model,j} = 1.2$. The initialization range is $[0, 1)$. The figure shows that the density function follows the 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.

6 Experimental Evaluation

In this section, we first describe the setup used to carry out our experiments. After that, we present and discuss the results of the empirical performance evaluation of the ISL-based PSO algorithms presented in Section 4.

6.1 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. This algorithm was included in the evaluation in order to measure the contribution of the incremental population component used in IPSO. This algorithm is labeled as PSO-X, where X is the population size.

2. A recently proposed PSO algorithm with time-varying population size (labeled EPUS) [32]. 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 unless the population size would 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 [32] in order not to confound the effects of the variable population size with those of these operators.

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 it is done in IPSOLS. 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). 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 EPUSLS, PSOLS and IPSOLS.

All algorithms were run on a set of twelve commonly used benchmark functions whose mathematical definition is shown in Table 1. In all cases, we used the 100-dimensional versions of the benchmark functions. In our experimental setup, each algorithm was run with the same parameter settings across all benchmark functions. The parameter settings used for each algorithm are listed in Table 2.

Our decision of adding one particle per iteration is based on preliminary results that show that the particle-addition schedule affects the exploration-exploitation behavior of IPSO [42]. Faster schedules encourage exploration while slower ones encourage exploitation. In IPSOLS, the exploitative behavior induced by the local search procedure needs to be balanced with an exploration-encouraging, fast particle-addition schedule.

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

Table 1: Benchmark functions

Name	Definition	Search Range $[x_{\min}, x_{\max}]^n$
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
Axis-parallel Hyper-ellipsoid	$\sum_{i=1}^{n}(ix_i)^2$	[-100,100]^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
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
Penalized function	$\frac{\pi}{n}\{10\sin^2(\pi y_1) + \sum_{i=1}^{n-1}(y_i-1)^2[1+10\sin^2(\pi y_{i+1})] + (y_n-1)^2\} + \sum_{i=1}^{n} u(x_i,10,100,4)$, where $y_i = 1+(x_i+1)/4$, $u(x,a,k,m) = \begin{cases} k(x_i-a)^m & \text{if } x_i > a \\ k(-x_i-a)^m & \text{if } x_i < a \\ 0 & \text{otherwise} \end{cases}$	[-50,50]^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	$\sum_{i=1}^{n}\left(\sum_{k=1}^{k_{max}} a^k \cos\left(2\pi b^k (x_i+0.5)\right)\right) - n\sum_{k=1}^{k_{max}}[a^k \cos(\pi b^k)]$, where $a=0.5$, $b=3$, $k_{max}=20$	[-0.5,0.5]^n

criteria was met: (i) 10^6 function evaluations had been performed, or (ii) the objective function value was less than or equal to 10^{-15}. However, it was still possible to find solutions with a lower value than this threshold because the stopping criteria were evaluated outside the local search procedure. To eliminate the effects of any possible search bias toward the origin of the coordinate system, at each run, a benchmark function was randomly shifted within the specified search range. Functions Schwefel and Step were not shifted as their optima are not at the origin of the coordinate system. 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.

PSOLS was run with fewer particles than PSO because larger populations 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, a large population would essentially behave as a random restart local search, which was included in the comparison.

Table 2: Parameter settings

Setting(s)	Algorithm(s)
Acceleration coefficients: $\varphi_1 = \varphi_2 = 2.05$	All except RLS
Constriction coefficient: $\chi = 0.729$	All except RLS
Maximum velocity: $V_{\max} = \pm X_{\max}$	All except RLS
Population size: $10, 100, 1000$ particles	PSO
Population size: $5, 10, 20$ particles	PSOLS
Population enlargement/reduction control parameter: $k = 2$	EPUS, EPUSLS
Minimum population size: 1 particle	EPUS, EPUSLS, IPSO, IPSOLS
Maximum population size: 1000 particles	EPUS, EPUSLS, IPSO, IPSOLS
Particle-addition schedule: 1 particle per iteration	IPSO, IPSOLS
Model particle for initialization: Best	IPSO, IPSOLS
Powell's method tolerance: 0.01	EPUSLS, IPSOLS, PSOLS, RLS
Powell's method maximum number of iterations: 10	EPUSLS, IPSOLS, PSOLS, RLS
Powell's method step size: 20% of the length of the search range	EPUSLS, IPSOLS, PSOLS, RLS

All particle swarm-based algorithms (PSO, PSOLS, EPUS, EPUSLS, IPSO 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.

6.2 Performance Evaluation Results

Algorithms for continuous optimization are often evaluated according to two different criteria. One of these criteria measures the quality of the solutions (through the objective function values associated with them) that algorithms are able to find given a maximum number of function evaluations. The other criterion measures the number of function evaluations needed by algorithms to reach a target objective function value. Since the algorithms used in our study are stochastic, both performance measures are also stochastic. For this reason, we look at the distribution (i) of the objective function values at different run lengths, and (ii) of the number of function evaluations needed to reach some target objective function values.[2] We also look at some central tendency

[2]In this paper's supplementary information web page [44], the reader can find the complete data that, for the sake of conciseness, are not included in the paper. Nevertheless, the main

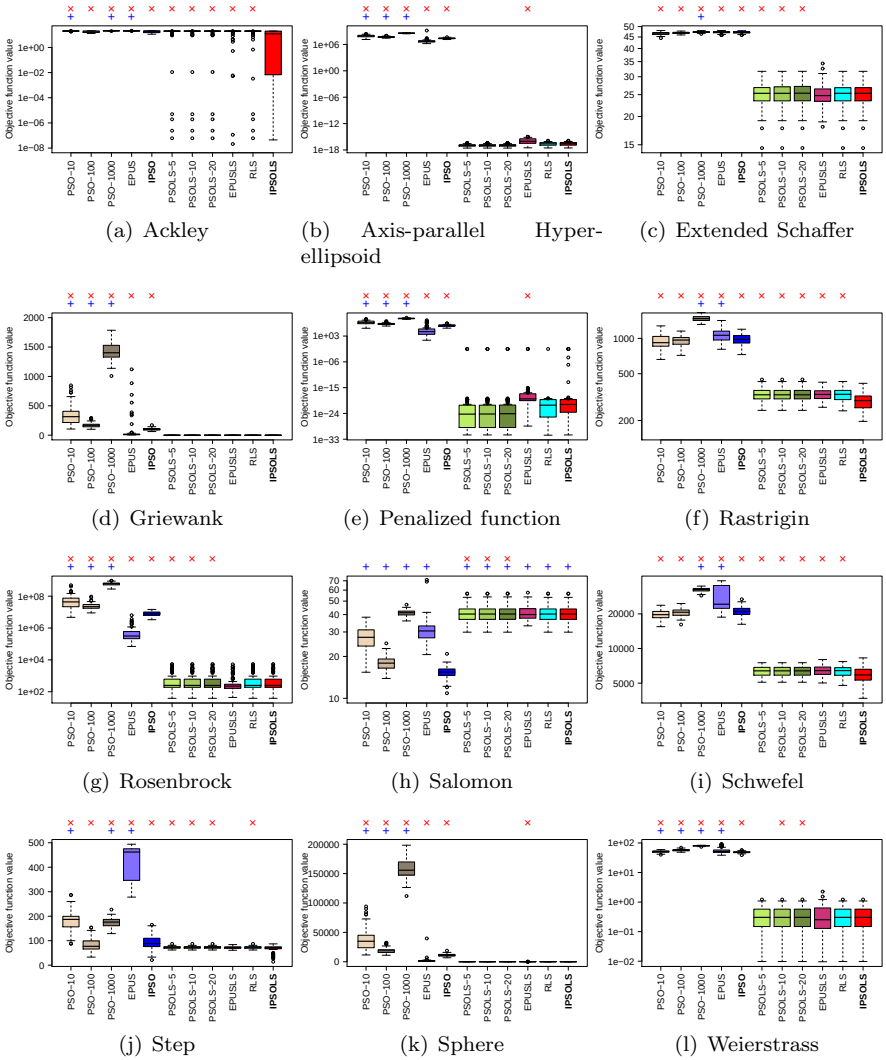
measures to have a more aggregated summary of the performance of the compared algorithms. Finally, we present a summary of the statistical data analyisis performed on all the data. In the discussion of the results, we pay particular attention to the two main components of the ISL-based algorithms, which are the variable population size and the use of a local search procedure.

6.2.1 Constant vs. variable population size

The distributions of the objective function values after 10^4 and 10^6 function evaluations are shown in Figures 3 and 4, respectively. On top of each box plot there may be 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 $\times$ 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 [44]. Significance was determined at the 5% level using a Wilcoxon test using Holm's correction method for multiple comparisons.

The performance of constant population size PSO algorithms without local search greatly depends on the population size. The results obtained with the traditional PSO algorithm confirm the tradeoff 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 after 10^6 function evaluations, the swarms with 1000 particles often return the best solutions. This tendency can also be seen in Table 3, which lists the median objective function values obtained by the tested algorithms on all benchmark functions at different run lengths.

results are discussed in the text.



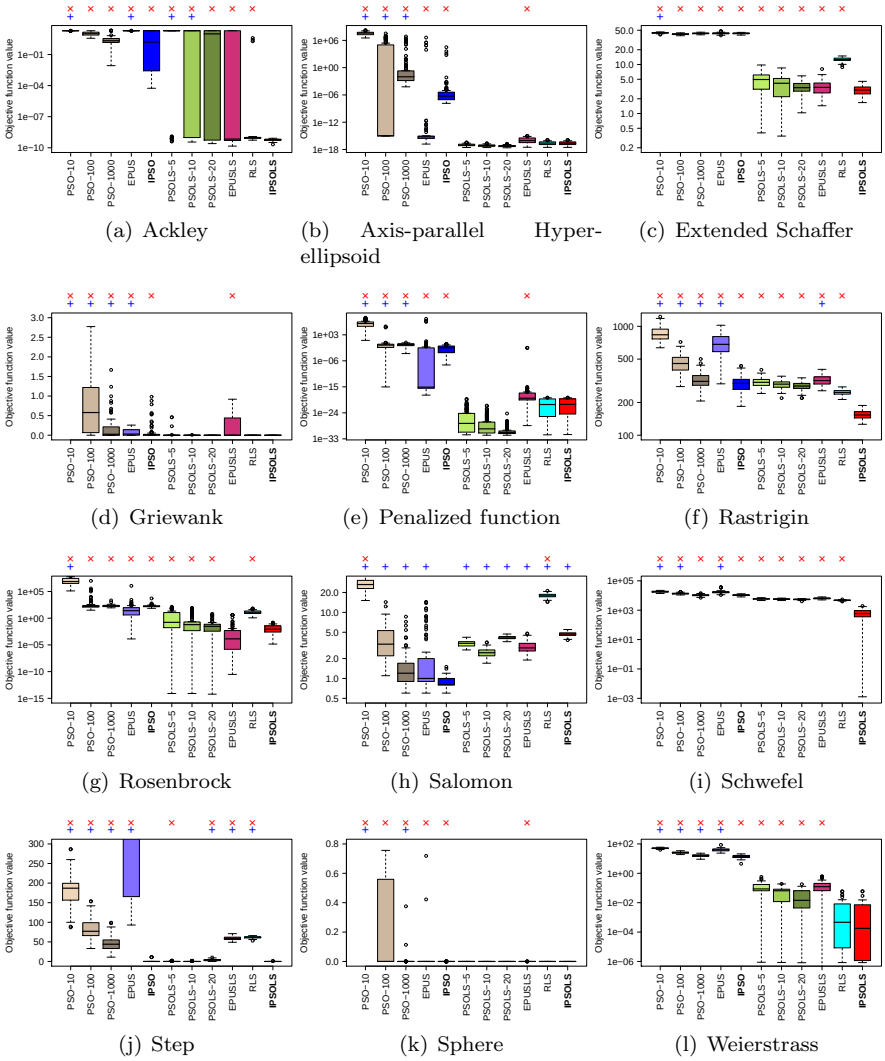
(a) Ackley (b) Axis-parallel Hyper-ellipsoid (c) Extended Schaffer

(d) Griewank (e) Penalized function (f) Rastrigin

(g) Rosenbrock (h) Salomon (i) Schwefel

(j) Step (k) Sphere (l) Weierstrass

Figure 3: The box plots show the distribution of the solution quality obtained with the compared algorithms for runs of up to 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) Axis-parallel Hyper-ellipsoid (c) Extended Schaffer

(d) Griewank (e) Penalized function (f) Rastrigin

(g) Rosenbrock (h) Salomon (i) Schwefel

(j) Step (k) Sphere (l) Weierstrass

Figure 4: The box plots show the distribution of the solution quality obtained with the compared algorithms for runs of up to 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 $\times$ symbol).

Table 3: Median objective function values at different run lengths[1]

Function Evaluations	Algorithm	Benchmark function											
		Ackley	Axis-parallel Hyper-ellipsoid	Extended Schaffer	Griewank	Penalized	Rastrigin	Rosenbrock	Salomon	Schwefel	Sphere	Step	Weierstrass
10^2	PSO-10[1]	**2.13e+01**	1.02e+09	**4.86e+01**	3.19e+03	9.20e+09	1.96e+03	2.73e+09	6.17e+01	3.65e+04	3.55e+05	**3.72e+02**	8.72e+01
	PSO-10[2]	2.14e+01	1.48e+09	**4.86e+01**	4.42e+03	1.88e+10	2.24e+03	4.99e+09	7.08e+01	3.73e+04	4.91e+05	4.78e+02	9.01e+01
	PSO-10[3]	2.14e+01	1.48e+09	**4.86e+01**	4.42e+03	1.88e+10	2.24e+03	4.99e+09	7.08e+01	3.73e+04	4.91e+05	4.78e+02	9.01e+01
	EPUS	2.14e+01	1.16e+09	4.87e+01	3.57e+03	1.02e+10	2.05e+03	3.17e+09	6.48e+01	3.86e+04	3.96e+05	5.02e+02	9.14e+01
	IPSO	**2.13e+01**	**9.54e+08**	**4.86e+01**	**3.00e+03**	**5.93e+09**	**1.84e+03**	**2.00e+09**	**5.83e+01**	**3.64e+04**	**3.33e+05**	4.04e+02	**8.63e+01**
	PSOLS-5	2.15e+01	1.89e+09	4.94e+01	5.03e+03	2.58e+10	2.45e+03	6.60e+09	7.72e+01	3.88e+04	5.59e+05	5.04e+02	9.15e+01
	PSOLS-10	2.15e+01	1.75e+09	4.91e+01	4.85e+03	2.41e+10	2.40e+03	6.12e+09	7.53e+01	3.85e+04	5.39e+05	4.98e+02	9.13e+01
	PSOLS-20	2.14e+01	1.66e+09	4.90e+01	4.74e+03	2.21e+10	2.36e+03	5.71e+09	7.39e+01	3.81e+04	5.27e+05	4.90e+02	9.10e+01
	EPUSLS	2.15e+01	2.15e+09	4.96e+01	5.23e+03	3.09e+10	2.51e+03	7.56e+09	8.07e+01	3.91e+04	5.81e+05	5.16e+02	9.21e+01
	RLS	2.15e+01	2.21e+09	4.97e+01	5.15e+03	3.01e+10	2.52e+03	7.39e+09	8.02e+01	3.91e+04	5.72e+05	5.14e+02	9.20e+01
	IPSOLS	2.15e+01	2.21e+09	4.97e+01	5.15e+03	3.01e+10	2.52e+03	7.39e+09	8.02e+01	3.91e+04	5.72e+05	5.14e+02	9.20e+01
10^3	PSO-10[1]	2.11e+01	2.96e+08	4.78e+01	1.12e+03	**1.06e+09**	1.38e+03	**4.59e+08**	**4.05e+01**	2.63e+04	1.24e+05	1.87e+02	6.77e+01
	PSO-10[2]	2.11e+01	5.76e+08	4.80e+01	2.12e+03	3.55e+09	1.66e+03	1.27e+09	5.02e+01	3.44e+04	2.35e+05	2.52e+02	8.37e+01
	PSO-10[3]	2.13e+01	1.28e+09	4.80e+01	3.99e+03	1.51e+10	2.12e+03	4.08e+09	6.76e+01	3.56e+04	4.43e+05	4.55e+02	8.71e+01
	EPUS	2.12e+01	4.27e+08	4.80e+01	1.58e+03	2.01e+09	1.61e+03	7.62e+08	4.91e+01	3.63e+04	1.70e+05	4.82e+02	7.99e+01
	IPSO	2.10e+01	3.01e+08	4.79e+01	1.22e+03	1.13e+09	1.53e+03	5.07e+08	4.09e+01	3.04e+04	1.35e+05	**1.86e+02**	7.43e+01
	PSOLS-5	**2.04e+01**	**9.34e-17**	**4.62e+01**	9.69e-01	2.99e+09	7.75e+02	1.37e+09	7.60e+01	**1.38e+04**	**0.00e+00**	2.44e+02	2.22e+01
	PSOLS-10	**2.04e+01**	**9.34e-17**	**4.62e+01**	9.69e-01	3.15e+09	7.86e+02	1.41e+09	7.46e+01	1.39e+04	**0.00e+00**	2.45e+02	2.25e+01
	PSOLS-20	2.05e+01	1.41e-16	4.63e+01	9.69e-01	3.52e+09	8.15e+02	1.50e+09	7.36e+01	1.40e+04	**0.00e+00**	2.49e+02	2.32e+01
	EPUSLS	**2.04e+01**	5.06e-16	4.63e+01	**9.56e-01**	2.27e+09	**7.56e+02**	1.12e+09	7.74e+01	1.39e+04	2.94e-22	2.44e+02	**2.18e+01**
	RLS	**2.04e+01**	**9.34e-17**	**4.62e+01**	9.69e-01	2.93e+09	7.65e+02	1.36e+09	7.75e+01	**1.38e+04**	**0.00e+00**	2.44e+02	**2.18e+01**
	IPSOLS	**2.04e+01**	**9.34e-17**	**4.62e+01**	9.69e-01	2.93e+09	7.65e+02	1.36e+09	7.75e+01	**1.38e+04**	**0.00e+00**	2.44e+02	**2.18e+01**
10^4	PSO-10[1]	2.01e+01	8.37e+07	4.67e+01	3.15e+02	4.69e+07	9.19e+02	4.37e+07	2.74e+01	1.97e+04	3.49e+04	1.87e+02	5.10e+01
	PSO-10[2]	1.72e+01	4.95e+07	4.70e+01	1.68e+02	1.62e+07	9.67e+02	2.20e+07	1.79e+01	2.07e+04	1.86e+04	7.70e+01	5.60e+01
	PSO-10[3]	2.09e+01	3.70e+08	4.74e+01	1.40e+03	1.39e+09	1.47e+03	6.16e+08	4.13e+01	3.25e+04	1.56e+05	1.76e+02	8.04e+01
	EPUS	2.06e+01	4.89e+06	4.72e+01	1.23e+01	3.48e+04	1.06e+03	3.14e+05	3.05e+01	2.43e+04	1.23e+03	4.62e+02	5.16e+01
	IPSO	2.00e+01	2.51e+07	4.71e+01	9.89e+01	5.34e+06	9.81e+02	7.78e+06	**1.54e+01**	2.10e+04	1.09e+04	8.80e+01	4.92e+01
	PSOLS-5	1.99e+01	**1.08e-17**	2.53e+01	**2.27e-01**	**5.59e-25**	3.31e+02	2.49e+02	4.03e+01	6.40e+03	**0.00e+00**	7.30e+01	3.04e-01
	PSOLS-10	1.99e+01	**1.08e-17**	2.53e+01	**2.27e-01**	6.17e-25	3.31e+02	2.49e+02	4.03e+01	6.40e+03	**0.00e+00**	7.30e+01	3.04e-01
	PSOLS-20	1.99e+01	**1.08e-17**	2.54e+01	**2.27e-01**	7.93e-25	3.31e+02	2.49e+02	4.03e+01	6.40e+03	**0.00e+00**	7.30e+01	3.04e-01
	EPUSLS	1.99e+01	9.68e-17	**2.48e+01**	2.33e-01	1.17e-19	3.33e+02	**2.26e+02**	4.00e+01	6.43e+03	1.59e-22	**7.20e+01**	**2.53e-01**
	RLS	1.99e+01	2.49e-17	2.53e+01	2.33e-01	7.56e-22	3.34e+02	2.49e+02	4.03e+01	6.43e+03	**0.00e+00**	7.30e+01	3.04e-01
	IPSOLS	**1.22e+01**	2.49e-17	2.53e+01	2.31e-01	1.33e-21	**2.95e+02**	2.49e+02	4.03e+01	**5.90e+03**	**0.00e+00**	**7.20e+01**	3.04e-01
10^5	PSO-10[1]	1.99e+01	3.84e+07	4.50e+01	1.04e+02	7.95e+06	8.36e+02	7.43e+06	2.63e+01	1.80e+04	1.18e+04	1.87e+02	5.10e+01
	PSO-10[2]	1.03e+01	2.12e+05	4.50e+01	1.07e+00	4.90e+00	4.57e+02	1.10e+03	3.75e+00	1.35e+04	1.31e+01	7.70e+01	2.55e+01
	PSO-10[3]	1.00e+01	9.08e+06	4.62e+01	3.04e+01	1.95e+05	6.86e+02	1.50e+06	9.61e+00	1.60e+04	3.27e+03	4.40e+01	4.27e+01
	EPUS	1.98e+01	1.46e-02	4.57e+01	3.32e-02	8.79e-01	7.96e+02	2.43e+02	9.50e+00	1.82e+04	8.57e-07	4.52e+02	4.11e+01
	IPSO	4.19e+00	8.31e+04	4.56e+01	1.13e+00	1.23e+01	5.73e+02	2.89e+03	**3.13e+00**	1.55e+04	1.37e+01	1.10e+01	2.45e+01
	PSOLS-5	1.99e+01	9.52e-18	1.06e+01	**0.00e+00**	2.20e-28	3.06e+02	1.56e+01	8.90e+00	5.90e+03	**0.00e+00**	2.10e+01	8.64e-02
	PSOLS-10	1.98e+01	7.11e-18	1.54e+01	**0.00e+00**	2.90e-30	2.94e+02	6.64e+01	2.13e+01	5.70e+03	**0.00e+00**	6.70e+01	6.98e-02
	PSOLS-20	9.81e+00	**5.91e-18**	1.54e+01	**0.00e+00**	**1.40e-31**	2.84e+02	6.64e+01	2.13e+01	5.47e+03	**0.00e+00**	6.70e+01	6.98e-02
	EPUSLS	1.98e+01	9.68e-17	**6.24e+00**	2.32e-01	1.17e-19	3.33e+02	**1.27e+00**	6.60e+00	6.43e+03	1.59e-22	7.00e+01	1.32e-01
	RLS	2.76e+00	2.49e-17	1.52e+01	**0.00e+00**	7.56e-22	2.81e+02	6.50e+01	2.14e+01	5.45e+03	**0.00e+00**	6.65e+01	**6.69e-02**
	IPSOLS	**8.54e-10**	2.49e-17	6.68e+00	**0.00e+00**	8.27e-22	**1.99e+02**	1.78e+00	6.00e+00	**2.81e+03**	**0.00e+00**	**9.00e+00**	7.27e-02
10^6	PSO-10[1]	1.99e+01	3.84e+07	4.43e+01	1.04e+02	7.95e+06	8.36e+02	7.35e+06	2.63e+01	1.80e+04	1.18e+04	1.87e+02	5.10e+01
	PSO-10[2]	1.02e+01	9.99e-16	4.17e+01	5.79e-01	2.77e-01	4.56e+02	1.64e+02	3.30e+00	1.35e+04	1.00e-15	7.70e+01	2.55e+01
	PSO-10[3]	2.13e+00	9.85e-03	4.31e+01	2.46e-02	4.50e-01	3.12e+02	1.98e+02	1.20e+00	1.06e+04	3.91e-07	4.40e+01	1.65e+01
	EPUS	1.98e+01	5.92e-16	4.34e+01	2.95e-02	7.97e-16	6.83e+02	2.55e+01	1.00e+00	1.65e+04	4.98e-16	4.30e+02	3.99e+01
	IPSO	1.60e+00	5.64e-07	4.32e+01	7.40e-03	3.20e-02	2.99e+02	1.86e+02	**8.00e-01**	1.06e+04	2.82e-11	**0.00e+00**	1.43e+01
	PSOLS-5	1.99e+01	9.52e-18	4.94e+00	**0.00e+00**	2.20e-28	3.06e+02	1.68e-01	3.40e+00	5.90e+03	**0.00e+00**	**0.00e+00**	8.64e-02
	PSOLS-10	1.98e+01	7.11e-18	4.14e+00	**0.00e+00**	2.90e-30	2.94e+02	6.31e-02	2.45e+00	5.70e+03	**0.00e+00**	**0.00e+00**	6.50e-02
	PSOLS-20	9.81e+00	**5.91e-18**	3.37e+00	**0.00e+00**	**1.40e-31**	2.83e+02	2.97e-02	4.10e+00	5.47e+03	**0.00e+00**	4.00e+00	1.45e-02
	EPUSLS	6.89e-10	9.35e-17	3.41e+00	6.41e-16	1.17e-19	3.18e+02	**1.37e-04**	2.90e+00	6.36e+03	1.59e-22	5.90e+01	1.24e-01
	RLS	8.92e-10	2.49e-17	1.27e+01	**0.00e+00**	7.56e-22	2.47e+02	1.28e+01	1.78e+01	**0.00e+00**	6.20e+01	4.64e-04	
	IPSOLS	**5.91e-10**	2.49e-17	**3.02e+00**	**0.00e+00**	8.27e-22	**1.53e+02**	8.89e-03	4.70e+00	**5.92e+02**	**0.00e+00**	**0.00e+00**	**1.78e-04**

[1] Results of PSO-based algorithms obtained with a fully-connected topology. The lowest median objective function values are highlighted in boldface.

Table 4: 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	17 (5)	22 (2)	22 (2)	22 (2)
10^3	1 (5)	23 (1)	24 (0)	19 (2)
10^4	10 (2)	17 (7)	24 (0)	10 (2)
10^5	14 (0)	3 (7)	22 (2)	8 (2)
10^6	23 (0)	10 (5)	19 (5)	9 (3)

[1] No worse cases shown in parenthesis.
[2] 12 problems × 2 topologies = 24 cases.

Regarding the algorithms with variable population size, it can be said that IPSO is the best among the studied algorithms for runs of up to 10^2 function evaluations. The data in Table 3 show that IPSO finds the best median objective function values for 11 out of the 12 functions used. IPSOLS and RLS find the best solutions for 6 out of the 12 possible cases for runs of up to 10^3 function evaluations; however, the best results are distributed among all the tested algorithms. For 10^4 or more function evaluations, algorithms that use local search are the ones that find the best solutions (except for the Salomon function). IPSOLS finds at least the same number of best solutions as the other local search-based algorithms. For runs of up to 1 million function evaluations, IPSOLS finds 8 out of the 12 possible best median solutions.

Data from Figures 3 and 4, and Table 3 suggest that in contrast with constant population size PSO algorithms, 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 mechanism used for varying the size of the population does have an impact on performance. This can be seen in Table 4, 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. In total, 24 cases are considered, which are the result of summarizing the results obtained on the 12 benchmark functions using both the fully connected and ring topologies. Also in Table 4, it can be seen 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. 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.

6.2.2 Use vs. no use of local search

The local search component plays a major role on the performance of the algorithms that include it. Table 3 and Figures 3 and 4 show that for runs of at least 10^3 function evaluations, 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 an algorithm without a local search component (IPSO) dominates is when solving the Salomon function. Speed is also affected by the use of a local search component. Table 5 lists the first, second, and third quartiles of the algorithms' run-length distributions [31]. A hyphen in an entry indicates that the target objective function value was not reached within the 10^6 function evaluations allocated for the experiments. Therefore, if there is a hyphen in a third quartile entry, this means that at least 25% of the runs did not find the target objective function value. A similar reasoning applies if there is a hyphen in a first or second quartile entry. The data in Table 5 show that the algorithms that combine a variable population size with a local search component (EPUSLS and IPSOLS) are the fastest and most reliable among the studied algorithms. EPUSLS and IPSOLS together are the fastest algorithms for 9 out of the 12 considered functions.

Table 5: First, second, and third quartiles of the number of function evaluations needed to find a target solution value[1]

Algorithm	Quartile	Benchmark function (Target value)											
		Ackley (10)	Axis-parallel Hyper-ellipsoid (0.01)	Extended Schaffer (10)	Griewank (0.01)	Penalized (0.001)	Rastrigin (1000)	Rosenbrock (0.01)	Salomon (10)	Schwefel (10000)	Sphere (0.01)	Step (10)	Weierstrass (0.01)
PSO-10[1]	3rd	-	-	-	-	-	1.64e+04	-	-	-	-	-	-
	2nd	-	-	-	-	-	4.41e+03	-	-	-	-	-	-
	1st	-	-	-	-	-	2.81e+03	-	-	-	-	-	-
PSO-10[2]	3rd	-	-	-	-	-	1.05e+04	-	3.09e+04	-	-	-	-
	2nd	-	3.65e+05	-	-	-	9.26e+03	-	2.61e+04	-	2.04e+05	-	-
	1st	3.02e+04	2.84e+05	-	-	-	8.01e+03	-	2.11e+04	-	1.65e+05	-	-
PSO-10[3]	3rd	1.28e+05	-	-	-	-	6.00e+04	-	1.04e+05	-	7.04e+05	-	-
	2nd	9.94e+04	-	-	-	-	5.44e+04	-	9.46e+04	-	6.21e+05	-	-
	1st	8.56e+04	9.16e+05	-	5.93e+05	-	4.74e+04	-	8.83e+04	4.23e+05	5.71e+05	-	-
EPUS	3rd	-	1.10e+05	-	-	-	2.18e+04	-	1.41e+05	-	5.49e+04	-	-
	2nd	-	1.02e+05	-	-	1.97e+05	1.33e+04	-	9.48e+04	-	5.15e+04	-	-
	1st	-	9.57e+04	-	5.38e+04	1.23e+05	8.45e+03	-	6.10e+04	-	4.75e+04	-	-
IPSO	3rd	-	7.10e+05	-	-	-	1.25e+04	-	**2.21e+04**	-	3.09e+05	2.26e+05	-
	2nd	3.41e+04	6.24e+05	-	3.44e+05	-	9.53e+03	-	**2.00e+04**	-	2.82e+05	1.53e+05	-
	1st	2.16e+04	5.68e+05	-	2.70e+05	9.64e+05	7.25e+03	-	**1.87e+04**	6.47e+05	2.65e+05	1.03e+05	-
PSOLS-5	3rd	-	1.39e+03	1.34e+05	**2.61e+04**	**1.23e+03**	9.37e+02	-	9.81e+04	1.70e+03	8.62e+02	1.98e+05	-
	2nd	-	9.38e+02	1.12e+05	**1.25e+04**	**1.12e+03**	8.93e+02	-	9.49e+04	1.60e+03	8.52e+02	1.59e+05	-
	1st	-	8.51e+02	9.54e+04	**6.89e+03**	**1.09e+03**	8.47e+02	-	9.24e+04	1.46e+03	8.48e+02	1.42e+05	-
PSOLS-10	3rd	-	1.39e+03	2.15e+05	**2.66e+04**	1.23e+03	9.42e+02	-	1.78e+05	1.71e+03	8.67e+02	3.68e+05	-
	2nd	-	9.44e+02	1.92e+05	**1.25e+04**	1.13e+03	8.98e+02	-	1.75e+05	1.60e+03	8.57e+02	2.92e+05	-
	1st	2.65e+04	8.56e+02	1.76e+05	**6.90e+03**	1.10e+03	8.52e+02	4.33e+05	1.72e+05	1.47e+03	8.53e+02	2.51e+05	-
PSOLS-20	3rd	-	1.40e+03	3.76e+05	**2.66e+04**	1.24e+03	9.52e+02	-	3.39e+05	1.72e+03	8.77e+02	6.68e+05	-
	2nd	-	9.54e+02	3.53e+05	**1.25e+04**	1.14e+03	9.08e+02	-	3.34e+05	1.61e+03	8.67e+02	5.80e+05	-
	1st	2.65e+04	8.66e+02	3.38e+05	**6.91e+03**	1.11e+03	8.62e+02	7.16e+05	3.31e+05	1.48e+03	8.63e+02	5.10e+05	1.85e+05
EPUSLS	3rd	-	**1.39e+03**	**4.75e+04**	-	**1.22e+03**	**9.10e+02**	**8.70e+05**	4.31e+04	1.69e+03	8.59e+02	-	-
	2nd	3.04e+05	**8.60e+02**	**3.44e+04**	1.41e+05	**1.12e+03**	**8.54e+02**	**4.87e+05**	3.21e+04	1.61e+03	8.49e+02	-	-
	1st	3.92e+04	**8.48e+02**	**2.99e+04**	6.66e+03	**1.10e+03**	**8.22e+02**	**3.20e+05**	2.86e+04	1.48e+03	8.45e+02	-	-
RLS	3rd	1.76e+05	1.38e+03	-	3.08e+04	**1.22e+03**	9.33e+02	-	-	**1.70e+03**	**8.58e+02**	-	**8.58e+05**
	2nd	7.01e+04	9.34e+02	-	1.40e+04	**1.12e+03**	8.89e+02	-	-	**1.59e+03**	**8.48e+02**	-	**3.80e+05**
	1st	3.36e+04	8.47e+02	-	6.91e+03	**1.09e+03**	8.43e+02	-	-	**1.46e+03**	**8.44e+02**	-	**1.37e+05**
IPSOLS	3rd	**1.42e+04**	1.38e+03	6.25e+04	2.46e+04	**1.22e+03**	9.33e+02	-	3.41e+04	**1.70e+03**	**8.58e+02**	**1.02e+05**	6.88e+05
	2nd	**1.03e+04**	9.34e+02	3.81e+04	1.36e+04	**1.12e+03**	8.89e+02	8.96e+05	3.10e+04	**1.59e+03**	**8.48e+02**	**8.87e+04**	4.11e+05
	1st	**7.50e+03**	8.47e+02	3.11e+04	6.82e+03	**1.09e+03**	8.43e+02	3.82e+05	2.86e+04	**1.46e+03**	**8.44e+02**	**4.30e+04**	1.83e+05

[1] Results of PSO-based algorithms obtained with a fully-connected topology. The target value for each function is indicated under its name in parentheses. The results with the lowest median number of function evaluations are highlighted in boldface.

Table 6: 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 (24)	0 (23)
10^3	15 (6)	15 (6)	15 (6)	0 (23)	0 (23)
10^4	14 (5)	15 (4)	15 (4)	12 (11)	8 (15)
10^5	15 (4)	14 (5)	14 (5)	19 (4)	14 (9)
10^6	12 (5)	10 (7)	12 (5)	18 (2)	14 (9)

[1] No worse cases shown in parenthesis.
[2] 12 problems × 2 topologies = 24 cases.

In terms of the quality of the solutions found by the local search-based algorithms, IPSOLS outperforms EPUSLS as seen in Tables 3 and 6. This last table 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.

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 6). 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 Axis-parallel Hyper-ellipsoid, Griewank, Penalized, Sphere and Weierstrass. In the case of the Sphere function, the local search procedure alone is able to find the optimum (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 [44]). Thus, IPSOLS's behavior is similar 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.

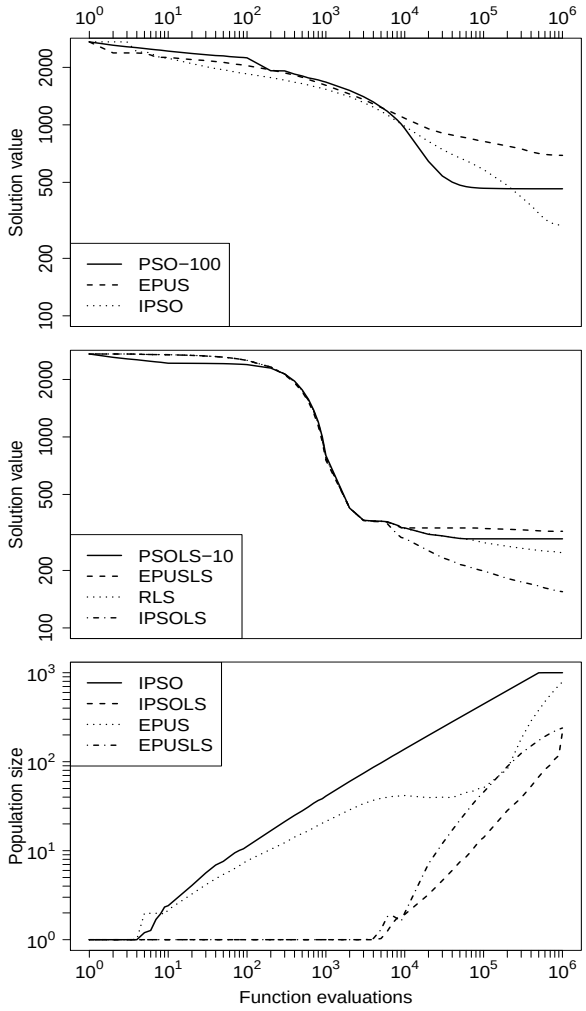


Figure 5: Solution development over time obtained by a selection of the compared algorithms (PSO-based algorithms using a fully-connected topology) on the Rastrigin function. Results without local search (upper plot). Results with local search (middle plot). Average population size growth in IPSO, EPUS,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

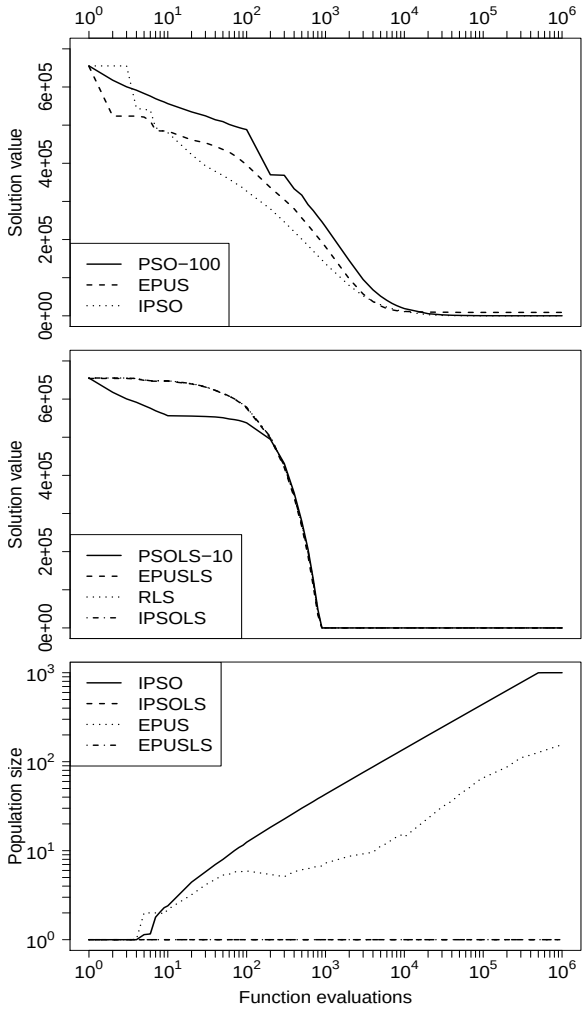


Figure 6: Solution development over time obtained by a selection of the compared algorithms (PSO-based algorithms using a fully-connected topology) on the Sphere function. Results without local search (upper plot). Results with local search (middle plot). Average population size growth in IPSO, EPUS,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

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 that 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 [44]).

An exception in the conclusions of the analysis of the results has been the Salomon function case. This function can be thought of as a multidimensional wave that is symmetric in all directions with respect to the optimum. We believe that the poor performance of all the tested local search-based algorithms is due to the undulatory nature of this function. When the local search is invoked in the proximity of the global optimum, valleys that are far away from it can actually attract the local search method. This can "deceive" the global optimization algorithm that calls the local search method. This phenomenon seems to be exacerbated when Powell's method is applied in high dimensions.

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 shown in [44]).

7 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 toward 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").

Table 7: 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	-	-

The experiments are carried out on problems derived from the Rastrigin function, each of which has different fitness distance correlation (FDC) [33]. 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 7.

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 [14], which for the case of two samples is defined as follows:

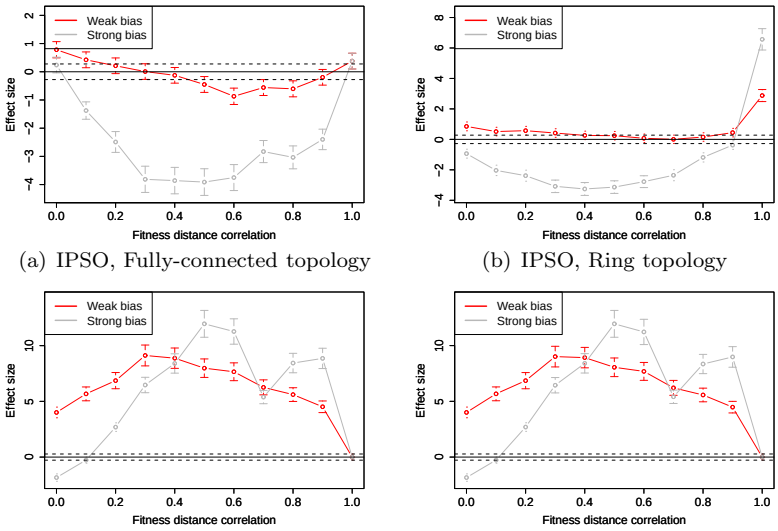
$$d = \frac{\hat{\mu_1} - \hat{\mu_2}}{\sigma_{pooled}}, \tag{6}$$

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 [47].

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 [57]. 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



(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.

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 [14].

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 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).

8 Conclusions and Future Work

In this paper, we have shown how the 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 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, IPSO, is a PSO algorithm with a growing population size, in which new particles are initialized biasing their initial position toward the best-so-far solution. The second algorithm, 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. That is, if a good-enough solution is found by local search alone, the algorithm is stopped. As a consequence, the number of particles remains constant as there is no need to iterate through the PSO component of IPSOLS. However, if the local search procedure does not find a satisfactory solution within the maximum number of iterations allocated to it, a new particle is added to the population and a form 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. Further 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 [53, 54]. Different particles using different local search algorithms may be a way to tackle problems of the kind posed by the Salomon function. An interesting line of research that has already produced some encouraging results, is the one of online parameter adaptation in PSO algorithms (see e.g. [61, 15]). We think that an adaptive version of IPSO or IPSOLS, in which particles are added or removed based on information gathered from the search process, is worth exploring.

An important issue that needs to be addressed in the future is the applicability of ISL to other population-based optimization techniques. In principle, ISL can be used with any population-based optimization algorithm. It is not known, however, what features exactly those algorithms need to have so that ISL can be of any advantage. For example, it is not straightforward to apply ISL to ant colony optimization algorithms [19], which have a centralized memory structure that already allows agents (in this case artificial ants) to share their search experience with others.

In this paper, we have tackled optimization problems. Thus, our conclusions only apply to this kind of problems. Further research is needed to understand whether it is possible to successfully use the incremental social learning frame-

work on traditional learning tasks such as classification or prediction.

Acknowledgments

The work described in this paper 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 (CEC 1994)*, 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, 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] Min-Rong Chen, Xia Li, Xi Zhang, and Yong-Zai Lu. A novel particle swarm optimizer hybridized with extremal optimization. *Applied Soft Computing*, 10(2):367–373, 2010.

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

[11] 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.

[12] 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.

[13] Leandro Dos Santos Coelho and Viviana Cocco Mariani. Particle swarm optimization with quasi-Newton local search for solving economic dispatch problem. In *Proceedings of the IEEE Congress on Systems, Man, and Cybernetics (SMC 2006)*, pages 3109–3113, Piscataway, NJ, 2006. IEEE Press.

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

[15] Yann Cooren, Maurice Clerc, and Patrick Siarry. Performance evaluation of TRIBES, an adaptive particle swarm optimization algorithm. *Swarm Intelligence*, 3(2):149–178, 2009.

[16] 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.

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

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

[19] Marco Dorigo and Thomas Stützle. *Ant Colony Optimization*. MIT Press, Cambridge, MA, 2004.

[20] Russell Eberhart and Yuhui Shi. Comparing inertia weights and constriction factors in particle swarm optimization. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2000)*, pages 84–88, Piscataway, NJ, 2000. IEEE Press.

[21] 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.

[22] 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.

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

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] Ling Hao and Lishuan Hu. Hybrid particle swarm optimization for continuous problems. In *Proceedings of the ISECS International Colloquium on Computing, Communication, Control and Management. CCCM 2009*, pages 283–286, Piscataway, NJ, 2009. IEEE Press.

[29] 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.

[30] 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.

[31] Holger H. Hoos and Thomas Stützle. *Stochastic Local Search: Foundations and Applications*. Morgan Kaufmann, San Francisco, CA, 2004.

[32] 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.

[33] 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.

[34] 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.

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

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

[37] 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.

[38] Wen-Fung Leong and Gary G. Yen. PSO-based multiobjective optimization with dynamic population size and adaptive local archives. *IEEE Transactions on Systems, Man, and Cybernetics. Part B: Cybernetics*, 38(5):1270–1293, 2008.

[39] J. J. Liang and P. N. Suganthan. Dynamic multi-swarm particle swarm optimizer with local search. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2005)*, pages 522–528, Piscataway, NJ, 2005. IEEE Press.

[40] 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.

[41] 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 (CEC 2008)*, pages 3663–3670, Piscataway, NJ, 2008. IEEE Press.

[42] 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.

[43] 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.

[44] 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/`.

[45] 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.

[46] 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.

[47] 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.

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

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

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

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

[52] 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.

[53] 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.

[54] 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.

[55] 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.

[56] 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.

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

[58] 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.

[59] 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.

[60] 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.

[61] Zhi-Hui Zhan, Jun Zhang, Yun Li, and Henry Shu-Hung Chung. Adaptive particle swarm optimization. *IEEE Transactions on Systems, Man, and Cybernetics. Part B: Cybernetics*, 39(6):1362–1381, 2009.

[62] 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.