Fuzzy Self-Tuning PSO: A Settings-Free Algorithm for Global Optimization

Marco S. Nobile[a,c,*], Paolo Cazzaniga[b,c], Daniela Besozzi[a,c], Riccardo Colombo[a,c], Giancarlo Mauri[a,c], Gabriella Pasi[a]

[a] *Department of Informatics, Systems and Communication, University of Milano-Bicocca, Milano, Italy*
[b] *Department of Human and Social Sciences, University of Bergamo, Bergamo, Italy*
[c] *SYSBIO.IT Centre of Systems Biology, Milano, Italy*

Abstract

Among the existing global optimization algorithms, Particle Swarm Optimization (PSO) is one of the most effective methods for non-linear and complex high-dimensional problems. Since PSO performance strongly depends on the choice of its settings (i.e., inertia, cognitive and social factors, minimum and maximum velocity), Fuzzy Logic (FL) was previously exploited to select these values. So far, FL-based implementations of PSO aimed at the calculation of a unique settings for the whole swarm. In this work we propose a novel self-tuning algorithm—called Fuzzy Self-Tuning PSO (FST-PSO)—which exploits FL to calculate the inertia, cognitive and social factor, minimum and maximum velocity independently for each particle, thus realizing a complete settings-free version of PSO. The novelty and strength of FST-PSO lie in the fact that it does not require any expertise in PSO functioning, since the behavior of every particle is automatically and dynamically adjusted during the optimization. We compare the performance of FST-PSO with standard PSO, Proactive Particles in Swarm Optimization, Artificial Bee Colony, Covariance Matrix Adaptation Evolution Strategy, Differential Evolution and Genetic Algorithms. We empirically show that FST-PSO can basically outperform all tested algorithms with respect to the convergence speed and is competitive concerning the best solutions found, noticeably with a reduced computational effort.

Keywords: Particle Swarm Optimization, adaptive algorithms, Fuzzy

*Corresponding author. email: nobile@disco.unimib.it

Logic, settings-free optimization, self-tuning algorithms

1. Introduction

Particle Swarm Optimization (PSO) is a population-based global optimization meta-heuristics, inspired by the collective movement of birds flocks and fish schools [1]. In PSO, a swarm of N individuals, called particles, moves inside a bounded search space and cooperates to identify the best solution for a given problem with respect to a given fitness function. Two components are considered in PSO to the aim of exploiting and controlling the cooperation within the swarm: (i) the social attraction, which favors the collaboration among particles; (ii) the cognitive attraction, which prompts a particle to rely on its individual experience. The former component drives the particle towards the (current) best particle in the swarm, or towards the best particle in a specified neighborhood; the latter component guides each particle towards the best position it autonomously found so far. Both components influence the velocity and the exploratory capabilities of particles within the search space. Two parameters are used to balance the influence of these attractors: the so-called social (c_{soc}) and cognitive (c_{cog}) factors. Moreover, to obtain an effective exploration of the search space, an inertia factor w is used to weigh the movement of particles, and the magnitude of their velocities is clamped to a given threshold $v_{\max}$. A vector $\mathbf{v}_{\max} = (v_{\max_1}, \ldots, v_{\max_M})$ can also be defined when possibly different velocity values are assigned along the M dimensions of the search space.

As in the case of many other optimization algorithms, the performance of PSO strongly depends on the proper settings of the aforementioned factors $(N, c_{\text{soc}}, c_{\text{cog}}, w, \mathbf{v}_{\max})$. Unfortunately, an analytic determination of the best settings is generally impossible, being strongly problem-dependent: only a thorough knowledge of the shape and roughness of the fitness landscape—the hyper-surface characterizing all local and global optima—might help in properly choosing the values for PSO settings [2]. In general, the identification of the best settings is complex, lengthy and time consuming, so that a big deal of research is devoted to the definition of self-tuning and adaptive versions of evolutionary and swarm intelligence algorithms, including PSO. For instance, TRIBES is a settings-free version of PSO that automatically changes at run-time the particles' behavior as well as the topology of the swarm [3]. The Parameter-Less Evolutionary Search—based on Genetic Algorithms—dynamically determines the settings by exploiting some statistical properties

of the population [4]. The plague technique applied to Genetic Programming [5] is a strategy consisting in automatically adjusting the number of individuals of the population according to the fitness variation. APSO-MAM is a PSO variant where, at each iteration, the settings of a particle are dynamically varied. Hu *et al.* [6] shown that this method performs better than other algorithms that exploit automatic mechanisms to control parameter values; however, APSO-MAM is based on a subgradient procedure to improve the quality of the best particle that can be calculated only for differentiable functions. This procedure is suitable for benchmark functions—based on simple equations whose gradient can be analytically calculated—but hampers a full applicability of APSO-MAM in the case of real-world applications, in which the gradient of the fitness function does not have an analytic formulation.

Other approaches to dynamically select the PSO settings make use of Fuzzy Logic (FL) [7], to the aim of analyzing the contingent situation of the swarm. For instance, Shi and Eberhart [8] proposed for the first time a Fuzzy Rule-Based System (FRBS) to drive the configuration of PSO settings. In general, a FRBS exploits fuzzy sets and FL to represent different forms of knowledge about the system, as well as to model the relationships existing between its variables [9]. In the case of PSO considered in [8], the performance of the current best candidate solution and the current inertia weight are evaluated at each iteration, and exploited as inputs of the FRBS to calculate a new inertia weight for the whole swarm. Fuzzy Adaptive Turbulence in Particle Swarm Optimization was then introduced by Abraham and Liu [10] to deal with the issue of the premature convergence of particles. This PSO version exploits FL to adaptively tune an additional PSO settings value, i.e., the minimum velocity of particles, in order to reduce the crowding of the swarm around the global best. Note that a vector $\mathbf{v}_{\min}$ could be defined, analogously to the case of the maximum velocity vector $\mathbf{v}_{\max}$, to set different minimum velocity values along each dimension of the search space. However, Fuzzy Adaptive Turbulence in PSO considers only a common minimum velocity value for the whole swarm. A Fuzzy Particle Swarm Optimization algorithm (FPSO) was instead introduced by Tian and Li [11] to adjust both the inertia weight and the learning coefficient (a parameter introduced on purpose to modulate the velocity of particles). In FPSO, the FRBS exploits two input variables: the improvement of the global best and the deviation of particles' fitness.

Castillo and Melin [12] reported on works describing different evolutionary algorithms improved by means of FL, while a survey on different meth-

ods used to tune PSO settings was presented by Razaee Jordehi and Jasni [13]. All these papers show that FL represents an advantageous approach to develop self-tuning strategies for global optimization algorithms. Anyway, previous works in the context of PSO considered only a subset of its overall settings.

Traditional PSO versions consider w, c_{soc}, c_{cog}, $\mathbf{v}_{\min}$ and $\mathbf{v}_{\max}$ as *global* settings, i.e., the velocity and position of *all* particles in the swarm are updated according to the same values. On the contrary, in the algorithm we present in this work *each* particle has its own settings, determined by means of a FRBS; therefore, the simple *reactive* individuals of classic PSO become *proactive* optimizing agents. The FRBS calculates the individual settings of particles by computing two functions, the distance from the global best and a normalized fitness incremental factor, as it was initially proposed in Proactive Particles in Swarm Optimization (PPSO) [14]. Here we introduce Fuzzy Self-Tuning Particle Swarm Optimization (FST-PSO), which guarantees user independence via a fully automatic FL-based methodology that calculates at run-time the settings of $w, c_{\text{soc}}, c_{\text{cog}}, \mathbf{v}_{\min}$ and $\mathbf{v}_{\max}$, independently for each particle in the swarm. In Section 3 we extensively discuss the features of FST-PSO with respect to PPSO.

The performances of FST-PSO are compared against standard PSO [1], PPSO [14] and state-of-the-art competitors like Artificial Bee Colony (ABC) [15], Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [16], Differential Evolution (DE) [17], and Genetic Algorithms (GA) [18]. All these algorithms are population-based optimization procedures, where the quality of the individuals is assessed by means of a proper fitness function.

According to the so-called *no free lunch theorem* [19], there exists no algorithm able to outperform *all* the other algorithms, on average, given *any* possible optimization problem. In this paper we empirically show that FST-PSO can basically outperform all tested competitors with respect to the convergence speed, and it is also competitive concerning the best solutions found. We also highlight that FST-PSO requires less computational effort with respect to the state-of-the-art competitors. Considering the aforementioned theorem and the results shown in this work, a remarkable advantage of FST-PSO is that it represents a completely settings-free "black-box" self-tuning algorithm, usable without any expertise in global optimization methods, and characterized by an excellent trade-off between exploratory capabilities and computational requirements.

The paper is structured as follows. In Section 2 we describe the standard

PSO algorithm, while FST-PSO is presented in Section 3. We show the results of our analyses in Section 4, empirically proving the better performance of FST-PSO for a set of multi-dimensional benchmark functions. Finally, in Section 5 we discuss some future developments of our methodology.

2. Particle Swarm Optimization

PSO is a population-based swarm intelligence meta-heuristics, especially suitable for optimization problems whose solutions can be encoded as real-valued vectors [1]. In PSO, a population (called swarm) of N candidate solutions (called particles) cooperates to identify the optimal solution by moving within a bounded M-dimensional search space, where M is the length of the real-valued vectors.

Each particle i, $i = 1, \ldots, N$, is characterized by two vectors in the search space: the position, $\mathbf{x}_i \in \mathbb{R}^M$, and the velocity, $\mathbf{v}_i \in \mathbb{R}^M$. Usually, the initial positions of particles are randomly selected with a uniform distribution over the search space, though different initialization strategies can be exploited [20]. The position and the velocity of each particle are updated, at each iteration of the optimization phase, according to two attractors: the best position found so far by the particle itself ($\mathbf{b}_i \in \mathbb{R}^M$), and the best position identified so far by the whole swarm ($\mathbf{g} \in \mathbb{R}^M$). These attractors are balanced by two non-negative PSO-specific settings, the so-called cognitive factor ($c_{\text{cog}} \in \mathbb{R}_+$) and social factor ($c_{\text{soc}} \in \mathbb{R}_+$).

Since a deterministic-driven movement of particles could funnel the particles into local optima, each attractor is multiplied by a vector of random numbers, sampled from the uniform distribution in $[0, 1)$. In addition, the update of the velocity is modulated by an inertia weight, $w \in \mathbb{R}_+$, in order to avoid chaotic movements of the swarm. Formally, for each $i = 1, \ldots, N$, the velocity of particle i at iteration t is given by:

$$\begin{aligned} \mathbf{v}_i(t) = w \cdot \mathbf{v}_i(t-1) + \\ + c_{\text{soc}} \cdot \mathbf{R}_1 \circ (\mathbf{x}_i(t-1) - \mathbf{g}(t-1)) + \\ + c_{\text{cog}} \cdot \mathbf{R}_2 \circ (\mathbf{x}_i(t-1) - \mathbf{b}_i(t-1)) , \end{aligned} \tag{1}$$

where $\circ$ denotes the component-wise multiplication operator between vectors, and $\mathbf{R}_1$ and $\mathbf{R}_2$ are two vectors of random numbers associated with the social and cognitive factor, respectively [21]. We highlight here that, in standard PSO, the values of $w, c_{\text{soc}}, c_{\text{cog}}$ are iteration-independent and are valid for all

particles in the swarm. Accordingly, the position of particle i at iteration t is updated by calculating:

$$\mathbf{x}_i(t) = \mathbf{x}_i(t-1) + \mathbf{v}_i(t). \tag{2}$$

The quality of each particle in the swarm—i.e., of each candidate solution for the given optimization problem—is evaluated by exploiting an adequate fitness function (hereby denoted by f). The fitness function drives the movement of all particles within the search space, since it is used to evaluate the values of the local and global attractors, $\mathbf{b}_i$ and $\mathbf{g}$, iteration by iteration. By calculating the fitness values of all particles over the set of feasible solutions, we obtain a hyper-surface called fitness landscape.

The methodology described so far might drive the particles outside the feasible space of solutions, or even towards the infinity. To avoid this drawback, in PSO the search space is bounded (according to domain knowledge), and specific boundary conditions are applied to all particles that reach the fixed limits. In what follows, we denote by $\mathbf{b}_{\min} \in \mathbb{R}^M$ and $\mathbf{b}_{\max} \in \mathbb{R}^M$ the vectors indicating the values of the minimum and maximum boundaries of the search space, respectively. If all components of these vectors have the same value—namely, all components in vector $\mathbf{b}_{\min}$ are equal to $b_{\min} \in \mathbb{R}$ and all components in vector $\mathbf{b}_{\max}$ are equal to $b_{\max} \in \mathbb{R}$—then we simply denote by $[b_{\min}, b_{\max}]^M$ the whole search space. We underline here that the boundaries of the search space are generally problem-dependent and cannot be determined with an automatic strategy. As boundary condition, we use the damping strategy [22]: each particle that goes outside the search space in any dimension is relocated at the boundary of the solution space in that dimension; moreover, the velocity component in that dimension is changed in the opposite direction and it is multiplied by a random factor chosen with uniform distribution in $[0, 1)$.

Another issue in PSO is that the velocity of particles might diverge during the optimization process. To avoid this problem, the velocity is usually clamped to a maximum value, $v_{\max_m} \in \mathbb{R}_+$, along each m-th dimension of the search space, with $m = 1, \ldots, M$ [21]. On the contrary, in standard PSO the particles are not characterized by a minimum velocity. However, to improve the exploratory capabilities of the swarm, the velocity of each particle can also be clamped to a minimum value, $v_{\min_m} \in \mathbb{R}_+$, along each dimension of the search space.

In general, the values for $N, c_{\text{soc}}, c_{\text{cog}}, w$, the vector of minimum velocity values $\mathbf{v}_{\min}$ and the vector of maximum velocity values $\mathbf{v}_{\max}$, are set by the

user. These PSO settings usually have a huge impact on the optimization performance [21], both in terms of convergence speed and quality of the best solution. To lower the possible negative effects of this choice, and provide unfamiliar PSO users with a settings-free global problem optimizer, in this work we propose a novel FL-based algorithm that automatically selects all these values.

3. Fuzzy Self-Tuning PSO

We describe here a fully-automated version of PSO, called Fuzzy Self-Tuning Particle Swarm Optimization (FST-PSO), where the values of PSO settings are dynamically controlled by means of FL. In particular, each particle is associated with its own values for the inertia, the social and cognitive factors, the maximum and minimum velocity. This work represents an alternative approach to previous FL-controlled versions of PSO [8, 10, 11], whose aim consisted in improving the performance of this optimization algorithm by carrying out the automatic determination of its *global* settings.

FST-PSO is an extended and improved version of the FL-based PSO that we previously proposed, named Proactive Particles in Swarm Optimization (PPSO) [14], whereby the velocity and position of every particle were updated according to individual particle settings—inertia, social and cognitive factors, each one associated with a fuzzy variable—independently from the values of the other particles in the swarm. In FST-PSO, also the maximum and minimum velocity of each particle are individually adjusted by means of FL, thanks to six additional rules, providing users with a completely settings-free version of PSO. To determine the swarm size, FST-PSO exploits the heuristic $N = \lfloor 10 + 2\sqrt{M} \rfloor$ which sets the number of particles N according to the number of dimensions M of the search space, as suggested by Hansen *et al.* [23] and also adopted in PPSO [14].

One of the main differences between FST-PSO and PPSO is that, in the latter, the velocity of all particles along each dimension of the search space was clamped to a common maximum value, proportional to a pre-defined value $\mathcal{U} \in (0, 1]$ (as default, $\mathcal{U} = 0.2$ was used in PPSO [14]). To be more precise, in PPSO [14] we considered

$$v_{\max_m} = \mathcal{U} \cdot (b_{\max_m} - b_{\min_m}) \,, \tag{3}$$

where $b_{\min_m}$ and $b_{\max_m}$ denote the boundaries of the m-th dimension of the search space. On the contrary, FST-PSO automatically determines this set-

ting independently for each particle, by replacing $\mathcal{U}$ with a linguistic value of a linguistic variable. Moreover, FST-PSO introduces an additional setting $\mathcal{L} \in (0, 1], \mathcal{L} < \mathcal{U}$, which is used to clamp also the minimum velocity of each particle along the m-th dimension of the search space to

$$v_{\min_m} = \mathcal{L} \cdot (b_{\max_m} - b_{\min_m}) \,. \tag{4}$$

FST-PSO automatically determines this setting independently for each particle, by replacing $\mathcal{L}$ with a linguistic value of a linguistic variable.

So doing, FST-PSO becomes a fully automated self-tuning algorithm, where each particle computes its current velocity according to its performance during the previous iteration of the optimization. In Section 4, a comparative evaluation of FST-PSO with respect to PPSO is reported, which shows the improved performance of the new algorithm. Moreover, additional comparative evaluations with respect to state-of-the-art optimization techniques are reported.

In what follows, we denote by $w_i(t), c_{\text{soc}_i}(t), c_{\text{cog}_i}(t), \mathcal{L}_i(t)$ and $\mathcal{U}_i(t)$ the inertia, social factor, cognitive factor, upper clamping value for maximum velocity and lower clamping value for minimum velocity of the i-th particle during iteration t, respectively. At each iteration, the current velocity of particle i is evaluated as:

$$\begin{aligned} \mathbf{v}_i(t) = w_i(t-1) \cdot \mathbf{v}_i(t-1) + \\ + c_{\text{soc}_i}(t-1) \cdot \mathbf{R}_1 \circ (\mathbf{x}_i(t-1) - \mathbf{g}(t-1)) + \\ + c_{\text{cog}_i}(t-1) \cdot \mathbf{R}_2 \circ (\mathbf{x}_i(t-1) - \mathbf{b}_i(t-1)) \,, \end{aligned} \tag{5}$$

where $\mathbf{R}_1$ and $\mathbf{R}_2$ are two vectors of random numbers associated with the social and cognitive factors, respectively; $\mathbf{x}_i$ and $\mathbf{b}_i$ are the current position and the best position found so far by particle i, respectively, and $\mathbf{g}$ is the global best position found so far by the swarm. We highlight that, differently from PSO, in FST-PSO the values of inertia, social and cognitive factors are iteration-dependent and they possibly assume a different value for each particle in the swarm.

To dynamically determine the values of $w_i(t)$, $c_{\text{soc}_i}(t)$, $c_{\text{cog}_i}(t)$, $\mathcal{L}_i(t)$ and $\mathcal{U}_i(t)$ in an automatic way, independently for each particle at each iteration t, we make use of a FRBS consisting of 15 fuzzy rules, reported in Table 1. These rules are based on two main concepts: the distance of the particle

Table 1: Fuzzy rules used by FST-PSO

Rule no.	*Rule definition*
1	if (ϕ is Worse or δ is *Same*) then (*Inertia* is Low)
2	if (ϕ is *Same* or δ is *Near*) then (*Inertia* is *Medium*)
3	if (ϕ is *Better* or δ is *Far*) then (*Inertia* is *High*)
4	if (ϕ is *Better* or δ is *Near*) then (*Social* is *Low*)
5	if (ϕ is *Same* or δ is *Same*) then (*Social* is *Medium*)
6	if (ϕ is *Worse* or δ is *Far*) then (*Social* is *High*)
7	if (δ is *Far*) then (*Cognitive* is *Low*)
8	if (ϕ is *Worse* or ϕ is *Same* or δ is *Same* or δ is *Near*) then (*Cognitive* is *Medium*)
9	if (ϕ is *Better*) then (*Cognitive* is *High*)
10	if (ϕ is *Same* or ϕ is *Better* or δ is *Far*) then ($\mathcal{L}$ is *Low*)
11	if (δ is *Same* or δ is *Near*) then ($\mathcal{L}$ is *Medium*)
12	if (ϕ is *Worse*) then ($\mathcal{L}$ is *High*)
13	if (δ is *Same*) then ($\mathcal{U}$ is *Low*)
14	if (ϕ is *Same* or ϕ is *Better* or δ is *Near*) then ($\mathcal{U}$ is *Medium*)
15	if (ϕ is *Worse* or δ is *Far*) then ($\mathcal{U}$ is *High*)

from the global best $\mathbf{g}$, and a function measuring the fitness improvement of each particle with respect to the previous iteration.

The distance between two particles i and j, for some $i, j = 1, \ldots, N$, is a function $\delta : \mathbb{R}^M \times \mathbb{R}^M \to \mathbb{R}_+$, defined as:

$$\delta(\mathbf{x}_i(t), \mathbf{x}_j(t)) = ||\mathbf{x}_i(t) - \mathbf{x}_j(t)||_2 = \sqrt{\sum_{m=1}^{M} (x_{i,m}(t) - x_{j,m}(t))^2}, \tag{6}$$

where $x_{i,m}, x_{j,m}$ denote the m-th components of the position vectors $\mathbf{x}_i, \mathbf{x}_j$ of particles i and j, respectively.

The normalized fitness incremental factor is a function $\phi : \mathbb{R}^M \times \mathbb{R}^M \to [-1, 1]$, which considers the positions of particle i at the current and previous iterations; it is defined as follows:

$$\phi(\mathbf{x}_i(t), \mathbf{x}_i(t-1)) = \frac{\delta(\mathbf{x}_i(t), \mathbf{x}_i(t-1))}{\delta_{\max}} \cdot \tag{7}$$

$$\cdot \frac{\min\{f(\mathbf{x}_i(t)), f_\triangle\} - \min\{f(\mathbf{x}_i(t-1)), f_\triangle\}}{|f_\triangle|},$$

where $\delta_{\max}$ is the length of the diagonal of the hyper-rectangle corresponding to the search space, while $f_{\triangle}$ represents the (estimated) worst fitness value for the optimization problem under investigation. In this work we consider minimization problems: the sign of function ϕ must therefore be inverted in the case of maximization problems, in order to exploit the set of fuzzy rules listed in Table 1.

We stress the fact that, since the fitness landscape of the optimization problem is generally unknown, an accurate estimate of the worst fitness value is, intuitively, as difficult as solving the optimization problem itself. For this reason, we consider $f_{\triangle}$ equal to the highest fitness value calculated during the first iteration of FST-PSO, according to the initial position of all particles. Then, during the optimization phase, we use the min functions in Equation 7 to clamp any fitness value worse than $f_{\triangle}$. More precisely, the second factor in Equation 7 considers the possible improvement of the current fitness value of the i-th particle with respect to the previous iteration; the variation of the fitness function is then normalized in $[-1, 1]$ by dividing by $|f_{\triangle}|$. Note that a lower value of $\phi(\mathbf{x}_i(t), \mathbf{x}_i(t-1))$ in $[-1, 1]$ corresponds to a lower fitness value of particle i with respect to the previous iteration: this means that the particle is moving towards a new position $\mathbf{x}_i$ that represents a better solution for the optimization problem. The first factor in Equation 7, instead, is needed to weigh function ϕ according to the distance between the current and the previous position of the particle. This distance is normalized by dividing the distance value by $\delta_{\max}$, so that the first factor takes values in the interval $(0, 1)$.

We describe now the rationale behind the set of fuzzy rules listed in Table 1. In the antecedent of these rules, we use two linguistic variables, named *Distance from* $\mathbf{g}$ and *Normalized fitness incremental factor*, denoted by δ and ϕ, respectively. The definition of a linguistic variable for δ, expressing the distance of a particle from the global attractor $\mathbf{g}$, allows to avoid the use of arbitrary thresholds (i.e., classic sets with crisp boundaries) to characterize the proximity to the global best. Similarly, the definition of a linguistic variable for ϕ allows to avoid arbitrary thresholds to characterize the improvement of a particle with respect to the fitness value it assumed in the previous iteration. In the consequent of rules, the output variables are called *Inertia*, *Social*, *Cognitive*, $\mathcal{L}$ and $\mathcal{U}$, which correspond to the respective settings of particle i in FST-PSO. We want to stress the fact that, during each iteration of FST-PSO, all particles compute independently their own values for δ and ϕ, which are used to calculate the output variables according to

the rules reported in Table 1.

The universe of discourse of the variable *Distance from* $\mathbf{g}$ consists of the numeric values of the distance between the position vector $\mathbf{x}_i$ and the global best $\mathbf{g}$, according to Equation 6. Thus, the base variable of δ corresponds to the interval $[0, \delta_{\max}]$. The term set of this variable is composed by three linguistic values, *Same*, *Near* and *Far*. The membership functions of δ, shown in Figure 1, depend on three parameters, $\delta_1, \delta_2, \delta_3 \in [0, \delta_{\max}]$, used to properly characterize the concept of fuzzy distance between the particle position and the global best. The values of $\delta_1, \delta_2, \delta_3$ are set according to the size of the search space; following our domain expertise related to PSO, we use the following heuristic: $\delta_1 = 0.2 \cdot \delta_{\max}$, $\delta_2 = 0.4 \cdot \delta_{\max}$, $\delta_3 = 0.6 \cdot \delta_{\max}$. These are general-purpose multipliers, created to avoid any overfitting to the benchmark functions used in this study, and implementing a general and fuzzy concept of distance from the global best.

The universe of discourse of variable *Normalized fitness incremental factor* consists of the numerical values of function ϕ, according to Equation 7; the base variable of ϕ therefore corresponds to the interval $[-1, 1]$. The term set of this variable is composed by three linguistic values, *Better*, *Same* and *Worse*. The membership functions of ϕ, shown in Figure 2, are three simple triangular fuzzy sets, which correspond to a simplified version of the membership functions previously defined in PPSO [14, 24]. FST-PSO relies on triangular fuzzy sets, for both δ and ϕ, because of their good trade-off between expressiveness, simplicity and computational efficiency [25].

The output variables (*Inertia*, *Social*, *Cognitive*, $\mathcal{L}, \mathcal{U}$) can assume three different linguistic values, *Low*, *Medium* and *High*. Since our FRBS is based on the Sugeno inference method [26]—which allows to define rules having fuzzy inputs and crisp outputs—each linguistic value of the five output variables is modeled as a fuzzy singleton, as given in Table 2. Formally, given a set $\mathcal{R}$ of R rules having the same output variable in their consequent (e.g., rules 1, 2, 3 in Table 1 for *Inertia*), the Sugeno method calculates the final numerical value of this output variable as the weighted average of the output of each rule in $\mathcal{R}$:

$$output = \frac{\sum_{r=1}^{R} \rho_r z_r}{\sum_{r=1}^{R} \rho_r}, \tag{8}$$

where ρ_r denotes the membership degree of the input variable of the r-th rule, and z_r represents the output crisp value for the r-th rule, as given in Table 2.

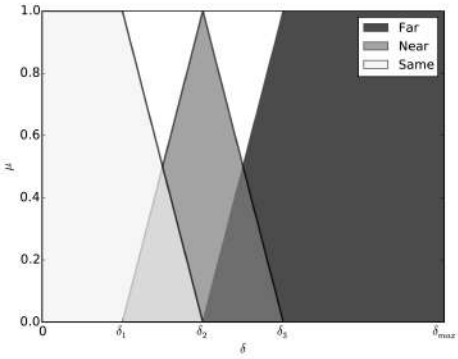

Figure 1: Membership functions of the linguistic values of the *Distance from* **g** variable. The membership function of *Same* is $trapezoid(\delta : 0, 0, \delta_1, \delta_2) = \{1, \text{ if } 0 \leq \delta < \delta_1; (\delta_2 - \delta)/(\delta_2 - \delta_1), \text{ if } \delta_1 \leq \delta < \delta_2; 0, \text{ if } \delta_2 \leq \delta \leq \delta_{\max}\}$. The membership function of *Near* is $triangle(\delta : \delta_1, \delta_2, \delta_3) = \{0, \text{ if } 0 \leq \delta < \delta_1; (\delta - \delta_1)/(\delta_2 - \delta_1), \text{ if } \delta_1 \leq \delta < \delta_2; (\delta_3 - \delta)/(\delta_3 - \delta_2), \text{ if } \delta_2 \leq \delta < \delta_3; 0, \text{ if } \delta_3 \leq \delta \leq \delta_{\max}\}$. The membership function of *Far* is $trapezoid(\delta : \delta_2, \delta_3, \delta_{\max}, \delta_{\max}) = \{0, \text{ if } 0 \leq \delta < \delta_2; (\delta - \delta_2)/(\delta_3 - \delta_2), \text{ if } \delta_2 \leq \delta < \delta_3; 1, \text{ if } \delta_3 \leq \delta \leq \delta_{\max}\}$.

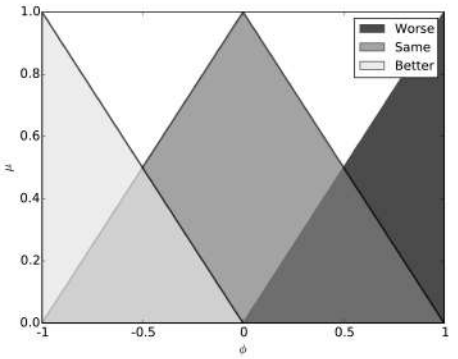

Figure 2: Membership function of the linguistic values of the *Normalized fitness incremental factor* variable. The membership function of *Better* is $triangle(\phi : -1, -1, 0) = \{1, \text{ if } \phi = -1; -\phi, \text{ if } -1 < \phi < 0; 0, \text{ if } 0 \leq \phi \leq 1\}$. The membership function of *Unvaried* is $triangle(\phi : -1, 0, 1) = 1 - |\phi|$. The membership function of *Worse* is $triangle(\phi : 0, 1, 1) = \{0, \text{ if } -1 \leq \phi < 0; \phi, \text{ if } 0 \leq \phi < 1; 1, \text{ if } \phi = 1\}$.

Figure 3 shows the resulting heatmaps of the five output variables, as computed by the Sugeno method, which are described in the following sections.

Table 2: Output variables and their defuzzification

Output variable	*Term*		
	Low	Medium	High
Inertia	0.3	0.5	1.0
Social	1.0	2.0	3.0
Cognitive	0.1	1.5	3.0
$\mathcal{L}$	0.0	0.001	0.01
$\mathcal{U}$	0.1	0.15	0.2

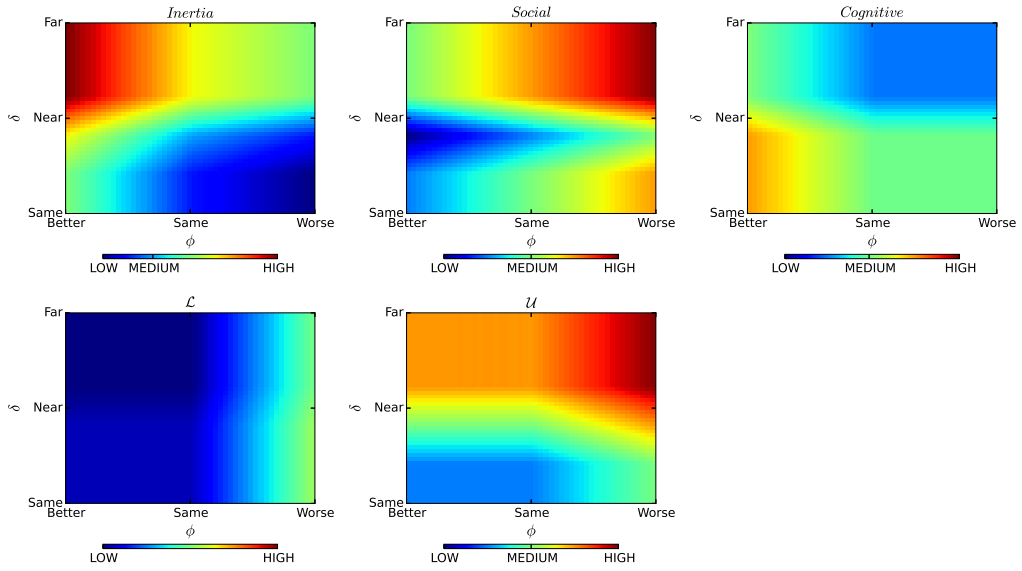

Figure 3: The heatmaps show the mapping between the fuzzy value of the input variables δ and ϕ and the defuzzified value of the five output variables of particle i: *Inertia* (*top left*), *Social* (*top middle*), *Cognitive* (*top right*), $\mathcal{L}$ (*bottom left*) and $\mathcal{U}$ (*bottom right*), according to Tables 1 and 2. Note that, in the plot of the output variable $\mathcal{L}$ (*bottom left*), the label "Medium" is not reported in the colorbar to avoid its overlap with the label "Low".

3.1. Inertia

The aim of the set of rules 1, 2 and 3, which adjust the value of the variable *Inertia* for every particle i, is to evaluate the contribution of the velocity value that particle i had in the previous iteration to update its velocity value at the current iteration. The rationale behind this is to increase the value of *Inertia*

when the particle is improving its fitness value with respect to the previous iteration, and to lower it otherwise. This is achieved by setting a *Low* value of *Inertia* when either ϕ is *Worse*, or the distance δ from the global best is *Same*. A *High* value is instead set when the particle is following the right direction (i.e., ϕ is *Better*), or it is far from the global best (i.e., δ is *Far*). Finally, *Inertia* is set to a "neutral choice" (i.e., the value *Medium*), when no relevant changes in the fitness value occur with respect to the previous iteration (i.e., ϕ is *Same*) or the distance is *Near*.

The heatmap for *Inertia* (Figure 3, top left) shows that the maximum value for this output variable is attained when δ is *Far* and ϕ is *Better*. The value of *Inertia* decreases when δ varies from *Far* to *Same* or ϕ varies from *Better* to *Worse*, reaching the lowest value when ϕ is *Worse* and δ is *Same*. In particular, we observe that in the case of δ values between *Near* and *Far*, there is a plateau in the values of *Inertia* when ϕ is *Better*.

3.2. Social attraction

Rules 4, 5 and 6 control the strength of the social information shared among particles. This is obtained by tuning the *Social* variable, which determines the attraction of particle i towards the global best position $\mathbf{g}$ in the swarm. If either the particle is finding better solutions for the optimization problem (i.e., ϕ is *Better*), or it is approaching the global best (i.e., δ is *Near*), then it is reasonable for it to "ignore" the information shared by the swarm. Therefore, *Social* is set to *Low* in these conditions. On the contrary, when the particle is not finding better solutions or it is not close to the global best, it should rather "follow the advice" of the other particles. Hence, we set *Social* to the *High* value. As in the case of *Inertia*, the FRBS assigns an intermediate value to *Social* if no relevant changes occur either in the fitness value or in the distance from $\mathbf{g}$ (i.e., ϕ and δ are *Same*).

The *Social* variable (Figure 3, top middle) assumes its highest value when δ is *Far* and ϕ is *Worse*. This heatmap can be described by considering different aspects. On the one hand, when the value of ϕ changes from *Worse* to *Better*, the value of *Social* linearly decreases, independently from the value of δ. On the other hand, when the value of δ changes from *Far* to *Same*, the heatmap of *Social* is characterized by a dip in correspondence of the value *Near* of δ. The modulation of *Social* is mainly driven by ϕ, and the lowest value of this variable is reached when δ is *Near* and ϕ lies between *Better* and *Same*.

3.3. Cognitive attraction

Rules 7, 8 and 9 are defined to calibrate the *Cognitive* variable, which weighs the attraction of particle i towards its personal best position $\mathbf{b}_i$. If the distance from the global best is *Far*, it is advisable that the particle limits the movement towards $\mathbf{b}_i$, therefore setting *Cognitive* to *Low*. On the contrary, if the particle is not improving its fitness value or it is not far from the global best (i.e., ϕ is *Same* or *Worse* and δ is *Same* or *Near*), then an intermediate value of *Cognitive* can weigh the particle tendency to move in the direction of its personal best position $\mathbf{b}_i$. Finally, if the particle is finding better solutions for the optimization problem (i.e., ϕ is *Better*), then the local exploration around its current position within the search space has to be encouraged by setting *Cognitive* to *High*.

The heatmap for *Cognitive* (Figure 3, top right) shows that the highest value for this factor is obtained when ϕ and δ assume the values *Better* and *Same*, respectively. As δ moves from *Same* to *Far* and ϕ moves from *Better* to *Worse*, the value of *Cognitive* decreases reaching the lowest values in the opposite configuration (i.e., when δ is *Far* and ϕ is *Worse*). Note that both input variables have a comparable impact on the *Cognitive* factor.

3.4. Lower and upper clamping values for velocity

Differently from PPSO, in FST-PSO the maximum and minimum velocities of each particle are clamped using Equations 3 and 4, respectively, in which the typical swarm-shared fixed settings of $\mathcal{U}$ and $\mathcal{L}$ are dynamically replaced by using particle-specific and iteration-dependent values. Rules 10, 11, 12 and 13, 14, 15 control the values of the output variables $\mathcal{L}$ and $\mathcal{U}$, respectively.

Concerning the $\mathcal{L}$ factor, it is advisable to have a *High* minimum velocity when the particle fitness ϕ is getting *Worse*. On the contrary, $\mathcal{L}$ is set to *Low* when ϕ is *Same* or *Better*, or when δ is *Far*, since in these situations the velocity of a particle could assume small values in order to perform a local exploration of the search space. Finally, when the particle position is close to the global best, the factor $\mathcal{L}$ is set to *Medium*. The heatmap of $\mathcal{L}$ factor is shown in Figure 3 (bottom left). We observe that this factor is mainly influenced by the value of ϕ. In particular, the highest value is reached when ϕ is *Worse*, then $\mathcal{L}$ decreases as ϕ reaches the *Same* value and, finally, there is a large plateau when ϕ is between *Same* and *Better*, irrespective of the δ value.

Regarding the $\mathcal{U}$ factor, the maximum velocity allowed is set to *High* in the case of a particle whose distance is far from the global best or when its fitness value is getting worse. On the contrary, this factor is set to *Low* only when the distance from the global optimum is *Same*. $\mathcal{U}$ is set to *Medium* when the fitness of the particle is improving or when the distance from the global best is *Near*. The heatmap of $\mathcal{U}$ factor is shown in Figure 3 (bottom right). In this case both ϕ and δ influence the value of this factor; in particular, there is a plateau characterized by medium values of $\mathcal{U}$ when ϕ is between *Better* and *Same* and δ is between *Near* and *Far*.

In general, by increasing the values of ϕ and δ, $\mathcal{U}$ increases and reaches its highest value in the case of ϕ equal to *Worse* and δ equal to *Far*.

4. Results

We investigated the performance of FST-PSO by testing the algorithm on 12 reference benchmark functions, listed in Table 3, which are all parametric in the number of dimensions M and have $\mathbb{R}^M$ as domain of definition. The graphical representation given in Figure 4, for $M = 2$, shows that these functions are emblematic examples of non-linear, multi-modal and rugged problems which can hardly be solved by means of classic optimization algorithms (e.g., gradient descent [27]).

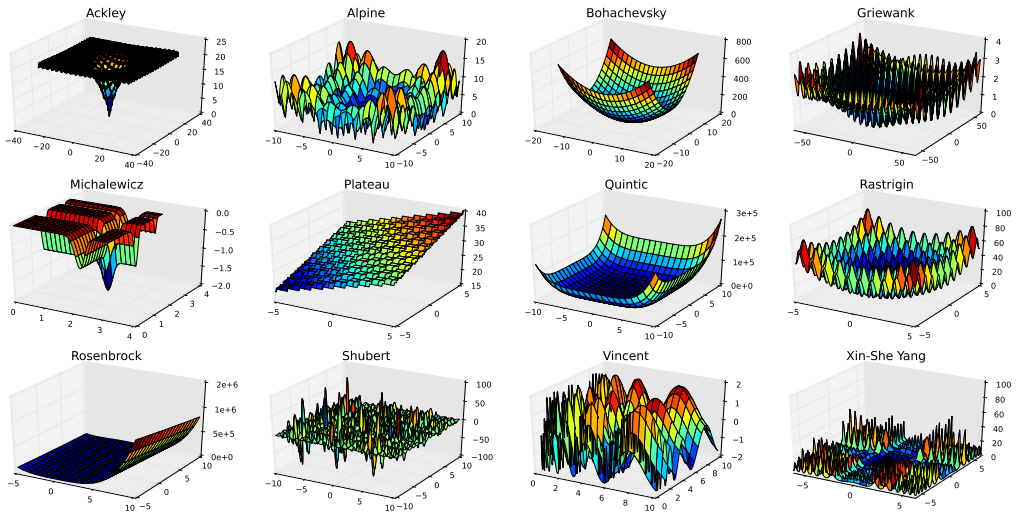

Figure 4: Plots of the benchmark functions (with $M = 2$) listed in Table 3.

Table 3: Benchmark functions

Function	*Equation*	*Search space*	*Value in global minimum**
Ackley	$f_{\text{Ack}}(\mathbf{x}) = 20 + e - 20\exp(-.2\sqrt{\frac{1}{M}\sum_{m=1}^{M} x_m^2}) - \exp(\frac{1}{M}\sum_{m=1}^{M}\cos(2\pi x_m))$	$[-30, 30]^M$	$f_{\text{Ack}}(\mathbf{0}) = 0$
Alpine	$f_{\text{Alp}}(\mathbf{x}) = \sum_{m=1}^{M} \lvert x_m \sin(x_m) + .1x_m \rvert$	$[-10, 10]^M$	$f_{\text{Alp}}(\mathbf{0}) = 0$
Bohachevsky	$f_{\text{Boh}}(\mathbf{x}) = \sum_{m=1}^{M-1}(x_m^2 + 2x_{m+1}^2 - .3\cos(3\pi x_m) - .4\cos(4\pi x_{m+1}) + .7)$	$[-15, 15]^M$	$f_{\text{Boh}}(\mathbf{0}) = 0$
Griewank	$f_{\text{Gri}}(\mathbf{x}) = \frac{1}{4000}\sum_{m=1}^{M} x_m^2 - \prod_{m=1}^{M}\cos(\frac{x_m}{\sqrt{m}}) + 1$	$[-600, 600]^M$	$f_{\text{Gri}}(\mathbf{0}) = 0$
Michalewicz	$f_{\text{Mic}}(\mathbf{x}) = -\sum_{m=1}^{M}\sin(x_m)\sin^{2k}(\frac{mx_m^2}{\pi})$, with $k = 10$ in this work	$[0, \pi]^M$	$f_{\text{Mic}}(\mathbf{0}) = -1.8013$
Plateau	$f_{\text{Pla}}(\mathbf{x}) = 30 + \sum_{m=1}^{M}\lfloor x_m \rfloor$	$[-5.12, 5.12]^M$	$f_{\text{Pla}}(\mathbf{-5.12}) = -6M + 30$
Quintic	$f_{\text{Qui}}(\mathbf{x}) = \sum_{m=1}^{M} \lvert x_m^5 - 3x_m^4 + 4x_m^3 + 2x_m^2 - 10x_m - 4 \rvert$	$[-10, 10]^M$	$f_{\text{Qui}}(\mathbf{-1}) = 0$
Rastrigin	$f_{\text{Ras}}(\mathbf{x}) = 10M + \sum_{m=1}^{M}(x_m^2 - 10\cos(2\pi x_m))$	$[-5.12, 5.12]^M$	$f_{\text{Ras}}(\mathbf{0}) = 0$
Rosenbrock	$f_{\text{Ros}}(\mathbf{x}) = \sum_{m=1}^{M-1}[100(x_m^2 - x_{m+1})^2 + (x_m - 1)^2]$	$[-5, 10]^M$	$f_{\text{Ros}}(\mathbf{1}) = 0$
Shubert	$f_{Shu}(\mathbf{x}) = \prod_{m=1}^{M}(\sum_{i=1}^{5} i\cos[(i+1)x_m + i])$	$[-10, 10]^M$	Many global minima
Vincent	$f_{\text{Vin}}(\mathbf{x}) = \sum_{m=1}^{M}\sin(10\log(x_m))$	$[0.25, 10]^M$	$f_{\text{Vin}}(\mathbf{7.706281}) = -M$
Xin-She Yang	$f_{\text{Xin}}(\mathbf{x}) = \sum_{m=1}^{M} \lvert x_m \rvert [\exp(\sum_{m=1}^{M}\sin(x_m^2)]^{-1}$	$[-2\pi, 2\pi]^M$	$f_{\text{Xin}}(\mathbf{0}) = 0$

*Boldface numbers refer to vectors whose M components have the same value

FST-PSO was compared against PSO, PPSO, ABC, CMA-ES, DE and GA, which are all population-based optimization methods where candidate solutions are encoded in a specific mode, according to each algorithm. The functioning of each algorithm, along with a brief description of the settings used for the tests, is summarized in the next section. Independently from the mathematical representation of the population individuals and the algorithm itself, in each optimization method the quality of each candidate solution is evaluated by means of the fitness function. In order to make a quantitative comparison between FST-PSO and each competitor algorithm, we exploited the *Average Best Fitness* (ABF), i.e., the mean of the fitness values of the best solution found at each iteration t, evaluated over a number Θ of runs. In all tests that follow, we set $\Theta = 30$.

Formally, denoting by $\mathcal{B}_\theta(t)$ the best candidate solution found at iteration t during the θ-th run, it holds

$$\text{ABF} = \frac{1}{\Theta}\sum_{\theta=1}^{\Theta} f(\mathcal{B}_\theta(t)). \tag{9}$$

To perform a fair comparison, for each algorithm the number of iterations

was fixed to $t_{\texttt{MAX}} = 400$ and the population size N was chosen by using the same heuristic defined for FST-PSO, so that the same number of fitness evaluations was performed for all methods.

We implemented FST-PSO using the Python language—exploiting the pyfuzzy package (website: http://pyfuzzy.sourceforge.net) as fuzzy engine, extended to fully support Sugeno inference—and relying on NumPy (website: http://www.numpy.org) (see [28] for further information). The source code of FST-PSO can be installed as PyPI package (`pip install fst-pso`).

PSO, ABC, CMA–ES, DE and GA were implemented using the PyGMO Python package [29]. Except for N, throughout all tests we relied on the default settings of PyGMO, in order to perform an "out-of-the-box" optimization.

4.1. Competitor algorithms

PSO and PPSO. The basics of PSO and PPSO were introduced in Sections 2 and 3. We used the following settings for PSO:

- inertia w linearly decrementing from 0.9 to 0.4;
- cognitive factor $c_{\text{cog}} = 2.05$;
- social factor $c_{\text{soc}} = 2.05$.

In the case of PSO, the $v_{\max_m}$ values were chosen using the same heuristic used for PPSO, as described in Section 3. For both PSO and PPSO, the $v_{\min_m}$ values were set equal to 0, that is, no minimum velocity was forced. Similarly to FST-PSO, damping boundary conditions were used for PSO and PPSO.

ABC. Artificial Bee Colony [15] is a swarm intelligence population-based optimization algorithm inspired by the foraging behavior of bees. In ABC, the population is composed of three different kinds of individuals: scout, employed and onlooker bees. Scout bees perform the exploration process, as they are randomly placed within the search space. Employed and onlookers carry out the exploitation process by means of a local search nearby the position identified by the scout. More precisely, scout bees randomly determine a new food source (i.e., a position in the search space) and, at this stage, they become employed. Free onlookers bees are then assigned to the position of employed food sources, proportionally to the fitness values of these positions.

After a given number of trials (in PyGMO this number is equal to 20), if onlookers did not improve their positions they go back to the hive and become scout again, so that they can be randomly placed in a different position.

CMA-ES. Covariance Matrix Adaptation Evolution Strategy [16] is generally considered one of the most effective evolutionary algorithms for single-objective real-valued optimization [30], and it is often regarded as the state-of-the-art for stochastic continuous optimizers. The basic ES algorithm exploits a mutation operator to explore the search space. In particular, the mutation creates novel individuals during each generation by perturbing the best individual, or by perturbing a novel individual created using a weighted average of the population. The perturbation is performed using multivariate normally distributed random deviates, having mean 0 and standard deviation σ (which is called the step-size). The Covariance Matrix Adaptation variant of ES is able to dynamically adapt such distribution to perform more effective mutation steps. In particular, CMA-ES relies on a $M \times M$ symmetric positive covariance matrix, which is used to determine pairwise dependencies between parameters of the problem under investigation. In other words, CMA-ES introduces a modified mutation operator, based on an M-dimensional distribution ellipsoid whose size and rotation are updated iteration by iteration, according to the optimization performances.

CMA-ES is meant to be a settings-free algorithm, although the population size N and the initial step-size can be selected by the user. In our tests, we used PyGMO's default setting $\sigma = 0.5$. For further reference on CMA-ES parameters we refer the reader to [31] and references therein.

DE. Differential Evolution [17] is a population-based algorithm where, at each generation, the individuals of the population evolve by means of two operators: mutation and crossover. In particular, three distinct candidate solutions are randomly selected and mixed using a weighted function (i.e., mutation). Then, the outcome is recombined by means of crossover with a fourth randomly chosen individual, to produce a new offspring called trial vector. This vector replaces the recombined individual in the next generation if it is characterized by an improved fitness value. The variant of DE used in the following tests makes use of random selection of mutation vectors (with self-adaptive weighing factor), one difference vector for trial vector generation, and exponential crossover with self-adaptive probability (this variant is called $rand/1/exp$, using the standard nomenclature [32]).

GA. Genetic Algorithms [18] are a population-based optimization strategy that mimics Darwinian processes like natural selection. In GA, a population of randomly generated individuals undergoes a selection mechanism and is modified by genetic operators (i.e., crossover and mutation). The selection strategy is used to choose the individuals according to their fitness values; these individuals will generate possibly improved offspring thanks to the application of the crossover operator. In order to better explore the search space, a random mutation is also performed on the offspring. The crossover and mutation operators are applied according to a specified probability. A relevant improvement to standard GA consists in the application of elitism, where the individual characterized by the best fitness value in the current generation is directly promoted to belong to the population at the next generation. PyGMO exploits the elitism operator, while the other individuals are selected using a roulette wheel strategy (i.e., selection probability is proportional to the fitness of the individual); the (single point) crossover operator exchanges parts of parents individuals and is applied with probability 0.95; mutation operator perturbs the individual exploiting a Gaussian distribution (with standard deviation equal to 0.1) and is applied with probability 0.02.

4.2. Optimization performance

To evaluate the optimization performance of each algorithm, in all executed tests we assumed $M = 100$ for every benchmark function listed in Table 3. In Figure 5 we show the comparison of the ABF of FST-PSO (black solid lines) against standard PSO (blue dashed lines), PPSO (gray dotted lines), ABC (red solid lines), CMA-ES (light blue solid lines), DE (green solid lines) and GA (orange solid lines). The standard deviation of each test is also pictured as a transparent filled area around the corresponding ABF curve, having the same color semantics. ABF and standard deviation values were measured during the $\Theta = 30$ runs and $t_{\texttt{MAX}} = 400$ iterations of each algorithm.

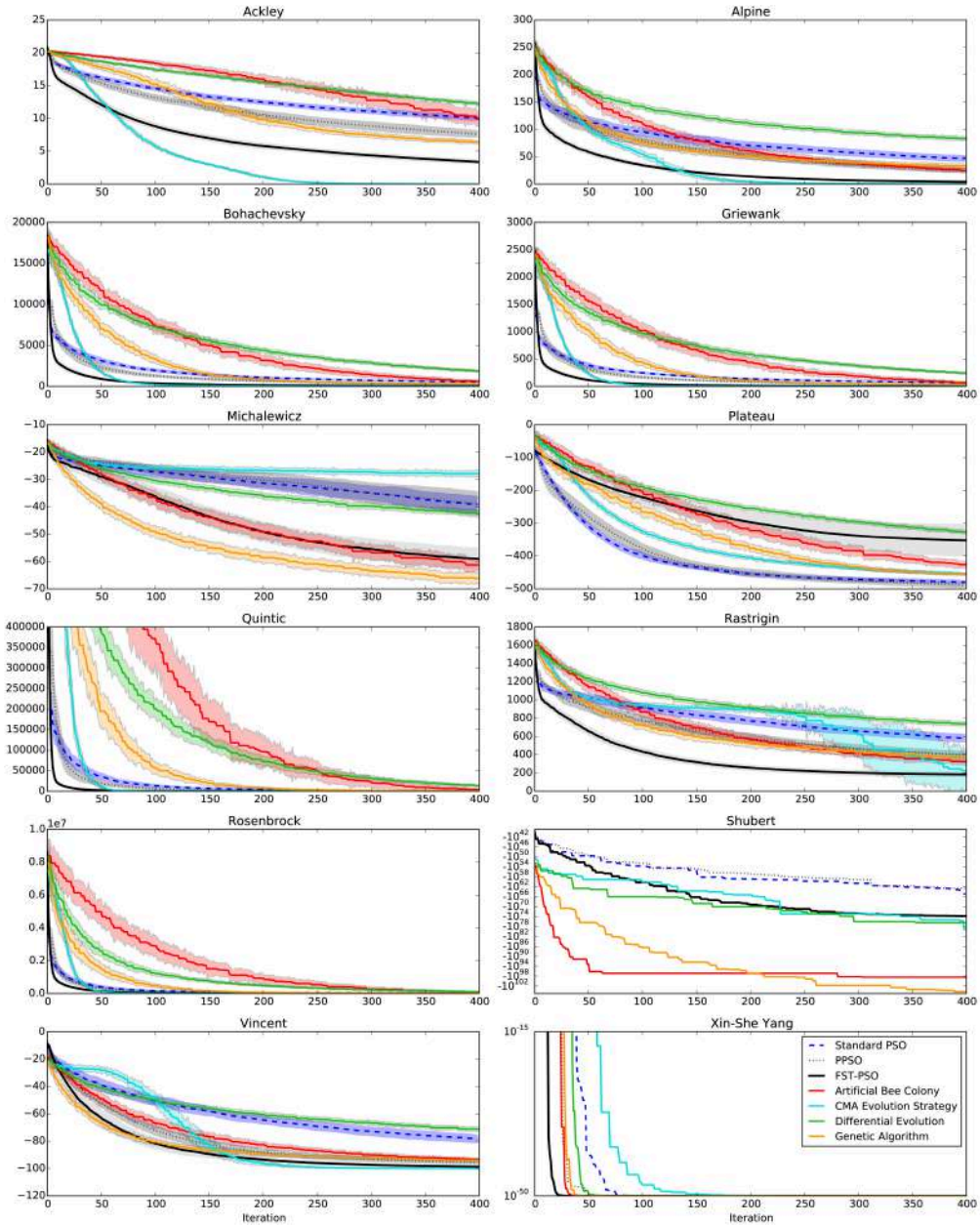

Figure 5: Comparison of the optimization performances of FST-PSO (black solid line), PPSO (gray dotted line), PSO (blue dashed line), ABC (red solid line), CMA-ES (light blue solid line), DE (green solid line) and GA (orange solid line), over the benchmark functions listed in Table 3. Lines and filled areas represent the ABF and the corresponding standard deviation, measured during 30 runs and 400 iterations of each algorithm (due to the logarithmic scale on the y axis, standard deviation is not represented for the Shubert and Xin-She Yang functions for the sake of readability). These results show that FST-PSO often achieves the best performance in terms of convergence speed and is competitive concerning the best solution found.

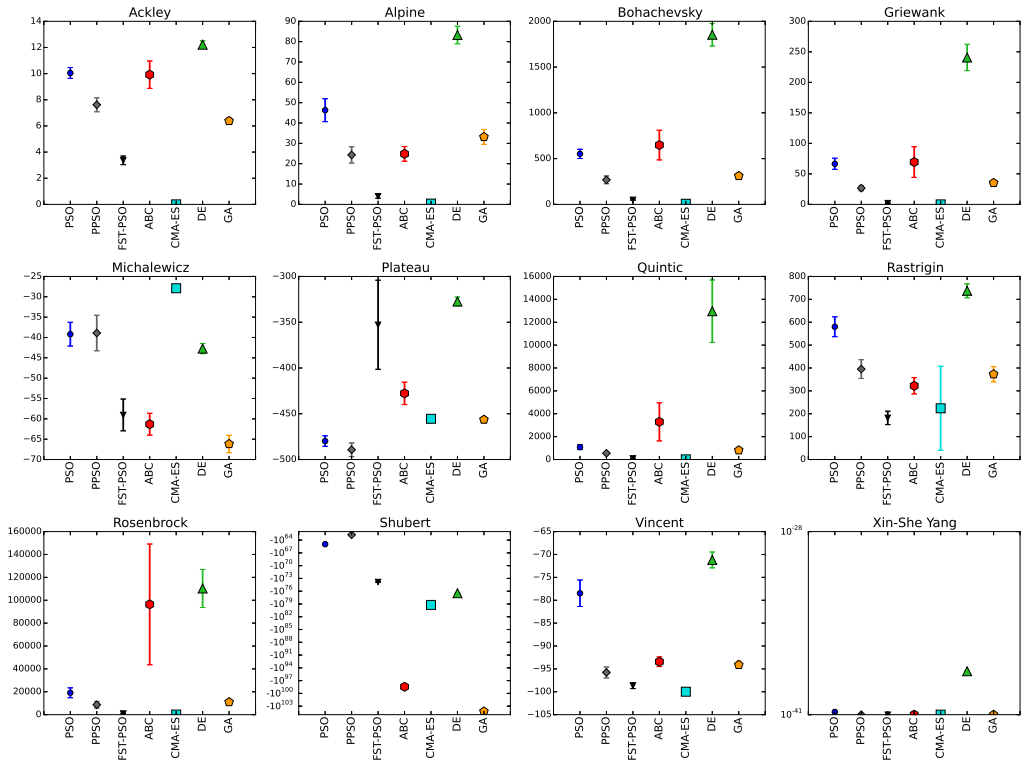

Figure 6: Comparison of the ABF and the standard deviations evaluated after $t_{\texttt{MAX}} = 400$ iterations of each optimization algorithm. FST-PSO (black triangles) ranks first in the case of Rastrigin and Xin-She Yang functions; it is close to CMA-ES in the case of Bohachevsky, Griewank, Quintic and Rosenbrock functions; it ranks second in the case of Ackley, Alpine and Vincent functions. In the case of the Plateau function, its performance is impaired because of the peculiar shape of the fitness landscape.

Our results show that, in the case of Bohachevsky, Griewank, Quintic, Rastrigin, Rosenbrock and Xin-She Yang functions, FST-PSO achieves the best performances in terms of convergence speed (Figure 5). To prove in a quantitative way the better performances of FST-PSO, from the point of view of the average convergence speed, we report in Table 4 the integrals of the ABF curves, calculated using the trapezoidal rule. Bold values correspond to a lower ABF achieved throughout the optimization, highlighting the faster convergence of our algorithm. According to these numerical results, FST-PSO is characterized by a faster convergence than the competitor algorithms also in the case of Alpine and Vincent functions. Moreover, the ABF of FST-PSO found after 400 iterations is competitive: as reported in Figure

Table 4: Convergence speed measured as the integral of the ABF curves, calculated on every benchmark function during 400 iterations of each optimization algorithm

Function	PSO	PPSO	FST-PSO	ABC	CMA-ES	DE	GA
Ackley	$5.19 \cdot 10^3$	$4.52 \cdot 10^3$	$2.84 \cdot 10^3$	$6.23 \cdot 10^3$	$\mathbf{1.63 \cdot 10^3}$	$6.27 \cdot 10^3$	$4.52 \cdot 10^3$
Alpine	$3.16 \cdot 10^4$	$2.48 \cdot 10^4$	$\mathbf{1.05 \cdot 10^4}$	$3.24 \cdot 10^4$	$1.41 \cdot 10^4$	$4.88 \cdot 10^4$	$2.66 \cdot 10^4$
Bohachevsky	$6.67 \cdot 10^5$	$4.99 \cdot 10^5$	$\mathbf{2.00 \cdot 10^5}$	$2.02 \cdot 10^6$	$4.27 \cdot 10^5$	$2.22 \cdot 10^6$	$1.07 \cdot 10^6$
Griewank	$8.49 \cdot 10^4$	$6.48 \cdot 10^4$	$\mathbf{2.44 \cdot 10^4}$	$2.69 \cdot 10^5$	$5.59 \cdot 10^4$	$2.96 \cdot 10^5$	$1.41 \cdot 10^5$
Michalewicz	$-1.24 \cdot 10^4$	$-1.21 \cdot 10^4$	$-1.82 \cdot 10^4$	$-1.83 \cdot 10^4$	$-1.04 \cdot 10^4$	$-1.37 \cdot 10^4$	$\mathbf{-2.17 \cdot 10^4}$
Plateau	$-1.65 \cdot 10^5$	$-1.64 \cdot 10^5$	$-1.09 \cdot 10^5$	$-1.16 \cdot 10^5$	$-1.44 \cdot 10^5$	$-9.44 \cdot 10^4$	$-1.35 \cdot 10^5$
Quintic	$7.46 \cdot 10^6$	$6.17 \cdot 10^6$	$\mathbf{2.31 \cdot 10^6}$	$9.40 \cdot 10^7$	$1.58 \cdot 10^7$	$6.55 \cdot 10^7$	$3.28 \cdot 10^7$
Rastrigin	$3.19 \cdot 10^5$	$2.57 \cdot 10^5$	$\mathbf{1.44 \cdot 10^5}$	$2.73 \cdot 10^5$	$3.18 \cdot 10^5$	$3.87 \cdot 10^5$	$2.48 \cdot 10^5$
Rosenbrock	$8.51 \cdot 10^7$	$7.64 \cdot 10^7$	$\mathbf{4.06 \cdot 10^7}$	$7.41 \cdot 10^8$	$1.17 \cdot 10^8$	$4.38 \cdot 10^8$	$2.34 \cdot 10^8$
Shubert	$-1.89 \cdot 10^{65}$	$-1.63 \cdot 10^{64}$	$-3.51 \cdot 10^{75}$	$\mathbf{-3.32 \cdot 10^{100}}$	$-4.22 \cdot 10^{79}$	$-2.12 \cdot 10^{78}$	$-3.57 \cdot 10^{105}$
Vincent	$-2.39 \cdot 10^4$	$-3.16 \cdot 10^4$	$\mathbf{-3.40 \cdot 10^4}$	$-3.02 \cdot 10^4$	$-3.09 \cdot 10^4$	$-2.28 \cdot 10^4$	$-3.32 \cdot 10^4$
Xin-She Yang	$1.36 \cdot 10^7$	$\mathbf{1.03 \cdot 10^7}$	$6.74 \cdot 10^8$	$8.25 \cdot 10^8$	$3.57 \cdot 10^8$	$3.02 \cdot 10^7$	$3.13 \cdot 10^7$

6, FST-PSO outperforms the other algorithms in the case of Rastrigin and Xin-She Yang functions, and it ranks second in 7 out of 12 test cases, that is, Ackley, Alpine, Bohachevsky, Griewank, Quintic, Rosenbrock and Vincent functions. Figure 6 also shows that CMA-ES and FST-PSO achieve very similar ABF in the case of Bohachevsky, Griewank, Quintic and Rosenbrock functions. Regarding the Ackley function, CMA-ES is the only algorithm able to find the global optimum within 400 iterations, while FST-PSO ranks second.

As evidenced by the standard deviations plotted in Figure 5, in the specific case of the Rastrigin function, CMA-ES is characterized by a peculiar behavior at the end of the optimization process. Despite an initial worse convergence speed, CMA-ES is able to collect—during some of the 30 runs of the optimization—enough statistical information for the covariance matrix adaptation, yielding better solutions than the other tested algorithms. However, FST-PSO converges more consistently to optimal solutions with a negligible variance.

It is worth noting that FST-PSO is more performant than PSO and PPSO on almost all tested benchmark functions: in particular, in the case of Quintic and Shubert functions, FST-PSO outperforms the other two algorithms by many orders of magnitude. A notable exception is instead represented by the Plateau function, whose piecewise "flat" fitness landscape (Figure 4) might induce the firing of fuzzy rules that mislead particles in the early phases of the optimization.

A different scenario concerns the Michalewicz function, where GA obtains the best performances, FST-PSO achieves the same performances of ABC,

while CMA-ES shows the worst performances. In the case of the Shubert function, GA and ABC achieve the best results, while FST-PSO and CMA-ES obtain performances similar to DE. These results highlight the validity of the no free lunch theorem [19], as mentioned in Section 1.

Alpine, Plateau and Vincent functions are those characterized by the most interesting results. For what concerns the optimization of the Plateau function, standard PSO and PPSO achieve the best results, while FST-PSO is characterized by a lower convergence speed and worse final solutions (only DE performances are worse than FST-PSO). In the case of Alpine function, FST-PSO has the highest convergence speed at the beginning of the optimization; however, after around 140 iterations CMA-ES overtakes FST-PSO but, by the end of the optimization, they both achieve the same ABF. Similarly, for what concerns the Vincent function, GA has the highest convergence speed at the beginning of the optimization, but after 100 iterations FST-PSO overtakes GA and finally achieves better solutions by the end of the optimization. Moreover, CMA-ES convergence speed is very slow at the beginning, but increases abruptly around iteration 100 and, by iteration 180, the ABF of this algorithm results to be the best one: CMA-ES indeed reaches the best solution by the end of the optimization process for this benchmark function.

In order to further investigate the performance of FST-PSO, we considered the 28 shifted/rotated benchmark functions proposed during the contest on real-parameters single-objective bound-constrained optimization of the 2013 IEEE Congress on Evolutionary Computation (CEC'13). For this batch of tests, we compared FST-PSO with standard PSO and with CMA-ES, which represents the FST-PSO's main competitor, given the previous results. For each test, we assumed $M = 100$ for every benchmark function.

As shown in Figure 7, FST-PSO always outperforms standard PSO, except for the case of function $f8$ (a rotated Ackley function). FST-PSO and CMA-ES exhibit similar performance for almost all benchmark functions, except for the case of functions $f3, f7, f9, f11, f12, f13, f19, f25$ and $f27$, where CMA-ES achieves, on average, better results. FST-PSO outperforms standard PSO and CMA-ES in the case of functions $f14$ (a strongly multimodal, rotated, non-separable, asymmetrical Schwefel function, in which the second better local optimum is placed very far from the global optimum to mislead the optimization) and $f16$ (the rotated Katsuura function: multimodal, non-separable, asymmetrical, and non-differentiable). It is worth noting that in three cases ($f15$, $f22$, and $f23$), FST-PSO is characterized

by better results throughout the whole optimization and it is outperformed by CMA-ES only in the latest stages of the execution. Stated otherwise, in such cases, FST-PSO can yield better results than CMA-ES when a limited budget of fitness evaluations or running time is allowed. This topic will be discussed in the following section.

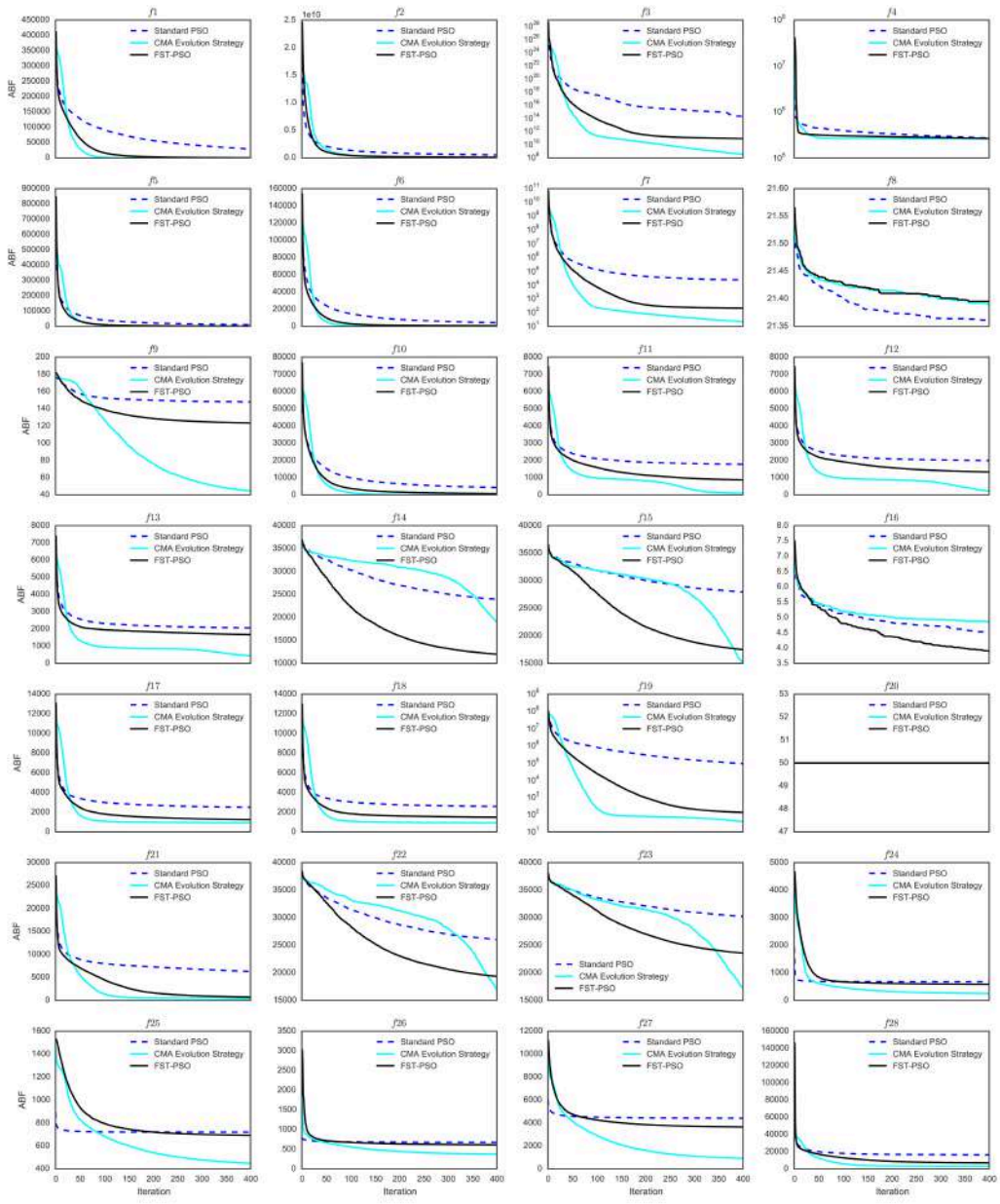

Figure 7: Comparison of the optimization performances of FST-PSO (black solid line), PSO (blue dashed line), and CMA-ES (light blue solid line) over the shifted/rotated CEC 2013 benchmark functions. These results show that FST-PSO outperforms standard PSO and it is largely competitive with CMA-ES.

4.3. Computational performance

Given the results presented in Section 4.2, it is interesting to assess the computational performances of the compared algorithms, especially in the case of FST-PSO and CMA-ES, which result to be the best optimization strategies for the benchmark functions tested in this work. To this aim, we executed additional tests to perform the optimization of the Plateau, Rastrigin and Vincent functions, for which FST-PSO and CMA-ES are characterized by the most divergent results: FST-PSO performs worse, better and initially better than CMA-ES on each of these benchmark functions, respectively (see Figure 5). The comparison between FST-PSO and CMA-ES was carried out by measuring their computational costs while increasing the number of dimensions of the search space—namely, $M = 1, 10, 20, 30, 50, 100$—of these three benchmark functions.

Table 5 reports the running time, expressed in seconds, required for the execution of 400 iterations of optimization of each fitness function using FST-PSO and CMA-ES on a workstation equipped with a Intel®Core™i7-3770 CPU @ 3.40GHz, 16GB RAM and running Ubuntu 14.10. According to our results, even in the case of low-dimension problems FST-PSO scales better than CMA-ES, which is tied to the numerous calculations for the covariance matrix adaptation. In the case of the Rastrigin function, FST-PSO is able to give, on average, better solutions than CMA-ES with lower computational costs. For instance, when $M = 100$, FST-PSO is approximately $70\times$ faster than CMA-ES. For the Plateau function, FST-PSO is strongly more efficient than CMA-ES, even for low dimension problems, being from $13\times$ to $77\times$ faster. The same holds for the Vincent function, where FST-PSO is around $81\times$ faster than CMA-ES when $M = 100$. Therefore, despite its additional computational costs due to the multiple Sugeno inferences, FST-PSO proves to be a very efficient optimization algorithm.

The computational efficiency of FST-PSO can also be analyzed from a different perspective. Let us assume to fix a total amount of computation time $\mathcal{T}$ to execute an optimization task. In this situation, although CMA-ES can (in principle) converge to better fitting solutions, according to our tests it would be outperformed by FST-PSO, which is able to execute a larger number of iterations in the same amount of time. To properly discuss this topic, we optimized the Plateau, Rastrigin and Vincent functions with FST-PSO and CMA-ES, performing 30 runs of each algorithm for each value of $M = 1, 10, 20, 30, 50, 100$. During each test, we stopped CMA-ES as soon as FST-PSO completed 400 iterations, keeping track of the fitness value of the

Table 5: Computational time (expressed in seconds) required to execute 400 iterations of FST-PSO and CMA-ES on the Plateau, Rastrigin and Vincent functions, with an increasing number of dimensions M

	Plateau		Rastrigin		Vincent	
M	CMA-ES	FST-PSO	CMA-ES	FST-PSO	CMA-ES	FST-PSO
1	15.04	1.13	1.24	1.21	6.42	1.19
10	26.04	1.81	5.41	2.04	28.11	1.92
20	39.65	2.41	12.26	2.82	46.06	2.60
30	58.81	3.16	26.70	3.78	72.67	3.36
50	134.90	4.77	88.03	5.94	159.69	5.08
100	703.22	9.14	702.07	10.16	785.96	9.68

best individuals identified during the last iteration of each run, which were later used to asses the ABF. The results of these analyses, reported in Table 6, prove that the higher efficiency of FST-PSO with respect to CMA-ES allows a better convergence when the same budget of computational time is used: the ABF is consistently better (except in the case of Rastrigin with $M = 10$ and Vincent with $M = 1$), especially in the case of high values of M.

Table 6: Comparison of the optimization performances (ABF $\pm$ standard deviation) of CMA-ES and FST-PSO, given the same amount of execution time, for the optimization of the Plateau, Rastrigin and Vincent benchmark functions

	Plateau		Rastrigin		Vincent	
M	CMA-ES	FST-PSO	CMA-ES	FST-PSO	CMA-ES	FST-PSO
1	24.88 ± 0.01	24.13 ± 0.34	0.01 ± 0.01	$4.8 \cdot 10^{-9} \pm 5.6 \cdot 10^{-9}$	-1.00 ± 0.0	$-0.99 \pm 2.3 \cdot 10^{-10}$
10	-14.87 ± 1.42	-19.27 ± 4.29	9.09 ± 4.39	11.41 ± 5.74	-7.72 ± 1.19	-9.91 ± 0.11
20	-50.77 ± 3.43	-61.34 ± 8.29	49.71 ± 44.22	33.96 ± 9.34	-12.76 ± 2.31	-19.74 ± 0.29
30	-82.08 ± 3.88	-105.80 ± 12.05	184.32 ± 44.86	53.23 ± 15.36	-15.94 ± 1.99	-29.07 ± 0.24
50	-126.02 ± 5.25	-194.16 ± 18.66	423.08 ± 22.67	95.42 ± 17.66	-19.99 ± 1.73	-49.41 ± 0.41
100	-166.08 ± 9.94	-352.84 ± 48.71	1036.09 ± 28.95	181.86 ± 29.37	-26.21 ± 2.48	-98.67 ± 0.63

4.4. Analysis of FST-PSO settings and input variables

In order to determine the robustness of the FST-PSO algorithm with respect to the numerical values used for the defuzzification of the output variables (Table 2), we performed a systematic analysis consisting in the variation of each defuzzification value for each output variable. Namely, the three crisp numerical values associated with the linguistic values *Low*, *Medium* and *High*, were sampled from a normal distribution, clamped in

zero, with mean equal to the reference value given in Table 2 and standard deviation equal to 0.1. Each test was repeated 30 times, in order to assess the ABF.

Figure 8 shows the effect of these perturbations on the optimization performances of FST-PSO. In general, we can observe that the values reported in Table 2 represent the best choice for the optimal functioning of FST-PSO: the perturbations allowed to achieve only similar or worse results, with the exception of Ackley and Quintic functions in which different settings of $\mathcal{U}$ allowed to obtain slightly better results with respect to those obtained with the values listed in Table 2. If we compare these results with those obtained with the competitor algorithms (see Figure 9), we can observe that the results described in Figures 5 and 6 are confirmed, since the ranking of the algorithms remains overall unchanged, thus proving the robustness of FST-PSO.

We also investigated the distribution of ϕ and δ values during the optimization process of FST-PSO. Figure 10 shows the distributions of ϕ and δ values calculated during the iterations of a single run of FST-PSO for the optimization of two peculiar benchmark functions: Rastrigin (top) and Michalewicz (bottom). Each point represents the value of ϕ (left) and δ (right) of a single particle at each iteration. The solid line shows the "trajectory" of such values for a single (randomly selected) particle. Even though both functions are multi-modal (as shown in Figure 4), the Rastrigin function is characterized by a high number of evenly distributed local minima with similar value, while the Michalewicz function has exactly $M!$ narrow local minima with different values. This peculiar difference in the fitness landscapes impinges on the distribution of ϕ (Figure 10, left): the particles moving in the search space of the Rastrigin function show small changes of ϕ, while the particles moving in the search space of the Michalewicz function are characterized by wide fluctuations of ϕ. This circumstance leads to the fast convergence of FST-PSO in the case of the Rastrigin function (faster than any competitor, as shown in Figure 6). On the contrary, in the case of the Michalewicz function, the execution of opposite rules might misguide particles and slow down the convergence speed.

5. Conclusion

In this paper we presented a novel self-tuning version of the PSO algorithm, starting from the concept of proactive particles in swarm optimization introduced by Nobile *et al.* [14]. The algorithm introduced here, named FST-

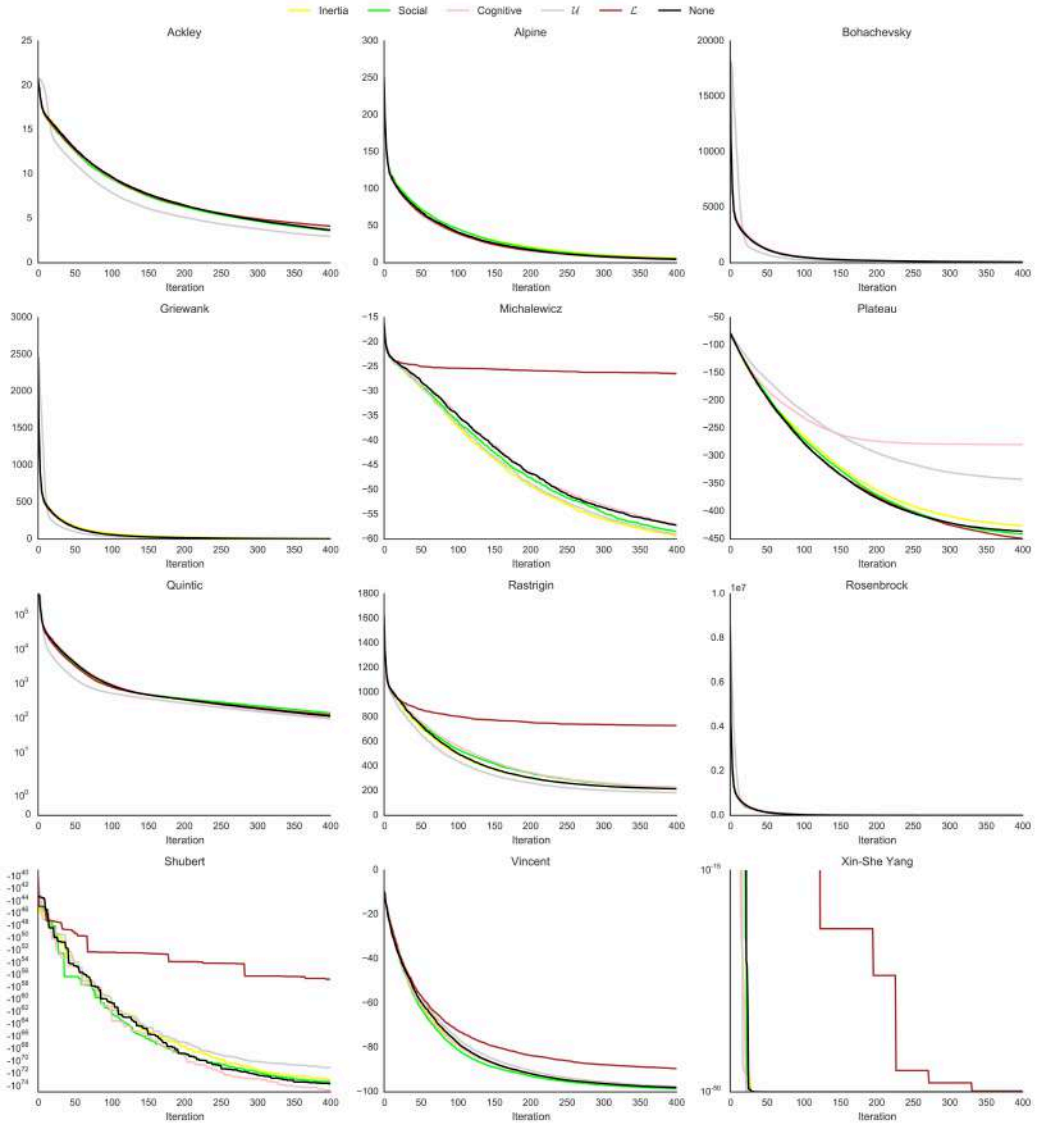

Figure 8: Comparison of the ABF obtained from different perturbations of the crisp values used in FST-PSO. Inertia (yellow lines), Social (light green lines), Cognitive (pink lines), $\mathcal{U}$ (silver lines) and $\mathcal{L}$ (brown lines) refer to the perturbation of the corresponding values listed in Table 2; "None" (black lines) refers to the results obtained with FST-PSO's default values.

PSO, exploits a FRBS to dynamically adjust the values of inertia, cognitive factor, social factor, lower and upper clamping values for minimum and max-

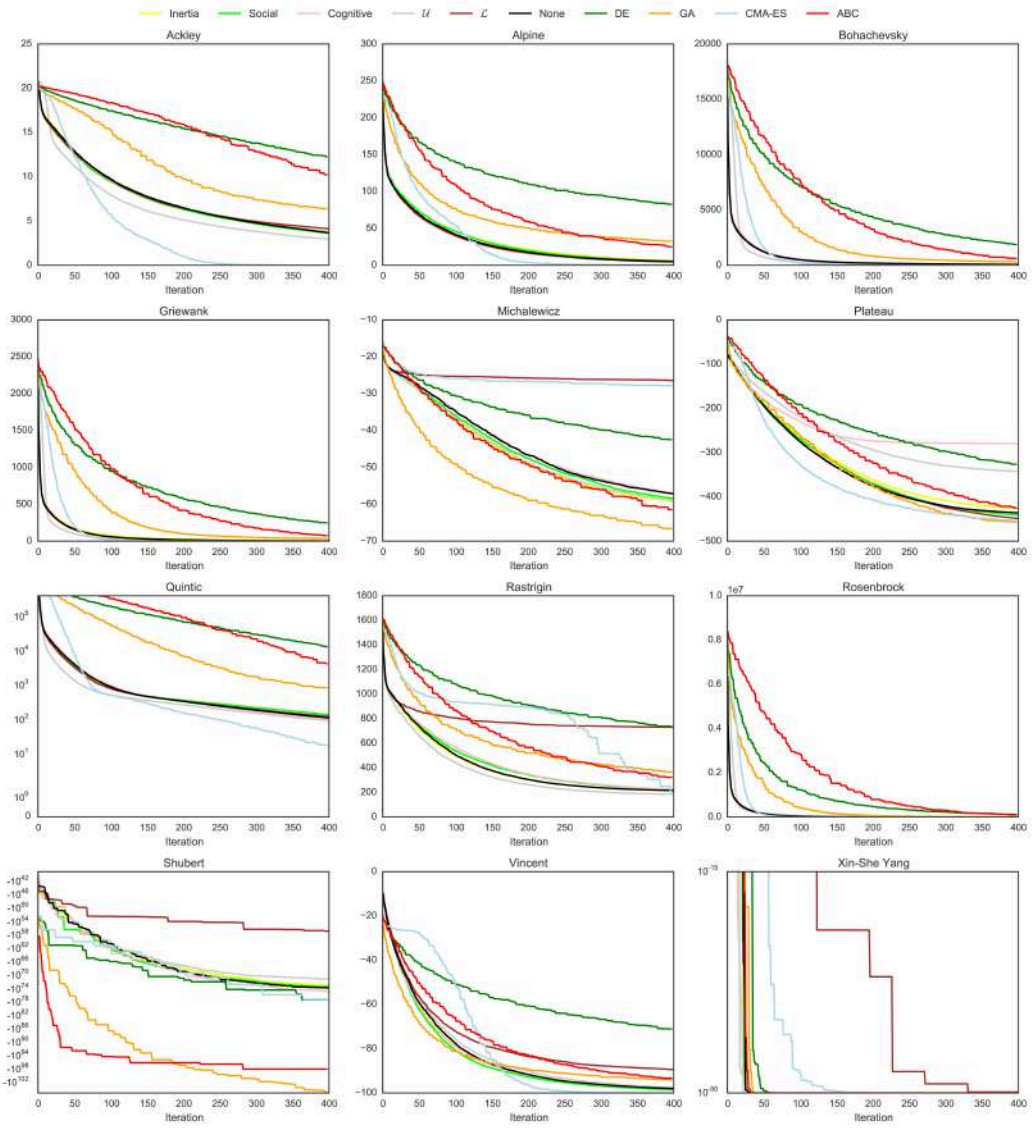

Figure 9: Comparison of the ABF obtained from different perturbations of the crisp values used in FST-PSO against the ABF obtained from the competitor algorithms considered in this work: DE (dark green lines), GA (orange lines), CMA-ES (light blue lines) and ABC (red lines). The perturbations were applied to the Inertia (yellow lines), Social Factor (light green lines), Cognitive Factor (pink lines), $\mathcal{U}$ (silver lines), $\mathcal{L}$ (brown lines). "None" (black lines) corresponds to the default values used by FST-PSO.

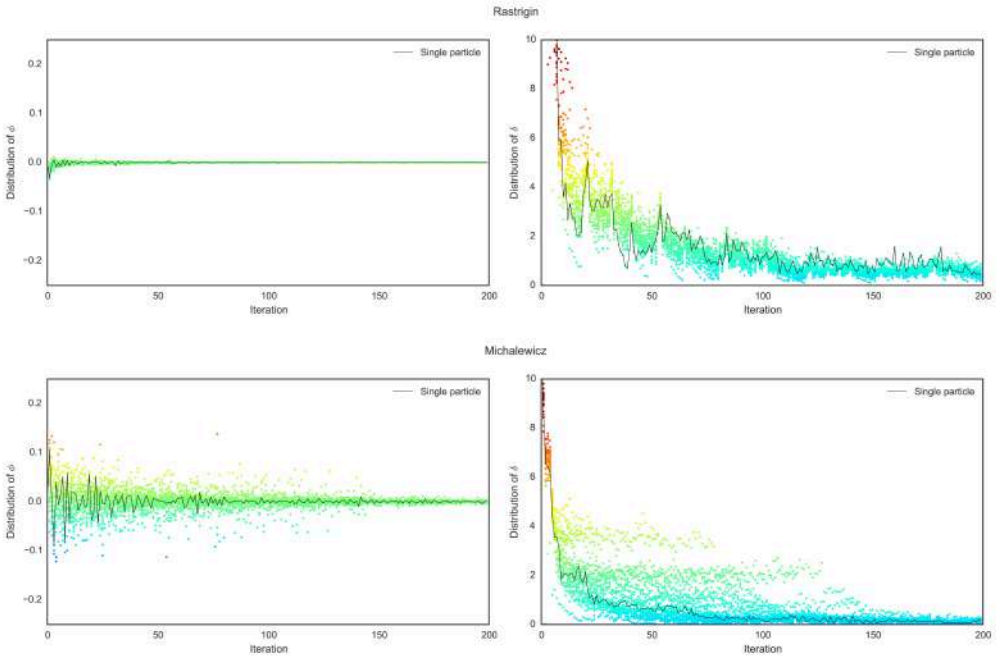

Figure 10: Values of ϕ (left) and δ (right) in the swarm during the optimization of the Rastrigin (top) and Michalewicz (bottom) functions. Each point corresponds to the ϕ and δ values of a single particle of the swarm, the black solid line is used to highlight how the ϕ and δ values of a (randomly chosen) particle change during the iterations of the optimization. The plots are limited to the first 200 iterations for the sake of clarity.

imum velocity of the swarm particles. In particular, each particle adjusts its own values during each iteration, according to its performance, therefore turning a swarm of reactive particles typical of PSO into a population of proactive agents. Thanks to this approach, FST-PSO does not require any manual user-setting and can be used "out-of-the-box". This represents a fundamental feature to facilitate the adoption of swarm intelligence methods by inexpert users, and it is especially useful in real case applications when no *a priori* knowledge is available about the optimization problem under investigation.

The performance of FST-PSO was compared against different optimization methods—standard PSO, PPSO, ABC, CMA-ES, DE and GA—by exploiting twelve well known multi-dimensional and multi-modal benchmark functions. In particular, we investigated the optimization performance by analyzing the convergence speed and the Average Best Fitness obtained through 30 repetitions of each algorithm, showing that FST-PSO turns out to be the best approach in terms of convergence speed and it is also competitive concerning the best solutions found. The main competitor of FST-PSO is CMA-ES, which achieves better results in the case of Ackley, Alpine and Vincent functions. We also showed the competitiveness of FST-PSO with respect to CMA-ES on shifted/rotated benchmark functions using the CEC'13 suite: CMA-ES resulted more performing than FST-PSO only on 9 functions out of 28. In these tests, FST-PSO also outperformed standard PSO in all but one benchmark functions. However, considering the computational time required by the two methods, we observed that FST-PSO is up to 80 times faster than CMA-ES, making our method more suitable for the application to real-world problems characterized by a high number of dimensions. Moreover, since FST-PSO was implemented in pure Python code, there is still room for a further improvement of computational performances by re-implementing the method with more performing languages like C or C++.

As a future research direction, we plan to investigate alternative fuzzy inference methods, exploring the impact of different implication operators on the FST-PSO performance. We are also planning to investigate a differential modification of proactive particle's behavior—instead of the direct change of the functioning settings, which is commonly used in all versions of fuzzy PSO—in order to prevent any situation in which the consecutive application of contradictory rules might slow down the exploration capability of a particle. In addition, FST-PSO might be extended with rules able to assign negative values to the *Social* factor, to the aim of preserving the diversity of

the swarm throughout the optimization process, thus avoiding the necessity of reboot strategies for particles [33]. An additional interesting issue regards the execution of a global sensitivity analysis [34] on the vertexes of the fuzzy sets for the two input variables considered in this work.

PSO is a very popular optimization method because, despite the lack of a proper convergence theorem, it has been successfully applied to many real-life problems (see [35] and references therein), including the problem of kinetic parameters estimation (PE) of biological systems. As a matter of fact, PSO was empirically shown to be one of the most suitable algorithm for PE [36, 37]. In particular, Dräger *et al.* [37] showed that PSO outperforms many of the algorithms that have also been tested in this paper, assumed that its functioning settings are properly selected. Therefore, as a future application we plan to integrate FST-PSO in the multi-swarm methodology for PE that we previously defined [38], in order to provide the community of Computational Systems Biology with a fully automatic methodology to determine the missing kinetic values of biochemical reaction networks.

References

[1] J. Kennedy, R. C. Eberhart, Particle swarm optimization, in: Proceedings IEEE International Conference on Neural Networks, Vol. 4, 1995, pp. 1942–1948.

[2] K. M. Malan, A. P. Engelbrecht, A survey of techniques for characterising fitness landscapes and some possible ways forward, Information Sciences 241 (2013) 148–163.

[3] M. Clerc, Particle swarm optimization, in: International Scientific and Technical Encyclopaedia, Wiley, Hoboken, NJ, 2006.

[4] G. Papa, Parameter-less algorithm for evolutionary-based optimization, Computational Optimization and Applications 56 (1) (2013) 209–229.

[5] M. Tomassini, L. Vanneschi, J. Cuendet, F. Fernández, A new technique for dynamic size populations in genetic programming, in: Proceedings 2004 IEEE Congress on Evolutionary Computation, Vol. 1, 2004, pp. 486–493.

[6] M. Hu, T.-F. Wu, J. D. Weir, An adaptive particle swarm optimization with multiple adaptive methods, IEEE Transactions on Evolutionary Computation 17 (5) (2013) 705–720.

[7] J. Yen, R. Langari, Fuzzy Logic: Intelligence, Control, and Information, Prentice Hall, Upper Saddle River, NJ, 1998.

[8] Y. Shi, R. C. Eberhart, Fuzzy adaptive particle swarm optimization, in: Proceedings 2001 IEEE Congress on Evolutionary Computation, Vol. 1, 2001, pp. 101–106.

[9] L. Magdalena, Fuzzy Rule-Based Systems, in: J. Kacprzyk, W. Pedrycz (Eds.), Springer Handbook of Computational Intelligence, Springer, 2015, pp. 203–218.

[10] A. Abraham, H. Liu, Turbulent particle swarm optimization using fuzzy parameter tuning, in: A. Abraham, A.-E. Hassanien, P. Siarry, A. Engelbrecht (Eds.), Foundations of Computational Intelligence Volume 3, Springer, Berlin, Germany, 2009, pp. 291–312.

[11] D. Tian, N. Li, Fuzzy particle swarm optimization algorithm, in: Proceedings JCAI'09. International Joint Conference on Artificial Intelligence, 2009, pp. 263–267.

[12] O. Castillo, P. Melin (Eds.), Fuzzy Logic Augmentation of Nature-Inspired Optimization Metaheuristics, Springer, Berlin, Germany, 2015.

[13] A. Rezaee Jordehi, J. Jasni, Parameter selection in particle swarm optimisation: a survey, Journal of Experimental and Theoretical Artificial Intelligence 25 (4) (2013) 527–542.

[14] M. S. Nobile, G. Pasi, P. Cazzaniga, D. Besozzi, R. Colombo, G. Mauri, Proactive particles in swarm optimization: a self-tuning algorithm based on fuzzy logic, in: Proceedings 2016 IEEE International Conference on Fuzzy Systems, 2015, pp. 1–8.

[15] D. Karaboga, B. Basturk, A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm, Journal of Global Optimization 39 (3) (2007) 459–471.

[16] N. Hansen, A. Ostermeier, Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation, in: Proceedings of 1996 IEEE International Conference on Evolutionary Computation, IEEE, 1996, pp. 312–317.

[17] R. Storn, K. Price, Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces, Journal of Global Optimization 11 (4) (1997) 341–359.

[18] J. H. Holland, Adaptation in Natural and Artificial Systems, The University of Michigan Press, Ann Arbor, Michigan, USA, 1975.

[19] D. H. Wolpert, W. G. Macready, No free lunch theorems for optimization, IEEE Transactions on Evolutionary Computation 1 (1) (1997) 67–82.

[20] P. Cazzaniga, M. S. Nobile, D. Besozzi, The impact of particles initialization in PSO: Parameter estimation as a case in point, in: Proceedings of the 2015 IEEE Conference on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB), IEEE, 2015, pp. 1–8.

[21] R. Poli, J. Kennedy, T. Blackwell, Particle swarm optimization, Swarm Intelligence 1 (1) (2007) 33–57.

[22] S. Xu, Y. Rahmat-Samii, Boundary conditions in particle swarm optimization revisited, IEEE Transactions on Antennas and Propagation 55 (2007) 760–765.

[23] N. Hansen, R. Ros, N. Mauny, M. Schoenauer, A. Auger, Impacts of invariance in search: When CMA-ES and PSO face ill-conditioned and non-separable problems, Applied Soft Computing 11 (8) (2011) 5755–5769.

[24] A. Tangherloni, L. Rundo, M. S. Nobile, Proactive Particles in Swarm Optimization: a settings-free algorithm for real-parameter single objective optimization problems, in: Evolutionary Computation (CEC), 2017 IEEE Congress on, IEEE, 2017, pp. 1940–1947.

[25] W. Pedrycz, Why triangular membership functions?, Fuzzy Sets and Systems 64 (1) (1994) 21–30.

[26] M. Sugeno, Industrial Applications of Fuzzy Control, Elsevier Science Inc., New York, NY, 1985.

[27] R. Fletcher, Practical Methods of Optimization, John Wiley & Sons, New York, NY, 2013.

[28] T. E. Oliphant, Python for scientific computing, Computing in Science and Engineering 9 (3) (2007) 10–20.

[29] D. Izzo, PyGMO and PyKEP: open source tools for massively parallel optimization in astrodynamics (the case of interplanetary trajectory optimization), in: Proceedings of the Fifth International Conference on Astrodynamics Tools and Techniques, ICATT, 2012.

[30] N. Hansen, A. Auger, R. Ros, S. Finck, P. Pošík, Comparing results of 31 algorithms from the black-box optimization benchmarking BBOB-2009, in: Proceedings of the 12th annual conference companion on Genetic and evolutionary computation, ACM, 2010, pp. 1689–1696.

[31] N. Hansen, The CMA evolution strategy: a comparing review, in: J. Lozano, P. Larranaga, I. Inza, E. Bengoetxea (Eds.), Towards a New Evolutionary Computation. Advances on Estimation of Distribution Algorithms, Springer, 2006, pp. 75–102.

[32] E. Mezura-Montes, J. Velázquez-Reyes, C. A. Coello Coello, A comparative study of differential evolution variants for global optimization, in: Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation, ACM, 2006, pp. 485–492.

[33] S. Spolaor, A. Tangherloni, L. Rundo, M. S. Nobile, P. Cazzaniga, Reboot strategies in particle swarm optimization and their impact on parameter estimation of biochemical systems, 2017 IEEE Conference on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB) (Accepted).

[34] A. Saltelli, M. Ratto, T. Andres, F. Campolongo, J. Cariboni, D. Gatelli, M. Saisana, S. Tarantola, Global Sensitivity Analysis: The Primer, Wiley-Interscience, New York, NY, 2008.

[35] R. Poli, Analysis of the publications on the applications of particle swarm optimisation, Journal of Artificial Evolution and Applications 2008 (2008) 2–10.

[36] D. Besozzi, P. Cazzaniga, G. Mauri, D. Pescini, L. Vanneschi, A comparison of genetic algorithms and particle swarm optimization for parameter estimation in stochastic biochemical systems, in: C. Pizzuti, M. D. Ritchie, M. Giacobini (Eds.), Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics, Vol. 5483 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2009, pp. 116–127.

[37] A. Dräger, M. Kronfeld, M. J. Ziller, J. Supper, H. Planatscher, J. B. Magnus, Modeling metabolic networks in *C. glutamicum*: a comparison of rate laws in combination with various parameter optimization strategies, BMC Systems Biology 3 (2009) 5.

[38] M. S. Nobile, D. Besozzi, P. Cazzaniga, G. Mauri, D. Pescini, A GPU-based multi-swarm PSO method for parameter estimation in stochastic biological systems exploiting discrete-time target series, in: M. Giacobini, L. Vanneschi, W. Bush (Eds.), Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics, Vol. 7246 of Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2012, pp. 74–85.