PSO-*X*: A Component-Based Framework for the Automatic Design of Particle Swarm Optimization Algorithms

Christian L. Camacho-Villalón, Marco Dorigo, *Fellow, IEEE*, and Thomas Stützle, *Fellow, IEEE*

***Abstract*—The particle swarm optimization (PSO) algorithm has been the object of many studies and modifications for more than 25 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 article, 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 25 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.**

***Index Terms*—Automatic algorithm design, continuous optimization, particle swarm optimization (PSO).**

I. 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 [1], [2]. 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 [3]. 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 [5], [6], which includes automatic configuration tools 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 [3] algorithms are seen as a particular combination of algorithm components. The design of algorithms using a component-based approach relies on three key elements: 1) a software framework from which algorithm components can be selected; 2) a set of rules indicating a coherent way to combine the components in the software framework; and 3) 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 algorithms, such as ant colony optimization [5] and artificial bee colony [7], there are very few previous work attempting the automatic design of PSO algorithms. The two most relevant of these works are [8] and [9], where the authors used grammatical evolution (GE) to evolve novel velocity update rules in PSO. The main limitation of these works is the low number of different components that can be combined. In [8], only the social and cognitive components of the velocity update rule can be automatically designed; in [9], the list of components includes also the topology and swarm size, but the grammar that defines the rules to combine components is based on the standard version of PSO and makes difficult to include recent algorithm components.

To overcome these limitations, in this article, 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. Most of such flexibility is achieved through the use of a generalized

Manuscript received February 22, 2021; revised June 4, 2021; accepted July 29, 2021. Date of publication August 5, 2021; date of current version May 30, 2022. This work was supported by the Belgian F.R.S.–FNRS. *(Corresponding author: Christian L. Camacho-Villalón.)*

The authors are with the Institut de Recherches Interdisciplinaires et de Développements en Intelligence Artificielle (IRIDIA), Université Libre de Bruxelles, 1050 Bruxelles, Belgium (e-mail: christian.camacho.villalon@ulb.be; mdorigo@ulb.ac.be; thomas.stuetzle@ulb.be).

This article has supplementary downloadable material available at https://doi.org/10.1109/TEVC.2021.3102863, provided by the authors.

Digital Object Identifier 10.1109/TEVC.2021.3102863

velocity update rule (GVUR)—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 GVUR, 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 six automatically generated PSO implementations with ten of the best-known variants proposed in the literature over a set of 50 benchmark problems for evaluating continuous optimizers.

The remainder of this article is structured as follows. In Section II, we present a review of the main concepts of PSO and of the mechanism used by `irace` to automatically create high-performing implementations. In Section III, we identify particular design choices proposed since the earliest PSO publication and discuss their functional purpose in the implementation of the algorithm. In Section IV, we introduce our PSO algorithm template and, from Sections IV-A–IV-E, we explain how it can be used to create PSO variants. The experimental procedure we followed is explained in Section V. The results of the experiments conducted to evaluate the performance of the PSO-*X* algorithms are presented in Section VI. In Section VII, we conclude this article and mention future research directions.

II. Preliminaries

A. Continuous Optimization Problems

In this article, 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 $f : S \subseteq \mathbb{R}^d \to \mathbb{R}$ by finding a vector $\vec{o} \in S$ such that $\forall\, \vec{x} \in S$, $f(\vec{o}) \leq f(\vec{x})$. The search space S is a subset of $\mathbb{R}^d$ in which a solution is represented by a real-valued vector $\vec{x}$, and each component x^j of $\vec{x}$ 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 $\vec{o}$ represents the solution for which the evaluation function $f(\cdot)$ returns the minimum value.

B. Particle Swarm Optimization

Particle swarm optimization [10] 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: 1) the position $\vec{x}_t^{\,i}$; 2) the velocity $\vec{v}_t^{\,i}$; and 3) the personal best position $\vec{p}_t^{\,i}$. The vector $\vec{x}_t^{\,i}$ 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 [11]—and the sets I^i— which we refer to as *models 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 (StaPSO) [12], the rule used to update particles' position is

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

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

The velocity update rule of StaPSO is defined as follows:

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

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 (ACs) that control the influence of $(\vec{p}_t^{\,i} - \vec{x}_t^{\,i})$ and $(\vec{l}_t^{\,i} - \vec{x}_t^{\,i})$. The goal of vectors $(\vec{p}_t^{\,i} - \vec{x}_t^{\,i})$ and $(\vec{l}_t^{\,i} - \vec{x}_t^{\,i})$, respectively, known as the cognitive influence (CI) and the social influence (SI), is to attract particles toward 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 $\mathcal{U}(0, 1]$; their function is to induce perturbation to the CI and SI vectors.

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

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

C. Automatic Algorithm Configuration

We employed a state-of-the-art offline configuration tool called `irace` [13]. This tool has been shown to be capable of dealing with the task of selecting, configuring, and generating high-performing algorithms by finding good algorithm configurations whose performance can be generalized

to unseen problem instances. To do so, `irace` implements a procedure called *iterated racing* [13], which is based on the machine learning model selection approach called racing [14] and on Friedman's nonparametric two-way analysis of variance. Iterated racing 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 identified using a statistical test based on Friedman's nonparametric two-way analysis of variance by ranks. During the configuration process, which is done sequentially and uses a given computational budget, `irace` adjusts the sampling distribution in order to bias new samplings toward 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 capable of handling the different types of parameters included in our framework, that is, numerical (e.g., ω, φ_1, or φ_2), categorical (e.g., *topology*), and subordinate parameters, that is, parameters that are only necessary for particular values of other parameters (e.g., when the size of the population changes in the implementation, it is necessary to configure the maximum and minimum number of particles in the swarm, but not when the size remains constant).

III. Design Choices in PSO

Many algorithm components have been proposed for PSO over the years [15], [16] 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 the PSO literature [17]. 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. Locally convergent implementations not only guarantee to find a local optimum in the search space but also prevent issues, such as 1) *swarm explosion*, which happens when a particle's velocity vector grows too large and the particle becomes incapable of converging to a point in the search space [19] and 2) *poor problem scalability*, which means that the algorithm performs poorly on high-dimensional problems [20]. In fact, the poor problem scalability issue has become very relevant in the last years because of the increasing number of problems involving large-dimensional spaces where PSO is applicable. It has been observed that unwanted particles roaming in high-dimensional spaces is a substantial part of this issue and that, in variants, such as StaPSO, parameter values for ω, φ_1, and φ_2 that perform well in low-dimensional spaces will most likely perform poorly in large dimensional ones [21]. A number of strategies have been proposed to address this issue, such as reinitialization [22], group-based random diagonal matrices [23], and perturbation mechanisms [20].

In group 2) are the algorithm components used to control the distribution of all next possible positions (DNPPs) 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 StaPSO), the spherical (used in standard PSO 2011 (SPSO11) [24]), and the additive stochastic, which comprises the recombination operators proposed for simple dynamic PSO algorithms [25]. 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—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 $\vec{p}_t^{\,i}$ or $\vec{x}_t^{\,i}$) 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 the 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. 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 [20], [26], avoid stagnation [27], and avoid divergence [28]. Additionally, some of these mechanisms allow to modify the DNPP of the particles; an example is the mechanism proposed in [20], 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 (PM): 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 [28], where a parameter decreases the PM when the best solution found so far has been constantly improving, whereas increases it when the algorithm

is stagnating. Another example is [20], 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 allows only to apply perturbation to individual positions used in the computation of the CI and SI. In the StaPSO algorithm, the random matrices [U_1^i and U_2^i, see (2)] are usually constructed as diagonal matrices with values drawn from $\mathcal{U}(0, 1)$; however, in some implementations of StaPSO (e.g., [16]), the matrices are replaced by two random values r_1^i and r_2^i—in this case, particles oscillate linearly between $\vec{p}_t^{\,i}$ and $\vec{l}_t^{\,i}$ without being able to move in different directions, preventing transformation variance [16] but affecting the performance of the algorithm. Using random rotation matrices (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 toward some particular reference frame. The two main methods that have been used to create RRMs in the context of PSO are exponential map [29] and Euclidean rotation [30].

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 the hierarchical and small-world network. In [31], 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 [32], 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 [33], [34]. The number of particles in the swarm has an impact on the tradeoff between solution quality and speed of the algorithm [16], [34]. 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 (FE) is expensive or when the number of possible FEs is limited.

IV. Designing PSO Algorithms From 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*.

Algorithm 1 Algorithm Template Used by PSO-*X*

Require: set of parameters
1: `swarm` ← INITIALIZE(Population, Topology, Model of influence)
2: **repeat**
3: **for** $i \leftarrow 1$ **to** size(`swarm`) **do**
4: $\vec{v}_{t+1}^{\,i} \leftarrow \omega_1 \vec{v}_t^{\,i} + \omega_2\, \mathsf{DNPP}(i,t) + \omega_3\, \mathsf{Pert}_{\mathsf{rand}}(i,t)$
5: apply velocity clamping %optional
6: $\vec{x}_{t+1}^{\,i} \leftarrow \vec{x}_t^{\,i} + \vec{v}_{t+1}^{\,i}$
7: **end for**
8: **for** $i \leftarrow 1$ **to** size(`swarm`) **do**
9: compute $f(\vec{x}_t^{\,i})$
10: update $\vec{p}_t^{\,i}$ using Eq. 3
11: **end for**
12: apply stagnation detection, particles reinitialization %optional
13: **if** type(Population) $\neq$ constant **then**
14: `swarm` ← UPDATEPOPULATION(`swarm`, Population)
15: **end if**
16: **if** type(Topology) = time-varying **or** type(Population) $\neq$ constant **then**
17: `swarm` ← UPDATETOPOLOGY(`swarm`, Topology, Model of influence)
18: **end if**
19: **until** termination criterion is met
20: **return** *global best solution*

A. 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 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 two *for* cycles of lines 3–7 and lines 8–11 correspond to the standard implementation of PSO—except for line 4 that shows our GVUR, defined as follows:

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

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 StaPSO [see (2)] and its value can be computed using the strategies that have been developed for this purpose (the list of the strategies available to compute its value are shown in Table 4 of the supplementary material [4]). The parameters ω_2 and ω_3 control the influence that will be given to the DNPP and $\mathsf{Pert}_{\mathsf{rand}}$ components; their values can be set equal to ω_1 or be computed using the random component, where $\omega_2, \omega_3 \sim \mathcal{U}[0.5, 1]$, or the constant component, where ω_2 and ω_3 are user selected constants in the interval [0, 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: 1) UPDATEPOPULATION(), that increases/decreases the size of the swarm according to the type of Population employed and 2) UPDATETOPOLOGY(), that connects newly added particles to a set of neighbors, or disconnect particles when the topology connectivity reduces over time.

All implementations that can be created using Algorithm 1 and combining algorithm components with different functionalities, such as different DNPPs or topologies, are considered valid implementations by PSO-*X*. This allows enough flexibility to explore many new designs without increasing too much the computational complexity of the implementations.

In particular, in PSO-X, we do not allow the recursive use of components, as it is done in the component-based framework using grammars [3]. In Table 2 of the supplementary material [4], we show the algorithm components and parameters in the framework, their domain and type, and the condition(s) under which each parameter is used in PSO-X.

B. 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:

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

where Mtx and $\text{Pert}_{\text{info}}$ are, as mentioned before, high-level representations of the different types of random matrices and informed perturbation mechanisms used in PSO. The DNPP-rectangular is by far the most commonly used in implementations of PSO, including StaPSO [12], the constriction coefficient PSO [19], the fully informed PSO (FiPSO) [35], etc. In the standard application of DNPP-rectangular (i.e., as in StaPSO), each term added in (5) is a vector located on a hyperrectangular surface whose side length depends on the distance between $\vec{p}_t^{\,k}$ and $\vec{x}_t^i$. However, when the perturbation component of DNPP-rectangular is an informed Gaussian—as in the locally convergent rotationally invariant PSO (LcRPSO) [20]—or an RRM—as in the diverse rotationally invariant PSO (DvRPSO) [29]—the surface on which the different vectors computed in (5) are located becomes hyperspherical or semi-hyperspherical, respectively.

Another option is DNPP-spherical [24], where a vector located on a hypersphere is used in the computation of a particle new position. The equation to compute the DNPP-spherical option is as follows:

$$\text{DNPP-spherical} = \mathcal{H}_i(\vec{c}_t^{\,i}, |\vec{c}_t^{\,i} - \vec{x}_t^i|) - \vec{x}_t^i \quad (6)$$

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

$$\vec{c}_t^{\,i} = \frac{\vec{x}_t^i + \vec{L}_t^i + \vec{P}_t^i}{3} \quad (7)$$

where

$$\vec{P}_t^i = \vec{x}_t^i + \varphi_{1t} \text{Mtx}_t \left(\text{Pert}_{\text{info}}(\vec{p}_t^{\,i}) - \vec{x}_t^i \right) \quad (8)$$

$$\vec{L}_t^i = \vec{x}_t^i + \sum_{k \in I_t^i \setminus \{i\}}^{n} \varphi_{2t}^k \text{Mtx}_t^k \left(\text{Pert}_{\text{info}}(\vec{p}_t^{\,k}) - \vec{x}_t^i \right) \quad (9)$$

and $\varphi_{2t}^k = (\varphi_{2t}/|I_t^i \setminus \{i\}|)$. The main difference between DNPP-spherical and the standard implementation of DNPP-rectangular is that the hypersphere $\mathcal{H}_i(\vec{c}_t^{\,i}, |\vec{c}_t^{\,i} - \vec{x}_t^i|)$ 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 $\mathcal{H}(\vec{c}_t^{\,i}, |\vec{c}_t^{\,i} - \vec{x}_t^i|)$ and the informants of i participate only in the computation of vector $\vec{L}_t^i$ [see (9)];[1] 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 (5).

The DNPP-standard, DNPP-discrete, DNPP-Gaussian, and DNPP-Cauchy–Gaussian options belong to the class of simple dynamic PSO algorithms [25], [36] and have the form $\vec{q}_t^{\,i} - \vec{x}_t^i$, where vector $\vec{q}_t^{\,i}$ is computed differently in each option

$$\text{DNPP-standard: } \vec{q}_t^{\,i} = \frac{\varphi_1 \vec{p}_t'^{\,i} + \varphi_2 \vec{p}_t'^{\,k}}{\varphi_1 + \varphi_2} \quad (10)$$

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

$$\text{DNPP-Gaussian: } \vec{q}_t^{\,i} = \mathcal{N}\left(\frac{\vec{p}_t'^{\,i} + \vec{p}_t'^{\,k}}{2}, |\vec{p}_t'^{\,i} - \vec{p}_t'^{\,k}|\right) \quad (12)$$

DNPP-Cauchy–Gaussian:

$$\vec{q}_t^{\,i} = \begin{cases} p_t^{i,j} + \mathcal{C}(1) \left| p_t'^{i,j} - p_t'^{k,j} \right|, & \text{if } \mathcal{U}[0,1] \leq r \\ p_t^{k,j} + \mathcal{N}(0,1) \left| p_t'^{i,j} - p_t'^{k,j} \right|, & \text{otherwise} \end{cases} \quad (13)$$

where $\eta_d \sim \mathcal{U}\{0, 1\}$ is a discrete random number drawn from a Bernoulli distribution, $\mathcal{C}(1)$ is a random number generated using a Cauchy distribution with scaling parameter 1, $\mathcal{N}(0, 1)$ 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 (13). Vectors $\vec{p}_t'^{\,i}$ and $\vec{p}_t'^{\,k}$ are computed using $\vec{p}_t'^{\,i} = \text{Pert}_{\text{info}}(\vec{p}_t^{\,i})$ and $\vec{p}_t'^{\,k} = \text{Pert}_{\text{info}}(\vec{p}_t^{\,k})$ 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 jth dimension of $\vec{q}_t^{\,i}$ is computed with probability r using $\vec{p}_t^{\,i}$ and a Cauchy distribution; and with probability $1 - r$ using $\vec{p}_t^{\,k}$ and a Normal distribution.

Although we kept the original definition of these DNPPs for the most part, we did two modifications: we included the $\text{Pert}_{\text{info}}$ component (i.e., vectors $\vec{p}_t'^{\,i}$ and $\vec{p}_t'^{\,k}$ instead of $\vec{p}_t^{\,i}$ and $\vec{p}_t^{\,k}$) 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.

C. $\text{Pert}_{\text{rand}}$ and $\text{Pert}_{\text{info}}$ Components

The two types of perturbation components included in PSO-X are: 1) $\text{Pert}_{\text{info}}$, which modifies an input vector and 2) $\text{Pert}_{\text{rand}}$, which generates a random vector that is added to the velocity vector. $\text{Pert}_{\text{info}}$, as explained in Section IV-B, is a component used by the DNPP component. Differently, $\text{Pert}_{\text{rand}}$ is used directly in the GVUR.

As shown in Table I, both $\text{Pert}_{\text{info}}$ and $\text{Pert}_{\text{rand}}$ are optional components in PSO-X that can be omitted from the implementation using the none option. The options for $\text{Pert}_{\text{info}}$, when the component is present in the implementation,

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

TABLE I
OPTIONS FOR COMPUTING $\text{PERT}_{\text{INFO}}$ AND $\text{PERT}_{\text{RAND}}$ COMPONENTS IN PSO-*X* WHEN THEY ARE USED IN THE IMPLEMENTATION

Component	Option	Definition
$\text{Pert}_{\text{info}}$ *	none	—
	$\text{Pert}_{\text{info}}$-Gaussian	$\mathcal{N}(\vec{r}, \sigma_t)$
	$\text{Pert}_{\text{info}}$-Lévy	$L_{\gamma_t}(\vec{r}, \sigma_t)$
	$\text{Pert}_{\text{info}}$-uniform	$\vec{r} + (\vec{s} \odot \vec{r})$, with $\vec{s} \sim \mathcal{U}[-b_t, b_t]$
$\text{Pert}_{\text{rand}}$ **	none	—
	$\text{Pert}_{\text{rand}}$-rectangular	$\tau_t\,(1 - 2 \cdot \mathcal{U}(0,1))$
	$\text{Pert}_{\text{rand}}$-noisy	$\mathcal{U}[-\delta_t/2, \delta_t/2]$

* In the options for computing $\text{Pert}_{\text{info}}$: $\vec{r}$ is the input vector; $\mathcal{N}(\vec{r}, \sigma_t)$ is a Normal distribution with mean $\vec{r}$ and variance σ_t; $L_\gamma(\vec{r}, \sigma_t)$ is a Lévy distribution with mean $\vec{r}$, variance σ_t, and scale parameter γ_t; and b_t is a real parameter.
** In the options for computing $\text{Pert}_{\text{rand}}$: τ_t and δ_t are two real parameters.

are $\text{Pert}_{\text{info}}$-Gaussian, $\text{Pert}_{\text{info}}$-Lévy, and $\text{Pert}_{\text{info}}$-uniform. $\text{Pert}_{\text{info}}$-Gaussian and $\text{Pert}_{\text{info}}$-Lévy compute a random vector by using a probability distribution whose center and dispersion are given by the input vector $\vec{r}$ and by the parameter σ_t that controls the magnitude of the perturbation. Similarly, in $\text{Pert}_{\text{info}}$-uniform, the PM depends on a parameter b_t, that controls the interval in which a random vector $\vec{s}$ will be generated using a uniform distribution. Regarding the $\text{Pert}_{\text{rand}}$ component, both $\text{Pert}_{\text{rand}}$-rectangular and $\text{Pert}_{\text{rand}}$-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 $\text{Pert}_{\text{info}}$-Lévy, the value of γ_t can be used to switch between a Gaussian and a Cauchy distribution [37]. 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 $\mathcal{U}\{10, 20\}$

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

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 [36] for computing the DNPP-Cauchy–Gaussian option assuming $r = 0.5$ to give the same probability to each case [see (13)].

Since the 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 $\text{Pert}_{\text{info}}$ and $\text{Pert}_{\text{rand}}$ components. These strategies are PM-constant value, PM-Euclidean distance, PM-obj.func. distance, and PM-success rate.

The PM-constant value strategy [26] is the simplest and consists in using a value that remains constant during the execution of the algorithm. This strategy guarantees that the PM 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 [20] 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:

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

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

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

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

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 [28] 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 $t \to t+1$ is a success if $f(\vec{g}_{t+1}) < f(\vec{g}_t)$, a failure otherwise. The PM-success rate strategy is defined as follows:

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

where the threshold parameters s_c and f_c are user defined.

D. *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 $\mathcal{U}(0, 1)$; 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 [29] option is based on an approximation method called exponential map whereby RRMs can be constructed avoiding matrix multiplication, which is computationally expensive. Mtx-exponential map is defined as

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

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 $\max_\beta = 1$.

The Mtx-Euclidean rotation [30] rotates a vector in any combination of planes.[2] 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 $r_{ii} = r_{jj} = \cos\alpha$, $r_{ij} = -\sin\alpha$, and $r_{ji} = \sin\alpha$, 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 $\vec{v}$ is done as follows:

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

where v_k indicates the kth entry of vector $\vec{v}$. 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.

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.

The last option for the Mtx component is Mtx-Increasing group-based [23] that divides a random diagonal matrix into g_t groups and every element in each group has the same value, generated uniformly random. The number of groups at each iteration is computed using the following equation:

$$g_t = \frac{d-1}{t_{\max} - 1} \times (t-1) + 1 \tag{20}$$

where d is the number of problem dimensions and $t_{\max}$ is the iteration number at which the algorithm stops. Note the algorithm starts with $g_t = 1$ and the number increases over time until there are $g_t = d$ groups, which is equivalent to gradually transforming an Mtx-random linear component into a Mtx-random diagonal one.

E. Topology, Model of influence and Population Components

In addition to the well-known options for the Topology component discussed in Sections II-B and III and showed in Table II, we implemented in PSO-X the Top-hierarchical and Top-time-varying options.

TABLE II
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 [38], 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 $\vec{p}_t$ 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 $\vec{p}_t$ vector with that of its parent and switching places when it has higher quality.

The Top-time-varying [31] 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 [33], there is a maximum ($\mathrm{pop}_{\max}$) and minimum ($\mathrm{pop}_{\min}$) number of particles that can be in the swarm at any given time. Particles are added or removed according to two criteria: 1) add one particle if the best solution found has not improved in the previous k consecutive iterations and the swarm size is smaller than $\mathrm{pop}_{\max}$ and 2) remove the particle with the lowest quality if the best solution found has improved in the previous k consecutive iterations and the swarm size is larger than $\mathrm{pop}_{\min}$. Whenever criterion 1) is verified, but the

[2]For a d-dimensional vector $\vec{u}$, there is a composition of $d(d-1)/2$-D rotation matrices built up in order to rotate $\vec{u}$ in all possible combinations of planes. See [30, Appendix III] for further details.

swarm size is equal to $\texttt{pop}_{\max}$, the particle with the lowest quality is removed before adding the new random particle.

In Pop-incremental [34], 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

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

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

$$x'^{\text{new},j} = \mathcal{U}\left[lb^j, ub^j\right]$$
$$x^{\text{new},j} = x'^{\text{new},j} + \mathcal{U}[0,1) \cdot \left(g_t^j - x'^{\text{new},j}\right).$$

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.

1) *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.
2) *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.
3) *Particles Are Removed:* The topology is repaired to ensure that every particle has the right number of neighbors.

F. Acceleration Coefficients

The four strategies that can be used to compute the ACs in PSO-*X* are: 1) AC-constant; 2) AC-random; 3) AC-time-varying; and 4) AC-extrapolated. In AC-random, the value of φ_{1t} and φ_{2t} is drawn from $\mathcal{U}[\varphi_{\min}, \varphi_{\max}]$, 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 [39], where φ_1 decreases from 2.5 to 0.5 and φ_2 increases from 0.5 to 2.5. In the AC-extrapolated strategy, proposed in [40], the value of the ACs is a function of the iteration number and particles quality computed as follows:

$$\varphi_1 = e^{-(t/t_{\max})}$$
$$\varphi_2 = e^{(\varphi_1 \cdot \Lambda_t^i)} \tag{21}$$

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

G. 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 [41]. 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 $||\vec{v}_t^i|| + ||\vec{g}_t - \vec{x}_t^i|| \leq \mu$, 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:

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

where $\vec{r} \sim \mathcal{U}(0, 1]$.

The second component, particles reinitialization [22] 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 $\vec{p}_t$ 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 $\vec{g}_t$ is lower than 10^{-8} for $10 \cdot d/\texttt{pop}$ iterations and regenerating particles positions using the following equation:

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

The last one is velocity clamping [10], [42] 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:

$$\vec{v}_{t+1}^{\,j} \begin{cases} v^{\max}, & \text{if } \vec{v}_{t+1}^{\,j} > v^{\max} \\ -v^{\max}, & \text{if } \vec{v}_{t+1}^{\,j} < -v^{\max} \\ \vec{v}_{t+1}^{\,j}, & \text{otherwise} \end{cases} \tag{22}$$

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

V. Experimental Procedure

A. Benchmark Problems

We conducted experiments on a set of 50 static benchmark continuous functions belonging to the CEC'05 and CEC'14 "Special Session on Single-Objective Real-Parameter Optimization" [43], [44], and to the Soft Computing (SOCO'10) "Test Suite on Scalability of EAs and Other Metaheuristics for Large-Scale Continuous Optimization Problems" [45]. A detailed description of the benchmark functions can be found in the given references and in the supplementary material [4] of this article.

The test set of continuous functions—Table V-B—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.

B. 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 58 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. In order to present statistically meaningful results, we perform 50 independent runs of each algorithm 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 FEs. Both the tuning and the experiments were carried out on a single-core Intel Xeon E5-2680 running at 2.5 GHz 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.[3] The version of `irace` is 3.2.

VI. 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 50 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 as follows.

1) Enhanced rotation-invariant PSO [30]—a variant that uses the AC-random strategy, Mtx-Euclidean rotation, and α-adaptive.
2) Fully informed PSO [35]—a traditional PSO variant that uses the constriction coefficient velocity update rule[4] (CCVUR) and the Mol-fully informed.
3) Frankenstein's PSO [31]—a PSO variant that uses Top-time-varying, Mol-fully informed, and ω_1 = linear decreasing.
4) Gaussian "bare-bones" PSO [46] (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 [38] (HiePSO)—a variant based on Top-hierarchical that can be implemented using either ω_1 = linear decreasing or ω_1 = linear increasing.
6) Incremental PSO [34] (IncPSO)—a variant of PSO that uses the CCVUR and Pop-incremental with Init-horizontal.
7) Locally convergent rotation-invariant PSO [20]—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 [22] (ResPSO)—a variant of StaPSO using velocity clamping and particles reinitialization.
9) Standard PSO [12]—the PSO algorithm described in Section II-B that uses (1)–(3).
10) Standard PSO 2011—a variant of StaPSO that uses the DNPP-spherical option.

In Table IV, 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 StaPSO. 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, ω_1 = constant, $\omega_2 = 1.0$, $\omega_3 = 0$, and AC-constant.

A. Comparison of Automatically Generated PSO Algorithms

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

1) *PSO-X_{all}:* Pop-incremental with Init-random, Top-fully connected with Mol-best-of-neighborhood, DNPP-rectangular with $\text{Pert}_{\text{info}}$-Lévy and PM-success rate, Mtx-random diagonal, and velocity clamping.
2) *PSO-X_{hyb}:* Pop-constant, Top-Von Neumann with Mol-best-of-neighborhood, DNPP-rectangular with $\text{Pert}_{\text{info}}$-Lévy and PM-success rate, Mtx-random diagonal, and velocity clamping.
3) *PSO-X_{mul}:* Pop-incremental with Init-horizontal, Top-time-varying with Mol-best-of-neighborhood, DNPP-rectangular with $\text{Pert}_{\text{info}}$-Lévy and PM-success rate, Mtx-random linear, and stagnation detection.
4) *PSO-X_{uni}:* Pop-incremental with Init-random, Top-fully connected with Mol-best-of-neighborhood, DNPP-rectangular with $\text{Pert}_{\text{info}}$-Lévy and PM-success rate, Mtx-random diagonal, and velocity clamping.
5) *PSO-X_{cec}:* Pop-constant, Top-Von Neumann with Mol-best-of-neighborhood, 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.

[3]The source code of PSO-X can be downloaded from http://iridia.ulb.ac.be/supp/IridiaSupp2021-001/PSO-X.zip.

[4]This rule is defined as $\vec{v}_{t+1}^{\,i} = \chi(\vec{v}_t^{\,i} + \varphi_1 U_{1t}^i(\vec{p}_t^{\,i} - \vec{x}_t^{\,i}) + \varphi_2 U_{2t}^i(\vec{l}_t^{\,i} - \vec{x}_t^{\,i}))$, where $\chi = 0.7298$ is called the constriction coefficient [19]. It can obtained from (4) by setting $\omega_1 = \omega_2 = 0.7298$ and using the DNPP-rectangular option.

TABLE III
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 Gbl. 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 Num. 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

TABLE IV
PARAMETER SETTINGS OF THE TEN PSO VARIANTS INCLUDED IN OUR COMPARISON

Algorithm	Settings
ERiPSO$_{dft}$	$\mathrm{pop} = 20$, $\omega_1 = 0.7213475$, AC-random, $\varphi_{1min} = 0$, $\varphi_{1max} = 2.05$, $\varphi_{2min} = 0$, $\varphi_{2max} = 2.05$, Mtx-Euclidean rotation$_{all}$ with α-adaptive and $\zeta = 30$ and $\rho = 0.01$.
FiPSO$_{tmd}$	Top-ring, $\mathrm{pop} = 20$, $\omega_1 = \omega_2 = 0.729843788$, Mtx-random diagonal, $\varphi_1 = 2.1864$ and $\varphi_2 = 2.3156$.
FraPSO$_{dft}$	$\kappa = 60$, $\mathrm{pop} = 60$, $\omega_1 =$ linear decreasing, $\omega_{1min} = 0.4$, $\omega_{1max} = 0.9$, $t_{schd} = 600$, $\varphi_1 = 2.0$ and $\varphi_2 = 2.0$.
GauPSO$_{tmd}$	Top-time-varying with MoI-random informant, $\kappa = 150$ $\mathrm{pop} = 30$, $\omega_1 = 0$ and DNPP-additive stochastic DNPP-Gaussian.
HiePSO$_{tmd}$	$bd = 2$, $\mathrm{pop} = 114$, $\omega_1 =$ linear increasing, $\omega_{1min} = 0.3284$, $\omega_{1max} = 0.8791$, $\varphi_1 = 2.1105$ and $\varphi_2 = 1.0349$.
IncPSO$_{tmd}$	Top-time-varying, $\kappa = 2360$, Init-horizontal, $\mathrm{pop}_{ini} = 5$, $\mathrm{pop}_{fin} = 295$, $\xi = 10$, $\omega_1 = \omega_2 = 0.729843788$, $\varphi_1 = 1.9226$ and $\varphi_2 = 1.0582$.
LcRPSO$_{dft}$	$\mathrm{pop} = d$, $\omega_1 = 0.7298$, AC-random, $\varphi_{1min} = 0$, $\varphi_{1max} = 1.4962$, $\varphi_{2min} = 0$, $\varphi_{2max} = 1.4962$, Mtx-random linear, Pert$_{\text{info}}$-Gaussian with PM-Euclidean distance and $\epsilon = 0.46461/d^{0.79}$.
ResPSO$_{tmd}$	Top-ring with MoI-fully informed, $\mathrm{pop} = 10$, $\omega_1 =$ linear decreasing, $\omega_{1min} = 0.2062$, $\omega_{1max} = 0.6446$, $\varphi_1 = 1.5014$, $\varphi_2 = 2.2955$.
SPSO11$_{tmd}$	Top-time-varying with $\kappa = 1085$, $\mathrm{pop} = 155$, $\omega_1 = 0.6482$, $\varphi_1 = 2.2776$, $\varphi_2 = 2.1222$.
StaPSO$_{tmd}$	Top-Von Neumann, $\mathrm{pop} = 34$, $\omega_1 = 0.6615$, $\varphi_1 = 2.3706$, $\varphi_2 = 0.8914$.

* 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.

TABLE V
PARAMETER SETTINGS OF THE SIX AUTOMATICALLY GENERATED PSO-*X* ALGORITHMS

Algorithm	Settings
PSO-X_{all}	$\mathrm{pop}_{ini} = 4$, $\mathrm{pop}_{fin} = 20$, $\xi = 8$, $\omega_1 =$ convergence-based, $a = 0.7192$, $b = 0.9051$, $\omega_2 =$ random, AC-constant, $\varphi_1 = 1.7067$, $\varphi_2 = 2.2144$, PM $= 0.438$, $s_c = 11$ and $f_c = 40$.
PSO-X_{hyb}	$\mathrm{pop} = 41$, $\omega_1 =$ adaptive based onvelocity, $\omega_{1min} = 0.119$, $\omega_{1max} = 0.1378$, $\lambda = 0.608$, $\omega_2 = 1.0$ AC-random, $\varphi_{1min} = 1.0429$, $\varphi_{1max} = 2.1653$, $\varphi_{2min} = 1.0429$, $\varphi_{2max} = 2.3275$, PM $= 0.5333$, $s_c = 28$ and $f_c = 42$.
PSO-X_{mul}	$\kappa = 300$, $\mathrm{pop}_{ini} = 3$, $\mathrm{pop}_{fin} = 50$, $\xi = 2$, $\omega_1 =$ success-based, $\omega_{1min} = 0.4$, $\omega_{1max} = 0.9$, AC-constant, $\varphi_1 = 0.92$, $\varphi_2 = 1.6577$, PM $= 0.5114$, $s_c = 2$ and $f_c = 33$.
PSO-X_{uni}	$\mathrm{pop}_{ini} = 10$, $\mathrm{pop}_{fin} = 58$, $\xi = 3$, $\omega_1 =$ adaptive based onvelocity, $\omega_{1min} = 0.3531$, $\omega_{1max} = 0.7095$, $\lambda = 0.4832$, $\omega_2 = \omega_1$, AC-random, $\varphi_{1min} = 1.4217$, $\varphi_{1max} = 2.051$, $\varphi_{2min} = 0.8626$, $\varphi_{2max} = 1.4609$, PM $= 0.9865$, $s_c = 38$ and $f_c = 11$.
PSO-X_{cec}	$\mathrm{pop} = 42$, $\omega_1 =$ self-regulating, $\omega_{1min} = 0.1673$, $\omega_{1max} = 0.2317$, $\eta = 0.2468$, $\omega_2 =$ random, $\omega_3 =$ random, AC-random, $\varphi_{1min} = 1.8684$, $\varphi_{1max} = 1.9233$, $\varphi_{2min} = 0.2802$, $\varphi_{2max} = 1.5143$, PM$_1 = 0.4837$, $s_{c1} = 29$, $f_{c1} = 45$, PM$_2 = 0.8139$, $s_{c2} = 30$ and $f_{c2} = 43$.
PSO-X_{soco}	$\mathrm{pop} = 19$, $\omega_1 =$ adaptive based onvelocity, $\omega_{1min} = 0.6564$, $\omega_{1max} = 0.8201$, $\lambda = 0.2959$, AC-constant, $\varphi_1 = 0.7542$, $\varphi_2 = 1.9235$, 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.

6) *PSO-X_{soco}:* Pop-constant, Top-ring with MoI-ranked fully informed, DNPP-rectangular with Pert$_{\text{info}}$-Gaussian and PM-success rate, Mtx-random diagonal, and velocity clamping.

In Table VI, we report 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 ("≈") 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

TABLE VI
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		+	≈	≈	≈	≈

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

PSO-X_{soco} was the one that returned the best median result in the higher number of cases. The symbol "+" next to some of the median values in Table VI 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 than PSO-X_{all} in most functions (26 out of 50 functions), while PSO-X_{mul}, PSO-X_{cec}, PSO-X_{hyb}, and PSO-X_{soco} were worse in 23, 19, 17, and 14 functions, respectively. 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.

1) Comparison of the PSO-X Algorithms on Specific Function Classes: In order to know whether our PSO-X algorithms are 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 is the performance of an algorithm across the different classes based on the result of the winner of each function. In Table VII, 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}, that are the algorithms we would expect to obtain better results because of the functions used for creating them.

As shown in Table VII, 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.[5] Another possible reason is the presence of slightly overfitted models during the creation of these algorithms with `irace`. The effect of overfitting can be observed more clearly for PSO-X_{uni} than for the rest of the 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 an exceedingly large computational budget used with `irace`.

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 discuss in Section IV-B, 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 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

[5]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 VII
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

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 toward exploitation, and PSO-X_{uni} toward 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 $\mathsf{Pert}_{\mathsf{rand}}$ component that is present only in PSO-X_{cec}. The $\mathsf{Pert}_{\mathsf{rand}}$ 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 IV). Data from Table VI 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 best performing of the six. However, when PSO-X_{uni} faced less regular and multimodal landscapes, its performance declined significantly. Given the large number of parameters in PSO-*X* compared to most PSO variants in the literature, framework users could be interested in obtaining information about the sampling distribution of the parameters and the way in which they interact with each other. We present this information in Section 4.2 of the supplementary material [4].

B. 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 best performing of the two versions of each algorithm. However, since the computed p-values were larger than 0.05 for enhanced rotation-invariant PSO (ERiPSO), Frankenstein's PSO (FraPSO), and SPSO11, we selected the version that obtained the lower median value across the 50 functions. In Table VIII, we show the median of the 50 runs executed by each algorithm for each function and, in Table IX, 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 Section 6 of the supplementary material [4], 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 $\text{LcRPSO}_{\text{dft}}$ 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 IX, 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 $\text{IncPSO}_{\text{tnd}}$ and $\text{StaPSO}_{\text{tnd}}$ were capable of outperforming the results obtained by some of the PSO-*X* algorithms, especially in the rotated functions, where those two algorithms were as competitive as those 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.

According to Wilcoxon pairwise tests between PSO-X_{cec} and the PSO variants using the data presented in Table VIII, the median solution values obtained by $\text{FinPSO}_{\text{tnd}}$, $\text{StaPSO}_{\text{tnd}}$, and $\text{IncPSO}_{\text{tnd}}$ are not statistically different from PSO-X_{cec}. $\text{FinPSO}_{\text{tnd}}$ was the best performing of the PSO variants in the unimodal functions, and $\text{IncPSO}_{\text{tnd}}$ 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 IV) and, in the case of $\text{FinPSO}_{\text{tnd}}$ and $\text{IncPSO}_{\text{tnd}}$, they both use the CCVUR. While our experimental results show that only $\text{IncPSO}_{\text{tnd}}$ and $\text{StaPSO}_{\text{tnd}}$ are clearly better than one of our algorithms (PSO-X_{uni}) across the whole benchmark set, the three PSO variants ($\text{FinPSO}_{\text{tnd}}$, $\text{StaPSO}_{\text{tnd}}$, and $\text{IncPSO}_{\text{tnd}}$) 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 (i.e., $\text{ERiPSO}_{\text{dft}}$, $\text{FraPSO}_{\text{dft}}$, and $\text{LcRPSO}_{\text{dft}}$) performed as well as

TABLE VIII
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$_{tmd}$	ResPSO$_{tmd}$	ERiPSO$_{dft}$	FinPSO$_{tmd}$	FraPSO$_{dft}$	LcRPSO$_{dft}$	HiePSO$_{tmd}$	IncPSO$_{tmd}$	StaPSO$_{tmd}$	SPSO11$_{tmd}$
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$^+$
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	$\approx$	$\approx$	$\approx$	$\approx$	$\approx$		+	+	+	$\approx$	+	+	+	$\approx$	$\approx$	+

* 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 IX
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_{mul}	PSO-X_{hyb}	PSO-X_{soco}	PSO-X_{cec}	GauPSO$_{tmd}$	ResPSO$_{tmd}$	ERiPSO$_{dft}$	FinPSO$_{tmd}$	FraPSO$_{dft}$	LcRPSO$_{dft}$	HiePSO$_{tmd}$	IncPSO$_{tmd}$	StaPSO$_{tmd}$	SPSO11$_{tmd}$
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

* 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.

the ones that were configured with `irace` in terms of median solution quality. It is particularly interesting the case of StaPSO 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.

C. Are PSO-X Implementations Convergent?

Local convergence is one of the most salient characteristics of high-performing PSO implementations. It prevents unwanted roaming (which often results in particles leaving the search space) and allows particles to improve the initial solutions for any number of dimensions. Creating convergent implementation using PSO-X is possible because 1) the value of the three main parameters of PSO (ω_1, φ_1, and φ_2) is limited within the theoretical region where local convergence is expected to occur[6] and 2) PSO-X is implemented both a number of

[6]In Section 4 of the supplementary material [4], we report the value of these parameters for the six PSO-X algorithms and the ten PSO variants and the conditions for order-1 stability.

algorithm components that have been shown to result in particles local convergence (e.g., DNPP-spherical, $\text{Pert}_{\text{info}}$-Gaussian, $\text{Pert}_{\text{info}}$-Lévy, etc.) and strategies that limit the magnitude of the velocity vector (velocity clamping).

As it can be observed in Tables IV and V, the six PSO-*X* algorithm that PSO-*X* automatically created use the two main algorithm components proposed for the LcRPSO (i.e., DNPP-rectangular and $\text{Pert}_{\text{info}}$-Gaussian).[7] In [20], it was formally and experimentally demonstrated that LcRPSO is locally convergent because the $\text{Pert}_{\text{info}}$-Gaussian component satisfies the local convergence condition, which ensures that the mapping between the input vector $\vec{r}$ and the perturbed vector $\mathcal{N}(\vec{r}, \sigma_t)$ is located in any definable region of the search space (see [20, Appendix 1] for the formal definition of this condition). Although our PSO-*X* algorithms are six different specializations of LcRPSO, by using a number of algorithm components that were found to be good design choices during the configuration process, they exhibit better performance than any of the variants considered in this study. We believe this shows the power of combining automatic configuration and component-based framework to create high-performing algorithms.

VII. Conclusion

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 limitation in our work: since PSO is an 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 [47]) and from evolutionary computation (e.g., evolutionary random grouping [36]), which have been shown to be highly competitive and even the state of the art for many problems. The second direction consists of extending PSO-*X* with components from recent stochastic optimization algorithms, in particular those that are controversial (see [48], and the references in this article), in order to see if we can highlight similarities between those algorithms and what have been proposed in the context of PSO.

[7]Note that, in addition to $\text{Pert}_{\text{info}}$-Gaussian, it is possible to use the $\text{Pert}_{\text{info}}$-Lévy component, since the Lévy distribution is a generalization of the Gaussian.

References

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

[2] T. Bäck, D. B. Fogel, and Z. Michalewicz, *Handbook of Evolutionary Computation*. Bristol, U.K.: IOP Publ., 1997.

[3] T. Stützle and M. López-Ibáñez, "Automated design of metaheuristic algorithms," in *Handbook of Metaheuristics*, vol. 272. Cham, Switzerland: Springer, 2019, pp. 541–579.

[4] C. L. Camacho-Villalón, M. Dorigo, and T. Stützle, "PSO-X: A component-based framework for the automatic design of particle swarm optimization algorithms: Supplementary material," Inst. Recherches Interdisciplinaires Develop. Intell. Artificielle, Univ. Libre Bruxelles, Bruxelles, Belgium, Rep. TR/IRIDIA/2021-002, 2021. [Online]. Available: http://iridia.ulb.ac.be/supp/IridiaSupp2021-001/

[5] M. López-Ibáñez and T. Stützle, "The automatic design of multi-objective ant colony optimization algorithms," *IEEE Trans. Evol. Comp.*, vol. 16, no. 6, pp. 861–875, Dec. 2012.

[6] T. Stützle and M. López-Ibáñez, "Automatic (offline) configuration of algorithms," in *Proc. GECCO*, 2015, pp. 681–702.

[7] D. Aydın, G. Yavuz, and T. Stützle, "ABC-X: A generalized, automatically configurable artificial bee colony framework," *Swarm Intell.*, vol. 11, no. 1, pp. 1–38, 2017.

[8] R. Poli, W. B. Langdon, and O. Holland, "Extending particle swarm optimisation via genetic programming," in *Proc. Eur. Conf. Genet. Program.*, 2005, pp. 291–300.

[9] P. B. Miranda and R. B. Prudêncio, "GEFPSO: A framework for PSO optimization based on grammatical evolution," in *Proc. GECCO*, 2015, pp. 1087–1094.

[10] R. Eberhart and J. Kennedy, "A new optimizer using particle swarm theory," in *Proc. 6th Int. Symp. Micro Mach. Human Sci.*, 1995, pp. 39–43.

[11] J. Kennedy and R. Mendes, "Population structure and particle swarm performance," in *Proc. CEC*, 2002, pp. 1671–1676.

[12] Y. Shi and R. Eberhart, "A modified particle swarm optimizer," in *Proc. ICEC*, 1998, pp. 69–73.

[13] M. López-Ibáñez, J. Dubois-Lacoste, L. P. Cáceres, M. Birattari, and T. Stötzle, "The irace package: Iterated racing for automatic algorithm configuration," *Oper. Res. Perspect.*, vol. 3, pp. 43–58, Dec. 2016.

[14] O. Maron and A. W. Moore, "The racing algorithm: Model selection for lazy learners," *Artif. Intell. Res.*, vol. 11, nos. 1–5, pp. 193–225, 1997.

[15] R. Poli, J. Kennedy, and T. Blackwell, "Particle swarm optimization," *Swarm Intell.*, vol. 1, no. 1, pp. 33–57, 2007.

[16] M. R. Bonyadi and Z. Michalewicz, "Particle swarm optimization for single objective continuous space problems: A review," *Evol. Comput.*, vol. 25, no. 1, pp. 1–54, Mar. 2017.

[17] K. R. Harrison, A. P. Engelbrecht, and B. M. Ombuki-Berman, "Inertia weight control strategies for particle swarm optimization," *Swarm Intell.*, vol. 10, no. 4, pp. 267–305, 2016.

[18] I. C. Trelea, "The particle swarm optimization algorithm: Convergence analysis and parameter selection," *Inf. Process. Lett.*, vol. 85, no. 6, pp. 317–325, 2003.

[19] M. Clerc and J. Kennedy, "The particle swarm-explosion, stability, and convergence in a multidimensional complex space," *IEEE Trans. Evol. Comput.*, vol. 6, no. 1, pp. 58–73, Feb. 2002.

[20] M. R. Bonyadi and Z. Michalewicz, "A locally convergent rotationally invariant particle swarm optimization algorithm," *Swarm Intell.*, vol. 8, no. 3, pp. 159–198, 2014.

[21] E. T. Oldewage, A. P. Engelbrecht, and C. W. Cleghorn, "Movement patterns of a particle swarm in high dimensional spaces," *Inf. Sci.*, vol. 512, pp. 1043–1062, Feb. 2020.

[22] J. García-Nieto and E. Alba, "Restart particle swarm optimization with velocity modulation: A scalability test," *Soft Comput.*, vol. 15, no. 11, pp. 2221–2232, 2011.

[23] E. van Zyl and A. P. Engelbrecht, "Group-based stochastic scaling for PSO velocities," in *Proc. CEC*, 2016, pp. 1862–1868.

[24] M. Clerc. (2011). *Standard Particle Swarm Optimisation From 2006 to 2011*. [Online]. Available: https://hal.archives-ouvertes.fr/hal-00764996/document
[25] J. Peña, "Theoretical and empirical study of particle swarms with additive stochasticity and different recombination operators," in *Proc. GECCO*, 2008, pp. 95–102.
[26] Z. Xinchao, "A perturbed particle swarm algorithm for numerical optimization," *Appl. Soft Comput.*, vol. 10, no. 1, pp. 119–124, 2010.
[27] P. K. Lehre and C. Witt, "Finite first hitting time versus stochastic convergence in particle swarm optimisation," in *Advances in Metaheuristics*. New York, NY, USA: Springer, 2013, pp. 1–20.
[28] F. van den Bergh and A. P. Engelbrecht, "A new locally convergent particle swarm optimiser," in *Proc. SMC*, 2002, p. 6.
[29] D. N. Wilke, S. Kok, and A. A. Groenwold, "Comparison of linear and classical velocity update rules in particle swarm optimization: Notes on scale and frame invariance," *Int. J. Numer. Methods Eng.*, vol. 70, no. 8, pp. 985–1008, 2007.
[30] M. R. Bonyadi, Z. Michalewicz, and X. Li, "An analysis of the velocity updating rule of the particle swarm optimization algorithm," *J. Heuristics*, vol. 20, no. 4, pp. 417–452, 2014.
[31] M. A. Montes de Oca, T. Stützle, M. Birattari, and M. Dorigo, "Frankenstein's PSO: A composite particle swarm optimization algorithm," *IEEE Trans. Evol. Comput.*, vol. 13, no. 5, pp. 1120–1132, Oct. 2009.
[32] J. Jordan, S. Helwig, and R. Wanka, "Social interaction in particle swarm optimization, the ranked fips, and adaptive multi-swarms," in *Proc. GECCO*, 2008, pp. 49–56.
[33] S.-T. Hsieh, T.-Y. Sun, C.-C. Liu, and S.-J. Tsai, "Efficient population utilization strategy for particle swarm optimizer," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 39, no. 2, pp. 444–456, Apr. 2009.
[34] M. A. Montes de Oca, T. Stützle, K. Van den Enden, and M. Dorigo, "Incremental social learning in particle swarms," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 41, no. 2, pp. 368–384, Apr. 2011.
[35] R. Mendes, J. Kennedy, and J. Neves, "The fully informed particle swarm: Simpler, maybe better," *IEEE Trans. Evol. Comput.*, vol. 8, no. 3, pp. 204–210, Jun. 2004.
[36] X. Li and X. Yao, "Cooperatively coevolving particle swarms for large scale optimization," *IEEE Trans. Evol. Comput.*, vol. 16, no. 2, pp. 210–224, Apr. 2012.
[37] T. J. Richer and T. Blackwell, "The Lévy particle swarm," in *Proc. CEC*, 2006, pp. 808–815.
[38] S. Janson and M. Middendorf, "A hierarchical particle swarm optimizer and its adaptive variant," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 35, no. 6, pp. 1272–1282, Dec. 2005.
[39] A. Ratnaweera, S. K. Halgamuge, and H. C. Watson, "Self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients," *IEEE Trans. Evol. Comput.*, vol. 8, no. 3, pp. 240–255, Jun. 2004.
[40] M. S. Arumugam, M. V. C. Rao, and A. W. Tan, "A novel and effective particle swarm optimization like algorithm with extrapolation technique," *Appl. Soft Comput.*, vol. 9, no. 1, pp. 308–320, 2009.
[41] M. Schmitt and R. Wanka, "Particles prefer walking along the axes: Experimental insights into the behavior of a particle swarm," in *Proc. GECCO*, 2013, pp. 17–18.
[42] R. Eberhart and Y. Shi, "Comparing inertia weights and constriction factors in particle swarm optimization," in *Proc. CEC*, Jul. 2000, pp. 84–88.
[43] P. N. Suganthan *et al.*, "Problem definitions and evaluation criteria for the CEC 2005 special session on real-parameter optimization," Nanyang Technol. Univ., Singapore, Kanpur Genet. Algorithms Lab., IIT Kanpur, Kanpur, India, Rep. #2005005, 2005.
[44] J. J. Liang, B.-Y. Qu, and P. N. Suganthan, "Problem definitions and evaluation criteria for the CEC 2014 special session and competition on single objective real-parameter numerical optimization," Comput. Intell. Lab., Zhengzhou Univ., Zhengzhou, China, Nanyang Technol. Univ., Singapore, Rep. 201311, 2013.
[45] M. Lozano, D. Molina, and F. Herrera, "Editorial scalability of evolutionary algorithms and other metaheuristics for large-scale continuous optimization problems," *Soft Comput.*, vol. 15, pp. 2085–2087, 2011.
[46] J. Kennedy, "Bare bones particle swarms," in *Proc. SIS*, 2003, pp. 80–87.
[47] J. Gimmler, T. Stützle, and T. E. Exner, "Hybrid particle swarm optimization: An examination of the influence of iterative improvement algorithms on performance," in *Proc. ANTS*, 2006, pp. 436–443.
[48] C. L. Camacho-Villalón, T. Stützle, and M. Dorigo, "Grey wolf, firefly and bat algorithms: Three widespread algorithms that do not contain any novelty," in *Proc. ANTS*, 2020, pp. 121–133.

Christian L. Camacho-Villalón received the B.S. degree in informatics and the M.S. degree in sciences and information technologies from Universidad Autónoma Metropolitana, Mexico City, Mexico, in 2013 and 2017, respectively.

He is an ASP Doctoral Fellow of the Belgian F.R.S.–FNRS with the IRIDIA Laboratory, Université Libre de Bruxelles, Brussels, Belgium. His research interests include artificial intelligence techniques, such as metaheuristics and machine learning, and their application to optimization and the automated design of algorithms.

Marco Dorigo (Fellow, IEEE) received the Ph.D. degree in electronic engineering from the Politecnico di Milano, Milan, Italy, in 1992, and the title of Agrégé de l'Enseignement Supérieur from the Université Libre de Bruxelles (ULB), Brussels, Belgium, in 1995.

He is a Research Director of the F.R.S.–FNRS, the Belgian National Funds for Scientific Research, and the Co-Director of IRIDIA, the AI Lab of the ULB. He is the inventor of the ant colony optimization metaheuristic. His research interests include swarm intelligence, swarm robotics, and metaheuristics for optimization.

Dr. Dorigo received the Marie Curie Excellence Award in 2003, the Dr. A. De Leeuw-Damry-Bourlart Award in applied sciences in 2005, the Cajastur "Mamdani" Prize for Soft Computing in 2007, the ERC Advanced Grant in 2010, the IEEE Frank Rosenblatt Award in 2015, and the IEEE Evolutionary Computation Pioneer Award in 2016. He is the Editor-in-Chief of *Swarm Intelligence*. He is a Fellow of AAAI and ECCAI.

Thomas Stützle (Fellow, IEEE) received the Ph.D. degree in computer science from Technische Universität Darmstadt, Darmstadt, Germany, in 1998.

He is a Research Director of the Belgian F.R.S.–FNRS with the IRIDIA Laboratory, Université Libre de Bruxelles, Brussels, Belgium. He has published extensively over 250 peer-reviewed articles in journals, conference proceedings, or edited books in the area of metaheuristics. His research interests include stochastic local search (SLS) algorithms, large-scale experimental studies, SLS algorithm engineering, and automated design of algorithms.