Information Sciences 239 (2013) 96–121

Contents lists available at SciVerse ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Compact Particle Swarm Optimization

Ferrante Neri[a,b,*], Ernesto Mininno[b,1], Giovanni Iacca[c]

[a] *Centre for Computational Intelligence, School of Computer Science and Informatics, De Montfort University, The Gateway, Leicester LE1 9BH, England, United Kingdom*
[b] *Department of Mathematical Information Technology, University of Jyväskylä, P.O. Box 35, Agora 40014, Finland*
[c] *INCAS^3-Innovation Centre for Advanced Sensors and Sensor Systems, P.O. Box 797, 9400 AT Assen, The Netherlands*

A R T I C L E I N F O

Article history:
Received 8 October 2012
Received in revised form 18 February 2013
Accepted 10 March 2013
Available online 22 March 2013

Keywords:
Particle Swarm Optimization
Compact optimization
Limited memory problems
Swarm intelligence
Computational intelligence optimization
Real-time application

A B S T R A C T

Some real-world optimization problems are plagued by a limited hardware availability. This situation can occur, for example, when the optimization must be performed on a device whose hardware is limited due to cost and space limitations. This paper addresses this class of optimization problems and proposes a novel algorithm, namely compact Particle Swarm Optimization (cPSO). The proposed algorithm employs the search logic typical of Particle Swarm Optimization (PSO) algorithms, but unlike classical PSO algorithms, does not use a swarm of particles and does not store neither the positions nor the velocities. On the contrary, cPSO employs a probabilistic representation of the swarm's behaviour. This representation allows a modest memory usage for the entire algorithmic functioning, the amount of memory used is the same as what is needed for storing five solutions. A novel interpretation of compact optimization is also given in this paper. Numerical results show that cPSO appears to outperform other modern algorithms of the same category (i.e. which attempt to solve the optimization despite a modest memory usage). In addition, cPSO displays a very good performance with respect to its population-based version and a respectable performance also with respect to some more complex population-based algorithms. A real world application in the field of power engineering and energy generation is given. The presented case study shows how, on a model of an actual power plant, an advanced control system can be online and real-time optimized. In this application example the calculations are embedded directly on the real-time control system.

1. Introduction

Despite the development of modern computational devices, some applications require the solution of a complex optimization problem notwithstanding limited hardware conditions. These conditions are here classified in the following way:

- Cost and space limitations.
- Real-time requirements.
- Fault-tolerance requirements.

The first category includes those devices which require the solution of an optimization problem but cannot use a computational device due to cost and space limitations. For example, a vacuum cleaner robot is supposed to undergo a learning

* Corresponding author. Tel.: +358 14 260 1211; fax: +358 14 260 1021.
E-mail addresses: fneri@dmu.ac.uk, ferrante.neri@jyu.fi (F. Neri), ernesto.mininno@jyu.fi (E. Mininno), giovanniiacca@incas3.eu (G. Iacca).
[1] Tel.: +358 14 260 1211; fax: +358 14 260 1021.

0020-0255/$ - see front matter © 2013 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.ins.2013.03.026

process in order to locate where obstacles are placed in a room (e.g. a sofa, a table, etc.) and then perform an efficient cleaning of the accessible areas. The learning process can be of several kinds, e.g. a neural network training. Although the robot must contain a computational core, e.g. within a cheap control card, it clearly cannot contain all the full power components of a modern computer, since they would increase volume, complexity, and cost of the entire device, see [95,37,61].

The second category refers to those problems which require a very fast solution of the optimization problem. In order to avoid delays due to the communication time between a control/actuator device and an external computer, real-time necessities may impose that the optimization is embedded within the control/actuator device. This situation can happen in industrial plants for energy production, see [96,65], or in telecommunications, see [12].

The third category includes special applications where fault-tolerance is a high priority and/or rebooting of the device is to avoid. An extremely interesting example is the space shuttle control: despite the constant growth of the power in computational devices, relatively simple hardware is still used on space shuttles in order to reduce fault risks. For example, since over 20 years, National Aeronautics and Space Administration (NASA) employs, within space shuttles, IBM AP-101S computers, see [60]. These computers constitute an embedded system for performing the control operations. The memory of computational devices is only 1 Mb, i.e. much less capacious than any modern device. On the other hand, this choice allows a high reliability of the computational core. In addition to that, it must be remarked that the computational devices on board of a space shuttle should reliably work without any kind of rebooting for months or, in some cases, even for years. Thus, the necessity of having an efficient control, an thus an efficient optimization algorithm on-board, notwithstanding the hardware limitations arise.

Although the No Free Lunch Theorems suggest that for a given problem a tailored algorithm should be designed, see [93], during the latest years, a multitude of algorithms displaying a high performance on a diverse set of problems have been proposed. These algorithms employ multiple search operators, intelligent systems, machine learning structures, and attempt to adaptively modify their behaviour in the presence of different fitness landscapes. Two examples of recently proposed algorithms which employ multiple search operators (or search algorithms) by means of a set of activation probabilities are presented in [89,64]. A very similar concept is encompassed by the idea of ensemble, see [44], where multiple strategies concur, by means of a self-adaptive/randomized mechanism, to the optimization of the same fitness function. Another similar concept is given in [69,100], where multiple search strategies, a complex randomized self-adaptation, and a learning mechanism are framed within a Differential Evolution (DE) structure. In [91] a heuristic technique assists Particle Swarm Optimization (PSO) in selecting the desired solutions while solving multi-objective optimization problems. In [79], a Cellular Automata scheme is integrated in the basic PSO velocity update rule in order to modify the particle trajectories and avoid them being trapped in local optima. In [52] a success based learning is designed to guide the progresses of search in PSO frameworks. Another good example of this algorithmic philosophy is the Frankenstein PSO, see [51], which combines several successful variants of PSO in order to make an ultimate PSO version. These methods, although very efficient, are usually complex and hardware demanding, e.g. require the storage of a population of solutions and of additional memory for saving learning structures. Thus, these algorithms would not be a viable option when only a limited hardware is available.

An extensive study on population-based algorithms and their advantages over single solution (i.e. memory-saving) algorithms has been reported in [67]. Five distinct mechanisms by which a population-based algorithm has an advantage over a single solution algorithm are identified. The first mechanism is that a population offers a diversified pool of building blocks whose combination might generate new promising solutions. The second mechanism is the result of focusing of the search caused by crossover. Since most crossover operators have the property that, if both parents share the same value of a variable, then the child also has the same value in correspondence of that variable, crossover only explores the part of the search space where individuals disagree. In contrast, mutation explores the entire search space. This focusing of the search by crossover can dramatically reduce the time needed to find a good solution. The third mechanism is the capability of a population to act as a low-pass filter of the landscape, ignoring short-length scale features in the landscape (e.g. shallow basins of attractions). The fourth mechanism is the possibility to search different areas of the decision space. This mechanism can be seen also in a different way: since population-based algorithms naturally perform an initial multiple sampling, the chance that an unlucky initial sample jeopardizes the entire algorithmic functioning is significantly mitigated. The fifth mechanism is the opportunity of using the population to learn about good parameters of the algorithm, i.e. to find a proper balance between exploration and exploitation.

For the above-listed reasons, the employment of a population-based algorithm seems to be preferable when possible. Unfortunately, in many problems falling into the categories mentioned at the beginning of this section, there may be not enough memory to store a population composed of numerous candidate solutions. Compact algorithms represent an efficient compromise as they present some advantages of population-based algorithms but do not require the memory for storing an actual population of solutions. These algorithms simulate the behaviour of population-based algorithms by employing, instead of a population of solutions, its probabilistic representation. In this way, a much smaller number of parameters must be stored in the memory. Thus, a run of these algorithms requires much less capacious memory devices compared to their correspondent population-based structures.

Compact algorithms belong to the class of Estimation of Distribution Algorithms (EDAs) as the explicit representation of the population is replaced with a probability distribution, see [42]. The very first implementation of compact algorithms has been the compact Genetic Algorithm (cGA), defined in [29]. The cGA simulates the behaviour of a standard binary encoded Genetic Algorithm (GA). In [29], it can be seen that cGA has a performance almost as good as that of GA and that cGA requires a much less capacious memory. Paper [70] performs a convergence analysis of cGA by using Markov chains. The extended compact Genetic Algorithm (ecGA), proposed in [28] and developed in [30] is based on the idea that the choice of a good probability distribution is equivalent to linkage learning. The measure of a good distribution is based on Minimum Description Length (MDL) models: simpler distributions are better than complex ones. The probability distribution used in ecGA belongs to a class of probability models known as Marginal Product Models (MPMs). A theoretical analysis of the ecGA behavior is presented in [73]. A hybrid version of ecGA integrating the Nelder–Mead algorithm is proposed in [75]. A study on the scalability of ecGA is given in [74]. The cGA and its variants have been intensively used in hardware implementation, see [5,24,36]. A cGA application to neural network training is given in [23].

In [7] a memetic variant of cGA is proposed in order to enhance the convergence performance of the algorithm in the presence of a relatively high number of dimensions. Paper [2] analyses analogies and differences between cGAs and (1 + 1)-Evolution Strategy ((1 + 1)-ES) and extends a mathematical model of ES, [72], to cGA obtaining useful information on the performance. Moreover, [2] introduces the concept of elitism, and proposes two new variants, with strong and weak elitism respectively, that significantly outperform both the original cGA and (1 + 1)-ES. A real-encoded cGA (rcGA) has been introduced in [48]. A noise robust version of rcGA has been proposed in [58,41].

In [49] the compact Differential Evolution (cDE) algorithm has been introduced. Some enhancements of cDE have been proposed. A memetic variant composed of cDE and a local search component has been proposed in [57] and an unconventional memetic approach based on a cDE structure has been presented in [55]. In [32,33], the effects of super-fit scheme and opposition based learning, respectively, are studied for cDE schemes. A survey on compact optimization is given in [56].

As shown in [49], while the compact algorithms based on a GA framework tend to perform worse than their population-based versions, especially in high dimensions, several cDE implementations (multiple trials are shown for various mutation schemes) are competitive with their corresponding population-based algorithms. The success of cDE implementation has been explained in [49] as the combination of two factors. The first is that a DE scheme seems to benefit from the introduction of a certain degree of randomization due to the probabilistic model. The second is that the one-to-one spawning survivor selection typical of DE (the offspring replaces the parent) can be naturally encoded into a compact logic. Conversely, GA selection schemes such as tournament selection are not prone to a compact encoding.

This paper proposes the definition of compact Particle Swarm Optimization (cPSO). The proposed algorithm benefits from the same natural encoding of the selection scheme employed by DE. In addition, PSO natively employs another “ingredient” of compact optimization, i.e. a special treatment for the best solution ever detected (elite). On the other hand, since compact algorithms are normally thought as Evolutionary Algorithms (EAs), PSO has been here reinterpreted as an EA in order to propose a compact encoding of PSO.

It must be clarified that in literature some papers hybridize PSO and EDAs, see e.g. [1]. These works are very different from the proposed cPSO since they, unlike cPSO, are not compact algorithms but are population-based algorithms which combine search logics of both PSO and EDAs.

The remainder of this article is organized in the following way. Section 2 describes the PSO functioning and illustrates various enhancements of the basic PSO scheme. Section 3, after some general considerations about compact optimization, describes the cPSO encoding and highlights its implementation details. Section 4 displays the numerical results carried out on a diverse testbed while Section 5 shows a control engineering real-world application which imposes, due to real-time constraints, the use of compact algorithms. Results on the real-world case highlight the benefits of the proposed cPSO. Finally, Section 6 gives the conclusion of this work.

2. Particle Swarm Optimization

Without a loss of generality, in this paper we refer to the minimization problem of an objective function $f(\mathbf{x})$, where $\mathbf{x}$ is a vector of n design variables defined within a domain D, namely decision space. Arrays are indicated in bold face while scalar values are in italic.

Particle Swarm Optimization (PSO) [38] is a metaheuristic inspired by the collective behaviour of animals (such as birds) and widely applied to various problems in continuous and discrete optimization. This study will focus on continuous problems, see [18] for a broad literature review. A set of particles moves inside the decision space D in order to detect its promising areas. For each particle k, the most successful visited position $\mathbf{x_{lb-k}}$ is recorded. More formally, in order to optimize the objective function f, a population of particles (called swarm) is randomly initialized in the solution space. The objective function determines the quality of the solution represented by a particle's position. At any time step t, a particle k has an associated position vector $\mathbf{x_k^t}$ and a velocity vector $\mathbf{v_k^t}$. For each particle k, the best position, in terms of objective function value, that the particle has ever visited (called local best) is stored. In this way, a vector of local best solutions $\mathbf{x_{lb}^t}$ is stored and progressively updated. The best solution amongst all that were visited, i.e. the element of the vector $\mathbf{x_{lb}^t}$ characterized by the lowest objective function value is called global best and is indicated as $\mathbf{x_{gb}^t}$. In order to pass from the step t to the step $t + 1$ and generate more promising solutions, each particle is perturbed according to the following formulas:

$$\mathbf{v_k^{t+1}} = \phi_1 \mathbf{v_k^t} + \phi_2 (\mathbf{x_{lb-k}^t} - \mathbf{x_k^t}) + \phi_3 \left(\mathbf{x_{gb}^t} - \mathbf{x_k^t}\right) \tag{1}$$

and

$$\mathbf{x_k^{t+1}} = \mathbf{x_k^t} + \mathbf{v_k^{t+1}} \tag{2}$$

where as mentioned above, $\mathbf{x_k^t}$ indicates the current position of the particle, $\mathbf{x_{lb-k}}$ is the best position visited by the kth particle, $\mathbf{x_{gb}}$ is the best position ever detected by the entire set of particles. Finally, the vector $\mathbf{v_k^t}$ is a perturbation vector, namely velocity, and ϕ_1, ϕ_2, and ϕ_3 are three weight factors which might be constant or randomized.

2.1. Modified and improved versions of PSO: a brief review

During the latest years, a multitude of implementations and modified versions have been proposed to correct the general PSO scheme presented above. Although this paper does not aim at offering an exhaustive literature review on the topic, this subsection briefly summarizes some amongst the most popular versions of PSO. Many of these versions can be seen as a setting and re-arrangement of the weight factors ϕ_1, ϕ_2, and ϕ_3. One example, namely constricted PSO has been presented in [14] and modifies formula (1) as follows:

$$\mathbf{v_k^{t+1}} = \chi\left(\mathbf{v_k^t} + \beta_1\mathbf{U_1}\left(\mathbf{x_{lb-k}^t} - \mathbf{x_k^t}\right) + \beta_2\mathbf{U_2}\left(\mathbf{x_{gb}^t} - \mathbf{x_k^t}\right)\right) \tag{3}$$

where U_1 and U_2 are $n \times n$ random matrices, β_1 and β_2 are constant values usually set equal to 2.05, see [19,86]. Parameter χ is related to β_1 and β_2: $\chi = \frac{2}{\left|2-\beta-\sqrt{\beta^2-4\beta}\right|}$, where $\beta = \sum\beta_k$.

The concept of inertia weight, that is the constant ϕ_1 in formula (1), and its role and setting have been thoroughly studied in [77,78]. An efficient strategy for setting the inertia weight is by means of a time-decreasing law in order to have an algorithm that initially explores the search space and only later focuses on the most promising regions. This result is obtained by means of the following formula:

$$\phi_1^t = \frac{\phi_1 t_{max} - t}{\phi_1 t_{max}}\left(\phi_{1-max} - \phi_{1-min}\right) + \phi_{1-min} \tag{4}$$

where $\phi_1 t_{max} - t$ marks the time at which $\phi_1^t = \phi_{1-min}$, and ϕ_{1-max} and ϕ_{1-min} are the maximum and minimum values the inertia weight can take, respectively. Other dynamic schemes for the inertia weight have been proposed. For example in [103,102], a time-increasing scheme has been tested while in [20] a randomized inertia weight is proposed.

In [45] the Fully Informed Particle Swarm (FIPS) has been introduced. The latter algorithm employs the constricted PSO rule introduced in [14] but equally distributes over neighbour particles the β value. In other words, for a particle k the velocity update is given by the following formula:

$$\mathbf{v_k^{t+1}} = \chi\left(\mathbf{v_k^t} + \sum_{k\in N_k}\beta_k\mathbf{U_k^t}\left(\mathbf{x_{gb}^t} - \mathbf{x_k^t}\right)\right) \tag{5}$$

The Hierarchical Particle Swarm Optimizer with Time-Varying Acceleration Coefficients (HPSOTVAC), proposed in [71], works without the inertia term in the velocity-update rule. The first feature of this PSO modification is that if any component of the velocity of a particle is below a threshold (when it approaches zero) a reinitialization of the velocity occurs. This reinitialization (value assignment) is performed by using a time-varying parameter, namely maximum velocity. The second feature is that the coefficients ϕ_2 and ϕ_3, see formula (1), are time-varying, the first coefficient being time-decreasing, the second coefficient time-increasing.

Other PSO variants improve upon the PSO performance by hybridizing, in a memetic fashion (see [54,53,34]), the original framework with other algorithmic paradigms. In [4] PSO performance is enhanced by means of the hybridization with a GA selection strategy. In [3] an evolutionary mutation assists a PSO while in [97] an evolutionary crossover supports a PSO framework. In [101] PSO and DE are combined. In [10] a PSO is used to enhance the performance of an individual and generate a super-fit solution for DE. A cooperative approach is proposed in [87] while in [62] a repulsion technique has been developed to prevent premature convergence. In [22], a hybrid algorithm that merges potentials of PSO with the Bacterial Foraging Optimization is proposed with reference to an image-compression problem. In [11], the PSO performance is enhanced by re-organizing the particles in a parallel fashion.

The concept of topology in PSO is another widely discussed topic over the last years. For example, paper [82] proposes a variable topology in PSO and paper [39] proposes the employment of small neighbourhood for tackling complex problems. More recently, Adaptive Hierarchical Particle Swarm Optimization (AHPSO), proposed in [35], employs a variable topology. In the fashion of Genetic Programming, particles are arranged according to a tree topology structure where the most promising particles (better fitness values) are located in the upper nodes of the tree. At each step, the velocity update occurs taking into account the previous best performance of the particle under examination and the previous best performance of its parent (in the tree). Before the velocity update is performed, the previous best fitness value of any particle is compared with that of its parent. If it is better, child and parent swap their positions in the hierarchy.

A popular and efficient PSO variant is the Comprehensive Learning Particle Swarm Optimizer (CLPSO), proposed in [43]. In CLPSO the velocity update is given by the following formula:

$$\mathbf{v_k^{t+1}} = \phi_1\mathbf{v_k^t} + \phi_2\mathbf{U}\left(\mathbf{x_{lb-f}^t} - \mathbf{x_i^t}\right) \tag{6}$$

where $\mathbf{U}$ is an $n \times n$ random matrix and $\mathbf{x}^{\mathbf{t}}_{\mathbf{lb-f}}$ is a heterogeneous vector composed, along each design variable, of local best design variables. For the sake of clarity, formula (6) can be re-written for a generic design variable *i*, in the following way:

$$v_k^{t+1}[i] = w \cdot v_k^t[i] + c \cdot rand_i \left(x^t_{lb-f}[i] - x^t_k[i] \right) \tag{7}$$

where *w* and *c* are constant values while $rand_i$ is a randomly generated number between 0 and 1. In order to determine (select) each $x^t_{lb-f}[i]$ value, a probabilistic procedure is implemented. For each design variable, a random number is generated and compared with a threshold value P_c. If it is higher then, for the corresponding design variable, the local best particle is followed. If it is lower then two local best particles are randomly selected from the swarm and their fitness value compared. The winning particle donates the corresponding design variable to $\mathbf{x}^{\mathbf{t}}_{\mathbf{lb-f}}$. In order to set P_c, an empirical rule has been proposed in [43]: for each particle *i*, a P_{c-i} value is calculated by the following empirical expression:

$$P_{c-i} = 0.05 + 0.45 \frac{e^{\frac{10(i-1)}{SS-1}} - 1}{(e^{10} - 1)} \tag{8}$$

where *SS* is swarm size (number of particles). A successful application of CLPSO to an image processing problem is given in [68].

A simple PSO related algorithm has been proposed in [104] and named Intelligence Single Particle Optimization (ISPO). ISPO requires a modest memory employment as it perturbs a single particle rather than an entire swarm. More specifically, a particle is perturbed for each of the *n* design variables *H* times. For each generic variable *k* a learning factor *L* is used. At the beginning *L* is initialized to zero. Then, for $t = \{1,2,\ldots,H\}$, a velocity vector for the variable *i* is generated as follows:

$$v^{t+1}[i] = \frac{A}{(t+1)^P} \cdot rand_{(-0.5,0.5)} + B \cdot L^t \tag{9}$$

where *A*, *P*, and *B* are three parameters which we will refer to as acceleration, acceleration power factor, and learning coefficient. The velocity is then added to the corresponding variable of the particle as in a original PSO: $x[i]^{t+1} = x[i]^t + v[i]^{t+1}$. The velocity in formula (9) means that, for each variable, a randomized exploratory radius is generated. This radius is larger at the beginning and progressively shrinks as *t* is increased. When the new design variable has been perturbed, the fitness function value of the corresponding candidate solution is calculated and compared with that of the solution before the perturbation. If an improvement has been achieved (or the perturbed solution has the same fitness of that before perturbation), then the velocity value is assigned to the learning factor: $L^{t+1} = v^{t+1}[i]$. If there is no improvement, the learning factor is reduced by means of a shrinking factor $S_f : L^{t+1} = \frac{L^t}{S_f}$. If L^{t+1} is smaller than a precision value ϵ, then it is reinitialized to zero. This set of operations is performed separately for each variable *i*.

Recently, in [51], three modified PSO versions are combined to generate the so called Frankenstein PSO (FPSO). This algorithm contains a time-varying population topology proposed in [71], the mechanism for updating the velocity of the particles proposed in [45], see formula (5), and the time-decreasing inertia weight proposed in [77,78], see formula (4).

In [98] the particles are mapped according to their position in the decision space and on the basis of the position of the global best particle with respect to the other particles a set of fuzzy rules establish the operation to be performed. This fuzzy mechanism is combined with a set of adaptive rules for the parameter updates. These rules have been presented in literature for example in [78,94].

In [40] two complex mechanisms have been introduced in order to enhance the PSO performance in multidimensional problems. The first mechanism is the inter-dimensional navigation of swarm particles: instead of operating at a fixed dimension, this modified PSO works with a range of dimensions and seeks for both positional and dimensional optima. The second mechanism compensate the premature convergence effect introduced by the first one by means of the fractional global best formation: all the best dimensional components are collected and used to fractionally create an artificial global best particle that has the potential to be a better guide than the original global best of PSO.

Other recent achievements are in [17] where the concept of incremental social learning, typical of multi-agent systems, has been successfully applied in the context of PSO, in [99] where another learning component, the orthogonal learning, supports the PSO framework, and in [80] where PSO is enhanced by means of a hybridization with Broyden–Fletcher–Goldfarb–Shanno method. In [27] a valuable analysis on the interaction amongst particles in a PSO framework is given. Other recently proposed examples of PSO based algorithms obtained by modifying the original paradigm are shown in [90,84].

3. Compact Particle Swarm Optimization

This section presents the algorithmic features of the proposed cPSO. Before discussing the implementation details of cPSO, some considerations are done about compact optimization and encoding of population-based algorithms. As for the other compact algorithms with real encoding previously presented in literature, see [48,49], each variable is assumed to be normalized in the interval $[-1,1]$.

3.1. Virtual population in real-valued compact algorithms

As mentioned above, the main feature of compact algorithms is the fact that they are population-less. Instead of processing an actual population of solution a virtual population is used. This virtual population is probabilistic model of a population of solutions and is encoded within a data structure, namely Perturbation Vector and indicated with **PV**. In binary encoded compact algorithms such as cGA in [29] and elitist cGA in [2], **PV** is a vector of binary numbers. In real-valued compact algorithm, such as rcGA in [48] and cDE in [49], the so called perturbation vector is actually a $n \times 2$ matrix:

$$\mathbf{PV}^t = [\boldsymbol{\mu}^t, \boldsymbol{\sigma}^t] \tag{10}$$

where $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ are, respectively, vectors containing, for each design variable, mean and standard deviation values of a Gaussian Probability Distribution Function (PDF) truncated within the interval $[-1,1]$. The amplitude of the PDF is normalized in order to keep its area equal to 1. The apex t in formula (10) indicates the generation (number of performed comparison).

The initialization of the virtual population is performed in the following way. For each design variable i, $\mu^1[i] = 0$ and $\sigma^1[i] = \lambda$ where λ is set as a large positive constant ($\lambda = 10$). This initialization of $\sigma[i]$ values is done in order to initially have a truncated Normal distribution with a wide shape.

The sampling mechanism of a design variable $x[i]$ associated to a generic candidate solution **x** from **PV** is not a trivial procedure and requires an extensive explanation. For each design variable indexed by i, a truncated Gaussian PDF with mean value $\mu[i]$ and standard deviation $\sigma[i]$ is associated. The PDF is described by the following formula:

$$PDF(truncNorm(x)) = \frac{e^{-\frac{(x-\mu[i])^2}{2\sigma[i]^2}}\sqrt{\frac{2}{\pi}}}{\sigma[i]\left(\mathrm{erf}\left(\frac{\mu[i]+1}{\sqrt{2}\sigma[i]}\right) - \mathrm{erf}\left(\frac{\mu[i]-1}{\sqrt{2}\sigma[i]}\right)\right)} \tag{11}$$

where erf is the error function, see [26].

The PDF in formula (11) is then used to compute the corresponding Cumulative Distribution Function (CDF). The CDF is constructed by means of Chebyshev polynomials by following the procedure described in [15] (the codomain of CDF is $[0,1]$). The sampling of the generic design variable $x[i]$ from **PV** is performed by generating a random number $rand(0,1)$ from a uniform distribution and then computing the inverse function of CDF in $rand(0,1)$. The newly calculated value is $x[i]$. The sampling mechanism is graphically represented in Fig. 1. When the comparison between two individuals of the population (or better two individuals sampled from **PV**) is performed the winner solution biases the **PV**. Let us indicate with **winner** the vector that scores a better fitness value and with **loser** the individual losing the (fitness based) comparison. Regarding the mean values $\boldsymbol{\mu}$, the update rule for each of its elements is:

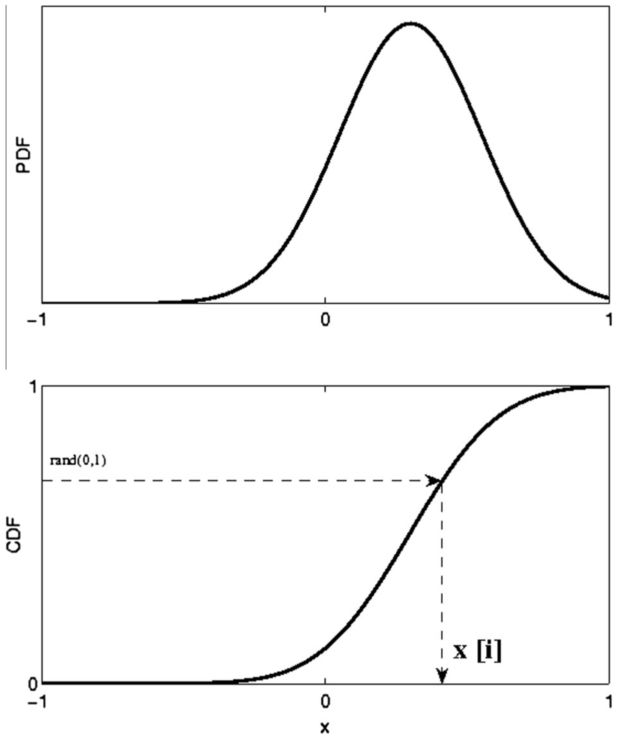

Fig. 1. Sampling mechanism.

$$\mu^{t+1}[i] = \mu^t[i] + \frac{1}{N_p}(winner[i] - loser[i]), \tag{12}$$

where N_p is virtual population size. Regarding σ values, the update rule of each element is given by:

$$(\sigma^{t+1}[i])^2 = (\sigma^t[i])^2 + (\mu^t[i])^2 - (\mu^{t+1}[i])^2 + \frac{1}{N_p}(winner[i]^2 - loser[i]^2). \tag{13}$$

Mathematical details regarding the construction of formulas (12) and (13) are given in [48]. Both persistent and non-persistent structures of rcGA have been tested, see results in [48]. Similar to the binary cGA case, it was impossible to assess whether one or another elitist strategy was preferable. As reported in [48], it is fundamental to remember that the virtual population size N_p is parameter typical of compact algorithms and does not strictly correspond to the population size in a population-based algorithm. The virtual population size, in real-valued compact optimization, is a parameter which biases the convergence speed of the algorithm. In [48] it has been mentioned that, for a given problem (with its dimensionality), this parameter should be set several times bigger than the population size of a corresponding population-based algorithm.

In elitist compact schemes, at each moment of the optimization process, the solution displaying the best performance is retained in a separate memory slot (with memory slot we mean the quantity of memory required to store one solution). This solution is named elite and here simply indicated as **elite**. If a new candidate solution is computed, the fitness based comparison between it and the elite is carried out. If the elite is a winner solution, it biases the **PV** as shown in formulas (12) and (13).

3.2. Design of a compact algorithm: from a general concept to compact encoding of Particle Swarm Optimization

At an abstract level, all the optimization algorithms can be considered as procedures combining operations of two kinds, search operations when one or more solutions are sampled within the search space and selection operations when one or more solutions are selected and retained. This definition is valid regardless of the fact the algorithm can process only a solution or an entire population and is important to understand the design procedure of compact algorithms. It must be remarked that to efficiently perform a proper algorithmic implementation, the population-based algorithm to be encoded as compact must be treated and represented in a special way.

Before entering into the details of compact algorithmic design, let us compare the above mentioned rcGA with the cDE scheme, see [48,49]. If we look at the two algorithms from an intuitive perspective we can conclude that cDE is basically a rcGA where the sampling rule has been slightly complicated. This modification appears to be very beneficial in terms of performance, as shown in [49]. From a deeper perspective, the difference in the performance can be justified by the fact that while compact encoding of DE is naturally performed, compact encoding of GA requires some major modifications of the original (population-based) algorithmic structure. DE employs the so called one-to-one spawning logic, i.e. in DE the survivor selection is applied by performing a pair-wise comparison between the performance of a parent solution and its corresponding offspring, see e.g. [66,59]. This logic is necessarily used in a compact algorithm as a limited amount of memory slots is available. On the contrary, the selection mechanism of GAs requires multiple comparisons of a list of solutions, e.g. in tournament selection. In order to perform the compact encoding of a GA without the need of allocating extra memory slots (which would jeopardize the memory saving logic), it is necessary to simulate a tournament selection having size two, which is unlikely to be a proper choice. This fact may turn into a poorer performance of cGA with respect to its population-based version especially for real-valued problems and when the dimensionality is not very low (e.g. more than 10 dimensions), while cDE tends to perform comparably (if not better) than its corresponding population-based version, see [48,49]. In other words, cDE works better than rcGA because its compact encoding is more “naturally” performed with respect to that of rcGA.

This concept can be considered as the basis for a suggestion on how the design of a compact algorithm should be performed in order to have a memory saving structure which is still capable of detecting good solutions. To do this, let us consider a generic population-based algorithm. At the generic step t, a set of candidate solutions are stored into a memory structure, namely population $\mathbf{Pop^t}$, which contains the current solutions. In order to improve upon the available candidate solutions, an operator U processes the population $\mathbf{Pop^t}$ and returns a new population of (trial) candidate solutions **Trials** after having applied the search logic of the algorithm:

$$\mathbf{Trials} = U(\mathbf{Pop^t}). \tag{14}$$

Subsequently, the selection operator S processes both the populations $\mathbf{Pop^t}$ and **Trials** and returns a new current population for the step $t+1$:

$$\mathbf{Pop^{t+1}} = S(\mathbf{Pop^t}, \mathbf{Trials}). \tag{15}$$

The sequential application of the operations in (14) and (15) can be seen as a general description of a population-based algorithm. For evolutionary algorithms, the above-mentioned formulas are already explicitly formulated as U is the application of variation operators (crossover and mutation) while S is the selection strategy which can be, for example, the selection of parents and generational replacement for classical GAs or the so called “plus” strategy in Evolution Strategies (ES), see [21]. For other algorithms, such as swarm intelligence algorithms, the representation by the operations above is still valid but less explicit than for evolutionary algorithms. As a general rule, we suggest to analyse the algorithms on the basis of the operations that they perform and thus look “beyond the metaphor” that inspired their first implementation.

In order to encode a compact version of a population-based algorithm, it must be considered that $\mathbf{Pop^t}$ is composed of **elite** and virtual population encoded in **PV**. The U operator samples a set of individuals (one by one to use only one memory slot) and generates a new candidate solution. The S operator selects the new **elite** and updates the virtual population. Formulas (14) and (15) can be re-written in the following way for the compact optimization case:

$$\mathbf{trial} = U([\mathbf{elite^t}, \mathbf{PV^t}]) \tag{16}$$

and

$$[\mathbf{elite^{t+1}}, \mathbf{PV^{t+1}}] = S([\mathbf{elite^t}, \mathbf{PV^t}], \mathbf{trial}) \tag{17}$$

where **trial** clearly is a trial vector attempting to outperform the current elite solution.

When the compact encoding is performed an important difficulty arises: the compact scheme imposes that the selection of the new elite solution is made only on the basis of a pair-wise comparison, in other words a tournament selection having size two, see [2]. This issue is an unavoidable limitation descending from the willingness of having a memory saving optimizer and, in some cases, can turn into a worsening of the performance with respect to the population-based case. This happens when the algorithmic structure requires the sorting of the population, e.g. the encoding of a GA employing tournament selection into rcGA, see [48]. Luckily, some population-based algorithms already natively employ the selection on the basis of pair-wise comparisons, as for example DE and PSO. In these cases, the compact encoding should not result into a significant performance worsening, see [49]. In this sense, some population-based algorithms can be straightforwardly and efficiently encoded into a compact scheme while some others are likely not to be prone to compact encoding.

PSO is based on the metaphor that a swarm of birds, while keeping memory of their most successful position, explore the decision space in order to find a better position by means of a search logic which involves other solutions (particles, birds) and a certain degree of randomization. By looking at the same algorithm from a different perspective, PSO is an Evolutionary Algorithm (EA) which processes a population of solutions, the local best individuals, combines them somehow, and applies the so called one-to-one spawning as a survivor selection scheme. In other words, like in DE, a PSO child solution replaces the parent solution which generated it (the new local best replaces the old local best).

More formally, PSO can be visualized as a population-based algorithm where the population $\mathbf{Pop^t}$ in formulas (14) and (15) is not composed of particles but of the best positions $\mathbf{x_{lb-k}}$. The role of the operator U is played by formulas (1) and (2) which perform a perturbation of each generic $\mathbf{x_{lb-k}}$ with the aim of detecting a solution with a higher performance. In PSO, if a new best position (with better performance) is detected, it replaces the old one. In other words, PSO natively employs single pair-wise comparison and replacement. Thus, unlike GA and similar to DE, the compact encoding of PSO is in this sense straightforward. In addition, the concept of elite is already contained in PSO as global best. This fact makes PSO extremely prone to a compact encoding as such encoding would not jeopardize the functioning principles of the algorithm. Considering that $\mathbf{elite^t}$ is $\mathbf{x^t_{gb}}$ and indicating with $\mathbf{PV^t_{lb}}$ a probabilistic representation of the population of local best positions, the compact encoding of PSO is obtained by:

$$\mathbf{trial} = U\left(\left[\mathbf{x^t_{gb}}, \mathbf{PV^t_{lb}}\right]\right) \tag{18}$$

and

$$\left[\mathbf{x^{t+1}_{gb}}, \mathbf{PV^{t+1}_{lb}}\right] = S\left(\left[\mathbf{x^t_{gb}}, \mathbf{PV^t_{lb}}\right], \mathbf{trial}\right). \tag{19}$$

3.3. Compact Particle Swarm Optimization: implementation details

The proposed cPSO employs a perturbation vector $\mathbf{PV^t_{lb}} = [\boldsymbol{\mu}^t, \boldsymbol{\sigma}^t]$ with the same structure as that shown in formula (10). At the beginning of the optimization process, by following the procedure described in [48], the **PV** initialization ($\forall i, \mu^1[i] = 0$ and $\sigma^1[i] = \lambda$, where $\lambda = 10$) is performed and each design variable is mapped to the continuous interval $[-1,1]$. In addition, position and velocity vectors, $\mathbf{x}$ and $\mathbf{v}$, are randomly initialized with components respectively in the normalized variable bounds and in the normalized velocity range $[0,1]$. Then, a new candidate solution $\mathbf{x_{lb}}$ is sampled from $\mathbf{PV^t_{lb}}$ by the procedure reported in formula (11), and represented in Fig. 1.

Velocity and position vectors are updated by means of slightly revisited PSO formulas:

$$\mathbf{v^{t+1}} = \phi_1\mathbf{v^t} + \phi_2\mathbf{U_1}\left(\mathbf{x^t_{lb}} - \mathbf{x^t}\right) + \phi_3\mathbf{U_2}(\mathbf{x_{gb}} - \mathbf{x^t}) \tag{20}$$
$$\mathbf{x^{t+1}} = \gamma_1\mathbf{x^t} + \gamma_2\mathbf{v^{t+1}} \tag{21}$$

It can be observed that velocity and position update formulas in (20) and (21) have the same structure of the PSO update formulas in (1) and (2). However, while in the population based PSO the k index appears to indicate the particle under examination, in its compact version, no index k appears. This fact can be seen because the algorithm works on a single particle x which is progressively perturbed by using one solution saved in memory ($\mathbf{x_{gb}}$) in addition to one sampled each time from a distribution ($\mathbf{x_{lb}}$).

The fitness value of the position $\mathbf{x}^{t+1}$ is calculated and compared with both $\mathbf{x}_{gb}$ and $\mathbf{x}_{lb}$. The competition with $\mathbf{x}_{lb}$ determines a **winner** and a **loser**. Formulas (12) and (13) are then applied to update the probability vector $\mathbf{PV}_{lb}^{t+1}$. If $f(\mathbf{x}^{t+1}) \leqslant f(\mathbf{x}_{gb})$, the value of the global best is then updated: $\mathbf{x}_{gb} = \mathbf{x}^{t+1}$ and the process is repeated over for the subsequent steps. Current values for $\mathbf{x}^{t+1}$ and $\mathbf{v}^{t+1}$ are retained for subsequent algorithm steps. For the sake of clarity, Algorithm 1 shows the working principles of cPSO.

Algorithm 1. cPSO pseudo-code

```
counter $t = 0$
for $i = 1:n$ do
    // $\mathbf{PV}_{lb}$ initialization//
    initialize $\mu[i] = 0$
    initialize $\sigma[i] = \lambda = 10$
end for
// Global Best Initialization// generate $\mathbf{x}_{gb}$ by means of $\mathbf{PV}_{lb}$
randomly generate $\mathbf{x}^t$ and $\mathbf{v}^t$
while termination condition is not satisfied do
    // New local best initialization//
    $\mathbf{x}_{lb}{}^{t}$ = generateFrom ($\mathbf{PV}_{lb}$)
    // Update position and velocity//
    $\mathbf{v}^{t+1} = \phi_1\mathbf{v}^t + \phi_2\mathbf{U_1}(\mathbf{x}_{lb}^t - \mathbf{x}^t) + \phi_3\mathbf{U_2}(\mathbf{x}_{gb} - \mathbf{x}^t)$
    $\mathbf{x}^{t+1} = \gamma_1\mathbf{x}^t + \gamma_2\mathbf{v}^{t+1}$
    // Local Best Selection//
    [winner,loser] = compete($\mathbf{x}^{t+1}$,$\mathbf{x}_{lb}$);
    // $\mathbf{PV}_{lb}$ Update//
    for $i = 1:n$ do
        $\mu^{t+1}[i] = \mu^t[i] + \frac{1}{N_p}(winner[i] - loser[i]);$
        $\sigma^{t+1}[i] = \sqrt{(\sigma^t[i])^2 + (\mu^t[i])^2 - (\mu^{t+1}[i])^2 + \frac{1}{N_p}(winner^2[i] - loser^2[i])};$
    end for
    // Global best update//
    [winner,loser] = compete($\mathbf{x}^{t+1}$,$\mathbf{x}_{gb}$);
    $\mathbf{x}_{gb}$ = winner;
    $t = t + 1$;
end while
```

It is important to remark that the compact encoding can be easily extended to many PSO variants/enhancements, for example the constricted PSO in formula (3) or the FIPS in formula (5).

4. Numerical results

The proposed cPSO has been tested on the 47 test problems listed in Appendix. For each problem, its features are specified. It must be observed that a set of various problems, in terms of dimensionality and problem features, have been selected. In addition, the set of test functions entirely contains the testbed from IEEE CEC 2008.

With reference to formulas (20) and (21) and Algorithm 1, the proposed cPSO has been run with the following parameters: $N_p = 300$, $\phi_1 = -0.2$, $\phi_2 = -0.07$, $\phi_3 = 3.74$, $\gamma_1 = 1$, and $\gamma_2 = 1$. These parameter values were selected by taking into considerations the indications given in [63] and then further tuning the parameters for the specific framework. More specifically, in order to tune cPSO, we considered a population-based PSO. As an initial guess we used the results of the meta-optimization reported in [63] and we assigned the best results reported for the 50 dimensional case. This choice has been made as a compromise among the dimensionality values of our benchmark. Then we performed a fine tuning meta-optimization for the parameters number of particles, ϕ_1, ϕ_2, and ϕ_3. This meta-optimization has been carried out letting the ϕ parameters vary within the interval $[-5,5]$ and N_p vary in the interval [20,500]. The implemented meta-optimization is a stochastic search having a randomized radius between 0.01 and 0.1 for the ϕ parameters and between 1 and 5 for N_p.

The performance of the set of parameters was checked by running PSO over five test problems: Sphere (f_1), Rosenbrock (f_3), Ackley (f_4), Rastrigin (f_6), and Schwefel (f_8). For each test problem five runs were performed. The Holm–Bonferroni procedure (see subSection 4.5) was then performed to establish whether the current best or the newly perturbed set of parameters leads to a better performance. When the Holm–Bonferroni procedure cannot establish a clear out-performance of one parameter setting, the result of the Wilcoxon Rank-Sum test [92] is considered for each of the five test problems. The setting with the highest amount of statistically significant victories is then selected for the following perturbation. If also the

Wilcoxon Rank-Sum test did not allow to clearly asses which parameter setting was preferable, a simple average based comparison, for each test problem, was carried out. We performed this meta-optimization procedure for 5000 parameter settings before selecting the ϕ values mentioned above. In addition, the number of particle suggested by the optimization procedure was 60. Then for fixed values of number of particles as well as ϕ_1, ϕ_2, and ϕ_3 we perturbed γ_1 and γ_2 with same procedure described above. As a result, we could not improve the performance by varying the γ values which have been set at their initial value 1.

When the parameter tuning was performed for PSO, we used the same parameters for the cPSO. Obviously, cPSO does not have a number of particles but has the virtual population size N_p. In order to set N_p we used the indication given in [48] (it should be set equal 2–4 time the population of the corresponding population-based algorithm) and performed, over the cPSO, a further meta-optimization. While using the same criteria above to evaluate the setting of an algorithm, we tested the cPSO performance for N_p equal to 40, 60, 80, 100, 150, 200, 300, and 400, respectively. On the basis of this test N_p was set equal to 300.

The cPSO algorithm has been compared with the following 11 algorithms:

- real coded Genetic Algorithm (rcGA) with persistent elitism proposed in [48] with $N_p = 300$, see original paper.
- compact Differential Evolution (cDE) with persistent elitism proposed in [49] with $N_p = 300$, $F = 0.5$, and $Cr = 0.3$, see original paper.
- Intelligence Single Particle Optimization (ISPO) proposed in [104] with $A = 1$, $P = 10$, $B = 2$, $S_f = 4$, $\epsilon = 1e{-}5$, and $H = 30$, see formula (9) and algorithmic description.
- J Adaptive Differential Evolution (JADE) proposed in [100] with $c = 0.1$ and $p = 0.05$, see original paper.
- Self-Adaptive Differential Evolution (SADE) proposed in [69] with $LP = 30$.
- Comprehensive Learning Particle Swarm Optimizer (CLPSO) proposed in [43].
- The original Particle Swarm Optimization with exactly the same parameter setting of the cPSO algorithm except the number of particles that is 60.
- The original Differential Evolution (DE/rand/1/bin) with exactly the same parameter setting of the cDE algorithms except the population size that is 60.
- The Covariance Matrix Adaptation Evolution Strategy (CMAES) proposed in [6].
- The Frankenstein PSO (FPSO) proposed in [51] with $k = 2000$, $w - Schedule = 5000$, $InitialI - w = 0.969$, $FinalI - w = 0.175$, and $velocityLimit = 0.75$.
- The Population-based Algorithm Portfolio (PAP) proposed in [64] with $\mu = 3$, $\lambda = 2$, $particleNumber = 10$, $migrSize = 1$, and $migrationInterval = 10$.

The total population size (number of particles) for all the population-based algorithms has been fixed to 60 individuals. In order to perform a fair comparison, for each competing algorithm, we considered the indication for parameter setting given in the original paper and subsequently we performed a parameter tuning over the problems under consideration in order to obtain the best performance. All the code related to this experimental section has been implemented in Java. The random number generator used for these experiments is the standard Java 48-bit linear congruential generator.

Table 1 summarizes the main algorithmic features of the algorithms used for comparison and gives an emphasis on the memory requirement for each algorithm. It can easily be observed that rcGA, cDE, ISPO and cPSO can be classified as memory-saving algorithms. The PSO algorithm with the same parameter setting of cPSO has been included in order to have a direct comparison between compact and population based versions. The DE has been included as a reference algorithm to study the variation in the performance when the compact encoding is performed, i.e. relationship between DE and cDE performance as well as PSO and cPSO performance. The remaining six algorithms are complex modern algorithms representing the state of the art in computational intelligence optimization.

In order to highlight the role of cPSO with respect to the other algorithms in this study, numerical results have been grouped into four categories: (1) comparison among memory saving algorithms, i.e. rcGA, cDE, ISPO, cPSO; (2) comparison of cPSO against algorithms based on a PSO framework, i.e. PSO, CLPSO, FPSO; (3) comparison with classical algorithms with a good performance, i.e. DE and CMAES; (4) comparison of cPSO against non PSO based state-of-the-art algorithms, i.e. JADE, SADE, PAP. Each algorithm has been executed for 30 independent runs. Each run has been continued for $5000 \times n$ fitness evaluations for each problem listed in Appendix 6. Tables 2, 3, and 5 show the results for each group, respectively. Each Table shows the average of the final results detected by each algorithm ± the corresponding standard deviation values calculated over the 30 runs. The best results are highlighted in bold face. In order to strengthen the statistical significance of the results, the Wilcoxon Rank-Sum test has also been applied according to the description given in [92], where the confidence level has been fixed at 0.95. A "+" indicates the case in which cPSO statistically outperforms the algorithm labelled on the top of the column for the corresponding test problem; a "=" indicates that a pairwise comparison leads to success of the Wilcoxon Rank-Sum test, i.e., the two algorithms have the same performance; a "−' ' indicates that cPSO is outperformed.

4.1. Comparison with memory-saving algorithms

The comparison among "lightweight" (memory-saving) algorithms show that cPSO displays a good performance. More specifically, it is clearly demonstrated that cPSO outperforms, on average, rcGA. It can be clearly seen that rcGA displays a

Table 1
Description of the algorithms contained in this study with respective memory employments.

Algorithm	Components	Memory slots
rcGA	compact GA based structure persistent elitism 1 sampling	4
cDE	compact DE based structure 3 samplings, crossover bin	4
ISPO	single particle optimization learning period for velocity	2
JADE	DE structure rand-to-best/1/bin Cr sampled from a normal distribution F sampled from a Cauchy distribution	$N_p + 1 + archive$
SADE	DE structure multiple mutation strategies self adaptive parameters	$N_p + 1 + learning$
CLPSO	PSO structure comprehensive learning strategy	$2N_p$
PSO	PSO structure	$2N_p$
DE	DE/rand/1/bin structure	N_p
CMA-ES	population-based ES structure covariance matrix	$N_p + n^2$
FPSO	PSO structure adaptive inertia weight reduced neighborhood velocity restriction	$2N_p + neighborhood$
PAP	structured population SaNSDE, wPSO, G3PCX, and CMA-ES migration of bests	$N_{SaNSDE} + 2N_{wPSO} +$ $N_{G3PCX} + N_{CMA-ES} + n$
cPSO	compact PSO based structure 1 sampling	5

good performance for low dimensional problems ($n = 10$) while it is not competitive with the other algorithms in problems characterized by a higher dimensionality. The comparison with cDE shows that cPSO has a respectable performance in many cases. cPSO outperforms cDE in 24 cases while it is outperformed for 21 test problems. This result can be put into relationship with the performance of population-based versions of PSO and DE. For some problems DE outperforms PSO and for other problems PSO displays a better performance. Thus, it is impossible to establish the superiority of one of the two algorithmic schemes. In the same way, for some problems cPSO employment is preferable to cDE and for other problems the converse situation occurs. In particular, cPSO is not so competitive with cDE for low dimensional problems while seems to often outperform cDE for $n = 50$ and $n = 100$, see the second column in Table 2. The search logic of cDE seems to be more efficient than that of cPSO in the presence of plateaus (f_{20} and f_{26}) and slightly more efficient in the presence of highly multi-modal problems with a strong basing of attraction in the center (e.g. f_{35} and f_{37}). On the other hand cPSO seems to be more efficient than cDE in other cases, when multiple strong basins of attraction are present in the decision space (e.g f_{25} and f_{31}). The comparison with ISPO also shows that cPSO displays, on average, a better performance (in Table 2 24+ and 17-appear). The performance of ISPO is good especially for separable problems. According to our interpretation, the good performance of ISPO for separable problems is due to the fact that ISPO explores the decision space by perturbing the variables one-by-one. For the sake of clarity, some examples of average performance trends are displayed in Figs. 2 and 3, respectively.

4.2. Comparison with PSO-based algorithms

Numerical results in Table 3 show that cPSO outperforms, on a regular basis, its population-based version. In our opinion, this fact is interesting indeed because cPSO employs the same search logic and parameter setting of PSO while it employs less memory resources. This result should be put into relationship with the comparison between DE and cDE presented in [49]. It was shown that cDE, given the parameters and the chosen mutation and crossover operators, outperforms its population-based version. As a conjecture, it was stated that a certain degree of randomization, given by the probabilistic model, is beneficial to the DE structure, see also [59]. An analogous conjecture could be done for PSO. The presence of a randomization within the terms ϕ_1, ϕ_2, and ϕ_3 in formula (1) seems to be a crucial point to guarantee a high performance. According to our interpretation, the randomization contained in the samples solution $\mathbf{x}^t_{lb}$ in formula (20) allows the detection of alternative search directions, thus enhancing upon PSO performance.

The cPSO algorithm, although a way less demanding in terms of memory employment, displays a comparable performance with respect to FPSO. In this case, it is hard to track a clear pattern between performance and problem features. It can be observed that cPSO outperforms and is outperformed by FPSO for both low and high dimensional problems, separable and non-separable problems, unimodal and multi-modal problems. The comparison between cPSO and CLPSO shows that

Table 2
Comparison among memory-saving algorithms.

Test problem	rcGA	W	cDE	W	ISPO	W	cPSO
f_{1}	1.426e+04 ± 9.26e+03	+	8.731e−29 ± 1.87e−28	−	**8.438e−31 ± 3.30e−31**	−	6.471e+01 ± 2.28e+01
f_{2}	2.825e+04 ± 6.57e+03	+	3.779e+03 ± 1.84e+03	+	**1.183e+01 ± 5.90e+00**	−	2.560e+03 ± 2.36e+03
f_{3}	1.281e+09 ± 1.59e+09	+	**1.291e+02 ± 1.83e+02**	−	2.026e+02 ± 3.29e+02	−	1.320e+05 ± 7.46e+04
f_{4}	1.874e+01 ± 3.58e−01	+	**8.694e−02 ± 2.96e−01**	−	1.942e+01 ± 1.55e−01	+	3.728e+00 ± 3.71e−01
f_{5}	6.343e−03 ± 1.30e−02	+	4.288e−03 ± 1.37e−02	+	1.123e+01 ± 1.75e+01	+	**9.636e−08 ± 3.07e−08**
f_{6}	1.962e+02 ± 2.84e+01	+	7.943e+01 ± 1.49e+01	+	2.547e+02 ± 4.22e+01	+	**2.940e+01 ± 7.94e+00**
f_{7}	2.311e+03 ± 2.46e+03	+	4.982e+03 ± 3.79e+03	+	2.253e+03 ± 8.61e+02	+	**4.614e+02 ± 2.40e+02**
f_{8}	3.193e+03 ± 8.00e+02	=	**1.672e+03 ± 4.49e+02**	−	5.767e+03 ± 5.37e+02	+	3.160e+03 ± 9.75e+02
f_{9}	1.009e+04 ± 2.36e+03	−	**8.549e+03 ± 2.13e+03**	−	2.754e+04 ± 6.09e+03	+	1.344e+04 ± 1.74e+03
f_{10}	3.696e+05 ± 1.79e+05	−	4.264e+04 ± 2.34e+04	−	**4.327e+03 ± 4.53e+03**	−	1.040e+06 ± 1.16e+05
f_{11}	1.850e+01 ± 4.36e−01	+	**1.707e+00 ± 1.10e+00**	−	1.949e+01 ± 1.88e−01	+	3.699e+00 ± 3.53e−01
f_{12}	5.768e−02 ± 1.04e−01	+	2.394e−01 ± 2.02e−01	+	**0.000e+00 ± 0.00e+00**	−	9.567e−08 ± 2.69e−08
f_{13}	2.153e+02 ± 3.95e+01	+	1.313e+02 ± 1.86e+01	+	2.565e+02 ± 4.14e+01	+	**3.924e+01 ± 2.31e+01**
f_{14}	3.245e+01 ± 4.52e+00	−	**2.989e+01 ± 3.48e+00**	−	4.776e+01 ± 4.33e+00	+	3.942e+01 ± 1.15e+00
f_{15}	5.251e+00 ± 5.19e+00	=	2.313e−16 ± 5.65e−16	−	**1.183e−16 ± 2.89e−17**	−	1.777e+00 ± 4.27e−01
f_{16}	**−1.000e+02 ± 4.42e−09**	=	**−1.000e+02 ± 1.63e−09**	=	**−1.000e+02 ± 8.38e−15**	=	**−1.000e+02 ± 8.45e−05**
f_{17}	1.450e+00 ± 1.87e+00	=	**2.811e−23 ± 3.51e−23**	−	9.993e−01 ± 1.55e+00	−	1.702e+00 ± 7.09e−01
f_{18}	−5.484e−01 ± 1.10e+00	+	**−1.150e+00 ± 4.99e−16**	−	−2.259e−01 ± 1.27e+00	+	−1.030e+00 ± 7.55e−01
f_{19}	4.337e+02 ± 4.74e+01	+	2.602e+02 ± 3.03e+01	+	4.043e+02 ± 4.14e+01	+	**4.403e+01 ± 3.44e+01**
f_{20}	−1.516e+01 ± 2.75e+00	+	−3.346e+01 ± 1.86e+00	−	**−3.348e+01 ± 1.63e+00**	−	−2.063e+01 ± 2.33e+00
f_{21}	8.371e+03 ± 1.61e+03	+	5.342e+03 ± 8.46e+02	+	9.678e+03 ± 1.08e+03	+	**4.784e+03 ± 1.09e+03**
f_{22}	2.013e+01 ± 1.47e−01	+	1.786e+01 ± 2.88e−01	+	1.950e+01 ± 7.50e−02	+	**3.899e−01 ± 5.18e−01**
f_{23}	1.646e+02 ± 2.35e+01	+	4.041e+01 ± 1.40e+01	+	**1.246e−13 ± 1.00e−14**	−	4.657e−02 ± 2.39e−02
f_{24}	8.487e+04 ± 8.13e+03	+	2.942e+03 ± 1.58e+03	+	**1.251e−30 ± 3.08e−31**	−	6.918e−02 ± 2.54e−02
f_{25}	−6.348e−03 ± 3.23e−04	+	−9.162e−03 ± 6.26e−04	+	−4.552e−03 ± 3.78e−04	+	**−7.858e−01 ± 1.60e−14**
f_{26}	−2.177e+01 ± 3.08e+00	+	−4.937e+01 ± 3.53e+00	−	**−6.557e+01 ± 3.19e+00**	−	−2.920e+01 ± 2.53e+00
f_{27}	2.523e+05 ± 2.59e+04	+	1.050e+04 ± 6.30e+03	+	**3.497e−30 ± 8.74e−31**	−	2.127e−02 ± 4.04e−03
f_{28}	1.165e+03 ± 7.34e+01	+	4.219e+02 ± 3.71e+01	+	7.941e+02 ± 7.68e+01	+	**8.776e−03 ± 2.87e−03**
f_{29}	6.905e+10 ± 1.37e+10	+	5.642e+08 ± 4.99e+08	+	3.502e+02 ± 3.90e+02	=	**1.220e+02 ± 2.81e+01**
f_{30}	1.296e+11 ± 2.62e+10	+	7.065e+10 ± 1.18e+10	+	9.701e+09 ± 3.25e+09	+	**4.928e+06 ± 6.56e+05**
f_{31}	2.149e+04 ± 2.50e+03	+	1.841e+04 ± 1.28e+03	+	1.970e+04 ± 1.29e+03	+	**1.045e+04 ± 2.94e+03**
f_{32}	1.590e+03 ± 1.26e+03	+	1.061e−05 ± 9.77e−06	−	**2.685e−30 ± 4.74e−31**	−	1.531e−02 ± 3.80e−03
f_{33}	1.257e+02 ± 6.45e+00	+	8.949e+01 ± 6.17e+00	+	1.772e+02 ± 5.90e+00	+	**7.370e+01 ± 3.32e+00**
f_{34}	5.332e+10 ± 3.50e+10	+	8.040e+09 ± 4.89e+09	+	**2.475e+02 ± 2.12e+03**	−	4.896e+05 ± 2.21e+05
f_{35}	9.383e+02 ± 1.79e+02	+	**5.577e+02 ± 8.52e+01**	−	1.611e+03 ± 2.52e+02	+	6.701e+02 ± 6.36e+01
f_{36}	7.461e+02 ± 2.33e+02	+	2.421e+02 ± 8.71e+01	+	**−1.272e+02 ± 3.76e+00**	−	−1.082e+02 ± 4.21e+00
f_{37}	5.507e+02 ± 1.82e−01	+	**5.479e+02 ± 9.64e−01**	−	5.499e+02 ± 4.63e−02	+	5.492e+02 ± 2.50e−01
f_{38}	−1.200e+03 ± 4.76e+01	+	**−1.406e+03 ± 3.22e+01**	−	−1.266e+03 ± 5.16e+01	=	−1.284e+03 ± 3.90e+01
f_{39}	6.156e+04 ± 1.53e+04	+	4.987e−27 ± 4.21e−27	−	**1.444e−30 ± 5.57e−31**	−	4.314e−03 ± 1.24e−03
f_{40}	7.517e+04 ± 1.07e+04	+	3.135e+04 ± 8.11e+03	+	5.664e+02 ± 2.18e+02	+	**4.375e+00 ± 9.83e−01**
f_{41}	1.043e+10 ± 4.33e+09	+	1.097e+03 ± 1.85e+03	+	2.574e+02 ± 3.10e+02	=	**8.941e+01 ± 5.26e+01**
f_{42}	1.948e+01 ± 2.58e−01	+	8.004e+00 ± 4.30e+00	+	1.948e+01 ± 1.49e−01	+	**1.277e+00 ± 3.68e−01**
f_{43}	2.979e−01 ± 3.72e−01	−	**1.353e−01 ± 2.30e−01**	−	6.858e+00 ± 1.05e+01	=	1.084e+00 ± 3.16e−01
f_{44}	4.706e−03 ± 7.39e−03	+	**0.000e+00 ± 0.00e+00**	=	**0.000e+00 ± 0.00e+00**	=	**0.000e+00 ± 0.00e+00**
f_{45}	4.257e+04 ± 4.14e+04	+	2.533e+04 ± 6.29e+03	+	4.065e+03 ± 9.65e+02	+	**5.051e+01 ± 4.27e+01**
f_{46}	2.369e+04 ± 3.44e+03	=	**2.000e+04 ± 3.03e+03**	−	3.775e+04 ± 6.48e+03	+	2.320e+04 ± 3.88e+03
f_{47}	2.086e+06 ± 7.96e+05	+	4.587e+05 ± 1.68e+05	−	**1.588e+04 ± 1.73e+04**	−	1.359e+06 ± 1.14e+06

CLPSO has a better performance for most of the problems. CLPSO makes use of the population diversity amongst actual solutions and cannot be easily encoded in a compact way. In this case, population-based algorithms seem to have an advantage in terms of performance and should be used when the hardware allows their implementation.

4.3. Comparison with classical optimization algorithms

Results in Table 4 show that cPSO outperforms, on a regular basis, DE/rand/1/bin. The comparison with CMA-ES shows that although outperformed, still cPSO displays a respectable performance. The cPSO algorithm is outperformed in roughly half of the cases. For the other test problems, either cPSO outperforms CMA-ES or displays a comparable performance. In conclusion, considering that cPSO memory requirement is 5 memory slots regardless of the problem dimensionality, while for CMA-ES the memory requirement grows quadratically with the dimensionality, the cPSO results appear to be satisfactory.

4.4. Comparison with non-PSO based state-of-the-art algorithms

The comparison of cPSO with the-state-of-the-art optimization algorithms shows (Table 5) that it is generally outperformed by the other algorithms considered in this study. On the other hand, considering that, as shown in Table 1, cPSO

Table 3
Comparison among PSO based algorithms.

Test problem	CLPSO	W	PSO	W	FPSO	W	cPSO
f_1	**6.671e−19 ± 5.83e−19**	–	1.252e+04 ± 5.30e+03	+	1.048e−06 ± 5.08e−06	–	6.471e+01 ± 2.28e+01
f_2	1.077e+03 ± 3.04e+02	=	4.231e+04 ± 6.97e+03	+	**8.029e+02** ± *3.98e+02*	–	2.560e+03 ± 2.36e+03
f_3	**6.951e+00 ± 1.79e+00**	–	1.102e+09 ± 5.07e+08	+	5.109e+02 ± 1.30e+03	–	1.320e+05 ± 7.46e+04
f_4	**1.637e−10 ± 7.95e−11**	–	1.638e+01 ± 1.21e+00	+	1.873e−05 ± 5.32e−05	–	3.728e+00 ± 3.71e−01
f_5	5.328e−16 ± 9.78e−16	–	**0.000e+00 ± 0.00e+00**	–	8.891e−06 ± 3.02e−05	+	9.636e−08 ± 3.07e−08
f_6	**2.808e−10 ± 3.11e−10**	–	2.886e+02 ± 3.27e+01	+	1.262e+01 ± 4.41e+00	–	2.940e+01 ± 7.94e+00
f_7	**1.598e−10 ± 1.27e−10**	–	1.320e+05 ± 1.02e+04	+	1.773e+01 ± 5.30e+00	–	4.614e+02 ± 2.40e+02
f_8	**3.818e−04 ± 8.42e−13**	–	6.676e+03 ± 6.43e+02	+	5.687e+03 ± 1.70e+03	+	3.160e+03 ± 9.75e+02
f_9	4.024e+03 ± 0.00e+00	=	1.304e+04 ± 3.16e+03	=	**2.740e+03 ± 7.44e+02**	–	1.344e+04 ± 1.74e+03
f_{10}	**6.292e+04 ± 2.57e+04**	–	9.715e+05 ± 1.56e+05	=	1.123e+06 ± 1.13e+05	+	1.040e+06 ± 1.16e+05
f_{11}	2.138e−02 ± 2.63e−02	–	1.706e+01 ± 1.72e+00	+	**4.995e−05 ± 1.21e−04**	–	3.699e+00 ± 3.53e−01
f_{12}	5.131e−06 ± 9.30e−06	+	1.138e+01 ± 3.07e+01	+	4.237e−04 ± 1.40e−03	+	**9.567e−08 ± 2.69e−08**
f_{13}	1.091e+02 ± 1.41e+01	+	3.154e+02 ± 2.18e+01	+	**1.184e+01 ± 3.10e+00**	–	3.924e+01 ± 2.31e+01
f_{14}	**3.929e+01 ± 1.24e+00**	=	3.965e+01 ± 1.17e+00	=	3.959e+01 ± 1.52e+00	=	3.942e+01 ± 1.15e+00
f_{15}	**1.539e−12 ± 8.72e−13**	–	4.081e+00 ± 2.22e+00	+	**1.495e−26 ± 2.10e−26**	–	1.777e+00 ± 4.27e−01
f_{16}	−8.635e+01 ± 1.87e+00	+	**−1.000e+02 ± 0.00e+00**	–	−5.617e+01 ± 2.43e+00	+	**−1.000e+02 ± 8.45e−05**
f_{17}	1.508e−21 ± 7.55e−22	–	1.045e+01 ± 5.09e+00	+	**4.712e−32 ± 5.59e−48**	–	1.702e+00 ± 7.09e−01
f_{18}	**−1.150e+00 ± 5.14e−16**	–	3.501e+03 ± 9.84e+03	+	**−1.150e+00 ± 6.80e−16**	–	−1.030e+00 ± 7.55e−01
f_{19}	2.796e+02 ± 1.69e+01	+	6.106e+02 ± 3.42e+01	+	**2.789e+01 ± 9.61e+00**	–	4.403e+01 ± 3.44e+01
f_{20}	**−4.453e+01 ± 5.82e−01**	–	−1.935e+01 ± 1.71e+00	+	−1.842e+01 ± 5.27e+00	=	−2.063e+01 ± 2.33e+00
f_{21}	**6.364e−04 ± 1.65e−12**	–	9.690e+03 ± 1.13e+03	+	1.244e+04 ± 2.49e+03	+	4.784e+03 ± 1.09e+03
f_{22}	**1.207e−10 ± 1.91e−11**	–	2.003e+01 ± 3.74e−01	+	2.390e+00 ± 4.59e−01	+	3.899e−01 ± 5.18e−01
f_{23}	**1.587e−03 ± 4.59e−04**	–	1.814e+02 ± 1.00e+01	+	7.412e−01 ± 5.97e−01	+	4.657e−02 ± 2.39e−02
f_{24}	**9.028e−20 ± 1.57e−20**	–	6.500e+04 ± 9.69e+03	+	2.139e+01 ± 1.50e+01	+	6.918e−02 ± 2.54e−02
f_{25}	−3.691e−01 ± 0.00e+00	=	−7.492e−03 ± 1.05e−03	+	−2.648e−01 ± 4.74e−02	+	**−7.858e−01 ± 1.60e−14**
f_{26}	**−8.104e+01 ± 1.06e+00**	–	−2.676e+01 ± 2.00e+00	+	−2.128e+01 ± 5.68e+00	+	−2.920e+01 ± 2.53e+00
f_{27}	**4.578e−19 ± 2.74e−19**	–	1.924e+05 ± 1.92e+04	+	5.020e+01 ± 4.44e+01	+	2.127e−02 ± 4.04e−03
f_{28}	**3.939e−07 ± 2.45e−07**	–	1.278e+03 ± 4.44e+01	+	6.190e+01 ± 1.17e+01	+	8.776e−03 ± 2.87e−03
f_{29}	**1.058e+02 ± 5.49e+01**	=	3.853e+10 ± 1.41e+10	+	2.495e+02 ± 1.91e+02	+	1.220e+02 ± 2.81e+01
f_{30}	2.737e+10 ± 2.49e+09	+	1.016e+11 ± 1.95e+10	+	2.551e+09 ± 6.26e+08	+	**4.928e+06 ± 6.56e+05**
f_{31}	**9.870e+01 ± 4.84e+01**	–	2.369e+04 ± 1.88e+03	+	3.505e+04 ± 2.51e+02	+	1.045e+04 ± 2.94e+03
f_{32}	**3.961e−19 ± 1.67e−19**	–	1.139e+04 ± 1.67e+03	+	7.895e−02 ± 7.41e−02	+	1.531e−02 ± 3.80e−03
f_{33}	**3.582e+01 ± 3.53e+00**	–	1.406e+02 ± 1.29e+01	+	3.709e+01 ± 4.47e+00	–	7.370e+01 ± 3.32e+00
f_{34}	**−7.318e+02 ± 6.40e+01**	–	7.764e+10 ± 2.16e+10	+	7.676e+04 ± 8.90e+04	–	4.896e+05 ± 2.21e+05
f_{35}	**−1.200e+02 ± 2.16e−09**	–	1.054e+03 ± 1.47e+02	+	−5.586e+00 ± 1.68e+01	–	6.701e+02 ± 6.36e+01
f_{36}	**−1.300e+02 ± 2.48e−13**	–	1.241e+03 ± 2.44e+02	+	−1.285e+02 ± 5.74e−01	–	−1.082e+02 ± 4.21e+00
f_{37}	**5.300e+02 ± 5.31e−10**	–	5.507e+02 ± 1.70e−01	+	5.326e+02 ± 5.75e−01	–	5.492e+02 ± 2.50e−01
f_{38}	**−1.621e+03 ± 9.24e+00**	–	−1.282e+03 ± 4.68e+01	=	−1.268e+03 ± 6.73e+01	=	−1.284e+03 ± 3.90e+01
f_{39}	**1.390e−18 ± 7.48e−19**	–	1.200e+03 ± 2.17e+02	+	4.782e−07 ± 1.00e−06	–	4.314e−03 ± 1.24e−03
f_{40}	7.118e+03 ± 0.00e+00	=	1.700e+04 ± 3.07e+03	+	9.090e+02 ± 3.72e+02	+	**4.375e+00 ± 9.83e−01**
f_{41}	**2.201e+01 ± 7.12e+00**	–	1.783e+07 ± 5.51e+06	+	8.578e+01 ± 6.92e+01	=	8.941e+01 ± 5.26e+01
f_{42}	**3.521e−10 ± 7.86e−11**	–	6.877e+00 ± 4.72e−01	+	7.044e−05 ± 1.48e−04	–	1.277e+00 ± 3.68e−01
f_{43}	**3.671e−09 ± 4.51e−09**	–	2.554e+02 ± 4.97e+01	+	2.783e+02 ± 9.58e+00	+	1.084e+00 ± 3.16e−01
f_{44}	4.706e−03 ± 7.39e−03	+	**0.000e+00 ± 0.00e+00**	=	**0.000e+00 ± 0.00e+00**	=	**0.000e+00 ± 0.00e+00**
f_{45}	**7.795e−07 ± 5.16e−07**	–	1.133e+06 ± 2.26e+05	+	1.568e+06 ± 1.01e+05	+	5.051e+01 ± 4.27e+01
f_{46}	**1.033e+04 ± 8.64e+02**	–	1.887e+04 ± 2.16e+03	–	1.665e+04 ± 8.73e+02	–	2.320e+04 ± 3.88e+03
f_{47}	**3.594e+05 ± 5.80e+04**	–	2.182e+06 ± 4.00e+05	+	5.573e+06 ± 4.20e+05	+	1.359e+06 ± 1.14e+06

has a very modest memory requirement with respect to the other algorithms and, from a programming point of view, it is much simpler even in terms of lines of code, as the latter require a population and other structures (e.g. archive), cPSO displays still a respectable performance. For example, cPSO succeeds at outperforming the PAP in about one fourth of the cases under study.

4.5. Ranking by means of Holm–Bonferroni procedure

In order to draw some statistically significant conclusions regarding the performance of the cPSO algorithm, the Holm–Bonferroni procedure, see [31,25], for the 12 algorithms under study and the 47 problems under consideration has been performed. The Holm–Bonferroni procedure consists of the following. Considering the results in the tables above, the 10 algorithms under analysis have been ranked on the basis of their average performance calculated over the 38 test problems. More specifically, a score R_i for $i = 1,\ldots,N_A$ (where N_A is the number of algorithms under analysis, $N_A = 12$ in our case) has been assigned. The score has been assigned in the following way: for each problem, a score of 12 is assigned to the algorithm displaying the best performance, 11 is assigned to the second best, 10 to the third and so on. The algorithm displaying the worst performance scores 1. For each algorithm, the scores obtained on each problem are summed up averaged over the amount of test problems (47 in our case). On the basis of these scores the algorithms are sorted (ranked). With the calculated R_i values,

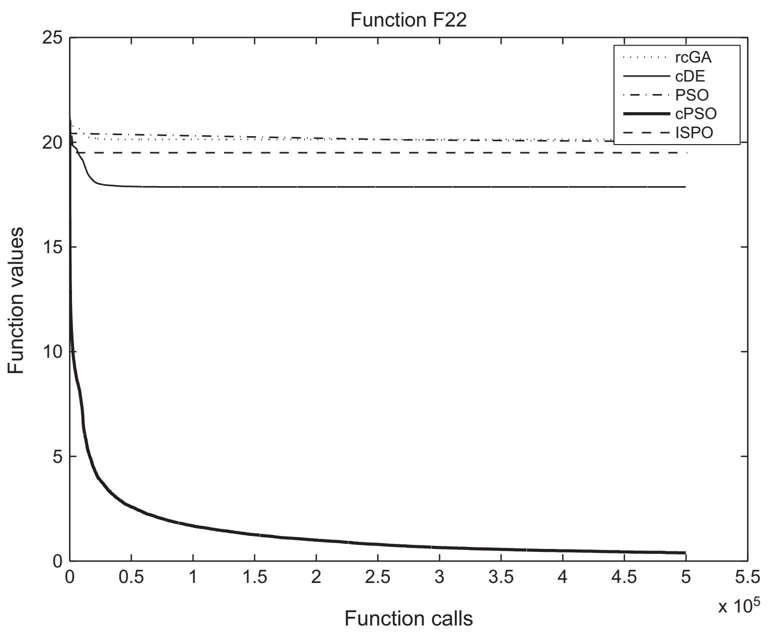

Fig. 2. Average performance trends of memory-saving algorithms and PSO for f_{22}.

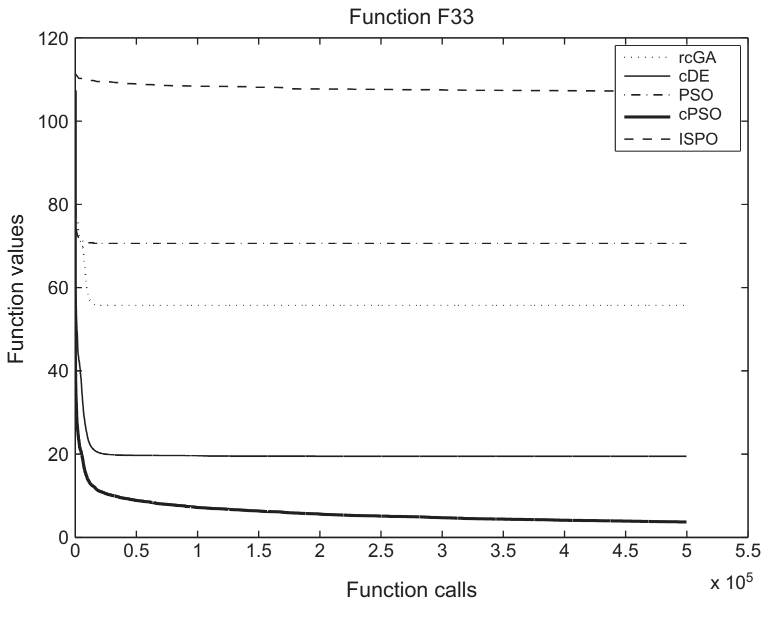

Fig. 3. Average performance trends of memory-saving algorithms and PSO for f_{33}.

cPSO has been taken as a reference algorithm. Indicating with R_0 the rank of cPSO, and with R_j for $j = 1, \ldots, N_A - 1$ the rank of one of the remaining 11 algorithms, the values z_j have been calculated as

$$z_j = \frac{R_j - R_0}{\sqrt{\frac{N_A(N_A+1)}{6N_{TP}}}} \tag{22}$$

where N_{TP} is the number of test problems in consideration ($N_{TP} = 47$ in our case). By means of the z_j values, the corresponding cumulative normal distribution values p_j have been calculated. These p_j values have then been compared with the corresponding $\delta/(N_A - j)$ where δ is the level of confidence, set to 0.05 in our case. Table 6 displays ranks, z_j values, p_j values, and corresponding $\delta/(N_A - j)$. The rank of cPSO is shown in parenthesis. The values of z_j and p_j are expressed in terms of

Table 4
Comparison of cPSO against classical optimization algorithms.

Test problem	DE	W	CMA-ES	W	cPSO
f_1	8.268e+01 ± 1.90e+01	+	**0.000e+00 ± 0.00e+00**	−	6.471e+01 ± 2.28e+01
f_2	3.062e+04 ± 3.70e+03	+	**1.395e−32 ± 2.23e−32**	−	2.560e+03 ± 2.36e+03
f_3	2.714e+06 ± 1.11e+06	+	**7.614e+02 ± 2.66e+03**	−	1.320e+05 ± 7.46e+04
f_4	4.071e+00 ± 1.98e−01	+	1.141e+01 ± 9.47e+00	=	**3.728e+00 ± 3.71e−01**
f_5	7.194e+01 ± 9.72e+00	+	2.465e−04 ± 1.35e−03	+	**9.636e−08 ± 3.07e−08**
f_6	2.150e+02 ± 9.09e+00	+	7.339e+01 ± 2.80e+01	+	**2.940e+01 ± 7.94e+00**
f_7	2.407e+05 ± 4.95e+04	+	**5.937e+01 ± 1.97e+01**	−	4.614e+02 ± 2.40e+02
f_8	6.329e+03 ± 2.35e+02	+	4.959e+03 ± 6.59e+02	+	**3.160e+03 ± 9.75e+02**
f_9	1.632e+04 ± 1.12e+03	+	**2.880e+03 ± 1.12e+03**	−	1.344e+04 ± 1.74e+03
f_{10}	8.507e+05 ± 9.23e+04	−	**8.570e+02 ± 1.23e+03**	−	1.040e+06 ± 1.16e+05
f_{11}	4.216e+00 ± 1.57e−01	+	1.082e+01 ± 9.63e+00	=	**3.699e+00 ± 3.53e−01**
f_{12}	6.535e+01 ± 1.01e+01	+	1.396e−03 ± 3.81e−03	+	**9.567e−08 ± 2.69e−08**
f_{13}	2.585e+02 ± 1.11e+01	+	7.210e+01 ± 2.23e+01	+	**3.924e+01 ± 2.31e+01**
f_{14}	4.002e+01 ± 1.07e+00	=	**8.432e+00 ± 9.38e+00**	−	3.942e+01 ± 1.15e+00
f_{15}	**7.441e−02 ± 1.88e−05**	+	1.355e+00 ± 7.42e+00	=	1.777e+00 ± 4.27e−01
f_{16}	−9.941e+01 ± 1.07e−01	+	**−1.000e+02 ± 9.0e−06**	=	**−1.000e+02 ± 8.45e−05**
f_{17}	**9.423e−08 ± 5.15e−08**	−	1.037e−02 ± 5.68e−02	−	1.702e+00 ± 7.09e−01
f_{18}	**−1.150e+00 ± 3.63e−07**	−	−1.116e+00 ± 6.62e−02	−	−1.030e+00 ± 7.55e−01
f_{19}	4.702e+02 ± 1.43e+01	+	1.290e+02 ± 2.97e+01	+	**4.403e+01 ± 3.44e+01**
f_{20}	−1.277e+01 ± 4.27e−01	+	−1.994e+01 ± 2.85e+00	=	**−2.063e+01 ± 2.33e+00**
f_{21}	1.269e+04 ± 3.61e+02	+	8.156e+03 ± 9.93e+02	+	**4.784e+03 ± 1.09e+03**
f_{22}	1.829e+01 ± 4.23e−01	+	1.521e+01 ± 7.97e+00	+	**3.899e−01 ± 5.18e−01**
f_{23}	1.610e+02 ± 6.38e+00	+	1.101e+01 ± 3.35e+00	+	**4.657e−02 ± 2.39e−02**
f_{24}	2.387e+04 ± 3.49e+03	+	**1.732e−41 ± 2.61e−41**	−	6.918e−02 ± 2.54e−02
f_{25}	−1.118e−02 ± 1.28e−03	+	−1.201e−02 ± 6.56e−03	+	**−7.858e−01 ± 1.60e−14**
f_{26}	−1.588e+01 ± 5.25e−01	+	**−3.144e+01 ± 6.55e+00**	−	−2.920e+01 ± 2.53e+00
f_{27}	8.898e+04 ± 8.78e+03	+	**0.000e+00 ± 0.00e+00**	−	2.127e−02 ± 4.04e−03
f_{28}	1.176e+03 ± 2.53e+01	+	3.610e+02 ± 8.97e+01	+	**8.776e−03 ± 2.87e−03**
f_{29}	2.635e+10 ± 5.08e+09	+	1.340e+02 ± 1.21e+02	+	**1.220e+02 ± 2.81e+01**
f_{30}	1.476e+11 ± 1.26e+10	+	**2.447e−01 ± 7.98e−02**	−	4.928e+06 ± 6.56e+05
f_{31}	3.025e+04 ± 4.78e+02	+	1.623e+04 ± 1.31e+03	+	**1.045e+04 ± 2.94e+03**
f_{32}	2.093e+05 ± 1.63e+04	+	**1.563e−13 ± 2.51e−14**	−	1.531e−02 ± 3.80e−03
f_{33}	1.185e+02 ± 2.83e+00	+	**5.603e+00 ± 6.00e+00**	−	7.370e+01 ± 3.32e+00
f_{34}	8.196e+10 ± 1.11e+10	+	**5.823e+01 ± 2.34e+03**	−	4.896e+05 ± 2.21e+05
f_{35}	1.399e+03 ± 4.24e+01	+	**1.995e+02 ± 4.44e+01**	−	6.701e+02 ± 6.36e+01
f_{36}	1.566e+03 ± 1.33e+02	+	**−1.300e+02 ± 2.53e−03**	−	−1.082e+02 ± 4.21e+00
f_{37}	5.506e+02 ± 1.24e−01	+	**5.478e+02 ± 6.04e+00**	−	5.492e+02 ± 2.50e−01
f_{38}	−1.055e+03 ± 1.08e+01	+	**−1.445e+03 ± 2.28e+02**	−	−1.284e+03 ± 3.90e+01
f_{39}	8.337e+03 ± 1.11e+03	+	**0.000e+00 ± 0.00e+00**	−	4.314e−03 ± 1.24e−03
f_{40}	8.968e+04 ± 7.74e+03	+	**1.488e−18 ± 1.47e−18**	−	4.375e+00 ± 9.83e−01
f_{41}	2.141e+09 ± 5.70e+08	+	**6.853e+01 ± 1.08e+02**	−	8.941e+01 ± 5.26e+01
f_{42}	1.363e+01 ± 4.45e−01	+	1.280e+01 ± 9.25e+00	+	**1.277e+00 ± 3.68e−01**
f_{43}	3.710e+02 ± 3.26e+01	+	**8.204e−04 ± 4.02e−03**	−	1.084e+00 ± 3.16e−01
f_{44}	4.683e+02 ± 1.34e+01	+	1.846e+02 ± 5.52e+01	+	**4.706e−03 ± 7.39e−03**
f_{45}	2.538e+06 ± 2.08e+05	+	1.140e+02 ± 1.84e+01	+	**5.051e+01 ± 4.27e+01**
f_{46}	3.151e+04 ± 1.19e+03	+	**4.725e+03 ± 1.03e+03**	−	2.320e+04 ± 3.88e+03
f_{47}	4.674e+06 ± 2.18e+05	+	**8.181e+03 ± 1.01e+04**	−	1.359e+06 ± 1.14e+06

z_{N_A-j} and p_{N_A-j} for $j = 1,\ldots,N_A - 1$. Moreover, it is indicated whether the null-hypothesis (that the two algorithms have indistinguishable performances) is "Rejected", i.e. cPSO statistically outperforms the algorithm under consideration, or "Accepted" if the distribution of values can be considered the same (there is no out-performance).

Table 6 shows that the proposed cPSO significantly outperforms its population-based variant and displays the best performance among the set of memory-saving algorithms. This fact makes cPSO a reliable option for applications plagued by a limited memory availability. It is also interesting to observe that cPSO and FPSO have a comparable performance despite the relevant spread in terms of computational resource requirements. The other population-based algorithms (CMA-ES, SADE, PAP, CLPSO, JADE) appear to outperform cPSO. As a conclusive remark, if the hardware (memory) allows the employment of a population, see [67], also hybridizing it with learning components, population-based algorithms should be used. However, when the hardware imposes the use of a memory-saving algorithm, cPSO seems to be an excellent compromise between the goal of having a high performance and the memory restrictions imposed by the application domain.

4.6. Time complexity

For all the algorithms under consideration, the average computational times (over 30 runs) have been computed. The time has been calculated for the Ackley function (f_{22} in Appendix). Several dimensionality levels, (10, 20, 30, 50, and 100

Table 5
Comparison of cPSO against non-PSO based state-of-the-art algorithms.

Test problem	JADE	W	SADE	W	PAP	W	cPSO
f_1	**5.161e−91 ± 2.53e−90**	−	1.837e−25 ± 3.72e−25	−	−2.226e−03 ± 8.71e−03	−	6.471e+01 ± 2.28e+01
f_2	**1.016e−17 ± 1.88e−17**	−	4.911e−04 ± 9.77e−04	−	9.702e−01 ± 1.45e+00	−	2.560e+03 ± 2.36e+03
f_3	**4.302e+00 ± 1.87e+01**	−	2.148e+01 ± 2.68e+01	−	1.470e+01 ± 2.25e+02	−	1.320e+05 ± 7.46e+04
f_4	1.552e−01 ± 3.55e−01	−	4.685e−01 ± 7.27e−01	−	**3.730e−14 ± 1.61e−14**	−	3.728e+00 ± 3.71e−01
f_5	4.825e−03 ± 6.39e−03	+	5.132e−04 ± 2.51e−03	+	1.906e−02 ± 5.42e−02	+	**9.636e−08 ± 3.07e−08**
f_6	**0.000e+00 ± 0.00e+00**	−	3.379e+01 ± 1.56e+01	=	1.273e+01 ± 3.98e+00	−	2.940e+01 ± 7.94e+00
f_7	4.146e−02 ± 2.03e−01	−	4.879e+01 ± 2.86e+01	−	**0.000e+00 ± 0.00e+00**	−	4.614e+02 ± 2.40e+02
f_8	**2.566e+02 ± 1.51e+02**	−	2.754e+03 ± 7.96e+02	=	1.035e+03 ± 5.73e+02	−	3.160e+03 ± 9.75e+02
f_9	**1.731e+03 ± 5.88e+02**	−	2.309e+03 ± 6.81e+02	−	3.327e+03 ± 6.10e+02	−	1.344e+04 ± 1.74e+03
f_{10}	**2.674e+04 ± 6.17e+03**	−	2.901e+04 ± 2.23e+04	−	5.030e+04 ± 4.21e+04	−	1.040e+06 ± 1.16e+05
f_{11}	1.666e−01 ± 4.53e−01	−	1.189e+00 ± 8.65e−01	−	**−5.577e+00 ± 7.02e+01**	−	3.699e+00 ± 3.53e−01
f_{12}	1.418e−01 ± 1.61e−01	=	5.276e−02 ± 1.13e−01	+	5.681e−02 ± 9.18e−02	+	**9.567e−08 ± 2.69e−08**
f_{13}	**2.651e+01 ± 6.72e+00**	−	3.685e+01 ± 1.07e+01	=	6.040e+01 ± 2.20e+01	+	3.924e+01 ± 2.31e+01
f_{14}	4.021e+01 ± 9.93e−01	+	3.434e+01 ± 4.01e+00	−	**3.408e+01 ± 5.36e+00**	−	3.942e+01 ± 1.15e+00
f_{15}	**7.683e−14 ± 7.99e−14**	−	2.529e−09 ± 1.88e−09	−	1.673e−07 ± 2.36e−07	−	1.777e+00 ± 4.27e−01
f_{16}	**−1.000e+02 ± 0.00e+00**	=	**−1.000e+02 ± 0.00e+00**	=	−9.994e+01 ± 3.48e−02	+	**−1.000e+02 ± 8.45e−05**
f_{17}	**8.260e−25 ± 3.47e−24**	−	1.360e−16 ± 3.52e−16	−	2.960e−06 ± 1.42e−05	−	1.702e+00 ± 7.09e−01
f_{18}	**−1.150e+00 ± 6.80e−16**	−	**−1.150e+00 ± 2.59e−15**	−	−6.569e−01 ± 3.54e−01	+	−1.030e+00 ± 7.55e−01
f_{19}	5.550e+01 ± 1.04e+01	+	4.589e+01 ± 1.52e+01	+	1.076e+02 ± 3.15e+01	+	**4.403e+01 ± 3.44e+01**
f_{20}	**−4.957e+01 ± 4.88e−02**	−	−4.118e+01 ± 1.44e+00	−	−4.530e+01 ± 7.95e−01	−	−2.063e+01 ± 2.33e+00
f_{21}	**4.540e+02 ± 2.81e+02**	−	6.692e+03 ± 8.18e+02	+	4.315e+03 ± 8.50e+02	=	4.784e+03 ± 1.09e+03
f_{22}	3.954e+00 ± 1.02e+00	+	6.422e+00 ± 1.46e+00	+	6.316e−01 ± 5.57e−01	=	**3.899e−01 ± 5.18e−01**
f_{23}	1.723e−12 ± 5.43e−12	−	3.081e−08 ± 1.45e−07	−	**5.875e−14 ± 4.04e−14**	−	4.657e−02 ± 2.39e−02
f_{24}	**7.620e−60 ± 3.73e−59**	−	2.467e−11 ± 8.27e−11	−	8.449e−26 ± 1.24e−25	−	6.918e−02 ± 2.54e−02
f_{25}	−9.938e−02 ± 2.40e−02	+	−6.956e−02 ± 1.21e−02	+	−1.639e−01 ± 3.06e−02	+	**−7.858e−01 ± 1.60e−14**
f_{26}	−7.728e+01 ± 9.48e−01	−	−4.418e+01 ± 5.48e+00	−	**−8.370e+01 ± 2.23e+00**	−	−2.920e+01 ± 2.53e+00
f_{27}	**9.000e−59 ± 4.41e−58**	−	4.956e−14 ± 1.82e−13	−	2.441e−25 ± 1.68e−25	−	2.127e−02 ± 4.04e−03
f_{28}	4.560e−01 ± 5.85e−01	=	1.322e+02 ± 3.02e+01	+	1.083e+02 ± 2.19e+01	+	**8.776e−03 ± 2.87e−03**
f_{29}	1.281e+02 ± 4.66e+01	+	2.103e+02 ± 1.25e+02	+	1.394e+02 ± 3.88e+01	+	**1.220e+02 ± 2.81e+01**
f_{30}	**1.309e+04 ± 3.11e+04**	−	6.017e+07 ± 4.51e+07	+	1.615e+09 ± 4.50e+08	+	4.928e+06 ± 6.56e+05
f_{31}	**8.241e+02 ± 4.10e+02**	−	1.519e+04 ± 1.54e+03	+	1.261e+04 ± 1.19e+03	+	1.045e+04 ± 2.94e+03
f_{32}	**1.023e−62 ± 4.34e−62**	−	4.563e−26 ± 6.23e−26	−	1.156e−24 ± 3.11e−24	−	1.531e−02 ± 3.80e−03
f_{33}	7.717e+01 ± 4.78e+00	+	8.844e+01 ± 5.92e+00	+	**6.882e+01 ± 3.44e+00**	−	7.370e+01 ± 3.32e+00
f_{34}	**−6.937e+02 ± 7.68e+01**	−	−4.252e+02 ± 4.15e+02	−	−6.614e+02 ± 6.57e+01	−	4.896e+05 ± 2.21e+05
f_{35}	**−1.188e+02 ± 1.94e+01**	−	1.767e+02 ± 7.50e+01	−	8.741e+01 ± 3.49e+01	−	6.701e+02 ± 6.36e+01
f_{36}	−1.299e+02 ± 1.96e−01	−	−1.297e+02 ± 6.07e−01	−	**−1.300e+02 ± 5.55e−03**	−	−1.082e+02 ± 4.21e+00
f_{37}	5.330e+02 ± 1.26e+00	−	5.407e+02 ± 3.75e+00	−	**5.306e+02 ± 5.93e−01**	−	5.492e+02 ± 2.50e−01
f_{38}	**−1.680e+03 ± 5.26e+00**	−	−1.489e+03 ± 3.51e+01	−	−1.575e+03 ± 2.90e+01	−	−1.284e+03 ± 3.90e+01
f_{39}	**1.177e−77 ± 2.20e−77**	−	6.824e−27 ± 7.07e−27	−	1.577e−26 ± 6.11e−27	−	4.314e−03 ± 1.24e−03
f_{40}	**4.230e−09 ± 4.36e−09**	−	6.328e+01 ± 6.75e+01	+	2.368e+02 ± 9.59e+01	+	4.375e+00 ± 9.83e−01
f_{41}	**5.053e+00 ± 5.10e+00**	−	4.439e+01 ± 2.63e+01	−	7.821e+01 ± 4.13e+01	=	8.941e+01 ± 5.26e+01
f_{42}	**7.994e−15 ± 0.00e+00**	−	1.465e−14 ± 1.19e−14	−	7.942e−02 ± 2.70e−01	−	1.277e+00 ± 3.68e−01
f_{43}	4.107e−04 ± 2.01e−03	−	**0.000e+00 ± 0.00e+00**	−	8.076e−03 ± 1.48e−02	−	1.084e+00 ± 3.16e−01
f_{44}	4.706e−03 ± 7.39e−03	=	**0.000e+00 ± 0.00e+00**	=	**0.000e+00 ± 0.00e+00**	=	**0.000e+00 ± 0.00e+00**
f_{45}	**2.448e−10 ± 1.41e−10**	−	1.597e+01 ± 8.51e+00	−	5.559e+01 ± 1.44e+01	=	5.051e+01 ± 4.27e+01
f_{46}	**2.575e+03 ± 4.63e+02**	−	5.826e+03 ± 7.87e+02	−	7.964e+03 ± 1.28e+03	−	2.320e+04 ± 3.88e+03
f_{47}	**2.096e+05 ± 2.94e+04**	−	5.176e+05 ± 2.59e+05	−	2.150e+05 ± 1.68e+05	−	1.359e+06 ± 1.14e+06

Table 6
Holm–Bonferroni procedure for the 10 algorithms over the 47 test problems under consideration.

$N_A - j$	Algorithm	$z_{N_A} - j$	$p_{N_A} - j$	$\delta/(N_A - j)$	Hypothesis	Rank
11	DE	−5.435e+00	2.736e−08	4.545e−03	Rejected	2.787e+00
10	PSO	−4.892e+00	4.998e−07	5.000e−03	Rejected	3.191e+00
9	rcGA	−4.606e+00	2.056e−06	5.556e−03	Rejected	3.404e+00
8	ISPO	−1.488e+00	6.844e−02	6.250e−03	Accepted	5.723e+00
7	cDE	−6.866e−01	2.462e−01	7.143e−03	Accepted	6.319e+00
6	FPSO	1.430e−01	5.569e−01	8.333e−03	Accepted	6.936e+00
5	CMA-ES	1.859e+00	9.685e−01	1.000e−02	Accepted	8.213e+00
4	SADE	2.203e+00	9.862e−01	1.250e−02	Accepted	8.468e+00
3	PAP	3.061e+00	9.989e−01	1.667e−02	Accepted	9.106e+00
2	CLPSO	3.519e+00	9.998e−01	2.500e−02	Accepted	9.447e+00
1	JADE	4.348e+00	1.000e+00	5.000e−02	Accepted	1.006e+01
	cPSO					(6.830e+00)

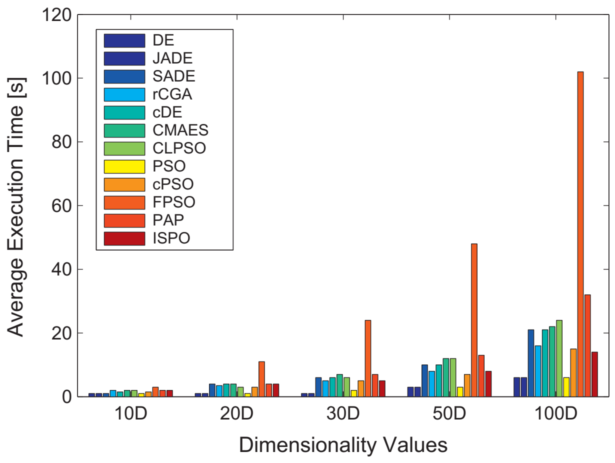

Fig. 4. Average execution times for the algorithms under investigation.

dimensions) have been considered. The computational times, for all the algorithms, have been calculated by means of a PC Intel Core 2 Duo 2.4 GHz with 4 GB RAM employing GNU/Linux Ubuntu 12.04 with Matlab implementation. It must be remarked that all the calculations for achieving the results reported in this paper have been performed in Java by means of the Kimeme platform [16]. However, for the representation of the overhead we used Matlab as we made use of its powerful post-processing tools. The execution times are shown in Fig. 4.

Results in Fig. 4 show that the algorithms may have a diverse overhead. In particular, FPSO appears to have a higher overhead with respect to the other algorithms. In addition, it can be observed that the compact versions are characterized by a higher overhead with respect to their corresponding population-based versions, see overhead corresponding to cDE and DE as well as cPSO and PSO. This fact is due to the computational cost of the sampling, see Fig. 1, that is higher than selecting an individual/particle from a list/population. Thus, cPSO has a higher computational cost than the original PSO. However, it is important to consider that the computational overhead of cPSO is rather low with respect to other algorithms in this study, e.g. FPSO. In this light, in those cases when only a limited memory is available (a population-based PSO cannot be used), cPSO can still be efficiently employed in real-time applications. In addition, cPSO tends to be computationally cheaper than the cDE since less samples are required to perturb an individual. More specifically, while cDE requires three samples from **PV** at each mutation, see [49], cPSO requires only one sampling, that is $\mathbf{x}_{lb}^{t}$ in formula (20).

Finally, it can be remarked that, unlike other algorithms such as CMAES and FPSO, the execution time of cPSO appears to grow linearly. For example, at 100D the execution time is about twice bigger than that at 50D (the budget per each run is proportionally connected to the dimensionality of the problem).

4.7. Results on a large scale testbed

In order to test the capability of cPSO to handle high dimensional fitness landscapes, the proposed algorithm has been run on the large scale testbed defined in [85] where the dimensionality level has been set equal to 1000. For the seven test problems in the testbed, the performance of cPSO has been compared with that of cDE. These test problems are constructed by using the same functions indicated with f_{32}–f_{38} in Appendix and then scaling them up to 1000 dimensions. In order to have a fair reference of modern algorithms on large scale problems, CLPSO and JADE have also been run for the testbed. The choice of these two algorithms is due to the fact that they displayed the best performance in the Holm–Bonferroni procedure shown in Table 6. Each algorithm has been run 30 times. The same parameter setting shown above has been used for this 1000 dimension test. Table 7 shows the results in terms of average performance, related standard deviation, and Wilcoxon Rank-Sum test.

Numerical results on the large scale problems show that compact algorithms display a mediocre performance. This finding confirms the conjecture reported in [48]. The exploration of a high dimensional space is especially hard for compact algorithms because the sampled solutions are extracted from the same *PV*. This produces that the sampled solutions, in the late stage of the optimization, are probably focused in a relatively narrow region of the decision space. This effect could prevent from a proper exploration that scattered multiple solutions could allow, see [67].

The comparison between cPSO and cDE leads to an interesting finding. The cPSO algorithm appear to significantly outperform for the entire testbed the cDE algorithm. This fact is, according to our interpretation, due to the PSO structure which allows a natural compact encoding, as explained in Section 3. In this light, cPSO is a compact structure which enhances the state-of-the-art of compact optimization for high dimensional problems.

Table 7
Results on the large scale testbed [85] in 1000 dimensions.

	cPSO	CLPSO	W	JADE	W	cDE	W
f_{32}	1.59e+06 ± 1.40e+05	**−3.04e+02 ± 3.19e+01**	−	1.00e+06 ± 3.07e+05	−	4.83e+06 ± 1.30e+05	+
f_{33}	−2.80e+02 ± 3.20e+00	**−3.72e+02 ± 5.41e−01**	−	−3.24e+02 ± 8.09e+00	−	−2.76e+02 ± 2.64e+00	+
f_{34}	1.30e+12 ± 5.15e+11	**4.04e+03 ± 2.75e+02**	−	4.24e+11 ± 2.25e+11	−	4.74e+12 ± 2.24e+11	+
f_{35}	1.70e+04 ± 9.21e+02	**−7.03e+01 ± 1.33e+01**	−	4.48e+03 ± 9.44e+02	−	2.06e+04 ± 4.73e+02	+
f_{36}	1.41e+04 ± 9.56e+02	**−1.78e+02 ± 2.43e−01**	−	8.58e+03 ± 2.70e+03	−	4.28e+04 ± 1.02e+03	+
f_{37}	−1.19e+02 ± 7.31e−02	**−1.40e+02 ± 9.77e−05**	−	−1.22e+02 ± 6.43e−01	−	−1.19e+02 ± 5.53e−02	+
f_{38}	−8.60e+03 ± 4.70e+02	**−1.33e+04 ± 4.73e+01**	−	−1.18e+04 ± 4.21e+02	−	−7.25e+03 ± 1.84e+02	+

5. Case study: power plant optimization

Circulating Fluidized Bed (CFB) boilers are devices for performing the fluidized bed combustion, see [76]. The latter is a combustion technology used in power plants. CFB boilers are very promising since, with respect to devices belonging to the same family, allow the power production with a high efficiency and in a clean way. In addition, these devices are very versatile since they are capable to burn a diverse range of combustibles, from bio-fuels to coal. Despite their popularity and effectiveness, the dynamics of the burning process in CFB is not much studied and has not been modelled yet. Obviously, a proper understanding of the combustion dynamics is a prerequisite for performing an optimal control. In particular, the combustion process is completely unknown when it involves the co-combustion of mixtures of diverse fuels. Variations and inhomogeneity of the fuels cause abrupt changes in the burning rate and oxygen concentration levels.

The efficiency and usability of the proposed cPSO has been tested to perform the real-time control of a real-world CFB boiler from Technical Research Center of Finland (VTT), Jyväskylä division. The CFB system has been observed by VTT institute and a set of data related to its functioning have been provided. The data set contains 28,829 samples, recorded with 1 Hz frequency. The CFB boiler can be seen as a system characterized by multiple inputs and one output. The inputs (or control inputs) are listed in the following:

- Velocity of fuel screw (i.e. amount of fuel per second).
- Additional velocity of fuel screw.
- Primary air flow.
- Secondary air flow.
- Tertiary air flow.

For a given set of input values, the total amount of O_2 is the recorded output of the system. In order to obtain a good performance of the boiler in terms of pollution efficiency, the amount of oxygen in the output of the system must be kept as constant as possible during the normal procedures. In order to pursue this aim, the CFB boiler is controlled by means of a classical Multiple Inputs Single Output (MISO) linear controller, designed with a trial and error procedure and capable of obtaining theoretical stability of the system, see [8]. Since the relationship among the input values and the output are just partially known, due to nonlinearities in the system and complex chemical dynamic behaviour, the exact value of the most suitable control vector cannot be determined a priori. Thus, the linear controller used in the application is just suboptimal. However the sampling time used in the real-time control of this plant (2 s), much higher than those typically used in other real-time control optimization application (see [9,10,57,55]), allowed us to propose here a novel implementation of optimization for control applications. In this paper it is proposed that in addition to the standard MISO controller, a feed-forward support virtual controller is used in the system. Fig. 5 shows the traditional MISO control scheme (white blocks) and the support virtual control scheme (grey blocks).

While the control of the power plant (white blocks) occurs with sample time of 2 s, within the grey blocks a prediction of the plant behaviour is performed. More specifically, a model of the plant, based on a static neural network, has been obtained by identification on the data, see [81]. The fast static model predicts for a given set of inputs the output (O_2) of the system. The predicted output is compared with the reference value (set point) chosen by the user. The cPSO algorithm is used to determine the set of inputs which minimizes the tracking error, i.e. the difference between the O_2 predicted by the model and that value of O_2 set by the user (this difference is the fitness of our problem). At the end of the 2 s interval, cPSO has performed 20,000 fitness evaluations and returns to the actual control system (white blocks) the optimized set of control inputs. The original controller is kept in order to guarantee the stability of the system and to compensate disturbance and parametric variation. This kind of novel control application has been made possible by the application of a computationally cheap, memory saving, optimization algorithm such as the one proposed in this paper. Since the entire optimization should be performed during a 2 s window, the algorithm must be implemented directly within the control card, see [48]. In this study, the proposed cPSO has been implemented and run within a Freescale 68HC11F1 micro-controller. The employment of an external computer would introduce some delays, due to the communication among devices, thus making the optimization unbearable. In other words, a full scale optimization algorithm would largely result in a run-time execution overrun during the normal functioning, making it completely unsuitable for this purpose.

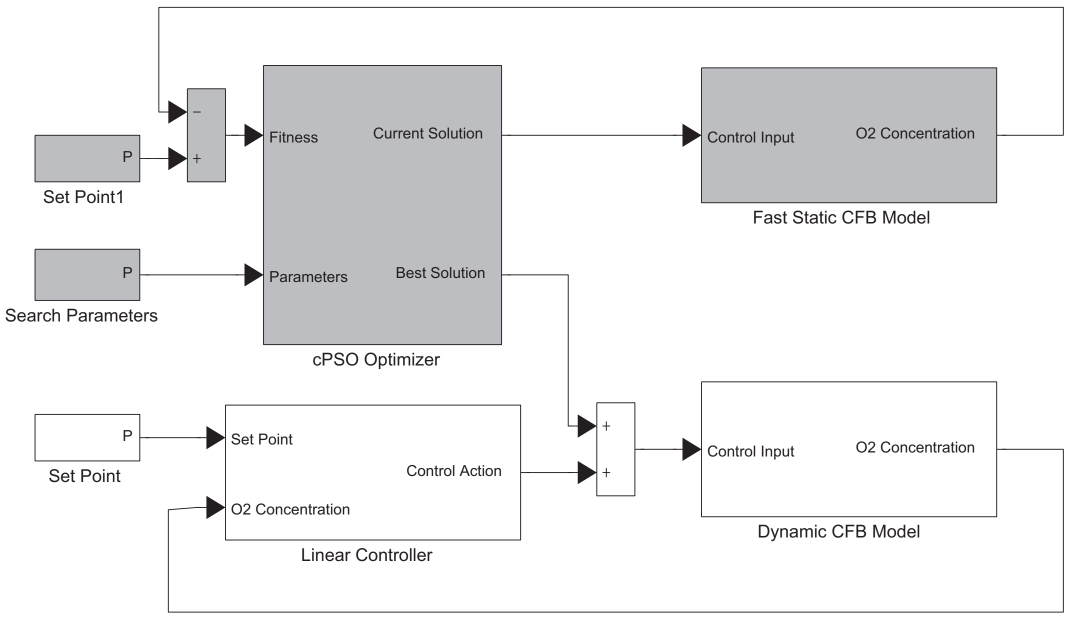

Fig. 5. Control scheme of the power plant.

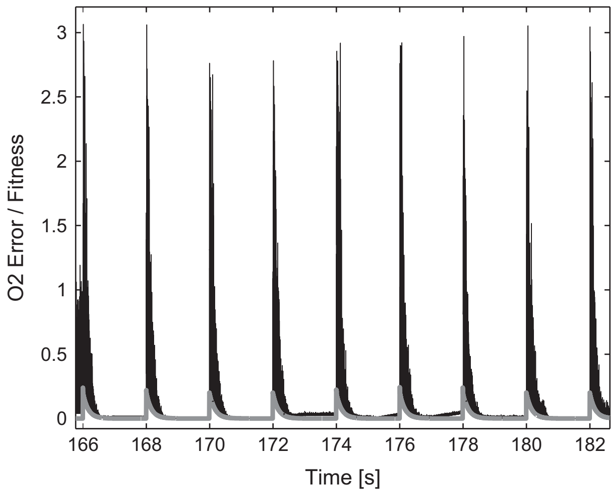

Fig. 6. Average value of O_2 over time.

The parameter setting for cPSO is the same mentioned above. The cPSO algorithm has been compared with cDE and ISPO. For cDE and ISPO, the same parameter settings shown above has been used. The three algorithms have been run 30 times. The average value of the O_2 over time during the execution of the optimization procedure is shown in Fig. 6. The reference value of O_2 and the correspondent value estimated by the static CFB model associated to a set of inputs selected by cPSO are shown in grey and black, respectively.

The performance comparison of the proposed cPSO against cDE and ISPO is shown in Fig. 7.

In order to highlight the convergence properties of cPSO, a typical evolution (a selected run) of the normalized σ values of the virtual population during the operation of the system is shown in Fig. 8. Each design variable is indicated with a different colour.

It can be observed that at the beginning of each 2 s window, the sigma values are high and during the interval, the population tends to quickly converge, $\sigma \to 0$. This behaviour demonstrates the efficiency of the proposed cPSO to quickly improve an initial trial solution and thus being valuable under real-time constraints.

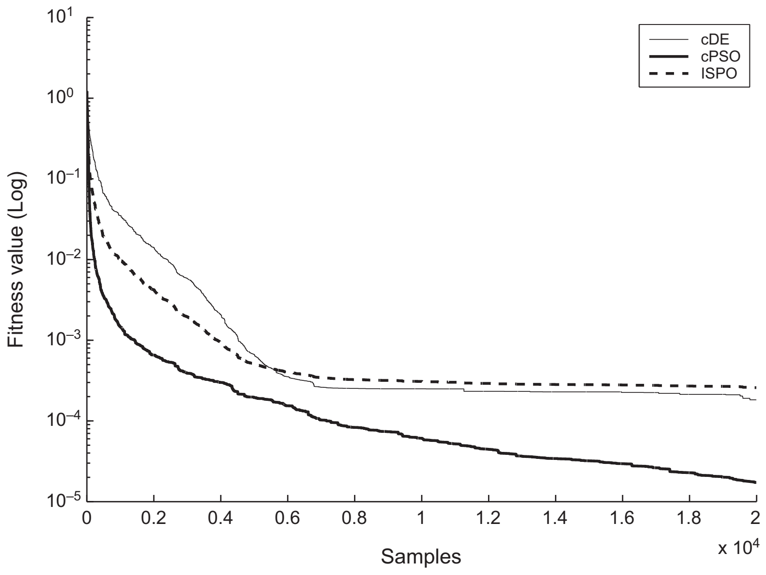

Fig. 7. Average performance trend (over 30 runs) of cPSO and cDE for the case study.

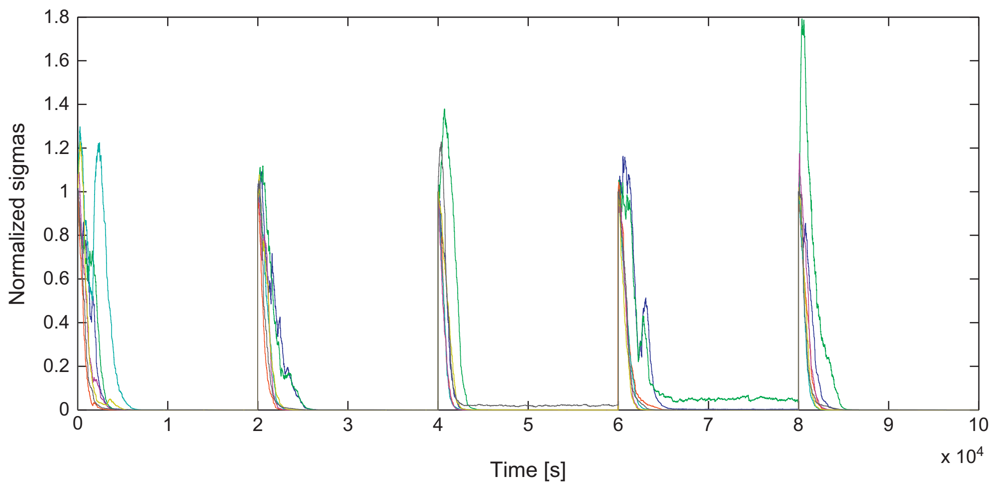

Fig. 8. Variation of normalized σ over time (each design variable corresponds to a colour). (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

6. Conclusion

This paper proposes a novel optimization algorithm, namely compact Particle Swarm Optimization (cPSO). This algorithm employs the search logic of PSO but instead of having an actual swarm of solutions, makes use of a probabilistic representation of the population. This feature is important for application problems characterized by a limited memory since it allows the embedded implementation in small and cheap devices. The implementation of optimization algorithms on embedded devices can have an interesting impact for every day applications as well as for real-time systems and special applications.

The cPSO algorithm displays a good performance with respect to the other compact algorithms previously presented in literature. The proposed algorithm appears to be especially efficient for fitness landscapes characterized by multiple strong basins of attraction. Numerical results on a set of various test problems show that, although outperformed by complex population-based algorithms, cPSO is still competitive in several cases. Thus, cPSO seems to be a valid alternative for optimization problems plagued by a limited memory.

A real-world case where in practice cPSO has been implemented is also shown. The real-world application is the on-line control design of a boiler. The control on the actual device is combined with a predictive control performed over a model. The real-time requirements impose that the optimization is instantly run. Since the employment of multiple interconnected devices would introduce an unacceptable delay, the optimization must be entirely performed directly on the control card, despite the modest memory available. Experimental results on this real-world application show the applicability of the proposed approach and highlights the good cPSO performance within the category of memory-saving algorithms.

Acknowledgements

This research is supported by the Academy of Finland, Akatemiatutkija 130600, Algorithmic Design Issues in Memetic Computing and Tutkijatohtori 140487, Algorithmic Design and Software Implementation: a Novel Optimization Platform. In addition, this research supported by Tekes – the Finnish Funding Agency for Technology and Innovation, grant 40214/08 (Dynergia).

INCAS[3] is co-funded by the Province of Drenthe, the Municipality of Assen, the European Fund for Regional Development and the Ministry of Economic Affairs, Peaks in the Delta.

Appendix A. Benchmark problems

The following test problems have been considered in this study.

f_1 Shifted sphere function: f_1 from [83] with $n = 30$.

$$f_1(x) = \sum_{i=1}^{n} z_i^2 \tag{23}$$

where $z = x - o$ and the shifted optimum $o = [o_1, o_2, \ldots, o_n]$. Decision space $D = [-100, 100]^n$. Properties: Unimodal, Shifted, Separable, Scalable.

f_2 Shifted Schwefel's Problem 1.2: f_2 from [83] with $n = 30$.

$$f_2(x) = \sum_{i=1}^{n} \left(\sum_{j=1}^{i} z_j \right)^2 \tag{24}$$

where $z = x - o$ and the shifted optimum $o = [o_1, o_2, \ldots, o_n]$. Decision space $D = [-100, 100]^n$. Properties: Unimodal, Shifted, Non-separable, Scalable.

f_3 Rosenbrock's function: f_3 from [69] with $n = 30$.

$$f_3(x) = \sum_{i=1}^{n-1} \left(100 \left(x_{i+1} - x_i^2 \right)^2 + (1 - x_i)^2 \right) \tag{25}$$

Decision space $D = [-100, 100]^n$. Properties: Multi-modal, Non-separable, Scalable.

f_4 Shifted Ackley's function: f_5 from [69] with $n = 30$.

$$f_4(x) = -20 e^{-0.2 \sqrt{1/n \sum_{i=1}^{n} z_i^2}} - e^{(1/n) \sum_{i=1}^{n} \cos(2\pi z_i)} + 20 + e \tag{26}$$

where $z = x - o$ and the shifted optimum $o = [o_1, o_2, \ldots, o_n]$. Decision space $D = [-32, 32]^n$. Properties: Multi-modal, Shifted, Non-separable, Scalable.

f_5 Shifted Griewank's function: f_7 from [69] with $n = 30$.

$$f_5(x) = \sum_{i=1}^{n} \frac{z_i^2}{4000} - \prod_{i=1}^{n} \cos \frac{z_i}{\sqrt{i}} + 1 \tag{27}$$

where $z = x - o$ and the shifted optimum $o = [o_1, o_2, \ldots, o_n]$. Decision space $D = [-600, 600]^n$. Properties: Multi-modal. Shifted, Non-separable, Scalable.

f_6 Shifted Rastrigin's function: f_9 from [83] with $n = 30$.

$$f_6(x) = 10n + \sum_{i=1}^{n} \left(z_i^2 - 10 \cos 2\pi z_i \right) \tag{28}$$

where $z = x - o$ and the shifted optimum $o = [o_1, o_2, \ldots, o_n]$. Decision space $D = [-5, 5]^n$. Properties: Multi-modal, Shifted, Separable, Scalable, huge number of local optima.

f_7 Shifted non continuous Rastrigin's function: f_{11} from [69] with $n = 30$.

$$f_7(x) = 10n + \sum_{i=1}^{n} \left(y_i^2 - 10 \cos 2\pi y_i \right) \tag{29}$$

$$y_i = \begin{cases} z_i & \text{if } |z_i| < 1/2 \\ round(2z_i)/2 & \text{if } |z_i| \geqslant 1/2 \end{cases}$$

where $z = x - o$ and the shifted optimum $o = [o_1, o_2, \ldots, o_n]$. Decision space $D = [-500, 500]^n$. Properties: Multi-modal, Shifted, Rotated, Separable, Scalable, huge number of local optima.

f_8 Schwefel's function: f_{12} from [69] with $n = 30$.

$$f_8(x) = 418.9829n + \sum_{i=1}^{n} \left(-x_i \sin \sqrt{|x_i|}\right). \tag{30}$$

Decision space $D = [-500, 500]^n$. Properties: Multi-modal, Separable, Scalable.

f_9 Schwefel's Problem 2.6 with Global Optimum on Bounds: f_5 from [83] with $n = 30$.

$$f_9(x) = max_i(|A_i x - B_i|) \tag{31}$$

where A is a $n \times n$ matrix, a_{ij} are integer random numbers in the range $[-500, 500]$, $det(A) \neq 0$, A_i is the ith row of A, $B_i = A_i \times o$, o is a $n \times 1$ vector, with o_i are random numbers in the range $[-100, 100]$, corresponding to the optimum. Decision space $D = [-100, 100]^n$. Properties: unimodal, non-separable, scalable.

f_{10} Schwefel's Problem 2.13: f_{12} from [83] with $n = 30$.

$$f_{10}(x) = \sum_{i=1}^{n} (A_i - B_i(x))^2 \tag{32}$$

where $A_i = \sum_{j=1}^{n} (a_{ij} sin\alpha_j + b_{ij} cos\alpha_j), B_i(x) = \sum_{j=1}^{n} (a_{ij} sinx_j + b_{ij} cosx_j)$, for $i = 1, \ldots, n$, A and B are two $n \times n$ matrices, a_{ij} and b_{ij} are integer random numbers in the range $[-100, 100]$, and the shifted optimum $\alpha = [\alpha_1, \alpha_2, \ldots, \alpha_n]$, with α_j random numbers in the range $[-\pi, \pi]$. Decision space $D = [-\pi, \pi]^n$. Properties: multi-modal, shifted, non-separable, scalable.

f_{11} Shifted rotated Ackley's function: f_6 from [69] with $n = 30$. The formula used is the same as Eq. (26), where $z = M(x - o)$, condition number of matrix M being equal to 1, and the shifted optimum $o = [o_1, o_2, \ldots, o_n]$. Decision space $D = [-32, 32]^n$. Properties: Multi-modal, Rotated, Shifted, Non-separable, Scalable.

f_{12} Shifted rotated Griewank's function: f_8 from [69] with $n = 30$. The formula used is the same as Eq. (27), where $z = M(x - o)$, condition number of matrix M being equal to 3, and the shifted optimum $o = [o_1, o_2, \ldots, o_n]$. Decision space $D = [-600, 600]^n$. Properties: Multi-modal, Rotated, Shifted, Non-separable, Scalable.

f_{13} Shifted rotated Rastrigin's function: f_{10} from [83] with $n = 30$. The formula used is the same as Eq. (28), where $z = M(x - o)$, condition number of matrix M being equal to 3, and the shifted optimum $o = [o_1, o_2, \ldots, o_n]$. Decision space $D = [-5, 5]^n$. Properties: Multi-modal, Shifted, Rotated, Separable, Scalable, huge number of local optima.

f_{14} Shifted Rotated Weierstrass Function: f_{11} from [83] with $n = 30$.

$$f_{14}(x) = \sum_{i=1}^{n} \sum_{k=0}^{k_{max}} (a^k cos 2\pi b^k (z_i + 0.5)) - n \sum_{k=0}^{k_{max}} (a^k cos 2\pi b^k \cdot 0.5) \tag{33}$$

where $a = 0.5$, $b = 0.3$, $k_{max} = 20$, $z = (x - o) \times M$, M a linear transformation matrix, and the shifted optimum $o = [o_1, o_2, \ldots, o_n]$. Decision space $D = [-0.5, 0.5]^n$. Properties: multi-modal, shifted, rotated, non-separable, scalable, continuous but differentiable only on a set of points.

f_{15} Schwefel Problem 2.22: f_2 from [88] with $n = 10$.

$$f_{15}(x) = \sum_{i=1}^{n} |x_i| + \prod_{i=1}^{n} |x_i|. \tag{34}$$

Decision space $D = [-10, 10]^n$. Properties: Unimodal, Separable, Scalable.

f_{16} Schwefel Problem 2.21: f_4 from [88] with $n = 10$.

$$f_{16}(x) = \max_i |x_i|. \tag{35}$$

Decision space $D = [-100, 100]^n$. Properties: Unimodal, Non-separable, Scalable.

f_{17} Generalized penalized function 1: f_{12} from [88] with $n = 10$.

$$f_{17}(\mathbf{x}) = \frac{\pi}{n} \left\{ 10 \sin^2 \pi y_1 + \sum_{i=1}^{n} ((y_i - 1)^2 (1 + 10 \sin^2 \pi y_i)) + (y_n - 1)^2 \right\} + \sum_{i=1}^{n} u(x_i, 10, 100, 4) \tag{36}$$

where

$$y_i = 1 + \frac{1}{4}(x_i + 1) \tag{37}$$

and

$$u(x, a, k, m) = \begin{cases} k(x_i - a)^m & \text{if } x_i > a \\ 0 \quad \text{if } |x_i| \leqslant a \\ k(-x_i - a)^m & \text{if } x_i < -a \end{cases} \tag{38}$$

Decision space $D = [-50, 50]^n$. Properties: Multi-modal, Separable, Scalable.

f_{18} Generalized penalized function 2: f_{13} from [88] with $n = 10$.

$$f_{18}(\mathbf{x}) = \frac{1}{10}\left\{\sin^2 3\pi x_1 + \sum_{i=1}^{n-1}((x_i - 1)^2(1 + \sin^2 3\pi x_{i+1}))\right\} + \frac{1}{10}\{(x_n - 1)(1 + \sin 2\pi x_n)^2\} + \sum_{i=1}^{n} u(x_i, 5, 100, 4) \tag{39}$$

Decision space $D = [-50, 50]^n$. Properties: Multi-modal, Separable, Scalable.

f_{19} Shifted rotated Rastrigin's function: the same as Eq. (28), with same bounds and $n = 50$. Properties: Multi-modal, Shifted, Rotated, Separable, Scalable, huge number of local optima.

f_{20} Michalewicz's function: from [47] with $n = 50$.

$$f_{20}(x) = -\sum_{i=1}^{n} \sin(x_i)\left[\sin\left(\frac{ix_i^2}{\pi}\right)\right]^{2m} \tag{40}$$

where $m = 10$. Decision space $D = [0, \pi]^n$. Properties: Multi-modal, Separable, Scalable.

f_{21} Schwefel's function (see Eq. (30)), with same bounds and $n = 50$. Properties: Multi-modal, Separable, Scalable.

f_{22} Shifted Ackley's function: f_5 from [69] (see Eq. (26)), with $n = 100$. Decision space $D = [-32, 32]^n$. Properties: Multi-modal, Shifted, Non-separable, Scalable.

f_{23} Alpine function from [13], with $n = 100$.

$$f_{23}(x) = \prod_{i=1}^{n} \sin(x_i)\sqrt{\prod_{i=1}^{n}(x_i)} \tag{41}$$

Decision space $D = [-10, 10]^n$. Properties: Multi-modal (with one global optimum), Non-separable, Scalable.

f_{24} Axis Parallel Hyper-ellipsoid function: f_2 from [50], with $n = 100$.

$$f_{24}(x) = \sum_{i=1}^{n}(i \cdot x_i^2) \tag{42}$$

Decision space $D = [-10, 10]^n$. Properties: Unimodal, Separable, Scalable.

f_{25} Drop Wave function: f_{18} from [50], with $n = 100$.

$$f_{25}(x) = -\frac{1 + \cos\left(12\sqrt{\sum_{i=1}^{n} x_i^2}\right)}{\frac{1}{2}\sum_{i=1}^{n} x_i^2 + 2} \tag{43}$$

Decision space $D = [-5.12, 5.12]^n$. Properties: Multi-modal, Separable, Scalable.

f_{26} Michalewicz's function (see Eq. (40)), with same bounds and $n = 100$. Properties: Multi-modal, Separable, Scalable.

f_{27} Moved Axis Hyper-ellipsoid function: f_{14} from [46], with $n = 100$.

$$f_{27}(x) = \sum_{i=1}^{n} 5i \cdot x_i^2 \tag{44}$$

Decision space $D = [-5.12, 5.12]^n$. Properties: Unimodal, Separable, Scalable.

f_{28} Shifted Rastrigin's function: f_9 from [83] (see Eq. (28)), with $n = 100$. Decision space $D = [-5, 5]^n$. Properties: Multi-modal, Shifted, Separable, Scalable, huge number of local optima.

f_{29} Rosenbrock's function: f_3 from [69] (see Eq. (25)), with $n = 100$. Decision space $D = [-100, 100]^n$. Properties: Multi-modal, Separable, Scalable.

f_{30} Rotated Hyper-ellipsoid function: f_3 from [50], with $n = 100$.

$$f_{30}(x) = \sum_{i=1}^{n}\sum_{j=1}^{i} x_j^2 \tag{45}$$

Decision space $D = [-65, 536, 65, 536]^n$. Properties: Unimodal, Non-separable, Scalable.

f_{31} Schwefel's function (see Eq. (30)), with same bounds and $n = 100$. Properties: Multi-modal, Separable, Scalable.

f_{32} Shifted sphere function: f_1 from [85] (see Eq. (23)), with same bounds and $n = 100$. Properties: Unimodal, Shifted, Separable, Scalable.

f_{33} Shifted Schwefel Problem 2.21: f_2 from [85] with $n = 100$.

$$f_{33}(x) = \max_i |z_i| \tag{46}$$

where $z = x - o$ and the shifted optimum $o = [o_1, o_2, \ldots, o_n]$. Decision space $D = [-100, 100]^n$. Properties: Unimodal, Shifted, Non-separable, Scalable.

f_{34} Shifted Rosenbrock's Function: f_3 from [85] (see Eq. (25)), with $n = 100$. Properties: Multi-modal, Shifted, Non-separable, Scalable.

f_{35} Shifted Rastrigin's Function: f_4 from [85] (see Eq. (28)), with same bounds and $n = 100$. Properties: Multi-modal, Shifted, Separable, Scalable, huge number is huge of local optima.

f_{36} Shifted Griewank's Function: f_5 from [85] (see Eq. (27)), with same bounds and $n = 100$. Properties: Multi-modal, Shifted, Non-separable, Scalable.

f_{37} Shifted Ackley's Function: f_6 from [85] (see Eq. (26)), with same bounds and $n = 100$. Properties: Multi-modal, Shifted, Non-separable, Scalable.

f_{38} FastFractal DoubleDip Function: f_7 from [85] with $n = 100$.

$$f_{38}(x) = \sum_{i=1}^{n} fractal1D(x_i + twist(x_{(i \bmod n)+1})) \tag{47}$$

$$twist(x) = 4(y^4 - 2^3 + y^2) \tag{48}$$

$$fractal1D(x) \approx \sum_{k=1}^{3} \sum_{1}^{2^{k-1}} \sum_{1}^{ran2(o)} doubledip\left(x, ran1(o), \frac{1}{2^{k-1}(2 - ran1(o))}\right) \tag{49}$$

$$doubledip(x, c, s) = \begin{cases} (-6144(x-c)^6 - 3088(x-c)^4 - 392(x-c)^2 + 1)s & -0.5 < x < 0.5 \\ 0 & \text{otherwise} \end{cases} \tag{50}$$

where $ran1(o)$ and $ran2(o)$ are, respectively, a double and an integer, pseudo-randomly chosen, with seed o, with equal probability from the interval $[0,1]$ and the set $\{0,1,2\}$. Decision space $D = [-1,1]^n$. Properties: Multi-modal, Non-separable, Scalable.

f_{39} Shifted sphere function: f_1 from [83] (see Eq. (23)), with same bounds and $n = 50$. Properties: Unimodal, Shifted, Separable, Scalable.

f_{40} Shifted Schwefel's Problem 1.2: f_2 from [83] (see Eq. (24)), with same bounds and $n = 50$. Properties: Unimodal, Shifted, Non-separable, Scalable.

f_{41} Rosenbrock's function: f_3 from [69] (see Eq. (25)), with same bounds and $n = 50$. Properties: Multi-modal, Non-separable, Scalable.

f_{42} Shifted Ackley's function: f_5 from [69] (see Eq. (26)), with same bounds and $n = 50$. Properties: Multi-modal, Shifted, Non-separable, Scalable.

f_{43} Shifted Griewank's function: f_7 from [69] (see Eq. (27)), with same bounds and $n = 50$. Properties: Multi-modal. Shifted, Non-separable, Scalable.

f_{44} Shifted Rastrigin's function: f_9 from [83] (see Eq. (28)), with same bounds and $n = 50$. Properties: Multi-modal, Shifted, Separable, Scalable, huge number of local optima.

f_{45} Shifted non continuous Rastrigin's function: f_{11} from [69] (see Eq. (29)), with same bounds and $n = 50$. Properties: Multi-modal, Shifted, Rotated, Separable, Scalable, huge number of local optima.

f_{46} Schwefel's Problem 2.6 with Global Optimum on Bounds: f_5 from [83] (see Eq. (24)), with same bounds and $n = 50$. Properties: unimodal, non-separable, scalable.

f_{47} Schwefel's Problem 2.13: f_{12} from [83] (see Eq. (32)), with same bounds and $n = 50$. Properties: multi-modal, shifted, non-separable, scalable.

References

[1] C.W. Ahn, H.-T. Kim, Estimation of particle swarm distribution algorithms: bringing together the strengths of PSO and EDAs, in: Proceedings of the ACM 11th Annual conference on genetic and evolutionary computation, 2009, pp. 1817–1818.

[2] C.W. Ahn, R.S. Ramakrishna, Elitism based compact genetic algorithms, IEEE Transactions on Evolutionary Computation 7 (4) (2003) 367–385.

[3] P.S. Andrews, An investigation into mutation operators for particle swarm optimization, in: Proceedings of the IEEE Congress on Evolutionary Computation, 2006, pp. 1044–1051.

[4] P.J. Angeline, Using selection to improve particle swarm optimization, in: Proceedings of the IEEE Congress on Evolutionary Computation, 1998, pp. 84–89.

[5] C. Aporntewan, P. Chongstitvatana, A hardware implementation of the Compact Genetic Algorithm, in: Proceedings of the IEEE Congress on Evolutionary Computation, vol. 1, 2001, pp. 624–629.

[6] A. Auger, N. Hansen, A restart CMA evolution strategy with increasing population size, In: Proceedings of the IEEE Congress on Evolutionary Computation, 2005, pp. 1769–1776.

[7] R. Baraglia, J.I. Hidalgo, R. Perego, A hybrid heuristic for the traveling salesman problem, IEEE Transactions on Evolutionary Computation 5 (6) (2001) 613–622.

[8] S. Boyd, C. Barratt, Linear Controller Design—Limits of Performance, Prentice-Hall, 1991.

[9] A. Caponio, G.L. Cascella, F. Neri, N. Salvatore, M. Sumner, A fast adaptive memetic algorithm for on-line and off-line control design of PMSM drives, IEEE Transactions on System Man and Cybernetics-Part B, Special Issue on Memetic Algorithms 37 (1) (2007) 28–41.

[10] A. Caponio, F. Neri, V. Tirronen, Super-fit control adaptation in memetic differential evolution frameworks, Soft Computing-A Fusion of Foundations, Methodologies and Applications 13 (8) (2009) 811–831.

[11] J.-F. Chang, S.-C. Chu, J.F. Roddick, J.-S. Pan, A parallel particle swarm optimization algorithm with communication strategies, Journal of Information Science and Engineering 21 (4) (2005) 809–818.

[12] G. Cheung, W. Tan, T. Yoshimura, Real-time video transport optimization using streaming agent over 3G wireless networks, IEEE Transactions on Multimedia 7 (4) (2005) 777–785.

[13] M. Clerc, Particle Swarm Optimization Webpage. http://clerc.maurice.free.fr/pso/ (accessed 18.10.11).
[14] M. Clerc, J. Kennedy, The particle swarm-explosion, stability and convergence in a multidimensional complex space, IEEE Transactions on Evolutionary Computation 6 (1) (2002) 58–73.
[15] W.J. Cody, Rational Chebyshev approximations for the error function, Mathematics of Computation 23 (107) (1969) 631–637.
[16] Cyber Dyne Srl Home Page, Kimeme, 2012. <http://cyberdynesoft.it/>.
[17] M. de Oca, T. Stutzle, K. Van den Enden, M. Dorigo, Incremental social learning in particle swarms, IEEE Transactions on Systems, Man, and Cybernetics, Part B, Cybernetics 41 (2) (2011) 368–384.
[18] Y. del Valle, G. Venayagamoorthy, S. Mohagheghi, J.-C. Hernandez, R. Harley, Particle swarm optimization: basic concepts, variants and applications in power systems, IEEE Transactions on Evolutionary Computation 12 (2) (2008) 171–195.
[19] R. Eberhart, Y. Shi, Comparing inertia weights and constriction factors in particle swarm optimization, in: Proceedings of the IEEE Congress on Evolutionary Computation, 2000, pp. 84–88.
[20] R. Eberhart, Y. Shi, Tracking and optimizing dynamic systems with particle swarms, in: Proceedings of the IEEE Congress Evolutionary Computation, 2001, pp. 94–100.
[21] A.E. Eiben, J.E. Smith, Introduction to Evolutionary Computation, Springer-Verlag, Berlin, 2003.
[22] H.-M. Feng, J.-H. Horng, S.-M. Jou, Bacterial foraging particle swarm optimization algorithm based fuzzy-vq compression systems, Journal of Information Hiding and Multimedia Signal Processing 3 (2) (2012) 227–239.
[23] J.C. Gallagher, S. Vigraham, A modified compact genetic algorithm for the intrinsic evolution of continuous time recurrent neural networks, in: Proceedings of the Genetic and Evolutionary Computation Conference, 2002, pp. 163–170.
[24] J.C. Gallagher, S. Vigraham, G. Kramer, A family of compact genetic algorithms for intrinsic evolvable hardware, IEEE Transactions Evolutionary Computation 8 (2) (2004) 111–126.
[25] S. Garcn++a, A. Fernn++ndez, J. Luengo, F. Herrera, A study of statistical techniques and performance measures for genetics-based machine learning: accuracy and interpretability, Soft Computing 13 (10) (2008) 959–977.
[26] W. Gautschi, Error Function and Fresnel Integrals, In: M. Abramowitz, I.A. Stegun (Eds.), Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables, 1972, pp. 297-309 (Chapter 7).
[27] S. Ghosh, S. Das, D. Kundu, K. Suresh, A. Abraham, Inter-particle communication and search-dynamics of lbest particle swarm optimizers: an analysis, Information Sciences 182 (1) (2012) 156–168.
[28] Harik, G., 1999. Linkage Learning via Probabilistic Modeling in the ECGA. Tech. Rep. 99010, University of Illinois at Urbana-Champaign, Urbana, IL.
[29] G.R. Harik, F.G. Lobo, D.E. Goldberg, The compact genetic algorithm, IEEE Transactions on Evolutionary Computation 3 (4) (1999) 287–297.
[30] G.R. Harik, F.G. Lobo, K. Sastry, Linkage learning via probabilistic modeling in the extended compact genetic algorithm (ECGA), in: M. Pelikan, K. Sastry, E. Cantú-Paz (Eds.), Scalable Optimization via Probabilistic Modeling. Studies in Computational intelligence, vol. 33, Springer, 2006, pp. 39–61.
[31] S. Holm, A simple sequentially rejective multiple test procedure, Scandinavian Journal of Statistics 6 (2) (1979) 65–70.
[32] G. Iacca, R. Mallipeddi, E. Mininno, F. Neri, P.N. Suganthan, Super-fit and population size reduction mechanisms in compact differential evolution, in: Proceedings of IEEE Symposium on Memetic Computing, 2011, pp. 21–28.
[33] G. Iacca, F. Neri, E. Mininno, Opposition-based learning in compact differential evolution, in: Evo Applications 2011 Part I, Lecture Notes in Computer Science, vol. 6624, Springer, 2011, pp. 264–273.
[34] G. Iacca, F. Neri, E. Mininno, Y.-S. Ong, M.-H. Lim, Ockham's Razor in memetic computing: three stage optimal memetic exploration, Information Sciences 188 (2012) 17–43.
[35] S. Janson, M. Middendorf, A hierarchical particle swarm optimizer and its adaptive variant, IEEE Transactions on Systems Man and Cybernetics, B 35 (6) (2005) 1272–1282.
[36] Y. Jewajinda, P. Chongstitvatana, Cellular compact genetic algorithm for evolvable hardware, in: Proceedings of the International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology, vol. 1, 2008, pp. 1–4.
[37] M.-J. Jung, H. Myung, H.-K. Lee, S. Bang, Ambiguity resolving in structured light 2D range finder for SLAM operation for home robot applications, in: Proceedings of the IEEE Workshop on Advanced Robotics and its Social Impacts, 2005, pp. 18–23.
[38] J. Kennedy, R.C. Eberhart, Particle swarm optimization, in: Proceedings of IEEE International Conference on Neural Networks, 1995, pp. 1942–1948.
[39] J. Kennedy, R. Mendes, Population structure and particle swarm performance, in: Proceedings of the IEEE Congress on Evolutionary Computation, 2002, pp. 1671–1676.
[40] S. Kiranyaz, T. Ince, A. Yildirim, M. Gabbouj, Fractional particle swarm optimization in multidimensional search space, IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics 40 (2) (2010) 298–319.
[41] P. Lanzi, L. Nichetti, K. Sastry, D.E. Goldberg, Real-coded extended compact genetic algorithm based on mixtures of models, in: Linkage in Evolutionary Computation, Studies in Computational Intelligence, vol. 157, Springer, 2008, pp. 335–358.
[42] P. Larrañaga, J.A. Lozano, Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation, Kluwer, 2001.
[43] J.J. Liang, A.K. Qin, P.N. Suganthan, S. Baskar, Comprehensive learning particle swarm optimizer for global optimization of multimodal functions, IEEE Transactions on Evolutionary Computation 10 (3) (2006) 281–295.
[44] R. Mallipeddi, P.N. Suganthan, Q.K. Pan, M.F. Tasgetiren, Differential evolution algorithm with ensemble of parameters and mutation strategies, Applied Soft Computing 11 (2) (2011) 1679–1696.
[45] R. Mendes, J. Kennedy, J. Neves, The fully informed particle swarm: simpler, maybe better, IEEE Transactions on Evolutionary Computation 8 (3) (2004) 204–210.
[46] A.-M. Miao, X.-L. Shi, J.-H. Zhang, E.-Y. Wang, S.-Q. Peng, A modified particle swarm optimizer with dynamical inertia weight, In: B. Cao, T.-F. Li, C.-Y. Zhang (Eds.), Fuzzy Information and Engineering Volume 2, Advances in Intelligent and Soft Computing, vol. 62, Springer, 2009, pp. 767–776.
[47] Z. Michalewicz, Genetic Algorithms + Data Structures = Evolution Programs, Springer-Verlag, 1992.
[48] E. Mininno, F. Cupertino, D. Naso, Real-valued compact genetic algorithms for embedded microcontroller optimization, IEEE Transactions on Evolutionary Computation 12 (2) (2008) 203–219.
[49] E. Mininno, F. Neri, F. Cupertino, D. Naso, Compact differential evolution, IEEE Transactions on Evolutionary Computation 15 (1) (2011) 32–54.
[50] M. Molga, C. Smutnicki, Test functions for optimization needs (c), 2005, 1–43. <http://www.zsd.ict.pwr.wroc.pl/files/docs/functions.pdf>.
[51] M. Montes de Oca, T. Stutzle, M. Birattari, M. Dorigo, Frankenstein's PSO: a composite particle swarm optimization algorithm, IEEE Transactions on Evolutionary Computation 13 (5) (2009) 1120–1132.
[52] M. Nasir, D. Maity, S. Das, S. Sengupta, U. Haldar, P.N. Suganthan, A dynamic neighborhood learning based particle swarm optimizer for global numerical optimization, Information Sciences 209 (20) (2012) 16–36.
[53] F. Neri, C. Cotta, Memetic algorithms and memetic computing optimization: a literature review, Swarm and Evolutionary Computation 2 (2012) 1–14.
[54] F. Neri, C. Cotta, P. Moscato, Handbook of Memetic Algorithms, Studies in Computational Intelligence, vol. 379, Springer, 2011.
[55] F. Neri, G. Iacca, E. Mininno, Disturbed Exploitation compact differential evolution for limited memory optimization problems, Information Sciences 181 (12) (2011) 2469–2487.
[56] F. Neri, G. Iacca, E. Mininno, Compact optimization, in: I. Zelinka, V. Snàšel, A. Abraham (Eds.), Handbook of Optimization: From Classical to Modern Approach, vol. 38, Springer, 2013, pp. 337–364.
[57] F. Neri, E. Mininno, Memetic compact differential evolution for cartesian robot control, IEEE Computational Intelligence Magazine 5 (2) (2010) 54–65.
[58] F. Neri, E. Mininno, T. Kärkkäinen, Noise analysis compact genetic algorithm, in: Applications of Evolutionary Computation, Lecture Notes in Computer Science, vol. 6024, Springer, 2010, pp. 602–611.
[59] F. Neri, V. Tirronen, Recent advances in differential evolution: a review and experimental analysis, Artificial Intelligence Review 33 (1–2) (2010) 61–106.

[60] P.G. Norman, The new AP101S general-purpose computer (GPC) for the space shuttle, IEEE Proceedings 75 (1987) 308–319.
[61] A. Okazaki, T. Senoo, J. Imae, T. Kobayashi, G. Zhai, Real-time optimization for cleaner-robot with multi-joint arm, in: Proceedings of the International Conference on Networking, Sensing and Control, 2009, pp. 885–890.
[62] K.E. Parsopoulos, M.N. Vrahatis, On the computation of all global minimizers through particle swarm optimization, IEEE Transactions on Evolutionary Computation 8 (3) (2004) 221–224.
[63] M.E.H. Pedersen, Good Parameters for Particle Swarm Optimization, Tech. Rep. HL1001, Hvass Lab., 2010.
[64] F. Peng, K. Tang, G. Chen, X. Yao, Population-based algorithm portfolios for numerical optimization, IEEE Transactions on Evolutionary Computation 14 (5) (2010) 782–800.
[65] S.A. Pourmousavi, M.H. Nehrir, C.M. Colson, C. Wang, Real-time energy management of a stand-alone hybrid wind-microturbine energy system using particle swarm optimization, IEEE Transactions on Sustainable Energy 1 (3) (2010) 193–201.
[66] K.V. Price, R. Storn, J. Lampinen, Differential Evolution: A Practical Approach to Global Optimization, Springer, 2005.
[67] A. Prügel-Bennett, Benefits of a population: five mechanisms that advantage population-based algorithms, IEEE Transactions on Evolutionary Computation 14 (4) (2010) 500–517.
[68] P. Puranik, P. Bajaj, A. Abraham, P. Palsodkar, A. Deshmukh, Human perception-based color image segmentation using comprehensive learning particle swarm optimization, Journal of Information Hiding and Multimedia Signal Processing 2 (2) (2011) 227–235.
[69] A.K. Qin, V.L. Huang, P.N. Suganthan, Differential evolution algorithm with strategy adaptation for global numerical optimization, IEEE Transactions on Evolutionary Computation 13 (2009) 398–417.
[70] R. Rastegar, A. Hariri, A step forward in studying the compact genetic algorithm, Evolutionary Computation 14 (3) (2006) 277–289.
[71] A. Ratnaweera, S.K. Halgamuge, H.C. Watson, Self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients, IEEE Transactions on Evolutionary Computation 8 (3) (2004) 240–255.
[72] G. Rudolph, Self-adaptive mutations may lead to premature convergence, IEEE Transactions on Evolutionary Computation 5 (4) (2001) 410–414.
[73] K. Sastry, D.E. Goldberg, On Extended Compact Genetic Algorithm. Tech. Rep. 2000026, University of Illinois at Urbana-Champaign, Urbana, IL, 2000.
[74] K. Sastry, D.E. Goldberg, D.D. Johnson, Scalability of a hybrid extended compact genetic algorithm for ground state optimization of clusters, Materials and Manufacturing Processes 22 (5) (2007) 570–576.
[75] K. Sastry, G. Xiao, Cluster Optimization Using Extended Compact Genetic Algorithm. Tech. Rep. 2001016, University of Illinois at Urbana-Champaign, Urbana, IL. 2001.
[76] L.J. Shadle, J.C. Ludlow, J.S. Mei, C. Guenther, Circulating fluid-bed technology for advanced power systems, in: Vision 21 Program Review Meeting, 2001.
[77] Y. Shi, R. Eberhart, A modified particle swarm optimizer, in: Proc. IEEE Int. Conf. Evolutionary Computation, 1998, pp. 69–73.
[78] Y. Shi, R. Eberhart, Empirical study of particle swarm optimization, in: Proc. IEEE Int. Conf. Evolutionary Computation, 1999, pp. 1945–1950.
[79] Y. Shi, H. Liu, L. Gao, G. Zhang, Cellular particle swarm optimization, Information Sciences 181 (20) (2011) 4460–4493.
[80] L. Shutao, T. Mingkui, I. Tsang, J.-Y. Kwok, A hybrid PSO-BFGS strategy for global optimization of multimodal functions, IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics 41 (4) (2011) 1003–1014.
[81] Z. Song, A. Kusiak, Multiobjective optimization of temporal processes, IEEE Transactions on Systems, Man, and Cybernetics, Part B 40 (3) (2010) 845–856.
[82] P.N. Suganthan, Particle swarm optimizer with neighborhood operator, in: Proceedings of the IEEE Congress on Evolutionary Computation, 1999, pp. 1958–1962.
[83] P.N. Suganthan, N. Hansen, J.J. Liang, K. Deb, Y.-P. Chen, A. Auger, S. Tiwari, Problem Definitions and Evaluation Criteria for the CEC 2005 Special Session on Real-Parameter Optimization, Tech. Rep. 2005005, Nanyang Technological University and KanGAL, Singapore and IIT Kanpur, India, 2005.
[84] C.L. Sun, J.C. Zeng, J.S. Pan, An improved vector particle swarm optimization for constrained optimization problems, Information Sciences 181 (6) (2011) 1153–1163.
[85] K. Tang, X. Yao, P.N. Suganthan, C. MacNish, Y.P. Chen, C.M. Chen, Z. Yang, Benchmark Functions for the CEC'2008 Special Session and Competition on Large Scale Global Optimization, Tech. Rep., Nature Inspired Computation and Applications Laboratory, USTC, China, 2007
[86] I.C. Trelea, The particle swarm optimization algorithm: convergence analysis and parameter selection, Information Processing Letters 85 (6) (2003) 317–325.
[87] F. van den Bergh, A.P. Engelbrecht, A cooperative approach to particle swarm optimization, IEEE Transactions on Evolutionary Computation 8 (3) (2004) 225–239.
[88] J. Vesterstrøm, R. Thomsen, A comparative study of differential evolution particle swarm optimization and evolutionary algorithms on numerical benchmark problems, in: Proceedings of the IEEE Congress on Evolutionary Computation, vol. 3, 2004, pp. 1980–1987.
[89] J. Vrugt, B. Robinson, J. Hyman, Self-adaptive multimethod search for global optimization in real-parameter spaces, IEEE Transactions on Evolutionary Computation 13 (2) (2009) 243–259.
[90] H. Wang, Z. Wu, S. Rahnamayan, Y. Liu, M. Ventresca, Enhancing particle swarm optimization using generalized opposition-based learning, Information Sciences 181 (20) (2011) 4699–4714.
[91] Y. Wang, Y. Yang, Particle swarm optimization with preference order ranking for multi-objective optimization original research article, Information Sciences 179 (12) (2009) 1944–1959.
[92] F. Wilcoxon, Individual comparisons by ranking methods, Biometrics Bulletin 1 (6) (1945) 80–83.
[93] D. Wolpert, W. Macready, No free lunch theorems for optimization, IEEE Transactions on Evolutionary Computation 1 (1) (1997) 67–82.
[94] T. Yamaguchi, K. Yasuda, Adaptive particle swarm optimization: self-coordinating mechanism with updating information, in: Proceedings of the IEEE International Conference on System Man, Cybernetics, 2006, pp. 2303–2308.
[95] S.X. Yang, C. Luo, A neural network approach to complete coverage path planning, IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics 34 (1) (2004) 718–724.
[96] R.E. Young, Petroleum refining process control and real-time optimization, IEEE Control Systems 26 (6) (2006) 73–83.
[97] Y.P. Chen, W.C. Peng, M.C. Jian, Particle swarm optimization with recombination and dynamic linkage discovery, IEEE Transactions on System, Man, and Cybernetics B 37 (6) (2007) 1460–1470.
[98] Z.-H. Zhan, J.Z.Y. Li, H.S.-H. Chung, Adaptive particle swarm optimization, IEEE Transactions on Systems, Man, and Cybernetics B 39 (6) (2009) 1362–1381.
[99] Z.-H. Zhan, J. Zhang, Y. Li, Y.-H. Shi, Orthogonal learning particle swarm optimization, IEEE Transactions on Evolutionary Computation 15 (6) (2011) 832–847.
[100] J. Zhang, A.C. Sanderson, JADE: adaptive differential evolution with optional external archive, IEEE Transactions on Evolutionary Computation 13 (5) (2009) 945–958.
[101] W.J. Zhang, X.F. Xie, DEPSO: Hybrid particle swarm with differential evolution operator, in: Proceedings of the IEEE International Conference on System Man, Cybernetics, 2003, pp. 3816–3821.
[102] Y.-L. Zheng, L.-H. Ma, L.-Y. Zhang, J.-X. Qian, Empirical study of particle swarm optimizer with an increasing inertia weight, in: Proc. IEEE Congr. Evol. Comput., 2003, pp. 221–226.
[103] Y.-L. Zheng, L.-H. Ma, L.-Y. Zhang, J.-X. Qian, On the convergence analysis and parameter selection in particle swarm optimization, in: Proc. IEEE Int. Conf. Mach. Learning Cybern., 2003, pp. 1802–1807.
[104] J. Zhou, Z. Ji, L. Shen, Simplified intelligence single particle optimization based neural network for digit recognition, in: Proceedings of the Chinese Conference on Pattern Recognition, 2008, 10311847.