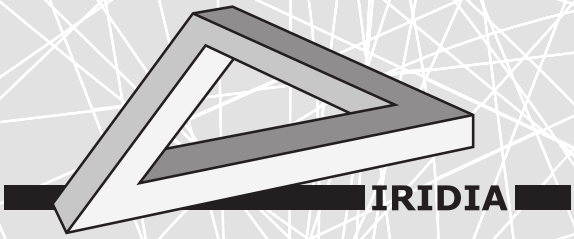

Université Libre de Bruxelles

Institut de Recherches Interdisciplinaires
et de Développements en Intelligence Artificielle

PSO-*X*: A component-based framework for the automatic design of particle swarm optimization algorithms

C.L. Camacho Villalón, M. Dorigo, and T. Stützle

IRIDIA – Technical Report Series

Technical Report No.
TR/IRIDIA/2021-002

March 2021

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/2021-002

PSO-*X*: A component-based framework for the automatic design of particle swarm optimization algorithms

Christian Leonardo Camacho Villalón · **Marco Dorigo** · **Thomas Stützle**

Abstract The particle swarm optimization (PSO) algorithm has been the object of many studies and modifications for more than twenty years. Ranging from small refinements to the incorporation of sophisticated novel ideas, the majority of modifications proposed to this algorithm have been the result of a manual process in which developers try new designs based on their own knowledge and expertise. However, manually introducing changes is very time consuming and makes the systematic exploration of all the possible algorithm configurations a difficult process. In this paper, we propose to use automatic design to overcome the limitations of having to manually find performing PSO algorithms. We develop a flexible software framework for PSO, called PSO-*X*, which is specifically designed to integrate the use of automatic configuration tools into the process of generating PSO algorithms. Our framework embodies a large number of algorithm components developed over more than twenty years of research that have allowed PSO to deal with a large variety of problems, and uses `irace`, a state-of-the-art configuration tool, to automatize the task of selecting and configuring PSO algorithms starting from these components. We show that `irace` is capable of finding high performing instances of PSO algorithms never proposed before.

Keywords Particle swarm optimization · automatic algorithm design · continuous optimization · computational intelligence.

C.L. Camacho Villalón, M. Dorigo, and T. Stützle
IRIDIA, Université Libre de Bruxelles, Brussels, Belgium
E-mail: {ccamacho,mdorigo,stuetzle}@ulb.ac.be

1 Introduction

Computational intelligence algorithms, such as particle swarm optimization (PSO) and evolutionary algorithms (EAs), are widely used to tackle complex optimization problems for which exact approaches are often impractical [46, 3]. The application of these algorithms has been shown to be instrumental in a growing number of areas where efficiently using resources, obtaining a higher degree of automation, or finding support in decision-making are needed on a regular basis. While the application of computational intelligence algorithms usually seeks higher efficiency and automation, their development is, on the contrary, mostly done following a manual approach based on the intuition and expertise of the developers [7, 55]. The manual development of such algorithms presents a number of drawbacks: it is a slow process based on trial and error; it limits the number of design alternatives that an implementation designer can explore; and it does not provide a principled way to explore the space of possible algorithms, thus making the development process difficult to reproduce.

To alleviate these issues, it has been recently proposed a development framework based on components [55, 6, 39, 38], which includes automatic configuration tools (ACTs) for creating high-performing algorithms. As opposed to manual approaches, where algorithms are typically seen as monolithic blocks with a few numerical parameters whose design is modified based on the experience and knowledge of the algorithm designer, in a *component-based* approach [7, 28, 54] algorithms are seen as a particular combination of algorithm components. The design of algorithms using a component-based approach relies on three key elements: (i) a software framework from which algorithm components can be selected, (ii) a set of rules indicating a coherent way to combine the components in the software framework, and (iii) the use of an automatic configuration tool to evaluate the performance of different designs and parameter settings.

Compared to the number of works devoted to other widely used computational intelligence algorithms, such as ant colony optimization [38], artificial bee colony [2], and simulated annealing [22], to name a few, there is no previous work on the automatic design of particle swarm optimization algorithms. To fill this gap, we propose PSO-*X*, a flexible, component-based framework containing a large number of algorithm components previously proposed in the PSO literature. In PSO-*X* each algorithm component can assume a set of different values and PSO-*X* generates a specific PSO algorithm by selecting a value for each possible component. To do so, PSO-*X* uses a generalized PSO template that is flexible enough to combine the algorithm components in many different ways, and that is sufficient to synthesize many well-known PSO variants published in the last two decades.

In our work, the rules that allow combining components use algorithm templates, as opposed to many recent works on automatic design that use grammars [39, 55, 44]. Even though algorithm templates are considered less expressive than grammars, they are generally simpler to define and still offer great flexibility when they are used to model one single class of algorithms. The algorithm template for PSO that we propose in this paper was created with the goal of replicating a number of well-known PSO algorithms proposed in the literature as well as of being flexible enough to allow devising many new ones. Most of such flexibility is achieved through the use of a generalized velocity update rule—the core component of PSO. The goal of using a generalized velocity rule is to facilitate the abstraction of the elements typically used in this algorithm component in order to allow the combination of concepts that operate at different levels of the algorithm design. For example, with our template and the generalized velocity update rule, a high-level component such as the type of distribution of all next possible particle positions can interact with specific types of perturbation and a number of strategies to compute their magnitude.

PSO-X provides two important benefits when implementing PSO algorithms: first, the possibility of easily creating many different implementations combining a wide variety of algorithm components from a single framework; second, the possibility of using automatic configuration tools to tailor implementations of PSO to specific problems according to different scenarios. Here, we aim at showing that developing PSO algorithms using PSO-X is more efficient and produces implementations capable of outperforming manually designed PSO algorithms. To assess the effectiveness of our PSO-X framework, we compare the performance of four automatically generated PSO implementations with nine of the best known variants proposed in the literature over a set of fifty benchmark problems for evaluating continuous optimizers.

The rest of the article is structured as follows. In Section 2, we start by presenting a brief review of the most important PSO concepts. In Section 3, we identify particular design choices proposed since the earliest PSO publication and discuss their functional purpose in the implementation of the algorithm. In Section 4, we introduce our PSO algorithm template and, from Sections 4.1 to 4.5, we explain how it can be used to create PSO variants. The experimental procedure we followed is explained in Section 5. The results of the experiments conducted to evaluate the performance of the algorithms instantiated from the PSO-X framework are described in Section 6. Finally, in Section 7 we conclude the paper by highlighting the advantages and limitations of our work.

2 Preliminaries

2.1 Continuous optimization problems

In this paper, we consider the application of PSO to continuous optimization problems. Without loss of generality, we consider minimization problems where the goal is to minimize a d-dimensional continuous objective function $f : S \subseteq \mathbb{R}^d \rightarrow \mathbb{R}$ by finding a vector $\mathbf{o} \in S$ such that $\forall\, \mathbf{x} \in S,\ f(\mathbf{o}) \leq f(\mathbf{x})$. The search space S is a subset of $\mathbb{R}^d$ in which a solution is represented by a real-valued vector $\mathbf{x}$, and each component x^j of $\mathbf{x}$ is constrained by a lower and upper bound such that $lb^j \leq x^j \leq ub^j$, for $j = 1, \ldots, d$. The vector $\mathbf{o}$ represents the solution for which the evaluation function $f(\cdot)$ returns the minimum value.

2.2 Particle swarm optimization

Particle swarm optimization [20] is a stochastic search algorithm where a set of "particles" search for approximate solutions of continuous optimization problems. In PSO each particle moves in the search space by repeatedly applying *velocity* and *position* update rules. Each particle i has, at every iteration t, three associated vectors: the position $\mathbf{x}_t^i$, the velocity $\mathbf{v}_t^i$, and the personal best position $\mathbf{p}_t^i$. The vector $\mathbf{x}_t^i$ is a candidate solution to the optimization problem considered whose quality is evaluated by the objective function $f(\cdot)$.

In addition to these vectors, each particle i has a set N^i of neighbors and a set I^i of informants. Set N^i contains the particles from which i can obtain information, whereas $I^i \subseteq N^i$ contains the particles that will indeed provide the information used when updating i's velocity. The way the sets N^i—which define the *topology* of the swarm [33]—and the sets I^i—known as the *model of influence*—are defined are two important design choices in PSO. Sets N^i can be defined in many different ways producing a large number of possible different topologies; the two extreme cases are the fully-connected topology, in which

all particles are in the neighborhood of all other particles, and the ring topology, where each particle is a neighbor of just two adjacent particles. Examples of other partially connected topologies include lattices, wheels, random edges, etc. The model of influence can also be defined in different ways, but the vast majority of implementations employ either the *best-of-neighborhood* which contains the particle with the best personal best solution in the neighborhood of i (which includes particle i itself), or the *fully informed* model, in which $I^i = N^i$.

In the standard PSO (SPSO) [53], the rule used to update particles' position is

$$\mathbf{x}_{t+1}^i = \mathbf{x}_t^i + \mathbf{v}_{t+1}^i, \tag{1}$$

where the velocity vector $\mathbf{v}_{t+1}^i$ of the i^{th} particle at iteration $t+1$ is computed using an update rule that involves $\mathbf{v}_t^i$, $\mathbf{p}_t^i$, and $\mathbf{l}_t^i$. The vector $\mathbf{l}_t^i$ indicates the best among the personal best positions of the particles in the neighborhood of i; formally, it is equal to $\mathbf{p}_t^k$ where $k = \arg\min_{j \in N^i}\{f(\mathbf{p}_t^j)\}$. Note that when a fully-connected topology is employed, vector $\mathbf{l}_t^i$ becomes the *global best* solution and is indicated as $\mathbf{g}_t$.

The velocity update rule of SPSO is defined as follows:

$$\mathbf{v}_{t+1}^i = \omega \mathbf{v}_t^i + \varphi_1 U_{1t}^i \left(\mathbf{p}_t^i - \mathbf{x}_t^i\right) + \varphi_2 U_{2t}^i \left(\mathbf{l}_t^i - \mathbf{x}_t^i\right), \tag{2}$$

where ω is a parameter, called inertia weight, used to control the influence of the previous velocity, and φ_1 and φ_2 are two parameters known as the acceleration coefficients that control the influence of $(\mathbf{p}_t^i - \mathbf{x}_t^i)$ and $(\mathbf{l}_t^i - \mathbf{x}_t^i)$. The goal of vectors $(\mathbf{p}_t^i - \mathbf{x}_t^i)$ and $(\mathbf{l}_t^i - \mathbf{x}_t^i)$, respectively known as the cognitive influence (CI) and the social influence (SI), is to attract particles towards high quality positions found so far. U_{1t}^i and U_{2t}^i are two $d \times d$ diagonal matrices whose diagonal values are random values drawn from $\mathscr{U}(0,1]$; their function is to induce perturbation to the CI and SI vectors.

The rule to update the personal best position of particle i is

$$\mathbf{p}_{t+1}^i = \begin{cases} \mathbf{x}_{t+1}^i, & \text{if } f(\mathbf{x}_{t+1}^i) < f(\mathbf{p}_t^i) \\ \mathbf{p}_t^i & \text{otherwise.} \end{cases} \tag{3}$$

3 Design choices in PSO

Many algorithm components have been proposed for PSO over the years [4, 5, 48, 11] with the goal of improving its performance and enabling its application to a wider variety of problems. In this article, we have categorized these algorithm components into five different groups: (1) those used to set the value of the main algorithm parameters, (2) those that control the distribution of particles positions in the search space, (3) those used to apply perturbation to the velocity and/or position vectors, (4) those regarding the construction and application of the random matrices, and (5) those related to the topology, model of influence and population size.

Group (1) comprises the time-varying and adaptive/self-adaptive *parameter control strategies* used to compute the value of ω, φ_1 and φ_2. Time-varying strategies take place at specific iterations of the algorithm execution; while adaptive and self-adaptive strategies use information related to the optimization process (e.g., particles average velocity, convergence state of the algorithm, average quality of the solutions found, etc.) to adjust the value of the parameters. Because the value of ω, φ_1 and φ_2 heavily influences the exploration/exploitation behavior of the algorithm, parameter control strategies are abundant in PSO literature [25, 26].

In particular, a lot of attention has been given to control strategies focused on adjusting the value of ω, which is intrinsically related to the local convergence of the algorithm.[1] Locally convergent implementations not only guarantee to find a local optimum in the search space, but also prevent issues such as (i) *swarm explosion*, which happens when a particle's velocity vector grows too large and the particle becomes incapable to converge to a point in the search space [17]; and (ii) *poor problem scalability*, which means that the algorithm performs poorly on high dimensional problems [10].

In group (2) are the algorithm components used to control the distribution of all next possible positions (DNPP) of the particles. The chosen DNPP determines the way particles are mapped from their current position to the next one. We consider the three main DNPP proposed in the literature—the rectangular (used in SPSO), the spherical (used in SPSO-2011 [16]) and the additive stochastic, which comprises the recombination operators[2] proposed for simple dynamic PSO algorithms [45]. Although some DNPP mappings suffer from *transformation variance*—which happens when the algorithm performs poorly under mathematical transformations of the objective function, such as scale, translation, and rotation[3]—there are a number of algorithm components that have been developed to prevent this issue.

Group (3) is composed of the algorithm components that allow to apply perturbations to the particles velocity/position vectors. In general, in PSO perturbation mechanisms can be *informed* or *random*. Informed perturbation mechanisms receive a position vector as an input (typically $\mathbf{p}_t^i$ or $\mathbf{x}_t^i$) and use it to compute a new vector that replaces the one that was received. The typical way in which informed mechanisms work is by using the components of the input vector as center of a probability distribution and mapping random values around them; however, other options found in the literature include computing the Hadamard product between the input vector and a random one, or randomly modifying the components of the input vector.[4] Differently, random perturbation mechanisms add a random value to a particle's position or velocity. Perturbation mechanisms proposed for PSO are used to improve the diversity of the solutions [58, 10], avoid stagnation [34], and avoid divergence [18]. Additionally, some of these mechanisms allow to modify the DNPP of the particles; an example is the mechanism proposed in [10], where a Gaussian distribution is used to map random points on spherical surfaces centered around the position of the informants.

One of the main challenges in most perturbation mechanisms is the determination of the perturbation magnitude: a strong perturbation may prevent particles from efficiently exploiting high quality areas of the search space, while a weak one may not produce any improvement at all. In order to allow convergent implementations to take advantage of the perturbation mechanism, some magnitude control strategies take into account the state of the optimization process to adjust the magnitude at run time. An example is [18], where a parameter decreases the perturbation magnitude when the best solution found so far has been constantly improving, whereas increases it when the algorithm is stagnating. Another

[1] In this context, convergence means that, as the number of iterations grows larger, the probability of particles reaching a stable position in which $\mathbf{x}_t^i = \mathbf{p}_t^i = \mathbf{g}_t$ and $\mathbf{V}_t^i = 0$ for all i approaches 1. See [47, 11].

[2] A recombination operator is a mapping between a set of neighbors of i and the variable q, where the variable q can be defined in different ways. The concept of recombination operator comes from evolutionary computation [3], where it is used to generate new solutions by recombining solutions in the current population.

[3] Scale variance means that the performance of the algorithm is affected by uniformly scaling all variables of the problem, while *translation* and *rotation variance* mean that the performance of the algorithm is dependent on how the coordinated axes are placed in the search space.

[4] Note that this is equivalent to the application of a *mutation operator* in evolutionary computation [3].

example is [10], where the magnitude is computed based on the Euclidean distance between the particles so as to decrease it as particles converge to the best solution found so-far.

The algorithm components in group (4) corresponds to the *random matrices*, whose function, similarly to some perturbation mechanism of group (3), is to provide diversity to particles movement. The main difference between the random matrices and the perturbation mechanisms described above is that the former can be used to produce changes in the magnitude and direction of the CI and SI vectors, while the latter allow only to apply perturbation to individual positions used in the computation of the CI and SI. In the SPSO algorithm, the random matrices (U_1^i and U_2^i, see Eq. 2) are usually constructed as diagonal matrices with values drawn from $\mathscr{U}(0,1)$; however, in some implementations of SPSO (e.g., [9]), the matrices are replaced by two random values r_1^i and r_2^i—in this case particles oscillate linearly between $\mathbf{p}_t^i$ and $\mathbf{l}_t^i$ without being able to move in different directions, preventing transformation variance [11] but affecting the performance of the algorithm. Using random rotation matrices[5] (RRMs), instead of random diagonal matrices, is another way to address transformation variance in PSO. RRMs allow to apply random changes to the length and direction of the vectors in the velocity update rule without being biased towards some particular reference frame. The two main methods that have been used to create RRMs in the context of PSO are exponential map [57] and Euclidean rotation [12].

The last group of algorithm components we identified in our work, group (5), includes the *topology*, *model of influence* and *population size*. The topology plays an important role in the way the algorithm will modulate its exploration-exploitation capabilities. In addition to the well-known fully-connected, ring and von Neumann topologies, there are other topologies that have been explored in the PSO literature, such as hierarchical and small-world network. In [42], a topology that decreases connectivity over time was proposed. Concerning the model of influence, besides the best-of-neighborhood and the fully informed, another option is the ranked fully informed model of influence [31], in which the contribution of each informant is weighted according to its rank in the neighborhood. Concerning population size, it has recently been proposed to increase or decrease the number of particles according to some metrics [15], [43]. The number of particles in the swarm has an impact on the trade-off between solution quality and speed of the algorithm [43, 11]. In general, a large population should be used as it can produce better results. However, a small population may be the best option when the objective function evaluation is expensive or when the number of possible function evaluations is limited.

4 Designing PSO algorithms from an algorithm template

In this section, we explain the way in which the algorithm components reviewed in the previous section can be combined using the PSO-*X* framework. In the reminder of this article, we use Sans Serif font to indicate the name of the algorithm components and of their options as implemented in PSO-*X*.

4.1 Algorithm template for designing PSO implementations

Algorithm 1 depicts the PSO-*X*'s algorithm template. A swarm of particles (`swarm`) is created using the INITIALIZE() procedure that assigns to each particle a set N^i, a set I^i, an

[5] A rotation matrix is an orthogonal matrix whose determinant is 1.

Algorithm 1 Algorithm template used by PSO-X

```
1: swarm ← INITIALIZE(Population, Topology, Model of influence)
2: repeat
3:     for i ← 1 to size(swarm) do
4:         compute f(x_t^i)
5:         update p_t^i using Eq. 3
6:         v_{t+1}^i ← ω1 v_t^i + ω2 DNPP(i,t) + ω3 Pert_rand(i,t)
7:         x_{t+1}^i ← x_t^i + v_{t+1}^i
8:     end for
9:     if type(Population) ≠ constant then
10:        swarm ← UPDATEPOPULATION(swarm, Population)
11:    end if
12:    if type(Topology) = time-varying or type(Population) ≠ constant then
13:        swarm ← UPDATETOPOLOGY(swarm, Topology, Model of influence)
14:    end if
15: until termination criterion is met
16: return global best solution
```

initial position and an initial velocity based on the Population, Topology and Model of influence indicated by the framework user. Additionally, the INITIALIZE() procedure creates and initializes any variable required to use the algorithm components included in the implementation. The **for** cycle of lines 3 to 8 corresponds to the standard implementation of PSO—except for line 6 that shows our generalized velocity update rule (GVUR), defined as follows:

$$\mathbf{v}_{t+1}^i = \omega_1 \, \mathbf{v}_t^i + \omega_2 \, \mathsf{DNPP}(i,t) + \omega_3 \, \mathsf{Pert}_{\mathsf{rand}}(i,t), \tag{4}$$

where DNPP represents the type of mapping from a particle's current position to the next one, and $\mathsf{Pert}_{\mathsf{rand}}$ represents an additive perturbation mechanism. The parameter ω_1 is the same as the inertia weight in SPSO (see Eq. 2) and its value can be computed using the strategies that have been developed for this purpose (see list in Table 1). The parameters ω_2 and ω_3 control the influence that will be given to the DNPP and $\mathsf{Pert}_{\mathsf{rand}}$ components; their vales are computed using one of the strategies indicated in Table 1. We use three independent ω parameters so that it is easy to disable any of the GVUR components. For example, a velocity free PSO can be easily obtained by setting $\omega_1 = 0$.

After all particles have updated their position, two procedures can take place: UPDATEPOPULATION(), that increases/decreases the size of the swarm according to the type of Population employed; and UPDATETOPOLOGY(), that connects newly added particles to a set of neighbors, or disconnect particles when the topology connectivity reduces over time.

4.2 DNPP component

The six options defined in PSO-X for the DNPP component are DNPP-rectangular, DNPP-spherical, DNPP-standard, DNPP-discrete, DNPP-Gaussian and DNPP-Cauchy–Gaussian.

The DNPP-rectangular option is defined as follows:

$$\mathsf{DNPP\text{-}rectangular} = \sum_{k \in I_t^i}^{n} \varphi_t^k \, \mathsf{Mtx}_t^k \left(\mathsf{Pert}_{\mathsf{info}}(\mathbf{p}_t^k) - \mathbf{x}_t^i \right), \tag{5}$$

where Mtx and $\mathsf{Pert}_{\mathsf{info}}$ are, as mentioned before, high-level representations of the different types of random matrices and informed perturbation mechanisms used in PSO.

Table 1: Available strategies for computing the value of the GVUR parameters ω_1, ω_2 and ω_3 and their mathematical definition

Parameter	Strategy	Mathematical definition
ω_1 *	constant	self-explanatory
	linear decreasing	$\omega_{1t} = \omega_{1max} - (\omega_{1max} - \omega_{1min})\frac{t}{t_{max}}$
	linear increasing	$\omega_{1t} = \omega_{1min} - (\omega_{1min} - \omega_{1max})\frac{t}{t_{max}}$
	random	$\omega_{1t} \sim \mathscr{U}[0.5, 1)$
	self-regulating	$\omega_{1t}^i = \begin{cases} \omega_{1t-1}^i + \eta\Delta\omega & \text{if } \mathbf{p}_t^i = \mathbf{g}_t \\ \omega_{1t-1}^i - \Delta\omega & \text{otherwise} \end{cases}$, where $\Delta\omega = \frac{(\omega_{1max} - \omega_{1min})}{t_{max}}$ and η is a user selected parameter
	adaptive based on velocity	$\omega_{1t} = \begin{cases} \arg\max\{\omega_{1t-1} - \lambda, \omega_{1min}\} & \text{if } \overline{v_t} \geq v_{t+1}^{\text{ideal}} \\ \arg\min\{\omega_{1t-1} + \lambda, \omega_{1max}\} & \text{otherwise} \end{cases}$, where λ is a user selected parameter, $\overline{v_t} = \frac{1}{n \cdot d}\sum_{i=1}^{n}\sum_{j=1}^{d}\lvert v_t^{i,j}\rvert$, $v_t^{\text{ideal}} = v_s\left(\frac{1+\cos\left(\pi\frac{1}{0.95 \cdot t_{max}}\right)}{2}\right)$ and $v_s = \frac{x_{max} - x_{min}}{2}$
	double exponential self-adaptive	$\omega_{1t}^i = e^{-e^{-R_t^i}}$, where $R_t^i = \lvert\lvert\mathbf{p}_t^{\text{best}} - \mathbf{x}_t^i\rvert\rvert(\frac{t_{max} - t}{t_{max}})$
	rank-based	$\omega_{1t} = \omega_{1min} + (\omega_{1max} - \omega_{1min})\frac{\mathscr{R}_t(i)}{n}$, where $\mathscr{R}_t(i)$ is a function that returns the fitness rank of i
	success-based	$\omega_{1t} = \omega_{1min} + (\omega_{1max} - \omega_{1min})\frac{\sum_{i=1}^{n} S_t^i}{n}$, where $S_t^i = \begin{cases} 1 & \text{if } f(\mathbf{p}_t^i) < f(\mathbf{p}_{t-1}^i) \\ 0 & \text{otherwise} \end{cases}$
	convergence-based	$\omega_{1t}^i = 1 - \left\lvert\frac{a - C_t^i}{(1+D_t^i)(1+b)}\right\rvert$, where $a, b \in [0,1]$ are user selected parameters, $C_t^i = \frac{\lvert f(\mathbf{p}_{t-1}^i) - f(\mathbf{p}_t^i)\rvert}{f(\mathbf{p}_{t-1}^i) - f(\mathbf{p}_t^i)}$ and $D_t^i = \frac{\lvert f(\mathbf{p}_t^i) - f(\mathbf{p}_t^{\text{best}})\rvert}{f(\mathbf{p}_t^i) - f(\mathbf{p}_t^{\text{best}})}$
ω_2, ω_3 **	$= \omega_1$	$\omega_2 = \omega_1$, $\omega_3 = \omega_1$
	random	$\omega_{2t}, \omega_{3t} \sim \mathscr{U}[0.5, 1]$
	constant	ω_2, ω_3 are user selected constants in the interval $[0, 1]$

* In the strategies for computing ω_1, t_{max} indicate the maximum number of iterations set for the algorithm and ω_{1min}, ω_{1max} are user selected constants for the minimum and maximum allowable value of ω_1.

** Note that different combinations are possible, for example $\omega_2 = \omega_1$ and $\omega_3 = \text{random}$.

The DNPP-rectangular is by far the most commonly used in implementations of PSO, including SPSO [53], the constriction coefficient PSO [17], the fully informed PSO [40], etc. In the standard application of DNPP-rectangular (i.e., as in SPSO), each term added in Eq. 5 is a vector located on a hyper-rectangular surface whose side length depends on the distance between $\mathbf{p}_t^k$ and $\mathbf{x}_t^i$. However, when the perturbation component of DNPP-rectangular is an informed Gaussian—as in the locally convergent rotationally invariant PSO (LcRPSO) [10]—or a random rotation matrix (RRM)—as in the diverse rotationally invariant PSO (DvRPSO) [57]—the surface on which the different vectors computed in Eq. 5 are located becomes hyperspherical or semi-hyperspherical, respectively.

Another option is DNPP-spherical [16][59], where a vector located on a hyper-sphere is used in the computation of a particle new position. The equation to compute the DNPP-spherical option is the following:

$$\text{DNPP-spherical} = \mathscr{H}_i\big(\mathbf{c}_t^i, |\mathbf{c}_t^i - \mathbf{x}_t^i|\big) - \mathbf{x}_t^i, \tag{6}$$

where $\mathscr{H}(\mathbf{c}_t^i, |\mathbf{c}_t^i - \mathbf{x}_t^i|)$ is a random point drawn from a hyperspherical distribution with center $\mathbf{c}_t^i$ and radius $|\mathbf{c}_t^i - \mathbf{x}_t^i|$. The center $\mathbf{c}_t^i$ is computed as follows:

$$\mathbf{c}_t^i = \frac{\mathbf{x}_t^i + \mathbf{L}_t^i + \mathbf{P}_t^i}{3}, \tag{7}$$

where

$$\mathbf{P}_t^i = \mathbf{x}_t^i + \varphi_{1t}\,\mathsf{Mtx}_t\big(\mathsf{Pert}_{\mathsf{info}}(\mathbf{p}_t^i) - \mathbf{x}_t^i\big), \tag{8}$$

$$\mathbf{L}_t^i = \mathbf{x}_t^i + \sum_{k \in I_t^i \setminus \{i\}}^{n} \varphi_{2t}^k\,\mathsf{Mtx}_t^k\big(\mathsf{Pert}_{\mathsf{info}}(\mathbf{p}_t^k) - \mathbf{x}_t^i\big), \tag{9}$$

and $\varphi_{2t}^k = \frac{\varphi_{2t}}{|I_t^i \setminus \{i\}|}$. The main difference between DNPP-spherical and the standard implementation of DNPP-rectangular is that the hypersphere $\mathscr{H}_i\big(\mathbf{c}_t^i, |\mathbf{c}_t^i - \mathbf{x}_t^i|\big)$ is invariant to rotation around its center, whereas DNPP-rectangular is rotation variant unless another component is used to overcome this issue—e.g., a Gaussian perturbation, as done in the LcRPSO variant. While the DNPP-spherical and the LcRPSO combining the DNPP-rectangular with a Gaussian perturbation component use the same idea, they work in a different way. In the DNPP-spherical DNPP, there is a single vector mapped randomly in the hypersphere $\mathscr{H}(\mathbf{c}_t^i, |\mathbf{c}_t^i - \mathbf{x}_t^i|)$ and the informants of i participate only in the computation of vector $\mathbf{L}_t^i$ (see Eq. 9)[6]; whereas in the LcRPSO variant, there are n different vectors, one for each informant of i, each mapped on a spherical surface, and the new velocity of the particle is obtained by adding all n vector, as shown in Eq. 5.

The DNPP-standard, DNPP-discrete, DNPP-Gaussian and DNPP-Cauchy–Gaussian options belong to the class of simple dynamic PSO algorithms [45, 35] and have the form $\mathbf{q}_t^i - \mathbf{x}_t^i$, where vector $\mathbf{q}_t^i$ is computed differently in each option:

$$\text{DNPP-standard}: \ \mathbf{q}_t^i = \frac{\varphi_1 \mathbf{p}_t'^i + \varphi_2 \mathbf{p}_t'^k}{\varphi_1 + \varphi_2} \tag{10}$$

$$\text{DNPP-discrete}: \ \mathbf{q}_t^i = \eta_d\,\mathbf{p}_t'^i + (1 - \eta_d)\,\mathbf{p}_t'^k) \tag{11}$$

$$\text{DNPP-Gaussian}: \ \mathbf{q}_t^i = \mathscr{N}\big(\frac{\mathbf{p}_t'^i + \mathbf{p}_t'^k}{2}, |\,\mathbf{p}_t'^i - \mathbf{p}_t'^k|\big) \tag{12}$$

$$\text{DNPP-Cauchy–Gaussian}: \ \mathbf{q}_t^i = \begin{cases} p_t^{i,j} + \mathscr{C}(1)|p_t'^{i,j} - p_t'^{k,j}| & \text{if } \mathscr{U}[0,1] \leq r \\ p_t^{k,j} + \mathscr{N}(0,1)|p_t'^{i,j} - p_t'^{k,j}| & \text{otherwise} \end{cases} \tag{13}$$

[6] In the original definition of Eq. 9, vector $\mathbf{L}_t^i$ was defined considering a best-of-neighborhood model of influence. In this article, we have extended the computation of $\mathbf{L}_t^i$ to an arbitrary number of informants.

where $\eta_d \sim \mathscr{U}\{0,1\}$ is a discrete random number drawn from a Bernoulli distribution, $\mathscr{C}(1)$ is a random number generated using a Cauchy distribution with scaling parameter 1, $\mathscr{N}(0,1)$ is a random number from a Normal distribution with mean 0 and variance 1, and r is a parameter that allows the user to select the probability with which the Cauchy or the Normal distributions are used in Eq. 13. Vectors $\mathbf{p}_t'^i$ and $\mathbf{p}_t'^k$ are computed using $\mathbf{p}_t'^i = \mathsf{Pert}_{\mathsf{info}}(\mathbf{p}_t^i)$ and $\mathbf{p}_t'^k = \mathsf{Pert}_{\mathsf{info}}(\mathbf{p}_t^k)$ with $k \in I^i$.

Unlike options DNPP-standard, DNPP-discrete and DNPP-Gaussian, where the mapping between particles i and k is deterministic, in DNPP-Cauchy–Gaussian, the value of the j^{th} dimension of $\mathbf{q}_t^i$ is computed with probability r using $\mathbf{p}_t^i$ and a Cauchy distribution; and with probability $1-r$ using $\mathbf{p}_t^k$ and a Normal distribution. Although we kept the original definition of these DNPPs for the most part, we did two modifications: we included the $\mathsf{Pert}_{\mathsf{info}}$ component (i.e., vectors $\mathbf{p}_t'^i$ and $\mathbf{p}_t'^k$ instead of $\mathbf{p}_t^i$, $\mathbf{p}_t^k$) and the possibility of using a random informant model of influence (MoI-random informant), which consists in choosing a random particle from N^i and use it as informant.

4.3 $\mathsf{Pert}_{\mathsf{rand}}$ and $\mathsf{Pert}_{\mathsf{info}}$ components

The two types of perturbation components included in PSO-X are: $\mathsf{Pert}_{\mathsf{info}}$, which modifies an input vector; and $\mathsf{Pert}_{\mathsf{rand}}$, which generates a random vector that is added to the velocity vector. $\mathsf{Pert}_{\mathsf{info}}$, as explained in Section 4.2, is a component used by the DNPP component. Differently, $\mathsf{Pert}_{\mathsf{rand}}$ is used directly in the generalized velocity update rule.

Table 2: Options for computing $\mathsf{Pert}_{\mathsf{info}}$ and $\mathsf{Pert}_{\mathsf{rand}}$ components in PSO-X when they are used in the implementation

Component	Option	Definition
$\mathsf{Pert}_{\mathsf{info}}$ *	$\mathsf{Pert}_{\mathsf{info}}$-Gaussian	$\mathscr{N}(\mathbf{r},\sigma_t)$
	$\mathsf{Pert}_{\mathsf{info}}$-Lévy	$L_{\gamma_t}(\mathbf{r},\sigma_t)$
	$\mathsf{Pert}_{\mathsf{info}}$-uniform	$\mathbf{r}+(\mathbf{s}\odot\mathbf{r})$, with $\mathbf{s}\sim\mathscr{U}[-b_t,b_t]$
$\mathsf{Pert}_{\mathsf{rand}}$ **	$\mathsf{Pert}_{\mathsf{rand}}$-rectangular	$\tau_t\,(1-2\cdot\mathscr{U}(0,1))$
	$\mathsf{Pert}_{\mathsf{rand}}$-noisy	$\mathscr{U}[-\delta_t/2,\delta_t/2]$

* In the options for computing $\mathsf{Pert}_{\mathsf{info}}$: $\mathbf{r}$ is the input vector; $\mathscr{N}(\mathbf{r},\sigma_t)$ is a Normal distribution with mean $\mathbf{r}$ and variance σ_t; $L_\gamma(\mathbf{r},\sigma_t)$ is a Lévy distribution with mean $\mathbf{r}$, variance σ_t, and scale parameter γ_t; and b_t is a real parameter.

** In the options for computing $\mathsf{Pert}_{\mathsf{rand}}$: τ_t and δ_t are two real parameters.

As shown in Table 2, both $\mathsf{Pert}_{\mathsf{info}}$ and $\mathsf{Pert}_{\mathsf{rand}}$ are optional components in PSO-X that can be omitted from the implementation using the none option. The options for $\mathsf{Pert}_{\mathsf{info}}$, when the component is present in the implementation, are $\mathsf{Pert}_{\mathsf{info}}$-Gaussian, $\mathsf{Pert}_{\mathsf{info}}$-Lévy, and $\mathsf{Pert}_{\mathsf{info}}$-uniform. $\mathsf{Pert}_{\mathsf{info}}$-Gaussian and $\mathsf{Pert}_{\mathsf{info}}$-Lévy compute a random vector by using a probability distribution whose center and dispersion are given by the input vector $\mathbf{r}$ and by the parameter σ_t that controls the magnitude of the perturbation. Similarly, in $\mathsf{Pert}_{\mathsf{info}}$-uniform, the perturbation magnitude depends on a parameter b_t, that controls the interval in which a random vector $\mathbf{s}$ will be generated using a uniform distribution. Regarding the $\mathsf{Pert}_{\mathsf{rand}}$ component, both $\mathsf{Pert}_{\mathsf{rand}}$-rectangular and $\mathsf{Pert}_{\mathsf{rand}}$-noisy employ a random uniform distribution

to generate a random vector; the magnitude of the perturbation is controlled in this case by parameters τ_t and δ_t, respectively.

In $\mathsf{Pert}_{\mathsf{info}}$-Lévy, the value of γ_t can be used to switch between a Gaussian and a Cauchy distribution [50]. That is, when $\gamma_t = 1$, the Lévy distribution is equivalent to the Gaussian distribution, and when $\gamma_t = 2$, it is equivalent to the Cauchy distribution. In PSO-X, the value of γ_t is obtained sampling from the discrete uniform distribution $\mathscr{U}\{10,20\}$:

$$\gamma_t = \mathscr{U}\{10,20\}/10.$$

This allows to vary the probability of generating a random value in the tail of the distribution. This way of computing the value of γ_t is similar to the one used in [35] for computing the DNPP-Cauchy–Gaussian option assuming $r = 0.5$ to give the same probability to each case—see Eq. 13.

Since the perturbation magnitude (PM) plays a critical role in the effectiveness of perturbation components, setting its value (either offline or during the algorithm execution) is often challenging. In PSO-X we implemented four strategies for computing the PM that can be used with any of the $\mathsf{Pert}_{\mathsf{info}}$ and $\mathsf{Pert}_{\mathsf{rand}}$ components. These strategies are PM-constant value, PM-Euclidean distance, PM-obj.func. distance, and PM-success rate.

The PM-constant value strategy [58] is the simplest and consists in using a value that remains constant during the execution of the algorithm. This strategy guarantees that the perturbation magnitude is always greater than zero—a condition that has to be verified for all perturbation strategies. However, the main problem with the PM-constant value strategy is that using the same value may not be effective for the different stages of the optimization process. For example, particles that are farther away from the global best solution may benefit from a large PM value in order to move to higher quality areas, while for those particles that are near the global best solution, a small PM value would make exploitation easier.

The PM-Euclidean distance strategy [10] consists in using the Euclidean distance between the current position of particle i and the personal best of a neighbor k. This strategy is defined as follows:

$$\mathrm{PM}_t^{i,k} = \begin{cases} \varepsilon \cdot \mathrm{PM}_{t-1}^{i,k} & \text{if } \mathbf{x}_t^i = \mathbf{p}_t^k \\ \varepsilon \cdot \sqrt{\sum_{j=1}^{d} (\mathbf{x}_t^{i,j} - \mathbf{p}_t^{k,j})^2} & \text{otherwise} \end{cases}, \tag{14}$$

where $0 < \varepsilon \leq 1$ is a parameter used to weigh the distance between $\mathbf{x}_t^i$ and $\mathbf{p}_t^k$.

The PM-obj.func. distance is very similar to the PM-Euclidean distance, but the distance between particles is measured in terms of the quality of the solutions. The equation to compute the PM using PM-obj.func. distance is

$$\mathrm{PM}_t^i = \begin{cases} m \cdot \mathrm{PM}_{t-1}^i & \text{if } \mathbf{p}_t^i = \mathbf{l}_t^i \\ m \cdot \frac{f(\mathbf{l}_t^i) - f(\mathbf{x}_t^i)}{f(\mathbf{l}_t^i)} & \text{otherwise} \end{cases}, \tag{15}$$

where $0 < m \leq 1$ is a parameter. For particles whose quality is very similar to that of the local best, the PM will be small, enhancing exploitation; and for those whose quality is poor compared to that of the local best, the PM will be large allowing them move to far areas of the search space.

The mechanism implemented in PM-success rate [18] to compute the PM takes into account the success rate of the algorithm in terms of improving the best solution's quality. The value of the PM is adjusted depending on the number of consecutive iterations in which the swarm has succeeded (#successes) or failed (#failures) to improve the best solution found

so far, where iteration $t \rightarrow t+1$ is a success if $f(\mathbf{g}_{t+1}) < f(\mathbf{g}_t)$, a failure otherwise. The PM-success rate strategy is defined as follows:

$$\text{PM} = \begin{cases} \text{PM} \cdot 2 & \text{if \#successes} > s_c \\ \text{PM} \cdot 0.5 & \text{if \#failures} > f_c \\ \text{PM} & \text{otherwise} \end{cases}, \tag{16}$$

where the threshold parameters s_c and f_c are user defined.

4.4 Mtx component

The options for the Mtx algorithm component in PSO-X are Mtx-random diagonal, Mtx-random linear, Mtx-exponential map, and Mtx-Euclidean rotation. The Mtx-random diagonal and Mtx-random linear options are both $d \times d$ diagonal matrices whose values are drawn from a $\mathscr{U}(0,1)$; the only difference between them is that, in Mtx-random linear, one random value is repeated d times in the matrix diagonal, whereas, in Mtx-random diagonal, the matrix contains d independently sampled values.

The Mtx-exponential map [57] option is based on an approximation method called exponential map [41] whereby RRMs can be constructed avoiding matrix multiplication, which is computationally expensive. Mtx-exponential map is defined as:

$$\text{Mtx-exponential map} = I + \sum_{\beta=1}^{max_\beta} \frac{1}{\beta!} \Big(\frac{\alpha\pi}{180} \big(A - A^T\big) \Big), \tag{17}$$

where I is the identity matrix, α is a scalar representing the rotation angle, and A is an $n \times n$ random matrix with uniform random numbers in $[-0.5, 0.5]$. To keep the computational complexity low we set $max_\beta = 1$.[7]

The last option for the Mtx component is Mtx-Euclidean rotation [12] that rotates a vector in any combination of planes.[8] An Mtx-Euclidean rotation for rotating axis x_i in the direction of x_j by the angle α is given by a matrix $[r_{mn}]$ with $r_{ii} = r_{jj} = \cos\alpha$, $r_{ij} = -\sin\alpha$, $r_{ji} = \sin\alpha$, and the remaining values are set to 1 if they are on the diagonal or to zero otherwise. Since $[r_{mn}]$ is an identity matrix except for the entries at the intersections between rows i and j and columns i and j, the multiplication between $[r_{mn}]$ and $\mathbf{v}$ is done as follows:

$$[r_{mn}]\mathbf{v} = \begin{cases} v_k r_{ii} + v_j r_{ji} & \text{if } k = i \\ v_k r_{jj} + v_i r_{ij} & \text{if } k = j \\ v_k & \text{otherwise} \end{cases}, \tag{18}$$

where v_k indicates the k^{th} entry of vector $\mathbf{v}$. We use Mtx-Euclidean rotation$_{all}$ to indicate when Mtx-Euclidean rotation is used to rotate a vector in all possible combination of planes, and Mtx-Euclidean rotation$_{one}$ to indicate when it is used to rotate in only one plane.

[7] This choice determines a low accuracy in the computation of the rotation matrix; however, this is not important in our case as we are computing a random rotation matrix and therefore a high precision is not necessary.

[8] For a d-dimensional vector $\mathbf{u}$, there is a composition of $d(d-1)/2$ dimensional rotation matrices built up in order to rotate $\mathbf{u}$ in all possible combinations of planes. See [19] and [12, Appendix III] for further details.

The strategies to compute the rotation angle are α-constant, α-Gaussian and α-adaptive. In α-constant, the value of α is defined by the user, whereas in α-Gaussian and α-adaptive, it is obtained by sampling values from $\mathcal{N}(0,\sigma)$. The value of σ when the Gaussian distribution is used can be a user defined parameter, as in α-Gaussian, or be computed using an adaptive approach, as in α-adaptive, which is defined as follows:

$$\sigma = \frac{\zeta \times ir_t}{\sqrt{d}} + \rho, \tag{19}$$

where ζ and ρ are two parameters and ir_t is the number of improved particles in the last iteration divided by the population size.

4.5 Topology, Model of influence and Population components

In addition to the well-known options for the Topology component discussed in Section 2.2 and 3 and showed in Table 3, we implemented in PSO-X the Top-hierarchical and Top-time-varying options.

Table 3: Available options in PSO-X for Population, Topology and Model of influence algorithm components

Component	Option
Topology	Top-ring Top-fully-connected Top-Von Neumann Top-random edge Top-hierarchical Top-time-varying
Model of influence	MoI-best-of-neighborhood MoI-fully informed MoI-ranked fully informed MoI-random informant
Population	Pop-constant Pop-time-varying Pop-incremental
Initialization	{ Init-random, Init-horizontal }

In Top-hierarchical [30], particles are arranged in a regular tree—i.e., a tree graph with a maximum branching degree (bd) and height (h)—where they move up and down based on the quality of their $\mathbf{p}_t$ vector, and sets N^i contain only the particles that are in the same branch of the tree as particle i but in a higher position. The topology is updated at the end of each iteration starting from the root node, and consists of each particle comparing the quality of its $\mathbf{p}_t$ vector with that of its parent and switching places when it has higher quality.

The Top-time-varying [42] is a topology that reduces its connectivity over time: it starts as a fully-connected topology and every κ iterations a number of edges is randomly removed from the graph until the topology is transformed into a ring. The value of κ, which controls the velocity at which the topology is transformed, is a multiple of the number of particles

in the swarm, so that the larger the value of κ the faster the topology will be disconnected. Additionally, the number of edges to be removed follows an arithmetic regression pattern of the form $n-2, n-3, \ldots, 2$, where n is the swarm size.

The options for the Model of influence component are MoI-best-of-neighborhood, where sets I^i contains i and the local best particle in the neighborhood of i; the MoI-fully informed, where sets $I^i = N^i$; MoI-ranked fully informed, which is similar to the MoI-fully informed, but particles in I^i are ranked according to their quality so that the influence of a particle with rank r is twice the influence of a particle with rank $r-1$; and MoI-random informant, which allows particles to select a random neighbor from N^i to form set I^i.

The options for the Population component are Pop-constant, Pop-time-varying and Pop-incremental. In Pop-time-varying [29], there is a maximum ($\texttt{pop}_{max}$) and minimum ($\texttt{pop}_{min}$) number of particles that can be in the swarm at any given time. Particles are added or removed according to two criteria: (i) add one particle if the best solution found has not improved in the previous k consecutive iterations and the swarm size is smaller than $\texttt{pop}_{max}$; and (ii) remove the particle with lowest quality if the best solution found has improved in the previous k consecutive iterations and the swarm size is larger than $\texttt{pop}_{min}$. Whenever criterion (i) is verified, but the swarm size is equal to $\texttt{pop}_{max}$, the particle with lowest quality is removed before adding the new random particle.

In Pop-incremental [43], the algorithm starts with an initial number of particles ($\texttt{pop}_{ini}$) and, at each iteration, there are ξ new particles added to the swarm until a maximum number is reached ($\texttt{pop}_{fin}$).

The initial position of newly added particles (x^{new}) can be computed using Init-random or Init-horizontal. In Init-random,

$$x^{\text{new},j} = \mathscr{U}[lb^j, ub^j],$$

where lb^j and ub^j are the lower and upper bound of the jth dimension of the search space. In Init-horizontal, an horizontal learning approach is applied to $x^{\text{new},j}$ after it has been randomly initialized in the search space:

$$\begin{aligned} x'^{\text{new},j} &= \mathscr{U}[lb^j, ub^j], \\ x^{\text{new},j} &= x'^{\text{new},j} + \mathscr{U}[0,1) \cdot (g_t^j - x'^{\text{new},j}). \end{aligned}$$

Using a dynamic population requires that the topology is updated in order to assign newly added particles to a neighborhood or to reconnect particles that were connected to a particle that was removed. This is handled as follows:

(i) *Particles are added to a fixed topology*—the topology is extended by connecting a newly added particle with a set of neighbors randomly chosen. In Top-hierarchical, new particles are always placed at the bottom of the tree.
(ii) *Particles are added to a time-varying topology*—we assign $\hat{C}_t^i$ neighbors to every new particle, where $\hat{C}_t^i$ is the average number of neighbors that every particle in the swarm has at iteration t.
(iii) *Particles are removed*—the topology is repaired to ensure that every particle has the right number of neighbors.

4.6 Acceleration coefficients

There are four strategies that can be used to computed the acceleration coefficients (ACs) in our framework: AC-constant, AC-random, AC-time-varying and AC-extrapolated. In AC-random,

the value of φ_{1t} and φ_{2t} is drawn from $\mathscr{U}[\varphi_{min}, \varphi_{max}]$, where $0 \leq \varphi_{min} \leq \varphi_{max} \leq 2.5$ are user selected parameters. The AC-time-varying strategy is the one proposed in the "self-organizing hierarchical PSO with time-varying acceleration coefficients" [49], where φ_1 decreases from 2.5 to 0.5 while φ_2 increases from 0.5 to 2.5. In the AC-extrapolated strategy, proposed in [1], the value of the acceleration coefficients is a function of the iteration number and particles quality computed as follows:

$$\begin{aligned} \varphi_1 &= e^{-(t/t_{max.})} \\ \varphi_2 &= e^{(\varphi_1 \cdot \Lambda_t^i)} \end{aligned} \tag{20}$$

where $\Lambda_t^i = |(f(\mathbf{l}_t^i) - f(\mathbf{x}_t^i))/f(\mathbf{l}_t)|$ adjusts the value of φ_2 in terms of the difference between $f(\mathbf{x}_t^i)$ and $f(\mathbf{l}_t^i)$. This means that when $f(\mathbf{l}_t^i) \ll f(\mathbf{x}_t^i)$, the step size of the particle will be larger, and when $f(\mathbf{l}_t^i) \approxeq f(\mathbf{x}_t^i)$ it will be smaller.

4.7 Reinitialization components and velocity clamping

The last group of components in PSO-X have been proposed with the goal of avoiding performance issues that affect PSO, such as divergence and stagnation.

The first ones is stagnation detection [52]. It is used to perturb the velocity vector of a particle when its current position is too close to the global best solution, and the velocity magnitude is not large enough to let the particle move to other parts of the search space. That is, when $||\mathbf{v}_t^i|| + ||\mathbf{g}_t - \mathbf{x}_t^i|| \leq \mu$, where $\mu > 0$ is a user defined threshold for the perturbation to occur. When the stagnation condition is verified, the velocity vector of the particle is randomly regenerated as follows:

$$\mathbf{v}_t^i = (2\mathbf{r} - 1) \cdot \mu,$$

where $\mathbf{r} \sim \mathscr{U}(0, 1]$.

The second component, particles reinitialization [23] is used to regenerate the position vector of the particles in case of early stagnation or ineffective movement is occurring. Early stagnation is considered to be affecting the implementation when the standard deviation of the $\mathbf{p}_t$ vectors is lower than 0.001. In this case, each entry of the particles position vector is randomly reinitialized with probability $1/d$. The second criterion, which tries to identify when particles are moving ineffectively, consists in detecting when the overall change of $\mathbf{g}_t$ is lower than 10^{-8} for $10 \cdot d/\texttt{pop}$ iterations and regenerating particles positions using the following equation:

$$x_{t+1}^{i,j} = (g_t^j - x_t^{i,j})/2 \text{ for } j = 1, \ldots, d.$$

The last ones is velocity clamping [20, 21] and consists in restricting the values of each dimension in the velocity vector of a particle within certain limits to prevent overly large steps. This is done using the following equation:

$$\mathbf{v}_{t+1}^j \begin{cases} v^{max} & \text{if } \mathbf{v}_{t+1}^j > v^{max} \\ -v^{max} & \text{if } \mathbf{v}_{t+1}^j < -v^{max} \\ \mathbf{v}_{t+1}^j & \text{otherwise} \end{cases}, \tag{21}$$

where v_{max}^j and $-v_{max}^j$ are maximum and minimum allowable value for the particle's velocity in dimension j. The value $v_{max}^j = \frac{ub^j - lb^j}{2}$ is set according to the lower lb^j and upper ub^j bounds for dimension j on the search space.

5 Experimental procedure

5.1 Automatic configuration

To automatically create and configure implementations from PSO-X, we employed a state-of-the-art offline configuration tool called `irace` [37]. This tool implements a mechanism called iterated racing, that consists of the following steps. First, it samples candidate configurations from the parameter space. Second, it evaluates the candidate configurations on a set of instances by means of races, whereby each candidate configuration is run on one instance at a time. Third, it discards the statistically worse candidate configurations on the basis of statistical tests (e.g., Friedman's non-parametric two-way analysis of variance by ranks). During the configuration process, which is done sequentially for a given computational budget, `irace` adjusts the sampling distribution in order to bias new samplings towards the best configurations found so far. When the computational budget is over, `irace` returns the configuration that performed best over the set of training instances. `irace` is cable of handling the different types of parameters included in our framework, that is, numerical, categorical, and conditional parameters. Numerical parameters, for example, ω, φ_1 or φ_2, have an explicit order in their value, while categorical parameters do not have any order or sensible distance measure, such as the different options for Topology. On the other hand, conditional parameters are only necessary for particular values of other parameters, for example s_c and f_c, which have to be configured only when the PM-success rate strategy is used in the implementation.

5.2 Benchmark problems

We conducted experiments on a set of 50 benchmark continuous functions belonging to the CEC'05 and CEC'14 "Special Session on Single Objective Real-Parameter Optimization" [56, 36], and to the Soft Computing (SOCO'10) "Test Suite on Scalability of Evolutionary Algorithms and other Metaheuristics for Large Scale Continuous Optimization Problems" [27]. A detailed description of the benchmark functions can be found in the given references and in the supplementary material web page of this article [13].

The test set of continuous functions—Table 4—is composed of 12 unimodal functions (f_{1-12}), 14 multimodal functions (f_{13-26}), and 24 hybrid composition functions (f_{27-50}). With the exception of f_{41}, none of the hybrid composition functions is separable, and the ones from f_{42-50} include also a rotation in the objective function.

5.3 Experimental setup

The computational budget used with `irace` was of 50 000 executions for creating the PSO-X algorithms and of 15 000 executions for tuning the parameter of the PSO variants included in our comparison. The reason for using different budgets is that there are 57 parameters involved in the creation of the PSO-X algorithms, and only between 5 and 10 parameters in the tuning of the PSO variants. The functions employed for creating and configuring the algorithms with `irace` (i.e., the training instances) used $d = 30$, and the ones used for our experimental evaluation used $d = 50$ and $d = 100$, depending on the scalability of each function.

Table 4: Benchmark functions

$f_{\#}$	Name	Search Range	Suite	$f_{\#}$	Name	Search range	Suite
f_1	Shifted Sphere	[-100,100]	SOCO	f_{26}	Shifted Rotated HGBat	[-100,100]	CEC'14
f_2	Shifted Rotated High Conditioned Elliptic	[-100,100]	CEC'14	f_{27}	Hybrid Function 1 (N = 2)	[-100,100]	SOCO
f_3	Shifted Rotated Bent Cigar	[-100,100]	CEC'14	f_{28}	Hybrid Function 2 (N = 2)	[-100,100]	SOCO
f_4	Shifted Rotated Discus	[-100,100]	CEC'14	f_{29}	Hybrid Function 3 (N = 2)	[-5,5]	SOCO
f_5	Shifted Schwefel 22.1	[-100,100]	SOCO	f_{30}	Hybrid Function 4 (N = 2)	[-10,10]	SOCO
f_6	Shifted Rotated Schwefel 1.2	[-65.536,65.536]	SOCO	f_{31}	Hybrid Function 7 (N = 2)	[-100,100]	SOCO
f_7	Shifted Scfewels12 noise in fitness	[-100,100]	CEC'05	f_{32}	Hybrid Function 8 (N = 2)	[-100,100]	SOCO
f_8	Shifted Schwefel 2.22	[-10,10]	SOCO	f_{33}	Hybrid Function 9 (N = 2)	[-5,5]	SOCO
f_9	Shifted Extended f_{10}	[-100,100]	SOCO	f_{34}	Hybrid Function 10 (N = 2)	[-10,10]	SOCO
f_{10}	Shifted Bohachevsky	[-100,100]	SOCO	f_{35}	Hybrid Function 1 (N = 3)	[-100,100]	CEC'14
f_{11}	Shifted Schaffer	[-100,100]	CEC'05	f_{36}	Hybrid Function 2 (N = 3)	[-100,100]	CEC'14
f_{12}	Shchwefel 2.6 Global Optimum on Bounds	[-100,100]	CEC'05	f_{37}	Hybrid Function 3 (N = 4)	[-100,100]	CEC'14
f_{13}	Shifted Ackley	[-32,32]	SOCO	f_{38}	Hybrid Function 4 (N = 4)	[-100,100]	CEC'14
f_{14}	Shifted Rotated Ackley	[-100,100]	CEC'14	f_{39}	Hybrid Function 5 (N = 5)	[-100,100]	CEC'14
f_{15}	Shifted Rosenbrock	[-100,100]	SOCO	f_{40}	Hybrid Function 6 (N = 5)	[-100,100]	CEC'14
f_{16}	Shifted Rotated Rosenbrock	[-100,100]	CEC'14	f_{41}	Hybrid Composition Function	[-5,5]	CEC'05
f_{17}	Shifted Griewank	[-600,600]	SOCO	f_{42}	Rotated Hybrid Composition Function	[-5,5]	CEC'05
f_{18}	Shifted Rotated Griewank	[-100,100]	CEC'14	f_{43}	Rotated H. Composition F. with Noise in Fitness	[-5,5]	CEC'05
f_{19}	Shifted Rastrigin	[-100,100]	SOCO	f_{44}	Rotated Hybrid Composition F.	[-5,5]	CEC'05
f_{20}	Shifted Rotated Rastrigin	[-100,100]	CEC'14	f_{45}	Rotated H. Composition F. with a Narrow Basin for the Global Opt.	[-5,5]	CEC'05
f_{21}	Shifted Schwefel	[-100,100]	SOCO	f_{46}	Rotated H. Comp. F. with the Global Opt. On the Bounds	[-5,5]	CEC'05
f_{22}	Shifted Rotated Schwefel	[-100,100]	CEC'14	f_{47}	Rotated Hybrid Composition Function	[-5,5]	CEC'05
f_{23}	Shifted Rotated WeierStrass	[-100,100]	CEC'05	f_{48}	Rotated H. Comp. F. with High Condition Number Matrix	[-5,5]	CEC'05
f_{24}	Shifted Rotated Katsuura	[-100,100]	CEC'14	f_{49}	Non-Continuous Rotated Hybrid Composition Function	[-5,5]	CEC'05
f_{25}	Shifted Rotated HappyCat	[-100,100]	CEC'14	f_{50}	Rotated Hybrid Composition Function	[-5,5]	CEC'05

In order to present statistically meaningful results, we perform 50 independent runs of each algorithms on each function and report the median (MED) result—to measure the quality of the solutions produced by the algorithms—and the median error (MEDerr) with respect to the best solution found by any of the algorithms. In all cases, the algorithm was stopped after reaching $5000 \times d$ objective function evaluations (FEs). Both the tuning and the experiments were carried out on single core Intel Xeon E5-2680 running at 2.5GHz with 12 Mb cache size under Cluster Rocks Linux version 6.0/CentOS 6.3. The PSO-X framework was codified using `C++` and compiled with `gcc` 4.4.6.[9] The version of `irace` is 3.2.

6 Analysis of the results

The analysis of the results is divided into two parts. In the first part, we analyze the performance and capabilities of six automatically generated PSO-X algorithms, named PSO-X_{all}, PSO-X_{hyb}, PSO-X_{mul}, PSO-X_{uni}, PSO-X_{cec} and PSO-X_{soco}. Each of these PSO-X algorithms has been created using a set of training instances composed of different functions. For PSO-X_{all}, we used all the fifty functions (f_{1-50}), whereas for PSO-X_{uni} we used only the unimodal functions (f_{1-12}), for PSO-X_{mul} only the multimodal ones (f_{13-26}) and for PSO-X_{hyb} only the hybrid compositions (f_{27-50}). In the case of PSO-X_{cec} and PSO-X_{soco}, we used the entire set of functions of the CEC'05 and SOCO'10 test suites, respectively. Unlike the SOCO'10 test suite, the CEC'05 competition set includes many rotated objective functions and more complex hybrid compositions. The idea of using different training instances is to try to identify the algorithm components that result in higher performance when tackling functions of different classes.

In the second part, we compare the performance of our automatically generated PSO-X algorithms with ten well-known variants of PSO. We used two versions of each PSO variant: one whose parameters were tuned with `irace` (indicated by "*tnd*") and the other that uses the default parameter settings proposed by the original authors (indicated by "*dft*"). The variants included in our comparison are the following:

1. Enhanced rotation invariant PSO [12] (ERiPSO)—a variant that uses the AC-random strategy, Mtx-Euclidean rotation and α-adaptive.
2. Fully informed PSO [40] (FiPSO)—a traditional PSO variant that uses the constriction coefficient velocity update rule[10] (CCVUR) and the Mol-fully informed.
3. Frankenstein's PSO [42] (FraPSO)—a PSO variant that uses Top-time-varying, Mol-fully informed and $\omega_1 =$ linear decreasing.[11]
4. Gaussian "bare-bones" PSO [32] (GauPSO)—a variant that uses the DNPP-Gaussian option of the DNPP-additive stochastic as the only mechanism to update particles positions.
5. Hierarchical PSO [30] (HiePSO)—a variant based on Top-hierarchical that can be implemented using either $\omega_1 =$ linear decreasing or $\omega_1 =$ linear increasing.
6. Incremental PSO [43] (IncPSO)—a variant of PSO that uses the CCVUR and Pop-incremental with Init-horizontal.

[9] The source code of PSO-X will be made publicly available under Creative Commons license after the publication of this article.

[10] This rule is define as: $\mathbf{v}_{t+1}^i = \chi\left(\mathbf{v}_t^i + \varphi_1 U_{1t}^i(\mathbf{p}_t^i - \mathbf{x}_t^i) + \varphi_2 U_{2t}^i(\mathbf{l}_t^i - \mathbf{x}_t^i)\right)$, where $\chi = 0.7298$ is called constriction coefficient [17]. It can obtained from Eq. 4 by setting $\omega_1 = \omega_2 = 0.7298$ and using the DNPP-rectangular option.

[11] In FraPSO, t_{max} is replaced by a parameter t_{schd}, which indicates the iteration at which $\omega_{1t} = \omega_{1min}$ (see linear decreasing in Table 1).

7. Locally convergent rotation invariant PSO [10] (LcRPSO)—a more recent variant of PSO in which the $\text{Pert}_{\text{info}}$-Gaussian component is used together with the PM-Euclidean distance strategy, Mtx-random linear and the AC-random strategy.
8. Restart PSO [23] (ResPSO)—a variant of SPSO using velocity clamping and particles reinitialization.
9. Standard PSO [53] (StaPSO)—the PSO algorithm described in Section 2.2 that uses Eq. 1, 2 and 3.
10. Standard PSO 2011 (SPSO11)—a variant of SPSO that uses the DNPP-spherical option.

In Table 5, we show the parameter configuration of the versions that we used in the comparison. Note that, with the goal of simplifying their description, we have only mentioned the components that are different in these algorithms from those in SPSO. This means that, unless specified otherwise, we assumed that the following components and parameters setting are used in their implementation: Pop-constant, Top-fully-connected with Mol-best-of-neighborhood, DNPP-rectangular with Mtx-random diagonal and $\text{Pert}_{\text{info}} = \text{Pert}_{\text{rand}} =$ none, $\omega_1 =$ constant, $\omega_2 = 1.0$, $\omega_3 = 0$ and AC-constant.

Table 5: Parameter settings of the ten PSO variants included in our comparison.

Algorithm	Settings
ERiPSO_{dft}	$\text{pop} = 20$, $\omega_1 = 0.7213475$, AC-random, $\varphi_{1min} = 0$, $\varphi_{1max} = 2.05$, $\varphi_{2min} = 0$, $\varphi_{2max} = 2.05$, $\text{Mtx-Euclidean rotation}_{all}$ with α-adaptive and $\zeta = 30$ and $\rho = 0.01$.
FiPSO_{tnd}	Top-ring, $\text{pop} = 20$, $\omega_1 = \omega_2 = 0.729843788$, Mtx-random diagonal, $\varphi_1 = 2.1864$ and $\varphi_2 = 2.3156$.
FraPSO_{dft}	$\kappa = 60$, $\text{pop} = 60$, $\omega_1 =$ linear decreasing, $\omega_{1min} = 0.4$, $\omega_{1max} = 0.9$, $t_{schd} = 600$, $\varphi_1 = 2.0$ and $\varphi_2 = 2.0$.
GauPSO_{tnd}	Top-time-varying with Mol-random informant, $\kappa = 150$ $\text{pop} = 30$, $\omega_1 = 0$ and DNPP-additive stochastic DNPP-Gaussian.
HiePSO_{tnd}	$bd = 2$, $\text{pop} = 114$, $\omega_1 =$ linear increasing, $\omega_{1min} = 0.3284$, $\omega_{1max} = 0.8791$, $\varphi_1 = 2.1105$ and $\varphi_2 = 1.0349$.
IncPSO_{tnd}	Top-time-varying, $\kappa = 2360$, Init-horizontal, $\text{pop}_{ini} = 5$, $\text{pop}_{fin} = 295$, $\xi = 10$, $\omega_1 = \omega_2 = 0.729843788$, $\varphi_1 = 1.9226$ and $\varphi_2 = 1.0582$.
LcRPSO_{dft}	$\text{pop} = d$, $\omega_1 = 0.7298$, AC-random, $\varphi_{1min} = 0$, $\varphi_{1max} = 1.4962$, $\varphi_{2min} = 0$, $\varphi_{2max} = 1.4962$, Mtx-random linear, $\text{Pert}_{\text{info}}$-Gaussian with PM-Euclidean distance and $\varepsilon = 0.46461/d^{0.58}$.
ResPSO_{tnd}	Top-ring with Mol-fully informed, $\text{pop} = 10$, $\omega_1 =$ linear decreasing, $\omega_{1min} = 0.2062$, $\omega_{1max} = 0.6446$, $\varphi_1 = 1.5014$, $\varphi_2 = 2.2955$.
SPSO11_{tnd}	Top-time-varying with $\kappa = 1085$, $\text{pop} = 155$, $\omega_1 = 0.6482$, $\varphi_1 = 2.2776$, $\varphi_2 = 2.1222$.
StaPSO_{tnd}	Top-Von Neumann, $\text{pop} = 34$, $\omega_1 = 0.6615$, $\varphi_1 = 2.3706$, $\varphi_2 = 0.8914$.

* As reminder for the reader, ζ and ρ are parameters of α-adaptive; σ is a parameter of α-Gaussian; κ is a parameter of Top-time-varying; t_{schd} is a parameter of $\omega_1 =$ linear decreasing; bd is a parameter of Top-hierarchical; ξ is a parameter of Pop-incremental; and ε is a parameter of PM-Euclidean distance.

6.1 Comparison of automatically generated PSO algorithms

The algorithm components in the automatically generated PSO-X algorithms are listed below, and their configuration is given in Table 6.

- PSO-X_{all}: Top-fully-connected with Mol-best-of-neighborhood and Pop-incremental with Init-random; DNPP-rectangular with $\text{Pert}_{\text{info}}$-Lévy and PM-success rate, Mtx-random diagonal and velocity clamping.
- PSO-X_{hyb}: Top-Von Neumann with Mol-best-of-neighborhood and Pop-constant; DNPP-rectangular with $\text{Pert}_{\text{info}}$-Lévy and PM-success rate, Mtx-random diagonal and velocity clamping.
- PSO-X_{mul}: Top-time-varying with Mol-best-of-neighborhood and Pop-incremental with Init-horizontal; DNPP-rectangular with $\text{Pert}_{\text{info}}$-Lévy and PM-success rate, Mtx-random linear and stagnation detection.
- PSO-X_{uni}: Top-fully-connected with Mol-best-of-neighborhood and Pop-incremental with Init-random; DNPP-rectangular with $\text{Pert}_{\text{info}}$-Lévy and PM-success rate, Mtx-random diagonal and velocity clamping.
- PSO-X_{cec}: Top-Von Neumann with Mol-best-of-neighborhood and Pop-constant; DNPP-rectangular with $\text{Pert}_{\text{info}}$-Lévy and PM-success rate, $\text{Pert}_{\text{rand}}$-noisy with PM-success rate and Mtx-random diagonal.
- PSO-X_{soco}: Top-ring with Mol-ranked fully informed and Pop-constant; DNPP-rectangular with $\text{Pert}_{\text{info}}$-Gaussian and PM-success rate, Mtx-random diagonal and velocity clamping.

In Table 7, it is shown the median of the results obtained by the algorithms on each function. At the bottom of the table, we show the number of times each algorithm obtained the best result among the six ("Wins"), the average median value ("Av.MED"), the average ranking of the algorithm across all 50 functions ("Av.Ranking"), and whether the overall performance of any of the compared algorithm was significantly worse ("$+$") or equal ("$\approx$") than the best ranked algorithm according to a Wilcoxon's rank-sum test at 0.95 confidence interval with Bonferroni's correction. PSO-X_{all} was the algorithm that ranked best of the six followed by PSO-X_{cec} and PSO-X_{hyb}, while PSO-X_{soco} was the one that returned the best median result in the higher number of cases. The symbol "$+$" next to the some of the median value in Table 7 indicates the cases where we found a statistical difference function-wise in favor of PSO-X_{all} according also to a Wilcoxon-Bonferroni test with $\alpha = 0.05$. PSO-X_{uni}, which ranked last of the six, was the algorithm that performed statistically worse that PSO-X_{all} in most functions (26 of the 50 functions), while PSO-X_{mul} was worse in 23 functions, PSO-X_{cec} in 19 functions, PSO-X_{hyb} in 17 functions, and PSO-X_{soco} in 14 functions. In the following we examine the performance of the six PSO-X algorithms across the different function classes in our benchmark set, focusing on those that are specific to a function class, and the effect of their algorithm differences in their performance.

6.1.1 Comparison of the PSO-X algorithms on specific function classes

In order to know whether our PSO-X algorithms were able to obtain better results in specific function classes, we analyze their performance according to the average ranking (Av.Ranking) they obtained in the unimodal (f_{1-12}), multimodal (f_{13-26}), hybrid composition (f_{27-50}) and rotated ($f_{\text{rotated}} = f_{2-4,6,14,16,18,20,22-26,42-50}$) functions. The Av.Ranking gives us an indication of how good or bad was the performance of an algorithm across the different classes based on the result of the winner of each function. In Table 8, we present this information together with the algorithms average median error (Av.MEDerr). In our analysis, we pay

particular attention to the results of PSO-X_{uni}, PSO-X_{mul}, PSO-X_{hyb} and PSO-X_{cec}, which are the algorithms from which we would expect to see better results because of the functions used for creating them.

As shown in Table 8, according to the median solution quality, the performance of the algorithms is weakly correlated with the class of functions used with `irace`. Although PSO-X_{mul} ranked first in its function class of specialization, PSO-X_{uni} was outperformed by all the algorithms in the unimodal functions, PSO-X_{hyb} was outperformed by PSO-X_{all} in the hybrid compositions, and PSO-X_{cec} was outperformed by PSO-X_{hyb} and PSO-X_{mul} in the rotated functions. An analysis of the results using the average median error of the algorithms shows similar results, although, in this case, the performance of PSO-X_{uni} and PSO-X_{mul} was weakly correlated to the class of functions used in their training sets.

There are a few possible reasons why the use of different sets of functions did not have a stronger effect on the performance of our algorithms. The first one is the way in which we separated the functions, that captures some features of the functions, but neglects others, such as separability, noise, and different combination of objective functions transformations.[12] Another possible reason is the presence of slightly overfitted models during the

[12] Note, for example, that there are rotated functions in the training set of the six PSO-X algorithms, except for PSO-X_{soco} that includes only translations.

Table 6: Parameter settings of the six automatically generated PSO-X algorithms.

Algorithm	Settings
PSO-X_{all}	$\text{pop}_{ini} = 4$, $\text{pop}_{fin} = 20$, $\xi = 8$, $\omega_1 =$ convergence-based, $\omega_{1min} = 0.4218$, $\omega_{1max} = 0.6046$, $a = 0.7192$, $b = 0.9051$, $\omega_2 =$ random, AC-constant, $\varphi_1 = 1.7067$, $\varphi_2 = 2.2144$, $\text{PM}_{t=0} = 0.438$, $s_c = 11$ and $f_c = 40$.
PSO-X_{hyb}	$\text{pop} = 41$, $\omega_1 =$ adaptive based onvelocity, $\omega_{1min} = 0.119$, $\omega_{1max} = 0.1378$, $\lambda = 0.608$, AC-random, $\varphi_{1min} = 1.0429$, $\varphi_{1max} = 2.1653$, $\varphi_{2min} = 1.0429$, $\varphi_{2max} = 2.3275$, $\text{PM}_{t=0} = 0.5333$, $s_c = 28$ and $f_c = 42$.
PSO-X_{mul}	$\kappa = 300$, $\text{pop}_{ini} = 3$, $\text{pop}_{fin} = 50$, $\xi = 2$, $\omega_1 =$ success-based, $\omega_{1min} = 0.4$, $\omega_{1max} = 0.9$, AC-constant, $\varphi_1 = 0.92$, $\varphi_2 = 1.6577$, $\text{PM}_{t=0} = 0.5114$, $s_c = 2$ and $f_c = 33$.
PSO-X_{uni}	$\text{pop}_{ini} = 10$, $\text{pop}_{fin} = 58$, $\xi = 3$, $\omega_1 =$ adaptive based onvelocity, $\omega_{1min} = 0.3531$, $\omega_{1max} = 0.7095$, $\lambda = 0.4832$, $\omega_2 = \omega_1$, AC-random, $\varphi_{1min} = 1.4217$, $\varphi_{1max} = 2.051$, $\varphi_{2min} = 0.8626$, $\varphi_{2max} = 1.4609$, $\text{PM}_{t=0} = 0.9865$, $s_c = 38$ and $f_c = 11$.
PSO-X_{cec}	$\text{pop} = 42$, $\omega_1 =$ self-regulating, $\omega_{1min} = 0.1673$, $\omega_{1max} = 0.2317$, $\eta = 0.2468$, $\omega_2 =$ random, $\omega_3 =$ random, AC-random, $\varphi_{1min} = 1.8684$, $\varphi_{1max} = 1.9233$, $\varphi_{2min} = 0.2802$, $\varphi_{2max} = 1.5143$, $\text{PM}_{1,t=0} = 0.4837$, $s_{c1} = 29$, $f_{c1} = 45$, $\text{PM}_{2,t=0} = 0.8139$, $s_{c2} = 30$ and $f_{c2} = 43$.
PSO-X_{soco}	$\text{pop} = 19$, $\omega_1 =$ adaptive based onvelocity, $\omega_{1min} = 0.6564$, $\omega_{1max} = 0.8201$, $\lambda = 0.2959$, AC-constant, $\varphi_1 = 0.7542$, $\varphi_2 = 1.9235$, $\text{PM}_{t=0} = 0.8907$, $s_c = 22$ and $f_c = 49$.

* As reminder for the reader, ξ is a parameter of Pop-incremental; a and b are parameters of $\omega_1 =$ convergence-based, λ of $\omega_1 =$ adaptive based onvelocity, and η of $\omega_1 =$ self-regulating; s_c and f_c are parameters of PM-success rate; and κ is a parameter of Top-time-varying;.

Table 7: Median results of the PSO-X algorithms in f_{1-50} with $d = 50$.

f#	PSO-X_{all}	PSO-X_{uni}	PSO-X_{mul}	PSO-X_{hyb}	PSO-X_{cec}	PSO-X_{soco}
f_1	**0.00E+00**	**0.00E+00**	9.90E-09$^+$	**0.00E+00**	**0.00E+00**	**0.00E+00**
f_2	1.78E+06	**1.18E+06**	3.50E+06$^+$	2.89E+06$^+$	3.01E+06$^+$	8.33E+06$^+$
f_3	2.10E+03	6.49E+03	2.09E+03	**1.78E+03**	2.37E+03	3.07E+03
f_4	**1.46E+04**	2.31E+04$^+$	3.91E+04$^+$	1.49E+04	2.35E+04$^+$	1.65E+04
f_5	**3.76E-09**	1.05E-03$^+$	2.49E-04$^+$	5.69E-03$^+$	2.96E-07$^+$	1.55E-02$^+$
f_6	5.54E-04	**5.25E-06**	3.17E+01$^+$	2.64E+01$^+$	2.73E+00$^+$	5.64E+01$^+$
f_7	1.96E+04	4.15E+04$^+$	1.83E+04	1.45E+04	9.10E+03	**3.80E+03**
f_8	**0.00E+00**	**0.00E+00**	7.04E-04$^+$	**0.00E+00**	**0.00E+00**	**0.00E+00**
f_9	2.76E+01	1.89E+02$^+$	1.84E+02$^+$	7.98E+01$^+$	3.49E+01	**5.36E-03**
f_{10}	**0.00E+00**	1.05E+00$^+$	4.89E-07$^+$	**0.00E+00**	**0.00E+00**	**0.00E+00**
f_{11}	2.86E+01	1.95E+02$^+$	1.77E+02$^+$	7.94E+01$^+$	4.64E+01	**1.07E-02**
f_{12}	9.86E-05	**1.86E-05**	2.65E+01$^+$	6.27E-02$^+$	1.87E-01$^+$	3.03E-02$^+$
f_{13}	**-1.44E-16**	**-1.44E-16**	5.18E-05$^+$	**-1.44E-16**	**-1.44E-16**	**-1.44E-16**
f_{14}	2.11E+01	**2.00E+01**	**2.00E+01**	**2.00E+01**	**2.00E+01**	2.12E+01$^+$
f_{15}	4.60E+01	**3.35E+01**	4.64E+01	4.42E+01	4.39E+01	4.19E+01
f_{16}	4.78E+01	**4.70E+01**	**4.70E+01**	**4.70E+01**	**4.70E+01**	**4.70E+01**
f_{17}	7.40E-03	1.63E-19	2.03E-09	1.63E-19	1.63E-19	**5.42E-20**
f_{18}	**1.63E-19**	3.79E-19	1.10E-06	1.05E-12	1.20E-13	1.08E-12
f_{19}	**1.24E+01**	1.40E+02$^+$	3.11E+01$^+$	1.14E+02$^+$	8.71E+01$^+$	1.41E+02$^+$
f_{20}	2.63E+02	2.43E+02	2.03E+02	**1.95E+02**	2.18E+02	2.39E+02
f_{21}	2.44E+04	2.56E+04$^+$	**2.34E+04**	2.54E+04$^+$	2.51E+04$^+$	2.86E+04$^+$
f_{22}	2.86E+04	2.90E+04	**2.79E+04**	2.82E+04	2.81E+04	3.47E+04$^+$
f_{23}	3.60E+01	3.37E+01	2.78E+01	3.52E+01	3.74E+01	**2.29E+01**
f_{24}	2.63E-01	7.06E-01$^+$	**6.10E-02**	7.48E-01$^+$	6.20E-01$^+$	3.53E+00$^+$
f_{25}	6.60E-01	6.20E-01	**4.10E-01**	4.67E-01	4.48E-01	4.37E-01
f_{26}	3.40E-01	7.77E-01$^+$	3.22E-01	**2.92E-01**	2.94E-01	3.40E-01
f_{27}	1.89E+01	1.21E+02$^+$	3.97E+01$^+$	**1.34E-09**	3.08E+01$^+$	2.01E+01
f_{28}	**5.52E+01**	1.47E+02$^+$	1.06E+02$^+$	1.18E+02$^+$	1.10E+02$^+$	6.50E+01
f_{29}	**1.40E+01**	7.97E+01$^+$	3.39E+01$^+$	6.52E+01$^+$	5.99E+01$^+$	6.18E+01$^+$
f_{30}	6.36E-13	1.63E-07$^+$	6.61E-04$^+$	**0.00E+00**	1.33E-07$^+$	**0.00E+00**
f_{31}	**2.80E+01**	2.49E+02$^+$	1.09E+02$^+$	6.28E+01$^+$	1.41E+02$^+$	4.02E+01
f_{32}	1.79E+02	3.89E+02$^+$	2.68E+02	2.90E+02	2.87E+02	**7.17E+01**
f_{33}	1.92E+01	7.79E+01$^+$	4.26E+01$^+$	6.32E+01$^+$	6.41E+01$^+$	**9.75E+00**
f_{34}	3.81E-18	1.34E-07$^+$	4.20E-04$^+$	2.50E-19	5.15E-08$^+$	**0.00E+00**
f_{35}	1.27E+04	**1.23E+04**	**1.23E+04**	**1.23E+04**	**1.23E+04**	**1.23E+04**
f_{36}	1.49E+05	2.61E+05$^+$	**1.22E+05**	1.52E+05	1.48E+05	3.32E+06$^+$
f_{37}	**3.90E+01**	4.77E+01$^+$	4.05E+01	5.35E+01$^+$	5.23E+01$^+$	5.99E+01$^+$
f_{38}	**1.65E+05**	3.10E+05$^+$	2.14E+05	2.51E+05$^+$	2.70E+05$^+$	4.61E+05$^+$
f_{39}	5.28E+00	5.49E+00	**4.24E+00**	4.50E+00	4.30E+00	5.43E+00
f_{40}	1.08E+01	1.74E+01$^+$	1.61E+01$^+$	1.44E+01$^+$	1.28E+01$^+$	**9.58E+00**
f_{41}	3.46E+02	4.00E+02$^+$	3.37E+02	2.98E+02	**2.24E+02**	3.51E+02
f_{42}	2.00E+02	2.23E+02	1.20E+02	1.09E+02	**9.44E+01**	2.77E+02
f_{43}	**2.25E+02**	3.16E+02	3.01E+02	2.38E+02	2.35E+02	3.09E+02
f_{44}	9.55E+02	9.34E+02	**9.25E+02**	**9.25E+02**	9.27E+02	9.29E+02
f_{45}	9.56E+02	9.34E+02	**9.25E+02**	9.28E+02	**9.25E+02**	9.29E+02
f_{46}	9.59E+02	9.33E+02	**9.25E+02**	9.28E+02	**9.25E+02**	9.29E+02
f_{47}	**8.02E+02**	1.02E+03$^+$	1.01E+03$^+$	1.02E+03$^+$	1.02E+03$^+$	1.01E+03$^+$
f_{48}	9.68E+02	9.58E+02	9.34E+02	**9.09E+02**	9.18E+02	9.22E+02
f_{49}	**9.76E+02**	1.02E+03$^+$	1.02E+03$^+$	1.02E+03	1.02E+03	1.01E+03
f_{50}	9.42E+02	9.84E+02$^+$	1.11E+03$^+$	9.57E+02	9.85E+02	**9.38E+02**
Wins	16	10	12	14	11	**17**
Av.MED	4.42E+04	**3.79E+04**	7.94E+04	6.80E+04	7.08E+04	2.44E+05
Av.Ranking	**3.24**	4.68	3.54	3.42	3.38	3.68
Wilcoxon test		$+$	$\approx$	$\approx$	$\approx$	$\approx$

* The symbol $^+$ that appears next to the median value indicates the cases where there is a statistical difference in favor of PSO-X_{all}.

Table 8: Average ranking (Av.Ranking) and average median error (Av.MEDerr) obtained by the PSO-X algorithms in f_{1-12} (unimodal), f_{13-26} (multimodal), f_{27-50} (hybrid) and f_{rotated} with $d = 50$.

f#		PSO-X_{all}	PSO-X_{uni}	PSO-X_{mul}	PSO-X_{hyb}	PSO-X_{cec}	PSO-X_{soco}
f_{1-12}	Av.Ranking	**2.83**	4.25	4.83	3.67	3.75	3.83
	Av.MEDerr	1.30E+05	**8.21E+04**	2.75E+05	2.22E+05	2.32E+05	6.75E+05
f_{13-26}	Av.Ranking	3.27	4.54	**3.15**	3.62	3.54	3.69
	Av.MEDerr	2.63E+02	3.75E+02	**1.45E+02**	3.04E+02	2.79E+02	9.55E+02
f_{27-50}	Av.Ranking	**1.9**	4.58	3.1	2.58	4.66	4.2
	Av.MEDerr	**6.04E+03**	1.63E+04	6.89E+03	9.58E+03	1.02E+04	1.45E+05
f_{rotated}	Av.Ranking	3.91	4.36	3.05	**2.77**	3.14	4.09
	Av.MEDerr	7.01E+04	**4.31E+04**	1.49E+05	1.20E+05	1.26E+05	3.68E+05
f_{1-50}	Av.Ranking	**3.24**	4.68	3.54	3.42	3.38	3.68
	Av.MEDerr	3.42E+04	**2.80E+04**	6.94E+04	5.81E+04	6.09E+04	2.34E+05

creation of these algorithms with `irace`. The effect of overfitting can be observed more clearly for PSO-X_{uni} than for the rest of algorithms. For a number of functions (e.g., $f_{6,12}$) the median solution obtained by PSO-X_{uni} was significantly better than that of the other algorithms, which contributes to lower the value of the Av.MED and Av.MEDerr metrics, but not to improve its ranking in its respective classes of specialization. Among the possible causes for the overfitting are the use of training sets with different number of instances (PSO-X_{uni} has 12 instances, while the best ranked algorithm, PSO-X_{all}, has 50) and of a exceedingly large computational budget used with `irace`.

6.1.2 PSO-X algorithm differences

The first thing to note about the design of the six PSO-X algorithms is that, despite they were created using different sets of functions, they all share the same core components, i.e., DNPP-rectangular with $\text{Pert}_{\text{info}}$-Lévy or $\text{Pert}_{\text{info}}$-Gaussian. This combination of components, as we discusses in Section 4.2, has the ability of making the implementation rotation invariant, which is an important characteristic given that 22 out of the 50 functions in our benchmark test set have a rotation in their objective function. In all cases, the strategy to control the perturbation magnitude (PM) was PM-success rate and, with the exception of PSO-X_{uni}, they all have a parameter setting where f_c is larger than s_c. This setting allows to decrease rapidly the PM when particles have been constantly improving the global best solution, but makes harder to switch back to a larger PM if the algorithm happens to stagnate. In this sense, PSO-X_{all}, PSO-X_{hyb}, PSO-X_{mul}, PSO-X_{cec} and PSO-X_{soco} are biased towards exploitation, and PSO-X_{uni} towards exploration.

Although PSO-X_{hyb} and PSO-X_{cec} obtained similar results in most functions and ranked almost the same across the whole benchmark set, the performance of PSO-X_{cec} was better in the CEC'05 hybrid compositions (f_{41-50}), and worse in functions f_{27}, f_{30}, f_{31} and f_{34} that belong to the SOCO'10 test suite. Based on the components and parameter setting in PSO-X_{hyb} and PSO-X_{cec}, this difference can be attributed to the $\text{Pert}_{\text{rand}}$ component that is present only in PSO-X_{cec}. The $\text{Pert}_{\text{rand}}$ component was advantageous for PSO-X_{cec} to tackle the more complex search spaces of the CEC'05 hybrid compositions, where the algorithm performed its best, but affected its solutions quality in most of the SOCO'10 test suite hybrid compositions. Another interesting comparison can be done between PSO-X_{all} (ranked first) and PSO-X_{uni} (ranked last). These two algorithms have the exact same components and differ only in the population size, which is roughly three times larger in PSO-X_{uni} compared to PSO-X_{all}; parameter ω_1, which is equal to ω_2 in PSO-X_{uni} and random in PSO-X_{all};

and parameters f_c and s_c, whose value is inverted in PSO-X_{uni} compare to PSO-X_{all} (see Table 5). Data from Table 7 shows that the configuration of PSO-X_{uni} is quite performing to tackle functions with large plateaus and quite regular landscapes, such as Elliptic (f_2), Schwefel (f_6, f_8, and f_{12}) or Rosenbrock (f_{15} and f_{16}), where PSO-X_{uni} was the most performing of the six. However, when PSO-X_{uni} faced less regular and multimodal landscapes, its performance declined significantly.

6.2 Comparison with other PSO algorithms

We also compared our PSO-X algorithms with ten traditional and recently proposed PSO variants. As mentioned before, for each algorithm we collected data using both a default (*dft*) version—that uses the parameter settings proposed by the authors—and a tuned (*tnd*) version—whose parameters were configured with `irace`. Based on a Wilcoxon-Bonferroni test at $\alpha = 0.05$, we selected the most performing of the two versions of each algorithm. However, since the computed p-values were larger than 0.05 for ERiPSO, FraPSO and SPSO11, we selected the version that obtained the lower median value across the 50 functions. In Table 9, we show the median of the 50 runs executed by each algorithm for each function and, in Table 10, we show the mean ranking obtained by each algorithm according to the different classes in which we separated the functions in the benchmark set. To complement the information given in the tables, in the supplementary material of the article [13], we present the distribution of the results obtained by the 16 compared algorithms using box plots.

In terms of the median solution quality, except for PSO-X_{uni}, the performance of the automatically generated PSO-X algorithms was better than any of the PSO variants in our comparison. PSO-X_{cec} obtained the best ranking followed by PSO-X_{mul} and PSO-X_{hyb}, and it was also the algorithm that returned the best median value in most functions. Regarding the performance of the algorithms on specific problems classes, PSO-X_{cec} obtained the best ranking according to the Av.MED result in the unimodal and rotated functions, PSO-X_{mul} the best one in the multimodal functions, and PSO-X_{all} the best one in the hybrid functions; whereas the algorithms that obtained the lower Av.MEDerr were LcRPSO$_{dft}$ in the unimodal and rotated functions, and PSO-X_{mul} in the multimodal and hybrid composition functions. To put these results in context, in Table 10, we have used boxes to highlight the results of the PSO variants whose ranking was equally good, or better, than any of the PSO-X algorithms.

Note that only IncPSO$_{tnd}$ and StaPSO$_{tnd}$ were capable of outperforming the results obtained by some of the PSO-X algorithms, specially in the rotated functions, where those two algorithms were as competitive as the automatically generated. However, in the case of the hybrid composition functions the results are quite compelling in favor of PSO-X, since even the worst automatically generated algorithm performed significantly better than any of the PSO variants. This is a very strong point in favor of our PSO-X algorithms not only because half of the functions in our benchmark set are hybrid compositions, but also because these kind of functions are the hardest to solve and the most representative of real-world optimization problems.

Table 9: Median results returned by the six automatically generated PSO-X algorithms and ten other PSO variants in f_{1-40} with $d = 100$ and f_{41-50} with $d = 50$.

f#	PSO-X_{all}	PSO-X_{uni}	PSO-X_{mul}	PSO-X_{hyb}	PSO-X_{soco}	PSO-X_{cec}	GauPSO$_{ind}$	ResPSO$_{ind}$	ERiPSO$_{dft}$	FinPSO$_{ind}$	FraPSO$_{dft}$	LcRPSO$_{dft}$	HiePSO$_{ind}$	IncPSO$_{ind}$	StaPSO$_{ind}$	SPSO11$_{ind}$
f_1	**0.00E+00**	**0.00E+00**$^+$	1.45E-08$^+$	**0.00E+00**	1.19E-28$^+$	**0.00E+00**	1.55E+03$^+$	3.00E-24$^+$	5.14E-25$^+$	**0.00E+00**	2.07E+03$^+$	3.26E-27$^+$	1.01E+01$^+$	1.16E-12$^+$	6.69E-28$^+$	2.93E+00$^+$
f_2	1.92E+07$^+$	4.44E+06	1.73E+07$^+$	1.18E+07$^+$	4.05E+07$^+$	8.82E+06	5.16E+08$^+$	9.84E+06	5.55E+06	7.69E+07$^+$	4.36E+07$^+$	**2.87E+06**	2.63E+08$^+$	1.47E+07$^+$	2.59E+07$^+$	2.28E+07$^+$
f_3	**2.83E+03**	4.85E+03	4.86E+03	3.96E+03	3.46E+03	3.00E+03	2.80E+09$^+$	5.53E+03	3.51E+03	4.88E+03	1.03E+04$^+$	3.58E+03	4.58E+07$^+$	5.38E+03	2.92E+03	4.75E+06$^+$
f_4	1.06E+04	7.67E+03	1.57E+04	1.02E+04	7.78E+03	1.98E+04	2.01E+05$^+$	1.87E+04	**9.95E+02**	7.61E+03	2.78E+04	3.82E+03	1.72E+05$^+$	2.86E+04	1.78E+04	2.01E+05$^+$
f_5	6.61E-03	4.19E+00$^+$	1.87E-02	2.22E+00$^+$	4.44E+00$^+$	**3.76E-03**	7.81E+01$^+$	6.93E+01$^+$	5.91E+01$^+$	3.41E+01$^+$	1.65E+00$^+$	5.16E+01$^+$	3.88E+01$^+$	5.58E+01$^+$	3.33E+01$^+$	6.12E+01$^+$
f_6	5.71E+01	1.97E-01	6.61E+02$^+$	1.08E+03$^+$	5.35E+03$^+$	2.94E+02	9.76E+04$^+$	2.89E+03$^+$	8.14E-02	1.38E+04$^+$	6.65E+03$^+$	**5.74E-04**	2.90E+04$^+$	2.49E+03$^+$	1.13E+04$^+$	1.08E+03$^+$
f_7	1.02E+05$^+$	1.87E+05$^+$	7.37E+04$^+$	9.49E+04$^+$	3.59E+04	4.41E+04	3.62E+05$^+$	5.26E+04	2.84E+04	6.95E+04$^+$	4.90E+04	2.38E+04	1.09E+05$^+$	**2.10E+04**	1.36E+05$^+$	6.00E+04$^+$
f_8	**0.00E+00**$^+$	**0.00E+00**	1.91E-03$^+$	**0.00E+00**	**0.00E+00**	**0.00E+00**	4.57E+01$^+$	1.09E-11$^+$	3.07E+02$^+$	**0.00E+00**	**0.00E+00**	3.37E+01$^+$	1.42E+00$^+$	1.35E-10$^+$	1.33E-15$^+$	1.62E+01$^+$
f_9	4.60E+02$^+$	5.57E+02$^+$	3.67E+02	3.76E+02	**9.52E-01**	3.50E+02	7.79E+02$^+$	9.12E+01	6.66E+02$^+$	3.31E+00	5.61E+02$^+$	6.10E+02$^+$	2.56E+02	1.55E+02	3.71E+02	6.31E+02$^+$
f_{10}	**0.00E+00**	3.15E+00$^+$	1.14E-06$^+$	8.23E-33$^+$	1.46E+00$^+$	**0.00E+00**	1.47E+01$^+$	4.20E+00$^+$	6.30E+01$^+$	5.90E+00$^+$	**0.00E+00**	6.13E+01$^+$	2.02E+01$^+$	1.78E+01$^+$	3.15E+00$^+$	6.49E+01$^+$
f_{11}	4.49E+02	5.46E+02$^+$	3.79E+02	3.83E+02	**1.03E+00**	3.86E+02	7.80E+02$^+$	1.02E+02	6.74E+02$^+$	2.41E+00	5.76E+02$^+$	6.27E+02$^+$	2.60E+02	1.84E+02	3.75E+02	6.28E+02$^+$
f_{12}	**4.38E+00**	3.29E+01	3.57E+02$^+$	1.29E+02$^+$	1.26E+01	3.68E+01	3.30E+04$^+$	4.59E+04$^+$	1.98E+04$^+$	5.63E+02$^+$	2.13E+02$^+$	7.44E+03$^+$	1.66E+04$^+$	1.81E+04$^+$	1.60E+03$^+$	3.22E+04$^+$
f_{13}	6.21E+00$^+$	3.32E-15$^+$	4.70E-05$^+$	**-1.44E-16**	1.39E-15$^+$	**-1.44E-16**	1.83E+01$^+$	1.12E+00$^+$	1.94E+01$^+$	**-1.44E-16**	1.89E+01$^+$	7.76E+00$^+$	3.53E+00$^+$	1.32E+00$^+$	3.22E-15$^+$	2.01E+01$^+$
f_{14}	2.13E+01$^+$	**2.00E+01**$^+$	**2.00E+01**$^+$	**2.00E+01**	2.13E+01$^+$	**2.00E+01**	2.13E+01$^+$	2.13E+01$^+$	2.03E+01$^+$	2.13E+01$^+$	2.13E+01$^+$	2.03E+01$^+$	2.13E+01$^+$	2.12E+01$^+$	2.12E+01$^+$	2.13E+01$^+$
f_{15}	1.88E+02	**1.16E+02**	2.86E+02	1.58E+02	1.76E+02	1.92E+02	2.28E+08$^+$	2.11E+02	1.24E+03$^+$	1.38E+02	6.45E+07$^+$	3.92E+02	1.61E+04$^+$	1.92E+02	1.74E+02	1.13E+03$^+$
f_{16}	9.86E+01$^+$	9.24E+01	9.68E+01	9.37E+01	9.32E+01	9.54E+01	2.88E+03$^+$	9.87E+01$^+$	9.70E+01	**9.10E+01**	1.48E+02$^+$	9.49E+01	3.28E+02$^+$	9.72E+01	9.42E+01	9.89E+01$^+$
f_{17}	2.71E-19	4.34E-19	3.09E-09$^+$	2.71E-19	**1.08E-19**	3.25E-19	1.22E+01$^+$	1.91E-01$^+$	1.80E-11$^+$	1.36E-19	8.95E+00$^+$	3.25E-19	1.07E+00$^+$	7.82E-02$^+$	2.71E-19	8.38E-02$^+$
f_{18}	2.20E-15	5.96E-19	1.60E-06	3.39E-13	4.58E-14	1.82E-14	2.90E+01$^+$	1.40E-12$^+$	2.55E-10$^+$	**2.17E-19**	1.05E-06$^+$	4.34E-19	1.42E+00$^+$	3.09E-06$^+$	1.76E-13	9.96E-01$^+$
f_{19}	**2.19E+01**	3.81E+02$^+$	6.96E+01	3.09E+02$^+$	3.92E+02$^+$	2.40E+02	6.91E+02$^+$	1.75E+02	2.57E+03$^+$	4.86E+02$^+$	5.51E+02$^+$	1.62E+03$^+$	7.17E+02$^+$	3.65E+02$^+$	3.91E+02$^+$	1.32E+03$^+$
f_{20}	7.78E+02$^+$	6.74E+02$^+$	5.60E+02	4.40E+02	6.63E+02	5.61E+02	8.26E+02$^+$	6.98E+02	1.02E+03$^+$	8.60E+02$^+$	7.84E+02$^+$	8.05E+02$^+$	6.30E+02	**3.76E+02**	4.49E+02	7.95E+02$^+$
f_{21}	5.59E+04$^+$	5.27E+04	**4.75E+04**	5.32E+04	6.44E+04$^+$	5.23E+04	5.83E+04$^+$	4.88E+04	6.02E+04$^+$	6.58E+04$^+$	5.53E+04$^+$	5.94E+04$^+$	5.52E+04$^+$	5.75E+04$^+$	5.58E+04$^+$	5.88E+04$^+$
f_{22}	5.95E+04$^+$	5.84E+04	**5.69E+04**	5.76E+04	7.30E+04$^+$	5.72E+04	6.79E+04$^+$	7.36E+04$^+$	6.09E+04$^+$	7.36E+04$^+$	6.77E+04$^+$	5.97E+04$^+$	7.25E+04$^+$	5.90E+04$^+$	6.27E+04$^+$	5.92E+04$^+$
f_{23}	9.36E+01	9.14E+01	7.35E+01	9.62E+01	6.79E+01	9.28E+01	1.25E+02$^+$	**4.59E+01**	1.44E+02$^+$	7.53E+01	1.34E+02$^+$	1.21E+02$^+$	8.60E+01	5.90E+01	1.00E+02	1.11E+02$^+$
f_{24}	4.16E-01	1.26E+00	6.45E-01	1.52E+00	4.14E+00$^+$	1.32E+00	2.61E+00$^+$	4.16E+00$^+$	2.13E+00$^+$	4.17E+00$^+$	4.18E+00$^+$	1.65E+00	4.18E+00$^+$	**8.22E-03**	1.43E+00	7.96E-01
f_{25}	5.76E-01$^+$	7.03E-01$^+$	4.62E-01	5.23E-01$^+$	5.49E-01$^+$	4.46E-01	6.91E-01$^+$	6.04E-01$^+$	6.02E-01$^+$	5.41E-01$^+$	6.02E-01$^+$	6.06E-01$^+$	5.89E-01$^+$	**4.05E-01**	4.66E-01	6.19E-01$^+$
f_{26}	3.63E-01$^+$	8.34E-01$^+$	3.48E-01$^+$	3.28E-01	3.79E-01$^+$	**3.15E-01**	1.00E+00$^+$	3.76E-01$^+$	3.50E-01	3.74E-01$^+$	4.03E-01$^+$	3.47E-01	4.04E-01$^+$	3.43E-01	3.30E-01	3.93E-01$^+$
f_{27}	**3.30E+01**	2.44E+02$^+$	1.00E+02	1.13E+02	7.90E+01	1.36E+02	4.34E+02$^+$	2.21E+02$^+$	2.57E+02$^+$	7.81E+01	2.12E+03$^+$	2.25E+02$^+$	2.57E+02$^+$	1.90E+02$^+$	9.88E+01	2.43E+02$^+$
f_{28}	1.87E+02	3.28E+02$^+$	2.07E+02	2.69E+02	**1.84E+02**	2.33E+02	3.42E+07$^+$	4.26E+02$^+$	3.54E+02$^+$	2.23E+02	6.41E+07$^+$	2.98E+02$^+$	1.46E+03$^+$	2.83E+02$^+$	3.36E+02$^+$	5.88E+02$^+$
f_{29}	**4.34E+01**	2.08E+02$^+$	6.97E+01	1.69E+02	1.94E+02$^+$	1.50E+02	4.33E+02$^+$	1.48E+02	7.56E+02$^+$	3.01E+02$^+$	4.62E+02$^+$	6.22E+02$^+$	3.01E+02$^+$	3.00E+02$^+$	2.75E+02$^+$	8.75E+02$^+$
f_{30}	1.20E-10	2.07E-06$^+$	2.18E-03$^+$	5.44E-24	**0.00E+00**	7.64E-07	5.63E+01$^+$	5.29E-06	1.29E+03$^+$	2.06E-01	**0.00E+00**	6.12E+01$^+$	5.18E+00$^+$	4.70E-01$^+$	4.44E-16	5.21E+01$^+$
f_{31}	1.93E+02	5.04E+02$^+$	2.32E+02	3.00E+02	1.11E+02	2.99E+02	5.26E+02$^+$	3.97E+02$^+$	5.15E+02$^+$	**1.10E+02**	2.32E+02	4.63E+02$^+$	4.57E+02$^+$	3.45E+02	3.15E+02	5.06E+02$^+$
f_{32}	5.06E+02	7.51E+02$^+$	4.76E+02	6.01E+02	2.38E+02	5.89E+02	9.83E+02$^+$	6.19E+02	7.72E+02$^+$	**1.75E+02**	1.77E+02	7.19E+02$^+$	8.92E+02$^+$	5.67E+02	7.58E+02$^+$	9.72E+02$^+$
f_{33}	**8.91E+01**	1.65E+02$^+$	9.24E+01	1.41E+02	1.38E+02	1.41E+02	2.34E+02$^+$	1.02E+02	3.19E+02$^+$	1.49E+02	1.76E+02$^+$	2.42E+02$^+$	1.77E+02$^+$	1.57E+02$^+$	1.72E+02$^+$	3.18E+02$^+$
f_{34}	9.78E-13	1.05E+00$^+$	1.43E-03$^+$	2.30E-18	1.05E+00$^+$	3.94E-07	9.57E+00$^+$	2.57E+00$^+$	7.53E+01$^+$	3.15E+00$^+$	**0.00E+00**	6.48E+01$^+$	1.48E+01$^+$	9.16E+00$^+$	1.05E+00$^+$	7.59E+01$^+$
f_{35}	2.59E+04$^+$	**2.49E+04**	2.51E+04	2.52E+04$^+$	2.51E+04	**2.49E+04**	3.40E+04$^+$	2.51E+04	2.82E+04$^+$	2.53E+04$^+$	2.52E+04$^+$	2.56E+04$^+$	2.54E+04$^+$	2.51E+04	2.52E+04$^+$	2.70E+04$^+$
f_{36}	3.13E+05	5.68E+05$^+$	**2.53E+05**	2.98E+05	5.67E+06$^+$	2.79E+05	6.93E+05$^+$	3.79E+05$^+$	4.51E+05$^+$	1.64E+08$^+$	2.50E+06$^+$	3.88E+05$^+$	7.08E+05$^+$	2.86E+05	3.60E+05$^+$	6.83E+05$^+$

Table 9 continued.

f#	PSO-X_{all}	PSO-X_{uni}	PSO-X_{mid}	PSO-X_{hyb}	PSO-X_{soco}	PSO-X_{cec}	GauPSO$_{ind}$	ResPSO$_{ind}$	ERiPSO$_{dft}$	FinPSO$_{ind}$	FraPSO$_{dft}$	LcRPSO$_{dft}$	HiePSO$_{ind}$	IncPSO$_{ind}$	StaPSO$_{ind}$	SPSO11$_{ind}$
f_{37}	1.32E+02	1.13E+02	**9.85E+01**	1.26E+02	1.32E+02^{+}	1.20E+02	1.29E+02^{+}	1.32E+02^{+}	1.20E+02	1.32E+02^{+}	1.30E+02^{+}	1.02E+02	1.33E+02^{+}	1.24E+02	1.20E+02	1.33E+02^{+}
f_{38}	3.28E+05	5.24E+05	3.47E+05	5.33E+05	7.48E+05^{+}	5.68E+05	9.86E+05^{+}	8.18E+05^{+}	5.32E+05	8.83E+05^{+}	**3.02E+05**	7.01E+05^{+}	8.51E+05^{+}	3.98E+05	6.69E+05^{+}	7.50E+05^{+}
f_{39}	9.43E+00^{+}	1.76E+01^{+}	7.03E+00	7.20E+00	8.61E+00^{+}	**6.96E+00**	2.13E+04^{+}	9.06E+00^{+}	1.56E+01^{+}	8.37E+00^{+}	1.10E+01^{+}	1.15E+01^{+}	6.60E+01^{+}	7.10E+00	7.13E+00	9.59E+00^{+}
f_{40}	1.74E+01	3.22E+01^{+}	1.87E+01	2.14E+01	**1.33E+01**	2.31E+01	5.40E+02^{+}	2.04E+01	1.07E+02^{+}	1.62E+01	2.47E+01	4.30E+01^{+}	3.87E+01^{+}	2.60E+01^{+}	2.10E+01	1.01E+02^{+}
f_{41}	3.46E+02^{+}	4.00E+02^{+}	3.37E+02	2.98E+02	3.51E+02^{+}	**2.24E+02**	3.04E+02	4.20E+02^{+}	9.48E+02^{+}	3.36E+02^{+}	7.45E+02^{+}	8.54E+02^{+}	4.06E+02^{+}	4.08E+02^{+}	4.00E+02^{+}	6.14E+02^{+}
f_{42}	2.00E+02^{+}	2.23E+02^{+}	1.20E+02^{+}	1.09E+02	2.77E+02^{+}	**9.44E+01**	2.76E+02^{+}	3.59E+02^{+}	4.61E+02^{+}	2.95E+02^{+}	5.85E+02^{+}	4.00E+02^{+}	3.27E+02^{+}	1.16E+02^{+}	2.40E+02^{+}	3.37E+02^{+}
f_{43}	2.25E+02	3.16E+02	3.01E+02^{+}	2.38E+02	3.09E+02^{+}	2.35E+02	3.61E+02^{+}	3.32E+02^{+}	4.86E+02^{+}	3.23E+02^{+}	5.07E+02^{+}	4.34E+02^{+}	3.55E+02^{+}	**1.65E+02**	2.54E+02	3.67E+02^{+}
f_{44}	9.55E+02^{+}	9.34E+02	**9.25E+02**	**9.25E+02**	9.29E+02	9.27E+02	9.51E+02^{+}	9.44E+02^{+}	1.12E+03^{+}	9.38E+02^{+}	1.22E+03^{+}	1.08E+03^{+}	9.37E+02^{+}	**9.25E+02**	9.40E+02^{+}	9.38E+02^{+}
f_{45}	9.56E+02^{+}	9.34E+02^{+}	9.25E+02	9.28E+02	9.29E+02	9.25E+02	9.50E+02^{+}	9.39E+02^{+}	1.13E+03^{+}	9.37E+02^{+}	1.21E+03^{+}	1.10E+03^{+}	9.39E+02^{+}	**9.17E+02**	9.39E+02^{+}	9.36E+02^{+}
f_{46}	9.59E+02^{+}	9.33E+02^{+}	9.25E+02	9.28E+02	9.29E+02	9.25E+02	9.50E+02^{+}	9.39E+02^{+}	1.12E+03^{+}	9.37E+02^{+}	1.21E+03^{+}	1.10E+03^{+}	9.39E+02^{+}	**9.17E+02**	9.40E+02^{+}	9.36E+02^{+}
f_{47}	8.02E+02	1.02E+03	1.01E+03	1.02E+03	1.01E+03	1.02E+03	1.02E+03^{+}	1.01E+03	1.28E+03^{+}	6.57E+02	1.31E+03^{+}	1.23E+03	**6.34E+02**	1.01E+03	9.42E+02	1.02E+03
f_{48}	9.68E+02^{+}	9.58E+02	9.34E+02	**9.09E+02**	9.22E+02	9.18E+02	**9.09E+02**	9.80E+02^{+}	1.17E+03^{+}	9.70E+02^{+}	1.37E+03^{+}	1.11E+03^{+}	9.57E+02^{+}	9.56E+02	9.20E+02	9.92E+02^{+}
f_{49}	9.76E+02	1.02E+03^{+}	1.02E+03	1.02E+03	1.01E+03	1.02E+03	1.02E+03	1.01E+03	1.28E+03^{+}	9.50E+02	1.04E+03	1.26E+03^{+}	8.24E+02	1.02E+03	**7.29E+02**	1.02E+03
f_{50}	9.42E+02	9.84E+02	1.11E+03^{+}	9.57E+02	9.38E+02	9.85E+02	9.54E+02	9.39E+02	1.27E+03^{+}	**2.00E+02**	1.35E+03^{+}	1.30E+03^{+}	9.72E+02	1.15E+03	9.06E+02	9.79E+02
Wins	9	5.	6	6	7	**11**	1	1	1	8	5	2	1	8	1	0
Av.MED	4.02E+05	1.18E+05	3.63E+05	2.58E+05	9.43E+05	1.98E+05	7.16E+07	2.26E+05	1.35E+05	4.84E+06	3.51E+06	**8.32E+04**	6.22E+06	3.12E+05	5.45E+05	5.89E+05
Av.Ranking	6.38	7.6	5.66	5.7	6.44	**5.1**	13.46	9.1	12.02	7.66	11.44	10.5	11.4	7.16	7.18	12.4
Wilcoxon test	$\approx$	$\approx$	$\approx$	$\approx$	$\approx$		$+$	$+$	$+$	$\approx$	$+$	$+$	$+$	$\approx$	$\approx$	$+$

[a] In the row "Wilcoxon test", we use the symbol $\approx$ to indicate those cases in which the Wilcoxon test with confidence at 0.95 using Bonferroni's correction did not return a p-value lower than $\alpha = 0.05$; and the symbol $+$ to indicate those cases in which the difference was significant (i.e., $p < \alpha$).

Table 10: Average ranking (Av.Ranking) and average median error (Av.MEDerr) obtained by the six automatically generated PSO-X algorithms and ten other PSO algorithms in f_{1-12} (unimodal), f_{13-26} (multimodal), f_{27-50} (hybrid) and f_{rotated}.

f#		PSO-X_{all}	PSO-X_{uni}	PSO-X_{mid}	PSO-X_{hyb}	PSO-X_{soco}	PSO-X_{cec}	GauPSO$_{ind}$	ResPSO$_{ind}$	ERiPSO$_{dft}$	FinPSO$_{ind}$	FraPSO$_{dft}$	LcRPSO$_{dft}$	HiePSO$_{ind}$	IncPSO$_{ind}$	StaPSO$_{ind}$	SPSO11$_{ind}$
f_{1-12}	Av.Ranking	5.92	6.92	7.75	6.75	5.5	**5.25**	15.33	9.08	9.17	7.75	9.67	7.92	11.83	8.83	8.58	12.8
	Av.MEDerr	1.51E+06	2.82E+05	1.34E+06	8.92E+05	3.28E+06	6.36E+05	2.76E+08	7.27E+05	3.63E+05	6.31E+06	3.53E+06	**1.39E+05**	2.57E+07	1.13E+06	2.07E+06	2.21E+06
f_{13-26}	Av.Ranking	5.88	7.12	**4.54**	5.23	7.08	4.85	13.9	8.88	11.92	8.35	10.77	10.04	12.35	6.77	6.85	12.65
	Av.MEDerr	1.05E+03	7.72E+02	**3.38E+02**	7.42E+02	2.53E+03	6.61E+02	1.52E+07	4.58E+03	3.01E+03	2.72E+03	4.30E+06	1.91E+03	4.09E+03	2.32E+03	1.37E+03	3.53E+03
f_{27-50}	Av.Ranking	**3.28**	6.16	3.76	3.6	5.36	5.5	9.36	8.34	11.24	8.68	11.24	12.18	11.92	10.3	11.1	14.28
	Av.MEDerr	1.27E+04	3.07E+04	**1.10E+04**	2.02E+04	2.44E+05	2.08E+04	1.42E+06	3.49E+04	2.67E+04	6.59E+06	2.66E+06	3.07E+04	4.94E+04	1.44E+04	2.82E+04	4.45E+04
f_{rotated}	Av.Ranking	7.45	6.91	5.64	5	7.36	**4.68**	12.09	10	10.86	8.77	14.18	9.59	11	5.64	6.82	10.9
	Av.MEDerr	8.19E+05	1.46E+05	7.28E+05	4.83E+05	1.79E+06	3.46E+05	1.51E+08	3.94E+05	1.97E+05	3.44E+06	1.93E+06	**7.53E+04**	1.40E+07	6.16E+05	1.12E+06	1.20E+06
f_{1-50}	Av.Ranking	6.2	7.54	5.54	5.5	6.34	**4.92**	13.28	8.94	11.98	7.66	11.44	10.48	11.34	6.9	7.1	12.3
	Av.MEDerr	3.69E+05	8.33E+04	3.27E+05	2.24E+05	9.09E+05	1.63E+05	7.16E+07	1.92E+05	1.01E+05	4.81E+06	3.47E+06	**4.91E+04**	6.19E+06	2.79E+05	5.12E+05	5.53E+05

[a] We show in boxes the rankings of the PSO variants that are better than, or as performing as, any of the automatically generated PSO-X algorithm.

According to Wilcoxon pair-wise tests between PSO-X_{cec} and the PSO variants using the data presented in Table 9, the median solution values obtained by FinPSO$_{tnd}$, StaPSO$_{tnd}$ and IncPSO$_{tnd}$ are not statistically different from PSO-X_{cec}. FinPSO$_{tnd}$ was the most performing of the PSO variants in the unimodal functions, and IncPSO$_{tnd}$ in the multimodal and rotated functions. The three PSO variants have some commonalities regarding their design, including that they all use low connected topologies (Von Nemann and ring) during most of their execution (see Table 5) and, in the case of FinPSO$_{tnd}$ and IncPSO$_{tnd}$, they both use the CCVUR. While our experimental results show that only IncPSO$_{tnd}$ and StaPSO$_{tnd}$ are clearly better than one of our algorithms (PSO-X_{uni}) across the whole benchmark set, the three PSO variants produced results that are competitive with the PSO-X algorithms in some specific classes of functions.

Finally, it is worth pointing out that none of the default versions of the PSO variants that we included in our comparison (that is, ERiPSO$_{dft}$, FraPSO$_{dft}$ and LcRPSO$_{dft}$) was as performing as the ones that were configured with `irace`. It is particularly interesting the case of SPSO and FinPSO, whose performance improved dramatically after the configuration process. However, it is extremely common to see these two variants implemented with default parameters in many papers proposing and comparing new algorithms.

7 Conclusions

In this article, we have proposed PSO-X, a flexible, automatically configurable framework that combines algorithm components and automatic configuration tools to create high performing PSO implementations. Six PSO algorithms were automatically created from the PSO-X framework and compared with ten well-known PSO variants published in the literature. The results obtained after solving a set of 50 benchmark functions with different characteristics and complexity showed that the automatically created PSO-X algorithms exhibited higher performance than their manually created counterparts.

In PSO-X, we have incorporated many relevant ideas proposed in the literature for the PSO algorithm, including: different topologies, models of influence and ways of handling the population; several strategies to set the value of the algorithm parameters; a number of ways to construct and apply random matrices; and various kinds of distributions of particles positions in the search space. With PSO-X, we seek to provide a tool that can simplify the application of PSO to tackle continuous optimization problems, and also to bring clarity on the main design choices available when implementing it. There is, however, one clear limitations in our work: since PSO is a intensively studied algorithm with hundreds of variants, including in PSO-X the totality of the ideas proposed for this algorithm is challenging. Hence, a continuous effort must be done to keep adding new algorithms to PSO-X so that implementations remain competitive with the state-of-the-art.

As future work, we are planning to explore two directions. The first one is to create a version of PSO-X from which hybrid PSO algorithms can be created; we are particularly interested in including components from exact methods (e.g., Nelder-Mead Simplex method [24]) and from evolutionary computation (e.g., evolutionary random grouping [35]), which have been shown to be highly competitive and even the state-of-the-art for many problems. The second direction consists extending PSO-X with components from recent stochastic optimization algorithms, in particular those that are controversial (see [14] and the references in the article), in order to see if we can highlight similarities between those algorithms and what have been proposed in the context of PSO.

References

1. M Senthil Arumugam, Machavaram Venkata Chalapathy Rao, and Alan WC Tan. A novel and effective particle swarm optimization like algorithm with extrapolation technique. *Applied Soft Computing*, 9(1):308–320, 2009.
2. Doğan Aydın, Gürcan Yavuz, and Thomas Stützle. ABC-X: a generalized, automatically configurable artificial bee colony framework. *Swarm Intelligence*, 11(1):1–38, 2017.
3. Thomas Bäck, David B. Fogel, and Zbigniew Michalewicz. *Handbook of evolutionary computation*. IOP Publishing, 1997.
4. Alec Banks, Jonathan Vincent, and Chukwudi Anyakoha. A review of particle swarm optimization. part i: background and development. *Natural Computing*, 6(4):467–484, 2007.
5. Alec Banks, Jonathan Vincent, and Chukwudi Anyakoha. A review of particle swarm optimization. part ii: hybridisation, combinatorial, multicriteria and constrained optimization, and indicative applications. *Natural Computing*, 7(1):109–124, 2008.
6. Leonardo César Teonácio Bezerra, Manuel López-Ibáñez, and Thomas Stützle. Deconstructing multi-objective evolutionary algorithms: An iterative analysis on the permutation flowshop. In Panos M. Pardalos, Mauricio G. C. Resende, Chrysafis Vogiatzis, and Jose L. Walteros, editors, *Learning and Intelligent Optimization, 8th International Conference, LION 8*, volume 8426 of *Lecture Notes in Computer Science*, pages 57–172. Springer, Heidelberg, Germany, 2014.
7. Leonardo César Teonácio Bezerra, Manuel López-Ibáñez, and Thomas Stützle. Automatic component-wise design of multi-objective evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 20(3):403–417, 2016.
8. Christian Blum and Enrique Alba, editors. *Genetic and Evolutionary Computation Conference, GECCO 2013, Proceedings, Amsterdam, The Netherlands, July 6-10, 2013*. ACM Press, New York, NY, 2013.
9. Mohammad Reza Bonyadi, Xiang Li, and Zbigniew Michalewicz. A hybrid particle swarm with velocity mutation for constraint optimization problems. In Blum and Alba [8], pages 1–8.
10. Mohammad Reza Bonyadi and Zbigniew Michalewicz. A locally convergent rotationally invariant particle swarm optimization algorithm. *Swarm Intelligence*, 8(3):159–198, 2014.
11. Mohammad Reza Bonyadi and Zbigniew Michalewicz. Particle swarm optimization for single objective continuous space problems: a review, 2017.
12. Mohammad Reza Bonyadi, Zbigniew Michalewicz, and Xiang Li. An analysis of the velocity updating rule of the particle swarm optimization algorithm. *Journal of Heuristics*, 20(4):417–452, 2014.
13. Christian Leonardo Camacho-Villalón, Marco Dorigo, and Thomas Stützle. PSO-X: A component-based framework for the automatic design of particle swarm optimization algorithms: Supplementary material. `http://iridia.ulb.ac.be/supp/IridiaSupp2021-001/`, 2021.
14. Christian Leonardo Camacho-Villalón, Thomas Stützle, and Marco Dorigo. Grey wolf, firefly and bat algorithms: Three widespread algorithms that do not contain any novelty. In *International Conference on Swarm Intelligence*, pages 121–133. Springer, 2020.
15. DeBao Chen and ChunXia Zhao. Particle swarm optimization with adaptive population size and its application. *Applied Soft Computing*, 9(1):39–48, 2009.

16. Maurice Clerc. Standard particle swarm optimisation from 2006 to 2011. open archive HAL hal-00764996, HAL, 2011.
17. 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.
18. Frans Van den Bergh and Andries Petrus Engelbrecht. A new locally convergent particle swarm optimiser. In *Proceedings of the 2002 IEEE international conference on systems, man and cybernetics - SMC*, volume 3, pages 6–pp. IEEE Press, October 2002.
19. Kirk L Duffin and William A Barrett. Spiders: a new user interface for rotation and visualization of n-dimensional point sets. In *Proceedings Visualization'94*, pages 205–211. IEEE, 1994.
20. Russell Eberhart and James Kennedy. A new optimizer using particle swarm theory. In *Proceedings of the Sixth International Symposium on Micro Machine and Human Science*, pages 39–43, 1995.
21. Russell Eberhart and Yuhui Shi. Comparing inertia weights and constriction factors in particle swarm optimization. In *Proceedings of the 2000 Congress on Evolutionary Computation (CEC'00)*, pages 84–88, Piscataway, NJ, July 2000. IEEE Press.
22. Alberto Franzin and Thomas Stützle. Revisiting simulated annealing: A component-based analysis. *Computers & Operations Research*, 104:191 – 206, 2019.
23. José García-Nieto and Enrique Alba. Restart particle swarm optimization with velocity modulation: a scalability test. *Soft Computing*, 15(11):2221–2232, 2011.
24. 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, *Ant Colony Optimization and Swarm Intelligence, 5th International Workshop, ANTS 2006*, volume 4150 of *Lecture Notes in Computer Science*, pages 436–443. Springer, Heidelberg, Germany, 2006.
25. Kyle Robert Harrison, Andries Petrus Engelbrecht, and Beatrice M. Ombuki-Berman. Inertia weight control strategies for particle swarm optimization. *Swarm Intelligence*, 10(4):267–305, 2016.
26. Kyle Robert Harrison, Andries Petrus Engelbrecht, and Beatrice M. Ombuki-Berman. Self-adaptive particle swarm optimization: a review and analysis of convergence. *Swarm Intelligence*, 12(3):187–226, 2018.
27. Francisco Herrera, Manuel Lozano, and D. Molina. Test suite for the special issue of Soft Computing on scalability of evolutionary algorithms and other metaheuristics for large scale continuous optimization problems. `http://sci2s.ugr.es/eamhco/`, 2010.
28. Holger H. Hoos. Automated algorithm configuration and parameter tuning. In Y. Hamadi, E. Monfroy, and F. Saubion, editors, *Autonomous Search*, pages 37–71. Springer, Berlin, Germany, 2012.
29. 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, 2008.
30. Stefan Janson and Martin Middendorf. A hierarchical particle swarm optimizer and its adaptive variant. *IEEE Transactions on Systems, Man, and Cybernetics – Part B*, 35(6):1272–1282, 2005.
31. Johannes Jordan, Sabine Helwig, and Rolf Wanka. Social interaction in particle swarm optimization, the ranked fips, and adaptive multi-swarms. In Ryan [51], pages 49–56.
32. James Kennedy. Bare bones particle swarms. In *Proceedings of the 2003 IEEE Swarm Intelligence Symposium. SIS'03 (Cat. No. 03EX706)*, pages 80–87. IEEE, 2003.

33. James Kennedy and Rui Mendes. Population structure and particle swarm performance. In *Proceedings of the 2002 Congress on Evolutionary Computation (CEC'02)*, pages 1671–1676, Piscataway, NJ, 2002. IEEE Press.
34. Per Kristian Lehre and Carsten Witt. Finite first hitting time versus stochastic convergence in particle swarm optimisation. In *Advances in metaheuristics*, pages 1–20. Springer, 2013.
35. Xiaodong Li and Xin Yao. Cooperatively coevolving particle swarms for large scale optimization. *IEEE Transactions on Evolutionary Computation*, 16(2):210–224, 2011.
36. J.J. Liang, B.Y. Qu, and Ponnuthurai N. Suganthan. Problem definitions and evaluation criteria for the CEC 2014 special session and competition on single objective real-parameter numerical optimization. 2005.
37. Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Thomas Stützle, and Mauro Birattari. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58, 2016.
38. Manuel López-Ibáñez and Thomas Stützle. The automatic design of multi-objective ant colony optimization algorithms. *IEEE Transactions on Evolutionary Computation*, 16(6):861–875, 2012.
39. Franco Mascia, Manuel López-Ibáñez, Jérémie Dubois-Lacoste, and Thomas Stützle. Grammar-based generation of stochastic local search heuristics through automatic algorithm configuration tools. *Computers & Operations Research*, 51:190–199, 2014.
40. Rui Mendes, James Kennedy, and José Neves. The fully informed particle swarm: simpler, maybe better. *IEEE Transactions on Evolutionary Computation*, 8(3):204–210, 2004.
41. Cleve Moler and Charles Van Loan. Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later. *SIAM Review*, 45(1):3–49, 2003.
42. 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.
43. Marco A. Montes de Oca, Thomas Stützle, Ken Van den Enden, and Marco Dorigo. Incremental social learning in particle swarms. *IEEE Transactions on Systems, Man, and Cybernetics – Part B*, 41(2):368–384, 2010.
44. Federico Pagnozzi and Thomas Stützle. Automatic design of hybrid stochastic local search algorithms for permutation flowshop problems. *European Journal of Operational Research*, 276(2):409–421, 2019.
45. Jorge Peña. Theoretical and empirical study of particle swarms with additive stochasticity and different recombination operators. In Ryan [51], pages 95–102.
46. Riccardo Poli. Analysis of the publications on the applications of particle swarm optimisation. *Journal of Artificial Evolution and Applications*, 2008, 2008.
47. Riccardo Poli. Mean and variance of the sampling distribution of particle swarm optimizers during stagnation. *IEEE Transactions on Evolutionary Computation*, 13(4):712–721, 2009.
48. Riccardo Poli, James Kennedy, and Tim Blackwell. Particle swarm optimization. *Swarm Intelligence*, 1(1):33–57, 2007.
49. Asanga Ratnaweera, Saman K Halgamuge, and Harry C Watson. Self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients. *IEEE Transactions on Evolutionary Computation*, 8(3):240–255, 2004.
50. Toby J Richer and Tim Blackwell. The lévy particle swarm. In *Proceedings of the 2006 Congress on Evolutionary Computation (CEC 2006)*, pages 808–815, Piscataway, NJ, July 2006. IEEE Press.

51. Conor Ryan, editor. *Genetic and Evolutionary Computation Conference, GECCO 2008, Proceedings, Atlanta, Georgia, USA July 12-16, 2008*. ACM Press, New York, NY, 2008.
52. Manuel Schmitt and Rolf Wanka. Particles prefer walking along the axes: Experimental insights into the behavior of a particle swarm. In Blum and Alba [8], pages 17–18.
53. Yuhui Shi and Russell Eberhart. A modified particle swarm optimizer. In Patrick K. Simpson, Karen Haines, Jacek Zurada, and David Fogel, editors, *Proceedings of the 1998 IEEE International Conference on Evolutionary Computation (ICEC'98)*, pages 69–73, Piscataway, NJ, 1998. IEEE Press.
54. Thomas Stützle and Manuel López-Ibáñez. Automatic (offline) configuration of algorithms. In Juan Luis Jiménez Laredo, Sara Silva, and Anna I. Esparcia-Alcázar, editors, *GECCO (Companion)*, pages 681–702. ACM Press, New York, NY, 2015.
55. Thomas Stützle and Manuel López-Ibáñez. Automated design of metaheuristic algorithms. In Michel Gendreau and Jean-Yves Potvin, editors, *Handbook of Metaheuristics*, volume 272 of *International Series in Operations Research & Management Science*, pages 541–579. Springer, 2019.
56. Ponnuthurai N. Suganthan, Nikolaus Hansen, J. J. Liang, Kalyanmoy Deb, Y. P. Chen, A. Auger, and S. Tiwari. Problem definitions and evaluation criteria for the CEC 2005 special session on real-parameter optimization. Technical report, Nanyang Technological University, Singapore, 2005.
57. Daniel N Wilke, Schalk Kok, and Albert A Groenwold. Comparison of linear and classical velocity update rules in particle swarm optimization: Notes on scale and frame invariance. *International Journal for Numerical Methods in Engineering*, 70(8):985–1008, 2007.
58. Zhao Xinchao. A perturbed particle swarm algorithm for numerical optimization. *Applied Soft Computing*, 10(1):119–124, 2010.
59. Mauricio Zambrano-Bigiarin, Maurice Clerc, and Rodrigo Rojas. Standard particle swarm optimisation 2011 at cec-2013: A baseline for future pso improvements. In *Proceedings of the 2013 Congress on Evolutionary Computation (CEC 2013)*, pages 2337–2344, Piscataway, NJ, 2013. IEEE Press.