Information Sciences 297 (2015) 216–235

Contents lists available at ScienceDirect

Information Sciences

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

Cluster-Based Population Initialization for differential evolution frameworks

Ilpo Poikolainen [a,1], Ferrante Neri [a,b,*], Fabio Caraffini [a,b,1]

[a] Department of Mathematical Information Technology, University of Jyväskylä, P.O. Box 35 (Agora), Jyväskylä 40014, Finland
[b] Centre for Computational Intelligence, School of Computer Science and Informatics, De Montfort University, The Gateway, Leicester LE1 9BH, England, United Kingdom

ARTICLE INFO

Article history:
Received 30 April 2014
Received in revised form 30 October 2014
Accepted 5 November 2014
Available online 15 November 2014

Keywords:
Differential evolution
Memetic computing
Intelligent sampling
Evolutionary computation

ABSTRACT

This article proposes a procedure to perform an intelligent initialization for population-based algorithms. The proposed pre-processing procedure, namely Cluster-Based Population Initialization (CBPI) consists of three consecutive stages. At the first stage, the individuals belonging to a randomly sampled population undergo two subsequent local search algorithms, i.e. a simple local search that performs moves along the axes and Rosenbrock algorithm. At the second stage, the solutions processed by the two local searches undergo the *K*-means clustering algorithm and are grouped into sets on the basis of their euclidean distance. At the third stage the best individuals belonging to each cluster are saved into the initial population of a generic optimization algorithm. If the population has not been yet filled, the other individuals of the population are sampled within the clusters by using a fitness-based probabilistic criterion. This three stage procedure implicitly performs an initial screening of the problem features in order to roughly estimate the most interesting regions of the decision space. The proposed CBPI has been tested on multiple classical and modern Differential Evolution variants, on a wide array of test problems and dimensionality values as well as on a real-world problem. The proposed intelligent sampling appears to have a significant impact on the algorithmic functioning as it consistently enhances the performance of the algorithms with which it is integrated.

1. Introduction

Engineering and natural sciences often require the solution of multiple optimization problems. This fact makes the study of optimization methods extremely important in fields such as design and control engineering. Since only a very limited number of real-world optimization problems can be solved by exact methods, in the vast majority of cases an optimizer that does not require a specific hypothesis must be used. Over the past decades, computer scientists have designed a multitude of these types of algorithms for addressing real-world problems where an exact approach is almost never applicable. These methods, known as metaheuristics, do not offer guarantees regarding the convergence, but still are capable to detect high

* Corresponding author at: Centre for Computational Intelligence, School of Computer Science and Informatics, De Montfort University, The Gateway, Leicester LE1 9BH, England, United Kingdom. Tel.: +358 14 260 1211; fax +358 14 260 1021.

E-mail addresses: ilpo.poikolainen@jyu.fi (I. Poikolainen), fneri@dmu.ac.uk, ferrante.neri@jyu.fi (F. Neri), fcaraffini@dmu.ac.uk, fabio.caraffini@jyu.fi (F. Caraffini).

1 Tel.: +358 14 260 1211; fax +358 14 260 1021.

http://dx.doi.org/10.1016/j.ins.2014.11.026
0020-0255/© 2014 Elsevier Inc. All rights reserved.

quality solutions that can be of great interest for engineers and practitioners. Among the plethora of metaheuristics some are Evolutionary Algorithms (EAs) [26], Swarm Intelligence (SI) [25], and Memetic Computing (MC) [50].

For about two decades, i.e. from the 1970s to 1990s, computer scientists have put much effort to design metaheuristics with the intention of detecting an algorithm that could outperform all the other algorithms. After the publication of the No Free Lunch (NFL) Theorems [71], the view on optimization of scientists and practitioners underwent a radical modification. The NFL Theorems prove that all the optimization algorithms, under the hypotheses that they search within a set of finite candidate solutions and that the algorithms never visit the same point/candidate solution twice, display the same performance when averaged over all the possible optimization problems. As an immediate consequence, it was clear that it was no longer useful to discuss which algorithm was universally better or worse. Despite of the fact that the hypotheses of NFL Theorems are often not realistic (for example, it is very unlikely that an EA does not generate the same point twice during a run), a large portion of algorithmic design community started to propose algorithms, which were tailored to specific problems, see e.g. [64,14,53], instead of trying to propose universally applicable algorithms. On the other hand, by using the non-realism of NFL Theorems' hypotheses as an argument, another portion of the optimization community in recent years researchers have attempted to push towards the outer limit of these theorems by proposing relatively flexible algorithmic structures that combine (to some extent) robustness and high performance on various problems. This tendency is especially clear in continuous optimization and for those algorithms characterized by adaptively coordinated heterogeneous algorithmic components. For these two sub-fields, the NFL Theorems are proved to be not verified, see e.g. [3,58], respectively.

Since modern algorithms for continuous optimization are often composed of multiple adaptively coordinated operators, these two sub-fields are not disjointed. For example, in the context of Differential Evolution (DE), the optimizer proposed in [59] combines and coordinates multiple mutation strategies by making use of a learning period and a randomized success based logic (see also [22,52]). In [44] another DE based strategy namely *ensemble* has been presented on the basis of strategy used in Evolutionary Programming proposed in [43]. In the ensemble multiple mutation and crossover strategies, as well as the related parameters are encoded within the solutions and evolve with them. Other harmonic self-adaptive combinations of components within the DE framework are proposed in [8,7]. In the context of Particle Swarm Optimization (PSO), a harmonic coordination of multiple components is also a popular option to enhance the algorithmic robustness over a range of problems. An emblematic example of this strategy is the so-called Frankenstein's PSO [45]. A more elegant algorithm that coordinates, in a simplistic way, a perturbation logic with a variable decomposition is proposed in [37].

Some studies focus on the coordination techniques in order to have a robust behavior of the algorithm. Several nomenclatures are used in different contexts to express fairly similar concepts. With the term *portfolio* it is usually referred to algorithmic frameworks composed of optimizers that are alternatively selected during the run time. The selection criteria can be a simple schedule or a more sophisticated adaptive system. Some examples in the context of continuous optimization are given in [68,57]. In the context of combinatorial optimization, and more specifically for the maximum satisfiability problem, a popular portfolio named SATzilla platform, see [72,32], has been proposed. The difficulty of finding a trade-off between the search algorithms and the aim at determining an automatic coordination system is studied in [33]. A model of the behavior of optimizers in order to predict their run time is presented in [34]. Very closely related to the concept of portfolio, *hyper-heuristics* are composed of multiple algorithms usually coordinated by a machine learning algorithm which takes a supervisory role. This term is in the vast majority of cases, to combinatorial problems. Famous examples of hyper-heuristic have been proposed in [20,11] in the field of timetabling and rostering while in [12] graph coloring heuristics are coupled with a random ordering heuristic. An important concept in hyper-heuristic implementation is the choice function, that is a criterion that assigns a rewarding score to the most promising heuristic, see [20]. More sophisticated coordination schemes present in the literature make use of reinforcement learning in a stand alone or combined fashion, see e.g. [11,23], and memory-based mechanisms, see [10]. Elegant learning schemes coupled with multiple operators (multi-agents) for addressing complex optimization problems are presented in [2,1].

Closely related to hyper-heuristics and portfolio algorithms, Memetic Algorithms (MAs) are optimization algorithms composed of an evolutionary framework and a set of local searchers activated within the generation cycle, see [46,30]. In MAs, as for the related algorithmic families, optimization is carried out by multiple components/sub-algorithms but unlike them, emphasize the global and local search roles of its components. Although the one may argue that there is no clear definition of global and local search (e.g. a DE with a proper tuning can be used as a local search), the term MA is broadly used to refer population-based hybrid algorithms. Moreover, modern MA implementation ignore the original definition that the population-based framework should be evolutionary and refer as MAs also those algorithms based on a SI framework, see e.g. [69,66]. Recently, the concept of MA has been extended to single-solution algorithms or to any algorithm composed of multiple/heterogeneous components. In the latter case the subject is termed, by a part of the computer science community, as Memetic Computing (MC) and its implementation as MC structures, see e.g. [50,49,54,55].

Regardless of the used nomenclature, an important issue, that is also the focus of this paper, is the generation of an initial population in population-based hybrid algorithms. Nearly all the population-based metaheuristics start with the random sampling of a prefixed amount of points within the decision space. This choice can be explained by the motivation, "since we have no a priori knowledge on the problem, we give to each possible candidate solution the same chance to be in the starting population". Obviously, there is nothing wrong in this way of reasoning. Moreover, this initialization has the undoubted advantage that is computationally cheap as it does not require objective function evaluation nor other complex operations. On the other hand, for every problem, there likely exists many other strategies that can lead to much better results. Similar in the motivation, but very different in the implementation, a fully deterministic procedure that spreads the points in the decision space is also possible, see [41]. In the latter case, the motivation can be summarized as "since we have no a priori knowledge on the problem, we try to sample the initial points in a way that they cover the decision space as much as possible". This choice besides being computationally expensive includes an implicit drawback. The minimum amount of points necessary to cover the decision space grows exponentially with the dimensionality of the problem. Hence, in high dimensions a very (unreasonable) large amount of points is required to have a representative search space coverage. Some studies on the degree of randomization of the initial population have been reported in the literature, especially about EAs, see [42,61]. It is shown that in the many cases a deterministic initial sampling can lead to a performance deterioration. On the other hand, a random sampling within mapped areas of the decision space, i.e. a quasi random sampling, leads to a robust algorithmic behavior without excessively jeopardizing the algorithmic performance with respect to a simple (pseudo-) random sampling.

Whenever there is some knowledge about the problem, a sampling that uses this knowledge can be used to enhance the algorithmic performance, see [24]. For example, in control engineering a initial tuning of the control parameters usually allows an estimation of the instability region and a vague estimation of the region of interest of the algorithm. An initial sampling in this region of the decision space can bias the search towards a quick detection of the optimum, see e.g. [14].

Although in the vast majority of cases an a priori knowledge of the problem is not available, there is always the possibility to perform at runtime a problem characterization in order to extract some features to be exploited in the subsequent stages of optimization, see [18]. A pioneering study in this direction proposed the selective sampling, [5]. This procedure consists of performing an initial random initialization containing a large amount of points followed by a tournament selection in order to shrink the population to those individuals that display the best performance.

Within the context of DE, the sampling of extra points according to a central symmetry (opposition-based points [60]) appear to be beneficial to the algorithmic performance. Another approach consists of applying a local search to one or more solutions and then insert these improved solutions into the initial population of an optimizer. The schemes that improved one solution and inserts into a DE initial population is termed super-fit and displayed a very good performance with respect to the same algorithm making use of a random population and whose budget is entirely devoted to DE, see [15,35,19].

This article proposes a novel algorithmic component for pre-processing the initial solutions and generating an initial population for DE algorithms. The proposed component does not require any assumption on the optimization problem (except it being continuous). More specifically, an initial screening of the problem is implicitly performed in order to detect the most promising regions of the decision space. This result is achieved by a multiple stage procedure. At first a set of point is sampled at random. Subsequently two local searchers with very different features are consecutively applied to them with a shallow depth. The resulting points are then clustered. The population of the optimization algorithm is then composed of those individuals belonging to each cluster that display the highest performance and by other points, sampled from their neighborhood, according to a probabilistic criterion. Thus, the initial population is composed of points displaying a good performance and spread in different basins of attraction. A graphical representation of the entire framework is presented in Fig. 1.

In different ways, the combination of clustering techniques within DE framework for global optimization is a topic that has been investigated. For example, paper [70] makes use of a clustering technique over the individuals of a DE population in order to prevent a diversity loss and premature convergence. Paper [13] uses the one step k-means clustering as a multi-parent crossover. This idea is developed in [39] where the k-means clustering is associated to two novel crossover operators. In the context of dynamic optimization problem, the algorithm in [29] uses a multi-population where each sub-population covers a different area of the decision space. The number of clustered populations is dynamically varied during the optimization by means of an adaptive logic.

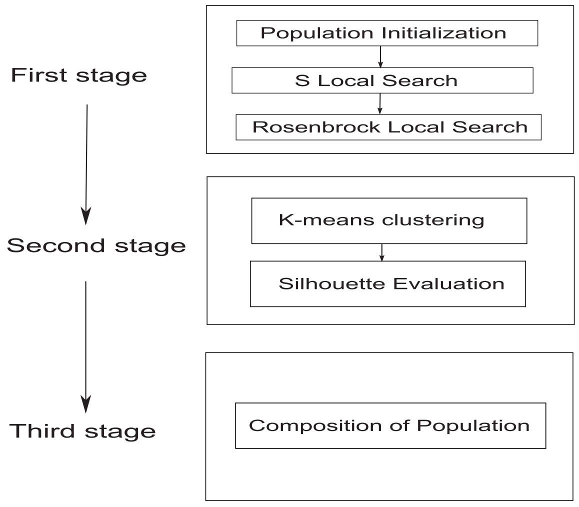

Fig. 1. Framework of three stages and their components.

The remainder of this article is organized in the following way. Section 2 gives a description of the proposed initialization procedure. Section 3 displays for a large set of problems the effect of the proposed initialization over multiple and diverse optimizers. Finally, Section 4 gives the conclusions of this work.

2. The proposed Cluster-Based Population Initialization

Without a loss of generality, in order to clarify the notation in this paper, we refer to the minimization problem of an objective function (or fitness) $f(\mathbf{x})$, where the candidate solution $\mathbf{x}$ is a vector of n design variables (or genes) in a decision space $\mathbf{D}$. Thus, the optimization problem considered in this paper consists of the detection of that solution $\mathbf{x}^* \in \mathbf{D}$ such that $f(\mathbf{x}^*) < f(\mathbf{x})$, and this is valid $\forall \mathbf{x} \in \mathbf{D}$. Array variables are highlighted in bold face throughout this paper.

Before entering into the implementation details, let us remark that, as explained in [51], optimization algorithms can be represented as the iterative application of a generation of a trial solution $\mathbf{x^t}$ (or population of solutions) by perturbation of a base point $\mathbf{x}$ (or population of points) and a selection strategy to update the base point(s).

2.1. First stage: Multiple local search

A set of S_{pop} initial points is randomly sampled within the decision space $\mathbf{D}$. Each point undergoes two consecutive local search components.

The first local search algorithm, here indicated with S, perturbs a single solution along its n axes, i.e. separately perturbs each design variable. This algorithm can be seen as a modification of a classical hill-descend algorithm and was used for the first time in [67] in order to address large scale problems. S perturbs the candidate by computing, for each coordinate i (each design variable), by applying the following formula:

$$x_i^t = x_i - \xi, \tag{1}$$

where ξ is the exploratory radius. Subsequently, if $\mathbf{x^t}$ outperforms $\mathbf{x}$, the solution $\mathbf{x}$ is updated (the values of $\mathbf{x^t}$ are copied in it), otherwise a half step in the opposite direction is performed:

$$x_i^t = x_i + \frac{\xi}{2}. \tag{2}$$

Again, $\mathbf{x^t}$ replaces $\mathbf{x}$ if it outperforms it. If there is no update, i.e. the exploration was unsuccessful, the radius ξ is halved. This exploration is repeated for all the design variables. The termination criterion of this local search is based only on the budget. For the sake of clarity, Algorithm 1 describes the working principles of the S operator.

Algorithm 1. Pseudo-code of the S local search.

```
1: INPUT $\mathbf{x}$
2: while condition on the local computational budget do
3:   for $i = 1 : n$ do
4:     $x_i^t = x_i - \xi$
5:     if $f(\mathbf{x^t}) \leqslant f(\mathbf{x})$ then
6:       $\mathbf{x} = \mathbf{x^t}$
7:     else
8:       $x_i^t = x_i + \frac{\xi}{2}$
9:       if $f(\mathbf{x^t}) \leqslant f(\mathbf{x})$ then
10:        $\mathbf{x} = \mathbf{x^t}$
11:      end if
12:    end if
13:  end for
14:  if $\mathbf{x} == \mathbf{x^t}$ then
15:    $\xi = \frac{\xi}{2}$
16:  end if
17: end while
18: OUTPUT $\mathbf{x}$
```

The second local search is the Rosenbrock algorithm (R), see [62]. This operator, has been proved to always converge to a local optimum, under specific conditions, see [4]. At the beginning of the optimization of this local search, R is similar to S as it explores each of the n directions, perturbing the input solution $\mathbf{x}$ with an initial step size vector $\mathbf{h}$. A matrix $\mathbf{A}$ is also initialized as the identity matrix. Each step of this algorithm consists of the following. As long as new improvements are found, for $j = 1, 2, \ldots, n$, a new trial point $\mathbf{x^t}$ is generated by perturbing each ith design variable of the solution $\mathbf{x}$ in the following way:

$$x_i^t = x_i + h_j \cdot A_{i,j} \tag{3}$$

for $i = 1, 2, \ldots, n$. In case of success (the trial solution outperforms the solution $\mathbf{x}$), $\mathbf{x}$ is updated and the step size is increased of a factor α ($h_j = \alpha \cdot h_j$), otherwise it is decreased by means of a factor β and the opposite direction is tried ($h_j = -\beta \cdot h_j$). As said, this procedure is repeated until it is possible to improve upon the solution $\mathbf{x}$. Once every possible success has been found and exploited in each base direction, the coordinate system is rotated towards the approximated gradient by means of the Gram-Schmidt orthogonalization procedure. This operation results into an update of the matrix $\mathbf{A}$. After the orthogonalization, the step size vector $\mathbf{h}$ is reinitialized and the procedure is repeated, using the rotated coordinate system, perturbing again the current elite $\mathbf{x}$ according to Eq. (3): it is important to notice that, when a rotated coordinate system is used, i.e. the matrix $\mathbf{A}$ is no longer an identity matrix, this trial generation mechanism corresponds to a diagonal move by following the direction given by the gradient. Besides, the normal conditions on a maximum computational budget, the Rosenbrock Algorithm may terminate when a stop criterion is met. The stop criterion is given by two conditions. The first criterion is based on the minimum element of the perturbation vector $\mathbf{h}$, the second is based on the minimum difference between $\mathbf{x^t}$ and $\mathbf{x}$ design variables. More specifically, R is continued until the following condition is true:

$$\min(|\mathbf{h}|) > \varepsilon OR \min(|\mathbf{x^t} - \mathbf{x}|) > \varepsilon \tag{4}$$

where min() is the minimum vector element. If for a step there is no improvement at all, only the first condition is considered. In order to avoid undesired problems of numerical instability, the improved version proposed in [56] has been used in this article. The details of the used implementation are reported in Algorithm 2.

Algorithm 2. Pseudo-code of the Rosenbrock local search.

```
1: INPUT x
2: $\mathbf{h} = \frac{1}{10} ones(n)$ ($n$-dimensional vector of 0.1)
3: $\mathbf{A} = eye(n, n)$ ($n \times n$ identity matrix)
4: while condition on the local computational budget and stop criterion do
5:   while at least one step is unsuccessful do
6:     for $i = 1 : n$ do
7:       for $j = 1 : n$ do
8:         $x_i^t = x_i + h_j \cdot A_{i,j}$
9:         if $f(\mathbf{x^t}) \leqslant f(\mathbf{x})$ then
10:           $\mathbf{x} = \mathbf{x^t}$
11:           $h_j = \alpha \cdot h_j$ ($\alpha = 3$ in [62], 2 in [16])
12:         else
13:           $h_j = -\beta \cdot h_j$ $\beta = 0.5$ [62])
14:         end if
15:       end for
16:     end for
17:   end while
18:   estimate the gradient direction
19:   apply Gram-Schmidt orthogonalization where the first direction is the gradient estimation
20:   modify the matrix A on the basis of the orthogonalization
21: end while
22: OUTPUT x
```

It is worthwhile commenting our choice of these two local search algorithms and their consecutive activation. It must be noted that S and Rosenbrock algorithm are characterized by a very different search logic. More specifically, while S attempts to optimize the objective function by performing moves along the axes, Rosenbrock algorithm by following the direction of the gradients and changing the coordinates moves diagonally within the decision space. As observed in [17,18], the combination of these two local searchers allows an effective local optimization as the two search logics compensate each other. The main benefit of this combination is that S can efficiently handle separable optimization problems or characterized by a weak correlation among the variables whereas Rosenbrock algorithm has a great potential on non-separable problems or characterized by a strong correlation among the variables, see [18]. Furthermore, Rosenbrock appears efficient usually for low and medium dimensionality values while S is capable of quickly improving upon the initial fitness on large scale problems. In this paper, the consecutive application of these two algorithms is proposed in order to enhance upon the initial solutions and have a rough estimate of the location and number of local optima.

2.2. Second stage: K-means clustering

The candidate solutions, after the double local search improvement undergo a clustering stage. Clustering is a task consisting of grouping objects on the basis of their common features. In our case, objects to be clustered are points (candidate solutions) in a multi-dimensional space and the clustering criterion is based on their location within this space. In other words, the second stage of this pre-processing component has, as an input the list of the (locally improved) solutions and, as an output, the same solutions divided into multiple groups. Each group represents a potential local optimum. Since the number of local optima is, in general, unknown in optimization, multiple clustering scenarios are here considered. More specifically, this second stage performs the clustering for $K = 2, 3, \ldots, M$ clusters. Then a Silhouette technique is applied to determine the most suitable amount of clusters.

Each clustering scenario is computed by means of the K-means clustering technique implementation described in [40]. Briefly, according to this technique an initial set of K points $\mathbf{m_1}, \mathbf{m_2}, \ldots, \mathbf{m_K}$ are randomly sampled within the decision space. These points are termed centroids of the clusters and each of them identifies a set/group/cluster. Each of the improved candidate solutions is assigned to the corresponding cluster $\mathbf{S_k}$ on the basis of a minimum distance criterion. More formally, each cluster $\mathbf{S_k}$ is a set defined as

$$\mathbf{S_k} = \{\mathbf{x} : \|\mathbf{x} - \mathbf{m_k}\|^2 \leqslant \|\mathbf{x} - \mathbf{m_p}\|^2 : \forall p = 1 : K,\ p \neq k\}. \tag{5}$$

The K-means clustering algorithm consists of progressively updating the centroids and recalculating the points in the new configuration. For the generic step t, the equations to pass to the step $t + 1$ are:

Step 1:

$$\mathbf{S_k^{(t)}} = \{\mathbf{x^i} : \|\mathbf{x^i} - \mathbf{m_k^{(t)}}\|^2 \leqslant \|\mathbf{x^i} - \mathbf{m_p^{(t)}}\|^2 : \forall p = 1 : K,\ p \neq k\} \tag{6}$$

Step 2:

$$\mathbf{m_k^{(t+1)}} = \frac{1}{|\mathbf{S_k^{(t)}}|} \sum_{\mathbf{x} \in \mathbf{S_k^{(t)}}} \mathbf{x} \tag{7}$$

where $\mathbf{x^i}$ is the i-th candidate solution, $|\mathbf{S_k^{(t)}}|$ is the number of points belonging to the cluster $\mathbf{S_k^{(t)}}$ and $\sum_{\mathbf{x} \in \mathbf{S_k^{(t)}}} \mathbf{x}$ is the sum of the points belonging to the cluster $\mathbf{S_k^{(t)}}$.

In order to select the best cluster scenario, i.e. in order to select the best approximated detection of local optima, the Silhouette technique proposed in [63] has been used. This technique, for a given clustering scenario consists of the following steps. For each point $\mathbf{x^i}$ belonging to a cluster $\mathbf{S_k}$ the average distance with respect to the other cluster points $\mathbf{x^j}$ is calculated:

$$\mathbf{a^i} = \frac{1}{|\mathbf{S_k^{(t)}}|} \sum_{\mathbf{x} \in \mathbf{S_k^{(t)}}} \|\mathbf{x^i} - \mathbf{x^j}\|. \tag{8}$$

Then, the average distance between $\mathbf{x^i}$ and the points belonging to each of the other clusters is also calculated. The lowest of these distances is indicated with $\mathbf{b^i}$. For example, if 6 clusters are considered, a^i is the average distance between $\mathbf{x^i}$ and the

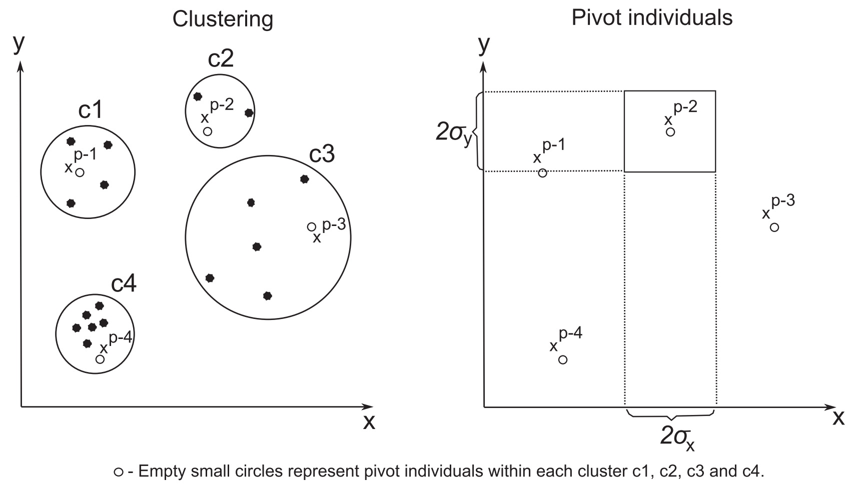
Clustering
Pivot individuals
y
x
c1
c2
c3
c4
p-1
p-2
p-3
p-4
$2\sigma_y$
$2\sigma_x$
○ - Empty small circles represent pivot individuals within each cluster c1, c2, c3 and c4.

Fig. 2. 2-dimensional illustration of clusters Ck and their respective pivot individuals x^{p-k}. In stage three pivot individuals are used as mean values with standard deviations σ_x and σ_y to fill in remaining population.

other points of the same cluster and 5 more numbers (average distances inter-clusters) are calculated. The lowest among these 5 distances is $\mathbf{b^i}$. The Silhouette related to the point $\mathbf{x^i}$ is defined as:

$$\mathbf{s^i} = \frac{\mathbf{a^i} - \mathbf{b^i}}{\max\{\mathbf{a^i}, \mathbf{b^i}\}}. \tag{9}$$

It can be easily observed that $\mathbf{s^i}$ can take values between -1 and 1. The Silhouette gives an estimate of the quality of the clustering with reference to the point $\mathbf{x^i}$, the more $\mathbf{s^i}$ is close to 1, the more accurate is the $\mathbf{x^i}$ classification. Hence, in order to perform an evaluation of the entire clustering the average Silhouette is calculated:

$$\langle \mathbf{s^i} \rangle = \frac{\sum_{\mathbf{i}=1}^{S_{pop}} \mathbf{s^i}}{S_{pop}}. \tag{10}$$

In order to select the most proper amount of clusters, the Silhouette is calculated for each cluster scenario and that one that has the best (closer to 1) Silhouette score is selected for the third stage. Algorithm 3 describes the working principles of the second stage.

Algorithm 3. Pseudo-code of the Second Stage.

```
1: INPUT $S_{pop}$ candidate solutions improved by two subsequent local search algorithms
2: for $K = 2 : M$ do
3:   generate $\mathbf{m_1}, \mathbf{m_2}, \ldots, \mathbf{m_K}$ random within the decision space
4:   for $i = 1 : S_{pop}$ do
5:     assess to what cluster $\mathbf{x^i}$
6:     update the corresponding centroid $\mathbf{m_k}$ (in the case of the $k$-th cluster)
7:   end for
8: end for
9: for $k = 1 : K$ do
10:   for $i = 1 : S_{pop}$ do
11:     compute the Silhouette $\mathbf{s^i}$
12:   end for
13:   compute and save the average Silhouette $\langle \mathbf{s^i} \rangle$
14: end for
15: select the best clustering scenario on the basis of the Silhouette value
16: OUTPUT $S_{pop}$ candidate solutions divided into $K$ clusters
```

2.3. Third stage: Composition of the population

In order to fill the population of the DE algorithm, two simple mechanisms are used. The first mechanism simply consists of deterministically including, for each cluster, the individual with the best fitness value, here termed cluster pivot and indicated with $\mathbf{x^{p-k}}$. Thus, the DE population will contain at least as many points as the number of clusters (K). In order to sample the other points, at first, a ranking of the clusters is performed on the basis of the fitness value of its best detected point. Then a scoring is assigned to each cluster. In our case, we have $K = 10$ clusters. In order to assign the scores we have heuristically chosen the so called "year 2000 formula-1 scoring". We have assigned the scores on the basis of the following schemes $\{10, 6, 5, 4, 3, 2, 1, 1, 1, 1\}$, i.e. the best cluster scores $sc^1 = 10$, the second $sc^2 = 6$, etc. When all the scores have been assigned, a probability is associated to each cluster according to the formula:

$$pc^k = \frac{sc^k}{\sum_{j=1}^{K} sc^j}. \tag{11}$$

On the basis of these probabilities, by means of a simple roulette wheel selection [26], the clusters are selected and a point is sampled from the selected cluster. The sampling occurs in the neighborhood of the points characterized by the best fitness. More specifically, each design variable of a point is sampled by a normal distribution whose mean value is the corresponding design variable of the cluster pivot and the standard deviation is $\sigma = 0.1 \cdot W_i$, where W is a width of design variable i. An example of the 2-dimensional case is shown in Fig. 2. The procedure is continued until the population contains the desired number of individuals.

For the sake of clarity, the pseudo-code showing the implementation details of the third stage is shown in Algorithm 4. For simplicity of the notation, it is assumed that the initial amount of sampled points S_{pop} is equal to the desired population size.

Algorithm 4. Pseudo-code of the Third Stage.

```
1: INPUT $S_{pop}$ candidate solutions divided into $K$ clusters
2: for $k = 1 : K$ do
3:    select, for each cluster the individual with the best fitness $\mathbf{x^{p-k}}$
4:    $\mathbf{Pop^k} = \mathbf{x^{p-k}}$
5: end for
6: for $k = 1 : K$ do
7:    assign to the cluster a score $sc^k$ on the basis of $f(\mathbf{x^{p-k}})$
8:    calculate $pc^k = \frac{sc^k}{\sum_{j=1}^{K} sc^j}$
9: end for
10: for $i = K + 1 : S_{pop}$ do
11:    generate a random number $rand(0,1)$
12:    select a cluster $k$ by means of the roulette wheel selection and the assigned probabilities
13:    for $j = 1 : n$ do
14:       $Pop_j^i = \mathcal{N}\left(x_j^{p-k}, \sigma\right)$
15:    end for
16: end for
17: OUTPUT Pop
```

2.4. Algorithmic philosophy, parameter setting, and budget issues

This section describes motivation and functioning of the proposed intelligent sampling software component in general and within the context of DE. As described above, the proposed CBPI is composed of three software sub-modules that sequentially operate in order to process an initially sampled random population. The first sub-module acting at the first stage has two roles: to improve upon the initial solutions and to attempt locally improve within the corresponding basin of attraction. The double local search is proposed in order to enhance the robustness of this operation. As mentioned above and reported in [17], the action of two local search algorithms with different search generation features allows a robust handling of problems who variables are strongly correlated and a quick improvement in case of weak correlation among variables, see [18]. Thus, the combined action of the two selected local search algorithms is likely to detect a point close to the corresponding local optimum. The second software sub-module, performing the second stage, groups the points in order to track a map of the strongest basins of attraction and areas near the local optima that display the best performance. A crucial role is played by the Silhouette test as it allows to check the clustering scenarios, *de facto* giving an estimate about the multi-modality of the problem. In other words, if the Silhouette suggests that the most efficient clustering is with one or two groups, it means that the interesting areas are few, possibly because the function is uni-modal (or bi-modal), or possibly because the problem is characterized by one (or two) strong basins of attraction. On the other hand, if the Silhouette suggests that many clusters approximate the data in the best way, the problem appears to be highly multi-modal with all the areas of potential interest for the optimization algorithm. The third software sub-module, uses the basins of attraction and an estimate of their quality to sample the points into the algorithm population. The deterministic mechanism ensures that at least one representative from each cluster is selected. This action guarantees that, even in the case of a cluster under-estimation, no basins of attraction are potentially excluded. The probabilistic selection, as often happens in stochastic algorithms, rewards the most promising areas of the decision space without completely excluding other areas that can be further exploited possibly leading to major improvements.

The CBPI software module hence performs an intelligent sampling on the basis of an implicit analysis of the problem features. Although, in this study, the result of the problem analysis is not explicitly used to design the optimization algorithm, the selection of the initial population on the basis of this criterion, as it will be shown in Section 3, can significantly bias the optimization performance. Thus, the work in this article can be seen as a continuation of the research line started in [18]. More specifically, this work can be seen as a first attempt to analyse/estimate the multi-modality in optimization problems. However, it must be remarked that the proposed CBPI does not aim at detecting all the local minima, On the contrary, CBPI is a pre-processing software module for facilitating optimization algorithms to solve global optimization problems despite of the difficulties due to the presence of multiple sub-optimal basins of attraction. Thus, rather than detecting all the local optima CBPI makes an estimate of the degree of multi-modality of the optimization problem. Furthermore, the main point of interest is not just the amount of optima but the amount of strong optima. A sub-optimal basin of attraction that is characterized by much worse values than those in other regions of the decision space is likely uninteresting for optimization purposes.

Although every algorithm could potentially benefit from some pieces of information about the multi-modality and the strength of the basins of attraction, this study focuses on DE and the ways to enhance its performance. As explained in

[52], DE suffers from a limited amount of search moves and the explicit or implicit integration of extra moves is likely to enhance upon its performance. Furthermore, the integration within the initial population of previously improved individuals, i.e. the generation of the super-fit, is known to be an efficient mechanism in DE frameworks, see [15,35,19]. In this case, the pre-processing does not only improve upon some solutions but structures the population in the decision space in order to have several solutions clustered in the most promising areas and other smaller clusters in other promising areas and still scattered with respect to each other. This configuration of the population is thought for the DE functioning. More specifically, since the generation of a trial point depends on the difference between pairs of points, e.g. DE/rand/1: $\mathbf{x^t} + F(\mathbf{x^r} - \mathbf{x^s})$, the size of the exploratory radius and thus the search logic depend on how the population is distributed. If the points are selected from the same cluster, the DE algorithm will perform a search in a narrow space, thus taking an exploitative behavior. This logic is, to some extent, similar to the population size reduction presented in [9] and very similar to the proximity mutation proposed in [27]. The proposed CBPI, like proximity mutation, encourages the recombination among points that are close to each other, still giving a chance to those points that are distant to recombine and explore the decision space. After the intelligent initialization, we can consider that two kinds of mutations are implicitly allowed. The first, that is an intra-cluster mutation (where the points belonging to the same cluster participate in the mutation) exploits those areas that appeared the most promising in the preliminary phase. The second, that is an inter-cluster mutation (where points belonging to different clusters participate in the mutation), explores new areas of the decision space by making use of a large step-size. Hence, a chance is given to this mechanism to allow the detection of other interesting areas that may have not been detected during the preliminary phase. Example of intra and inter-cluster mutations is shown in Fig. 3.

DE based schemes that perform some kinds of clustering have been proposed in literature in several occasions. Like the present work their goal is somehow to map the population into several areas of interest. However, while in the present paper the clustering aims at detecting a few most interesting areas (through their representative points) to ultimately detect the global optimum the approaches present in literature to tackle multi-modal problems in the sense of the search of all the local optima. One recent example is given in [28]. Another example for dynamic problems is given in [47].

An important issue when an automatic problem analyser is designed (as well an any extra components to be run on the top of an optimization algorithm) is its computational cost and impact on optimization. In order to perform a fair comparison, optimization algorithms must be run with the same computational budget. If an algorithm requires the activation of an extra component a fair comparison should include within the algorithm budget the fitness evaluations carried out by the extra component. In the present article, at most 30% of the computational budget, in terms of fitness evaluations, is assigned to the proposed CBPI while the remaining 70% or more is left to the DE based optimizer. Considering that the same computational budget is allocated to the two local searches, i.e. 15% to each of them, the computational budget for a single S run is $\frac{0.3t_b}{2S_{pop}}$ where t_b is the total budget of the entire run. The same budget is the maximum of R as well. However, the condition in formula 4 allows a stop criterion to R with a lower budget. Hence, the budget of the CBPI is between 15% and 30% of the total budget. The choice of this budget setting for local search is based on preliminary observation of the local search behavior. A compromise was sought between the willingness of letting the local search converge to the a close local optima and the necessity of having a relatively modest budget employment during the CBPI (and hence leave the majority of the budget to the DE optimization).

When compared to the other algorithms (e.g. the algorithms considered in Section 3 to which 100% of the budget is devoted to optimization), the proposed CBPI + DE based algorithm assigns shorter runs to the optimization (as 15–30% of the budget is assigned to the preprocessing and the remainder to the optimization). On the other hand, during the pre-processing phase, an optimization is *de facto*carried out and some solutions are improved. Hence, we can consider the

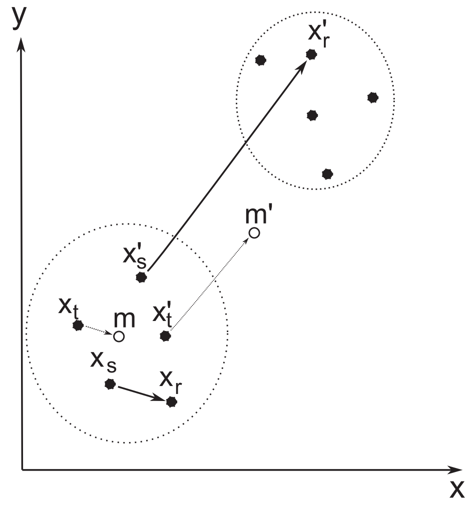

Fig. 3. Mutant m generated by points x_s, x_r and x_t represent intra-cluster mutation, while mutant m' generated by $x_{s'}$, $x_{r'}$ and $x_{t'}$ represent inter-cluster mutation.

Table 1
Average Fitness ± Standard Deviation and Wilcoxon rank-sum test on the Fitness (reference = CBPI variants) for CEC2013 in 10D.

(a) CBPI–DE vs DE

	CBPI–DE	DE	
f_1	**0.00e+00 ± 0.00e+00**	**0.00e+00 ± 0.00e+00**	=
f_2	**2.17e+06 ± 1.49e+06**	2.38e+06 ± 1.24e+06	=
f_3	1.01e+07 ± 9.14e+06	**1.88e+03 ± 1.82e+03**	–
f_4	1.38e+04 ± 4.36e+03	**1.25e+04 ± 3.03e+03**	=
f_5	**0.00e+00 ± 0.00e+00**	**0.00e+00 ± 0.00e+00**	=
f_6	3.82e+00 ± 3.83e+00	**2.57e+00 ± 4.08e+00**	–
f_7	2.39e+01 ± 7.27e+00	**5.62e-01 ± 5.60e–01**	–
f_8	2.04e+01 ± 5.71e–02	**2.04e+01 ± 8.96e–02**	=
f_9	**5.78e+00 ± 7.95e–01**	8.05e+00 ± 6.14e–01	+
f_{10}	1.19e+00 ± 1.96e–01	**6.06e–01 ± 1.06e–01**	–
f_{11}	**0.00e+00 ± 0.00e+00**	5.47e–01 ± 1.11e+00	+
f_{12}	**1.96e+01 ± 2.71e+00**	3.18e+01 ± 4.20e+00	+
f_{13}	**2.17e+01 ± 4.72e+00**	3.15e+01 ± 3.65e+00	+
f_{14}	**2.09e–01 ± 6.57e–01**	2.11e+02 ± 8.18e+01	+
f_{15}	**9.24e+02 ± 1.97e+02**	1.55e+03 ± 1.12e+02	+
f_{16}	**1.03e+00 ± 1.86e–01**	1.36e+00 ± 2.23e–01	+
f_{17}	**1.02e+01 ± 7.99e–02**	1.95e+01 ± 2.09e+00	+
f_{18}	**3.86e+01 ± 5.12e+00**	4.19e+01 ± 4.23e+00	+
f_{19}	**3.94e-01 ± 7.42e–02**	1.48e+00 ± 2.38e–01	+
f_{20}	3.30e+00 ± 2.73e–01	**3.30e+00 ± 2.19e–01**	=
f_{21}	**3.25e+02 ± 8.41e+01**	4.00e+02 ± 0.00e+00	+
f_{22}	**2.07e+01 ± 6.48e+00**	6.95e+02 ± 1.05e+02	+
f_{23}	**1.24e+03 ± 1.74e+02**	1.75e+03 ± 1.07e+02	+
f_{24}	**1.50e+02 ± 2.35e+01**	2.04e+02 ± 1.54e+01	+
f_{25}	2.05e+02 ± 2.22e+01	**2.03e+02 ± 4.85e+00**	–
f_{26}	**1.32e+02 ± 7.54e+00**	1.47e+02 ± 1.22e+01	+
f_{27}	3.91e+02 ± 3.16e+01	**3.00e+02 ± 4.12e–02**	–
f_{28}	**2.79e+02 ± 5.30e+01**	3.00e+02 ± 0.00e+00	=

(b) CBPI–JADE vs JADE

	CBPI–JADE	JADE	
f_1	**0.00e+00 ± 0.00e+00**	**0.00e+00 ± 0.00e+00**	=
f_2	**1.15e+05 ± 1.99e+05**	3.52e+05 ± 8.69e+05	=
f_3	1.07e+06 ± 2.51e+06	**5.35e+02 ± 2.44e+03**	–
f_4	**6.50e+03 ± 2.42e+03**	1.27e+04 ± 3.34e+03	+
f_5	**0.00e+00 ± 0.00e+00**	**0.00e+00 ± 0.00e+00**	=
f_6	**4.83e–01 ± 1.27e+00**	4.71e+00 ± 4.90e+00	=
f_7	1.17e+01 ± 9.31e+00	**4.13e+00 ± 4.33e+00**	–
f_8	**2.03e+01 ± 8.79e–02**	2.04e+01 ± 7.45e–02	+
f_9	**4.39e+00 ± 6.67e–01**	5.14e+00 ± 6.97e–01	+
f_{10}	6.81e–02 ± 3.60e–02	**5.15e–02 ± 1.62e–02**	=
f_{11}	**0.00e+00 ± 0.00e+00**	**0.00e+00 ± 0.00e+00**	=
f_{12}	1.06e+01 ± 2.40e+00	**9.43e+00 ± 2.29e+00**	=
f_{13}	1.57e+01 ± 5.14e+00	**1.20e+01 ± 4.32e+00**	–
f_{14}	1.62e–01 ± 6.65e–01	**2.00e–02 ± 3.41e–02**	=
f_{15}	**6.57e+02 ± 1.41e+02**	8.63e+02 ± 1.34e+02	+
f_{16}	**3.70e–01 ± 8.84e–02**	9.84e–01 ± 1.84e–01	+
f_{17}	**3.86e+00 ± 4.53e+00**	1.01e+01 ± 6.94e–07	+
f_{18}	**2.70e+01 ± 5.57e+00**	3.22e+01 ± 4.53e+00	+
f_{19}	**1.54e–01 ± 9.95e–02**	3.67e–01 ± 5.41e–02	+
f_{20}	**3.10e+00 ± 4.34e–01**	3.27e+00 ± 2.51e–01	=
f_{21}	**1.52e+02 ± 7.55e+01**	4.00e+02 ± 0.00e+00	+
f_{22}	**8.23e+00 ± 4.59e+00**	1.05e+01 ± 4.53e+00	=
f_{23}	**8.98e+02 ± 1.71e+02**	1.08e+03 ± 2.44e+02	+
f_{24}	**1.23e+02 ± 8.09e+00**	1.96e+02 ± 1.81e+01	+
f_{25}	**1.73e+02 ± 3.45e+01**	2.04e+02 ± 1.02e+01	+
f_{26}	**1.14e+02 ± 3.94e+00**	1.26e+02 ± 1.35e+01	+
f_{27}	3.07e+02 ± 2.37e+01	**3.05e+02 ± 1.96e+01**	–
f_{28}	**1.88e+02 ± 9.93e+01**	3.00e+02 ± 0.00e+00	+

(c) CBPI–MDE-pBX vs MDE-pBX

	CBPI–MDE-pBX	MDE-pBX	
f1	**0.00e+00 ± 0.00e+00**	**0.00e+00 ± 0.00e+00**	=
f2	2.50e+03 ± 7.36e+03	**4.47e+02 ± 8.73e+02**	–

(*continued on next page*)

Table 1 (*continued*)

(c) CBPI–MDE-pBX vs MDE-pBX

	CBPI–MDE-pBX	MDE-pBX	
f3	**6.90e+00 ± 1.41e+01**	9.56e+03 ± 4.38e+04	–
f4	4.20e+01 ± 3.01e+01	**8.25e–04 ± 5.54e–03**	–
f5	**0.00e+00 ± 0.00e+00**	**0.00e+00 ± 0.00e+00**	=
f6	**1.92e–01 ± 1.36e+00**	6.35e+00 ± 4.69e+00	+
f7	**3.84e+00 ± 7.96e+00**	7.24e+00 ± 9.11e+00	+
f8	**2.03e+01 ± 8.99e–02**	2.05e+01 ± 8.15e–02	+
f9	2.60e+00 ± 1.21e+00	**2.20e+00 ± 1.43e+00**	=
f10	**9.43e–02 ± 9.87e–02**	1.36e–01 ± 1.24e–01	+
f11	**7.02e–01 ± 8.43e–01**	2.48e+00 ± 1.70e+00	+
f12	1.45e+01 ± 4.79e+00	**1.09e+01 ± 5.21e+00**	–
f13	**1.37e+01 ± 8.69e+00**	2.03e+01 ± 9.51e+00	+
f14	**1.34e+01 ± 1.30e+01**	1.14e+02 ± 9.80e+01	+
f15	**6.30e+02 ± 1.89e+02**	7.99e+02 ± 2.85e+02	+
f16	**3.55e–01 ± 1.15e–01**	6.01e–01 ± 4.43e–01	+
f17	**3.57e+00 ± 3.78e+00**	1.29e+01 ± 2.05e+00	+
f18	2.46e+01 ± 5.69e+00	**1.97e+01 ± 4.99e+00**	–
f19	**1.73e–01 ± 1.35e–01**	6.48e–01 ± 2.22e–01	+
f20	**2.36e+00 ± 6.55e–01**	2.89e+00 ± 5.42e–01	+
f21	**1.55e+02 ± 6.36e+01**	4.00e+02 ± 0.00e+00	+
f22	**9.18e+01 ± 5.91e+01**	1.19e+02 ± 1.00e+02	=
f23	9.01e+02 ± 2.13e+02	**8.83e+02 ± 3.33e+02**	=
f24	**1.18e+02 ± 1.05e+01**	2.04e+02 ± 4.82e+00	+
f25	**1.76e+02 ± 3.59e+01**	2.01e+02 ± 2.12e+00	+
f26	**1.16e+02 ± 6.87e+00**	1.41e+02 ± 4.23e+01	=
f27	3.42e+02 ± 4.67e+01	**3.01e+02 ± 3.92e+00**	–
f28	**2.06e+02 ± 9.98e+01**	3.09e+02 ± 6.90e+01	+

Table 2
Average Fitness ± Standard Deviation and Wilcoxon rank-sum test on the Fitness (reference = CBPI variants) for CEC2013 in 30D.

(a) CBPI–DE vs DE

	CBPI–DE	DE	
f1	**0.00e+00 ± 0.00e+00**	**0.00e+00 ± 0.00e+00**	=
f2	**2.55e+07 ± 5.79e+06**	1.77e+08 ± 3.23e+07	+
f3	3.04e+09 ± 1.90e+09	**2.22e+08 ± 1.79e+08**	–
f4	6.95e+04 ± 1.27e+04	**6.32e+04 ± 6.42e+03**	=
f5	**0.00e+00 ± 0.00e+00**	**0.00e+00 ± 0.00e+00**	=
f6	3.00e+01 ± 6.71e+00	**1.69e+01 ± 3.20e+00**	–
f7	1.09e+02 ± 1.45e+01	**5.67e+01 ± 7.86e+00**	–
f8	**2.10e+01 ± 6.08e–02**	2.10e+01 ± 4.03e–02	=
f9	**2.98e+01 ± 1.53e+00**	4.01e+01 ± 1.05e+00	+
f10	**1.95e+01 ± 6.40e+00**	4.95e+01 ± 1.72e+01	+
f11	**7.96e–02 ± 2.70e–01**	1.29e+02 ± 1.03e+01	+
f12	**1.59e+02 ± 2.16e+01**	2.22e+02 ± 1.33e+01	+
f13	**2.07e+02 ± 1.84e+01**	2.17e+02 ± 1.25e+01	+
f14	**3.03e–01 ± 4.40e–01**	4.81e+03 ± 2.68e+02	+
f15	**4.81e+03 ± 3.92e+02**	7.59e+03 ± 2.16e+02	+
f16	**1.89e+00 ± 2.43e–01**	2.60e+00 ± 3.18e–01	+
f17	**3.05e+01 ± 1.53e–01**	1.64e+02 ± 5.22e+00	+
f18	**2.37e+02 ± 1.49e+01**	2.51e+02 ± 9.78e+00	+
f19	**1.52e+00 ± 1.53e–01**	1.55e+01 ± 1.05e+00	+
f20	1.41e+01 ± 5.25e–01	**1.38e+01 ± 2.88e–01**	–
f21	**2.41e+02 ± 4.35e+01**	2.61e+02 ± 4.81e+01	=
f22	**7.22e+01 ± 3.09e+01**	5.60e+03 ± 3.01e+02	+
f23	**5.75e+03 ± 4.58e+02**	7.99e+03 ± 2.72e+02	+
f24	2.86e+02 ± 4.48e+00	**2.85e+02 ± 1.37e+01**	=
f25	**3.01e+02 ± 5.90e+00**	3.25e+02 ± 3.32e+00	+
f26	**2.02e+02 ± 6.79e–01**	2.17e+02 ± 4.96e+00	+
f27	**7.45e+02 ± 3.48e+02**	1.33e+03 ± 3.61e+01	+
f28	3.00e+02 ± 4.64e–01	**3.00e+02 ± 0.00e+00**	+

(b) CBPI–JADE vs JADE

	CBPI–JADE	JADE	
f1	**0.00e+00 ± 0.00e+00**	**0.00e+00 ± 0.00e+00**	=
f2	7.25e+05 ± 1.70e+06	**2.20e+04 ± 1.14e+04**	=

Table 2 (*continued*)

(b) CBPI–JADE vs JADE

	CBPI–JADE	JADE	
f3	1.77e+07 ± 2.20e+07	**2.04e+06 ± 4.21e+06**	–
f4	**6.42e+04 ± 8.20e+03**	7.85e+04 ± 1.16e+04	+
f5	**0.00e+00 ± 0.00e+00**	**0.00e+00 ± 0.00e+00**	=
f6	4.16e+00 ± 6.25e+00	**1.06e+00 ± 5.17e+00**	=
f7	6.61e+01 ± 1.01e+01	**4.18e+01 ± 2.28e+01**	–
f8	2.10e+01 ± 4.89e−02	**2.10e+01 ± 4.73e−02**	=
f9	**2.67e+01 ± 1.57e+00**	3.00e+01 ± 1.58e+00	+
f10	**3.78e−02 ± 2.25e−02**	4.63e−02 ± 2.94e−02	=
f11	**0.00e+00 ± 0.00e+00**	**0.00e+00 ± 0.00e+00**	=
f12	6.83e+01 ± 1.63e+01	**5.28e+01 ± 8.68e+00**	–
f13	1.11e+02 ± 2.43e+01	**1.10e+02 ± 1.25e+01**	=
f14	5.94e−02 ± 3.92e−02	**5.01e−02 ± 3.49e−02**	=
f15	**3.47e+03 ± 3.26e+02**	4.63e+03 ± 2.76e+02	+
f16	**8.29e−01 ± 2.11e−01**	1.72e+00 ± 2.71e−01	+
f17	**2.58e+01 ± 1.06e+01**	3.04e+01 ± 1.22e−05	+
f18	1.89e+02 ± 2.27e+01	**1.49e+02 ± 1.65e+01**	–
f19	**1.10e+00 ± 2.37e−01**	1.44e+00 ± 1.18e−01	+
f20	1.48e+01 ± 4.34e−01	**1.45e+01 ± 4.51e−01**	–
f21	**2.00e+02 ± 1.44e−04**	3.05e+02 ± 6.04e+01	+
f22	**3.34e+01 ± 2.67e+01**	7.67e+01 ± 3.34e+01	+
f23	**4.22e+03 ± 4.91e+02**	5.43e+03 ± 3.68e+02	+
f24	2.65e+02 ± 1.04e+01	**2.18e+02 ± 1.14e+01**	–
f25	**2.96e+02 ± 4.46e+00**	3.00e+02 ± 6.07e+00	+
f26	**2.00e+02 ± 2.80e−01**	2.07e+02 ± 3.04e+01	=
f27	9.39e+02 ± 2.02e+02	**9.20e+02 ± 1.74e+02**	=
f28	**2.76e+02 ± 6.45e+01**	3.00e+02 ± 0.00e+00	=

(c) CBPI–MDE-pBX vs MDE-pBX

	CBPI–MDE-pBX	MDE-pBX	
f1	**0.00e+00 ± 0.00e+00**	**0.00e+00 ± 0.00e+00**	=
f2	1.05e+05 ± 6.43e+04	**8.57e+04 ± 5.00e+04**	=
f3	**5.12e+06 ± 8.18e+06**	1.48e+07 ± 2.46e+07	+
f4	3.52e+01 ± 2.75e+01	**1.33e+01 ± 3.92e+01**	–
f5	**0.00e+00 ± 0.00e+00**	**0.00e+00 ± 0.00e+00**	=
f6	**1.08e+01 ± 4.13e+00**	3.34e+01 ± 3.00e+01	+
f7	**4.13e+01 ± 2.04e+01**	6.09e+01 ± 1.73e+01	+
f8	**2.09e+01 ± 7.07e−02**	2.10e+01 ± 5.23e−02	+
f9	**2.13e+01 ± 2.43e+00**	2.27e+01 ± 4.25e+00	=
f10	**1.32e−01 ± 3.45e−02**	1.44e−01 ± 8.45e−02	=
f11	**1.11e+01 ± 3.51e+00**	4.70e+01 ± 1.32e+01	+
f12	1.21e+02 ± 1.85e+01	**7.18e+01 ± 1.85e+01**	–
f13	**9.71e+01 ± 3.58e+01**	1.46e+02 ± 3.27e+01	+
f14	**2.20e+02 ± 1.53e+02**	1.16e+03 ± 4.50e+02	+
f15	**3.26e+03 ± 2.88e+02**	3.92e+03 ± 6.42e+02	+
f16	**7.57e−01 ± 2.04e−01**	1.26e+00 ± 7.53e−01	+
f17	**4.59e+01 ± 8.15e+00**	7.09e+01 ± 1.28e+01	+
f18	**6.75e+01 ± 2.77e+01**	8.15e+01 ± 1.34e+01	+
f19	**2.65e+00 ± 6.95e−01**	9.86e+00 ± 7.61e+00	+
f20	1.15e+01 ± 1.52e+00	**1.08e+01 ± 7.21e−01**	–
f21	**1.92e+02 ± 2.69e+01**	3.23e+02 ± 8.67e+01	+
f22	**5.28e+02 ± 1.95e+02**	1.24e+03 ± 4.50e+02	+
f23	**4.10e+03 ± 4.85e+02**	4.48e+03 ± 7.47e+02	+
f24	2.50e+02 ± 1.68e+01	**2.32e+02 ± 1.10e+01**	–
f25	2.80e+02 ± 7.41e+00	**2.78e+02 ± 1.12e+01**	=
f26	**2.00e+02 ± 1.19e−02**	2.26e+02 ± 5.21e+01	=
f27	8.22e+02 ± 2.16e+02	**6.48e+02 ± 1.18e+02**	–
f28	**2.73e+02 ± 6.88e+01**	2.88e+02 ± 4.71e+01	=

system process as a two phase cascaded optimization, at first CBPI optimizes multiple solutions, groups them, and detect the interesting areas. Then, DE exploits these pieces of information for continuing the optimization.

3. Numerical results

In order to test the validity and potentials of the proposed CBPI, the following testbeds have been taken into account:

Table 3
Average Fitness ± Standard Deviation and Wilcoxon rank-sum test on the Fitness (reference = CBPI variants) for CEC2013 in 50D.

(a) CBPI–DE vs DE

	CBPI–DE	DE	
f1	**0.00e+00 ± 0.00e+00**	**0.00e+00 ± 0.00e+00**	=
f2	5.43e+07 ± 1.40e+07	**5.19e+07 ± 9.40e+06**	=
f3	1.98e+10 ± 4.50e+09	**1.82e+10 ± 6.16e+09**	=
f4	**1.13e+05 ± 1.27e+04**	1.22e+05 ± 1.17e+04	+
f5	**0.00e+00 ± 0.00e+00**	**0.00e+00 ± 0.00e+00**	=
f6	4.53e+01 ± 9.65e−01	**4.51e+01 ± 8.74e−01**	=
f7	1.59e+02 ± 1.09e+01	**1.51e+02 ± 1.20e+01**	−
f8	**2.12e+01 ± 3.40e−02**	2.12e+01 ± 2.62e−02	=
f9	**5.70e+01 ± 2.79e+00**	5.80e+01 ± 2.47e+00	=
f10	**5.29e+01 ± 1.39e+01**	5.95e+01 ± 1.27e+01	=
f11	**0.00e+00 ± 0.00e+00**	3.98e−02 ± 1.95e−01	=
f12	**4.39e+02 ± 4.34e+01**	4.47e+02 ± 4.28e+01	=
f13	5.23e+02 ± 3.83e+01	**5.19e+02 ± 5.24e+01**	=
f14	**5.14e−01 ± 4.25e−01**	6.65e−01 ± 6.42e−01	=
f15	9.42e+03 ± 3.26e+02	**9.37e+03 ± 4.81e+02**	=
f16	**2.39e+00 ± 3.90e−01**	2.45e+00 ± 2.40e−01	=
f17	**5.10e+01 ± 1.41e−01**	5.10e+01 ± 1.90e−01	=
f18	5.09e+02 ± 4.10e+01	**5.07e+02 ± 3.70e+01**	=
f19	2.69e+00 ± 1.87e−01	**2.67e+00 ± 2.02e−01**	=
f20	**2.42e+01 ± 4.39e−01**	2.42e+01 ± 5.22e−01	=
f21	**2.97e+02 ± 1.68e+02**	3.25e+02 ± 1.68e+02	=
f22	4.62e+01 ± 1.39e+01	**3.87e+01 ± 1.82e+01**	=
f23	**1.07e+04 ± 8.17e+02**	1.11e+04 ± 5.93e+02	=
f24	**3.60e+02 ± 7.79e+00**	3.63e+02 ± 7.65e+00	=
f25	4.00e+02 ± 5.91e+00	**3.98e+02 ± 8.34e+00**	=
f26	**2.05e+02 ± 9.85e−01**	2.05e+02 ± 7.25e−01	=
f27	**1.50e+03 ± 6.16e+02**	1.70e+03 ± 4.77e+02	=
f28	**4.00e+02 ± 8.20e−01**	4.00e+02 ± 1.16e+00	=

(b) CBPI–JADE vs JADE

	CBPI–JADE	JADE	
f1	**0.00e+00 ± 0.00e+00**	**0.00e+00 ± 0.00e+00**	=
f2	1.99e+05 ± 7.64e+05	**2.38e+04 ± 2.41e+04**	+
f3	1.22e+08 ± 9.18e+07	**1.12e+07 ± 1.45e+07**	−
f4	**1.09e+05 ± 1.27e+04**	1.37e+05 ± 1.65e+04	+
f5	**0.00e+00 ± 0.00e+00**	**0.00e+00 ± 0.00e+00**	=
f6	**3.48e+01 ± 1.17e+01**	4.22e+01 ± 8.66e+00	+
f7	1.04e+02 ± 8.66e+00	**6.67e+01 ± 1.56e+01**	−
f8	2.11e+01 ± 3.70e−02	**2.11e+01 ± 4.06e−02**	=
f9	**5.05e+01 ± 2.52e+00**	5.76e+01 ± 2.25e+00	+
f10	5.37e−02 ± 3.18e−02	**4.28e−02 ± 3.16e−02**	=
f11	**0.00e+00 ± 0.00e+00**	**0.00e+00 ± 0.00e+00**	=
f12	1.49e+02 ± 2.21e+01	**1.04e+02 ± 2.15e+01**	−
f13	2.36e+02 ± 3.75e+01	**2.34e+02 ± 3.17e+01**	=
f14	7.69e−02 ± 3.16e−02	**4.87e−02 ± 1.87e−02**	−
f15	**6.76e+03 ± 4.48e+02**	8.00e+03 ± 5.88e+02	+
f16	**1.19e+00 ± 1.85e−01**	1.91e+00 ± 2.74e−01	+
f17	**4.69e+01 ± 1.21e+01**	5.08e+01 ± 0.00e+00	=
f18	2.44e+02 ± 2.68e+01	**2.38e+02 ± 2.36e+01**	=
f19	**1.21e+00 ± 3.40e−01**	2.17e+00 ± 4.03e−01	+
f20	**2.40e+01 ± 7.50e−01**	2.44e+01 ± 3.49e−01	+
f21	**1.98e+02 ± 1.39e+01**	9.23e+02 ± 3.13e+02	+
f22	1.10e+01 ± 1.44e+00	**1.07e+01 ± 1.26e+00**	=
f23	**8.59e+03 ± 8.79e+02**	9.85e+03 ± 6.82e+02	+
f24	3.40e+02 ± 9.25e+00	**2.51e+02 ± 1.69e+01**	−
f25	**3.85e+02 ± 8.28e+00**	4.00e+02 ± 7.07e+00	+
f26	**2.01e+02 ± 2.00e−01**	2.41e+02 ± 8.74e+01	+
f27	1.68e+03 ± 9.44e+01	**1.66e+03 ± 2.60e+02**	+
f28	**4.00e+02 ± 0.00e+00**	**4.00e+02 ± 0.00e+00**	=

(c) CBPI–MDE-pBX vs MDE-pBX

	CBPI–MDE-pBX	MDE-pBX	
f1	**0.00e+00 ± 0.00e+00**	**0.00e+00 ± 0.00e+00**	=
f2	**4.55e+05 ± 1.78e+05**	4.59e+05 ± 2.43e+05	=
f3	**4.23e+07 ± 3.64e+07**	9.84e+07 ± 1.60e+08	=
f4	2.34e+02 ± 1.95e+02	**2.36e+01 ± 2.36e+01**	−

Table 3 (*continued*)

(c) CBPI–MDE-pBX vs MDE-pBX

	CBPI–MDE-pBX	MDE-pBX	
f5	**0.00e+00 ± 0.00e+00**	**0.00e+00 ± 0.00e+00**	=
f6	**3.47e+01 ± 1.03e+01**	5.57e+01 ± 2.33e+01	+
f7	6.72e+01 ± 2.17e+01	**6.46e+01 ± 1.37e+01**	=
f8	**2.11e+01 ± 4.31e−02**	2.12e+01 ± 4.37e−02	+
f9	4.53e+01 ± 3.17e+00	**4.43e+01 ± 7.26e+00**	=
f10	**1.40e−01 ± 5.98e−02**	1.53e−01 ± 2.67e−01	−
f11	**3.21e+01 ± 7.62e+00**	1.19e+02 ± 2.73e+01	+
f12	3.02e+02 ± 6.94e+01	**1.59e+02 ± 3.07e+01**	−
f13	**2.04e+02 ± 4.96e+01**	3.20e+02 ± 4.24e+01	+
f14	**8.31e+02 ± 2.76e+02**	2.76e+03 ± 7.65e+02	+
f15	**6.47e+03 ± 4.91e+02**	7.61e+03 ± 8.52e+02	+
f16	**1.10e+00 ± 2.32e−01**	1.85e+00 ± 8.83e−01	+
f17	**1.40e+02 ± 2.02e+01**	1.74e+02 ± 3.90e+01	+
f18	**1.33e+02 ± 2.07e+01**	1.84e+02 ± 2.96e+01	+
f19	**7.03e+00 ± 1.28e+00**	3.61e+01 ± 1.68e+01	+
f20	1.99e+01 ± 1.08e+00	**1.98e+01 ± 8.35e−01**	=
f21	**2.00e+02 ± 0.00e+00**	8.67e+02 ± 3.67e+02	+
f22	**1.49e+03 ± 5.24e+02**	3.24e+03 ± 1.15e+03	+
f23	**8.30e+03 ± 7.45e+02**	9.00e+03 ± 1.16e+03	+
f24	3.24e+02 ± 1.84e+01	**2.84e+02 ± 1.42e+01**	−
f25	3.68e+02 ± 1.30e+01	**3.67e+02 ± 1.44e+01**	=
f26	**2.00e+02 ± 4.59e−02**	3.46e+02 ± 8.23e+01	+
f27	1.51e+03 ± 1.17e+02	**1.28e+03 ± 1.63e+02**	−
f28	**4.00e+02 ± 0.00e+00**	5.42e+02 ± 7.05e+02	=

- The CEC2013 benchmark described in [38] in 10, 30, and 50 dimensions (28 test problems).
- The BBOB2010 benchmark described in [48] in 100 dimensions (24 test problems).
- The CEC2010 benchmark described in [65] in 1000 dimensions (20 test problems).

In addition, one real-world problem from [21] is also studied. Totally 129 problems for 5 dimensionality values have been considered. The CBPI has been always run with the following parameter setting: $M = 10$ and $S_{pop} = 30$. For S local searcher: exploratory radius $\xi = 0.4$ and for Rosenbrock's local search: $\epsilon = 10e^{-5}$, $\alpha = 2$ and $\beta = 0.5$.

The proposed CBPI has been tested on the following algorithms:

- Differential Evolution (DE). Classical DE/rand/1/exp with $Np = 30$, $F = 0.7$, $Cr = 0.5$,
- Self-Adaptive Control Parameters in Differential Evolution (jDE), proposed in [6], with $Np = 50.0$, $F_l = 0.1$, $F_u = 1.0$ and $\tau_1 = \tau_2 = 0.1$.
- Adaptive DE with optional external archive (JADE), proposed in [73], with $Np = 60$, $\mu_F = 0.1$ and $\mu_{Cr} = 0.05$.
- Self-Adaptive Differential Evolution (SADE), see [59], with $Np = 50$ and $LP = 20$.
- Differential Evolution with ensemble of mutation and crossover strategies (EPSDE), see [43], with $Np = 50, F \in \{0.5, 0.9\}$ and $Cr \in \{0.1, 0.5, 0.9\}$.
- Modified Differential Evolution with p-best crossover (MDE-pBX), see [36], with $Np = 100$ and group size $q = 0.15$.

Each of the algorithms listed above has been run with a budget of $5000 \times n$ function calls (fitness evaluations) and pairwise compare with the CBPI variant that uses 30% and 70% of the total budget ($5000 \times n$ function calls), for CBPI and optimization, respectively. Each algorithm, for each problem, has been run 50 times.

The complete set of results is reported in the website.[2]

In this article, detailed results about DE, JADE, and MDE-pBX are shown in addition to a summary table including all the six algorithms listed above.

Numerical results related to the effect of CBPI over the standard DE, JADE and MDE-pBX algorithms, respectively, are shown in Tables 1–5, for the five benchmarks under consideration. The best result of each pairwise comparison is highlighted in bold font. Moreover, the results of the Wilcoxon test are summarized in the final row of each table.

As shown in tables, in the vast majority of cases the CBPI version of DE significantly outperforms the basic DE variant. The action of CBPI component appears to be beneficial also in this case as it leads to a performance improvement in the vast of majority of cases. It must be remarked that while for JADE the CBPI application is not too evident for the testbed in 100D. On the contrary, the advantage of CBPI is not doubtable for the large scale testbed. Finally, the pairwise comparison between MDE-pBX and its CBPI version, shows that for almost all the problems under consideration CBPI–MDE-pBX outperforms its basic variant.

[2] https://sites.google.com/site/facaraff/home/Downloads/CBPI-DE_Detailed_Results.pdf.

Table 4
Average Fitness ± Standard Deviation and Wilcoxon rank-sum test on the Fitness (reference = CBPI variants) for BBOB2010 in 100D.

(a) CBPI–DE vs DE

	CBPI–DE	DE	
f1	**9.15e−14 ± 0.00e+00**	**9.04e−14 ± 0.00e+00**	=
f2	**1.56e−13 ± 1.42e−14**	**1.55e−13 ± 1.41e−14**	=
f3	**3.98e−02 ± 1.95e−01**	2.39e−01 ± 5.10e−01	=
f4	**7.56e−01 ± 7.05e−01**	1.19e+00 ± 1.01e+00	=
f5	2.49e−12 ± 1.74e−13	**2.44e−12 ± 2.26e−13**	=
f6	**1.40e+02 ± 3.75e+01**	4.43e+02 ± 9.08e+01	+
f7	**2.58e+02 ± 4.47e+01**	4.02e+02 ± 4.77e+01	+
f8	**6.05e+00 ± 1.44e+01**	9.06e+01 ± 1.80e+00	+
f9	**9.36e+01 ± 1.88e+00**	9.46e+01 ± 1.95e+00	=
f10	**8.84e+04 ± 7.61e+03**	4.10e+05 ± 2.69e+04	+
f11	**5.40e+02 ± 4.57e+01**	6.23e+02 ± 4.89e+01	+
f12	**5.64e−02 ± 7.96e−02**	7.16e+01 ± 4.82e+01	+
f13	**1.01e−01 ± 3.89e−02**	3.21e+00 ± 6.87e−01	+
f14	**8.46e−03 ± 8.53e−04**	1.94e−02 ± 1.13e−03	+
f15	**1.03e+03 ± 7.14e+01**	1.49e+03 ± 1.16e+02	+
f16	**1.80e+01 ± 1.29e+00**	2.36e+01 ± 1.78e+00	+
f17	**6.29e+00 ± 4.19e−01**	1.07e+01 ± 9.24e−01	+
f18	**2.43e+01 ± 2.30e+00**	4.14e+01 ± 3.28e+00	+
f19	**1.02e+01 ± 3.67e−01**	1.02e+01 ± 4.53e−01	=
f20	1.05e−02 ± 1.26e−02	**5.94e−04 ± 2.35e−03**	−
f21	**1.22e−01 ± 4.46e−01**	5.98e−01 ± 7.76e−01	+
f22	**1.20e+00 ± 6.19e−01**	1.79e+00 ± 2.74e−01	+
f23	**1.23e+00 ± 1.48e−01**	2.73e+00 ± 2.32e−01	+
f24	**1.11e+03 ± 8.25e+01**	1.60e+03 ± 1.14e+02	+

(b) CBPI–JADE vs JADE

	CBPI–JADE	JADE	
f1	**0.00e+00 ± 0.00e+00**	**0.00e+00 ± 0.00e+00**	=
f2	**0.00e+00 ± 0.00e+00**	**0.00e+00 ± 0.00e+00**	=
f3	**7.96e−01 ± 1.69e+00**	1.95e+00 ± 4.09e+00	=
f4	**4.06e+00 ± 4.24e+00**	1.66e+01 ± 1.74e+01	+
f5	**3.07e−06 ± 5.40e−07**	6.16e−06 ± 1.48e−06	+
f6	9.91e−01 ± 3.35e+00	**9.04e−01 ± 9.86e−01**	+
f7	**7.07e+01 ± 1.23e+01**	1.26e+02 ± 2.34e+01	+
f8	1.49e+01 ± 2.36e+01	**7.97e−01 ± 1.59e+00**	−
f9	**8.69e+01 ± 6.23e+00**	9.20e+01 ± 2.13e+01	=
f10	1.80e+03 ± 3.33e+02	**1.45e+03 ± 3.98e+02**	−
f11	**6.56e+02 ± 6.61e+01**	7.61e+02 ± 5.28e+01	+
f12	**3.95e−02 ± 4.28e−02**	9.27e+00 ± 9.42e+00	+
f13	**6.27e−03 ± 6.22e−03**	1.72e+00 ± 2.52e+00	+
f14	**4.97e−05 ± 6.18e−06**	5.29e−05 ± 6.57e−06	=
f15	5.25e+02 ± 6.35e+01	**4.67e+02 ± 5.29e+01**	−
f16	**1.80e+01 ± 1.75e+00**	2.37e+01 ± 2.02e+00	+
f17	4.51e+00 ± 6.16e−01	**2.03e+00 ± 6.69e−01**	−
f18	1.65e+01 ± 5.35e+00	**8.83e+00 ± 2.40e+00**	−
f19	6.78e+00 ± 3.48e−01	**6.72e+00 ± 3.47e−01**	=
f20	**8.64e−01 ± 3.20e−01**	1.22e+00 ± 2.30e−01	+
f21	**2.45e−01 ± 6.63e−01**	3.75e+00 ± 6.06e+00	+
f22	**1.40e+00 ± 6.27e−01**	8.30e+00 ± 1.04e+01	=
f23	**1.18e+00 ± 1.37e−01**	2.66e+00 ± 2.07e−01	+
f24	9.10e+02 ± 8.54e+01	**5.81e+02 ± 5.40e+01**	−

(c) CBPI–MDE-pBX vs MDE-pBX

	CBPI–MDE-pBX	MDE-pBX	
f1	**3.77e−13 ± 4.29e−13**	2.23e−08 ± 1.20e−07	+
f2	**7.01e−10 ± 3.48e−09**	1.82e−04 ± 6.63e−04	+
f3	**1.44e+02 ± 2.41e+01**	4.90e+02 ± 8.57e+01	+
f4	**1.49e+02 ± 2.60e+01**	8.81e+02 ± 1.59e+02	+
f5	**6.53e−04 ± 2.79e−04**	8.19e+00 ± 1.10e+01	+
f6	**1.22e+01 ± 9.55e+00**	4.69e+01 ± 4.04e+01	+
f7	**9.90e+01 ± 1.97e+01**	2.78e+02 ± 6.52e+01	+
f8	**1.55e+01 ± 2.24e+01**	1.71e+02 ± 8.03e+01	+
f9	**9.31e+01 ± 1.59e+00**	1.30e+02 ± 3.62e+01	+
f10	1.74e+04 ± 6.81e+03	**1.33e+04 ± 4.36e+03**	−
f11	4.01e+01 ± 1.09e+01	**1.77e+01 ± 6.45e+00**	−

Table 4 (*continued*)

(c) CBPI–MDE-pBX vs MDE-pBX

	CBPI–MDE-pBX	MDE-pBX	
f12	**5.92e−02 ± 1.02e−01**	5.03e+01 ± 2.22e+02	+
f13	**1.14e−02 ± 1.69e−02**	5.55e+00 ± 9.98e+00	+
f14	**1.44e−03 ± 4.42e−04**	2.78e−03 ± 2.53e−03	+
f15	**3.60e+02 ± 1.75e+02**	6.67e+02 ± 1.07e+02	+
f16	1.52e+01 ± 2.58e+00	**1.42e+01 ± 2.66e+00**	=
f17	**2.69e+00 ± 4.56e−01**	3.22e+00 ± 4.06e−01	+
f18	**9.78e+00 ± 1.85e+00**	1.17e+01 ± 1.48e+00	+
f19	3.44e+00 ± 1.46e+00	**2.40e+00 ± 9.01e−01**	–
f20	**1.18e+00 ± 1.21e−01**	2.09e+00 ± 1.14e−01	+
f21	**4.15e−02 ± 2.24e−01**	7.44e+00 ± 7.12e+00	+
f22	**1.41e+00 ± 6.26e−01**	9.42e+00 ± 1.02e+01	+
f23	**1.11e+00 ± 1.38e−01**	2.35e+00 ± 1.07e+00	+
f24	**3.28e+02 ± 1.19e+02**	3.29e+02 ± 3.23e+01	=

Table 5
Average Fitness ± Standard Deviation and Wilcoxon rank-sum test on the Fitness (reference = CBPI variants) for CEC2010 in 1000D.

(a) CBPI–DE vs DE

	CBPI–DE	DE	
f1	1.76e−12 ± 8.72e−13	**0.00e+00 ± 0.00e+00**	–
f2	**1.67e+00 ± 8.77e−01**	2.03e+00 ± 1.45e+00	=
f3	2.21e−09 ± 4.73e−10	**1.27e−12 ± 6.34e−14**	–
f4	**2.91e+13 ± 6.50e+12**	4.61e+13 ± 8.78e+12	+
f5	**2.68e+08 ± 3.50e+07**	4.84e+08 ± 5.24e+07	+
f6	**6.75e+06 ± 2.72e+06**	2.02e+07 ± 2.17e+05	+
f7	**2.66e+10 ± 3.08e+09**	3.54e+10 ± 4.05e+09	+
f8	**1.54e+07 ± 1.24e+07**	1.58e+07 ± 1.79e+07	=
f9	**5.29e+08 ± 2.62e+07**	9.39e+08 ± 3.30e+07	+
f10	**5.06e+03 ± 1.32e+02**	6.43e+03 ± 1.47e+02	+
f11	**1.84e+02 ± 4.52e+00**	2.05e+02 ± 3.37e+00	+
f12	**5.81e+05 ± 1.89e+04**	8.56e+05 ± 2.58e+04	+
f13	**1.20e+03 ± 2.18e+02**	1.49e+03 ± 8.88e+01	+
f14	**1.21e+09 ± 5.74e+07**	2.56e+09 ± 1.01e+08	+
f15	**1.03e+04 ± 1.49e+02**	1.42e+04 ± 2.44e+02	+
f16	**3.91e+02 ± 1.72e+00**	4.13e+02 ± 4.10e+00	+
f17	**1.19e+06 ± 2.11e+04**	2.02e+06 ± 4.51e+04	+
f18	**3.89e+03 ± 7.39e+02**	4.55e+03 ± 8.48e+02	+
f19	4.97e+06 ± 2.85e+05	**3.29e+06 ± 1.82e+05**	–
f20	1.38e+03 ± 1.14e+02	**9.87e+02 ± 1.09e+01**	–

(b) CBPI–JADE vs JADE

	CBPI–JADE	JADE	
f1	**6.05e−02 ± 9.32e−02**	1.13e+10 ± 6.62e+09	+
f2	**9.91e+00 ± 1.49e+01**	4.43e+03 ± 8.07e+02	+
f3	**2.61e+00 ± 4.21e+00**	1.77e+01 ± 5.54e−01	+
f4	**1.58e+12 ± 5.85e+12**	2.91e+12 ± 1.16e+12	+
f5	1.79e+08 ± 2.47e+07	**7.99e+07 ± 1.72e+07**	–
f6	**1.95e+01 ± 5.45e−02**	3.31e+06 ± 1.09e+06	+
f7	**1.85e+09 ± 6.28e+09**	3.86e+09 ± 3.14e+09	+
f8	**2.36e+07 ± 3.25e+07**	8.72e+13 ± 2.63e+14	+
f9	**7.68e+07 ± 9.78e+07**	1.63e+10 ± 4.53e+09	+
f10	**3.15e+03 ± 1.63e+02**	7.46e+03 ± 1.17e+03	+
f11	**1.78e+02 ± 5.29e+00**	1.95e+02 ± 6.08e+00	+
f12	**8.76e+03 ± 4.03e+03**	2.31e+06 ± 3.98e+05	+
f13	**9.81e+02 ± 1.96e+02**	8.62e+10 ± 4.36e+10	+
f14	**1.60e+08 ± 1.74e+08**	1.30e+10 ± 3.57e+09	+
f15	**6.23e+03 ± 2.18e+02**	8.53e+03 ± 1.20e+03	+
f16	**3.83e+02 ± 6.57e+00**	3.85e+02 ± 1.34e+01	=
f17	**4.54e+04 ± 4.67e+03**	2.22e+06 ± 5.26e+05	+
f18	**2.32e+03 ± 2.79e+02**	4.10e+11 ± 2.01e+11	+
f19	6.56e+06 ± 3.60e+06	**3.65e+06 ± 7.12e+05**	–
f20	**1.29e+03 ± 4.06e+02**	5.38e+11 ± 2.45e+11	+

(*continued on next page*)

Table 5 (*continued*)

(c) CBPI–MDE-pBX vs MDE-pBX

	CBPI–MDE-pBX	MDE-pBX	
f1	**4.63e−04 ± 6.68e−04**	1.61e+09 ± 9.23e+08	+
f2	**6.26e+03 ± 5.46e+02**	7.23e+03 ± 2.68e+02	+
f3	**1.56e+01 ± 1.38e+00**	1.94e+01 ± 6.67e−02	+
f4	**2.65e+12 ± 8.84e+11**	3.83e+12 ± 1.15e+12	+
f5	2.60e+08 ± 3.56e+07	**1.45e+08 ± 3.10e+07**	−
f6	**3.94e+06 ± 2.04e+06**	4.06e+06 ± 1.51e+06	=
f7	**5.41e+06 ± 5.61e+06**	1.32e+07 ± 1.86e+07	+
f8	**5.18e+07 ± 4.50e+07**	2.28e+08 ± 1.49e+08	+
f9	**1.18e+08 ± 2.00e+07**	2.31e+09 ± 1.38e+09	+
f10	8.44e+03 ± 1.27e+02	**7.42e+03 ± 2.79e+02**	−
f11	2.12e+02 ± 8.73e−01	**2.06e+02 ± 2.95e+00**	−
f12	**7.22e+04 ± 1.63e+04**	4.32e+05 ± 7.54e+04	+
f13	**1.60e+03 ± 3.15e+02**	3.23e+09 ± 2.45e+09	+
f14	**3.68e+08 ± 5.08e+07**	1.42e+09 ± 3.03e+08	+
f15	1.03e+04 ± 1.81e+02	**7.61e+03 ± 3.01e+02**	−
f16	3.92e+02 ± 6.06e−01	**3.85e+02 ± 1.10e+00**	−
f17	**1.98e+05 ± 3.27e+04**	5.70e+05 ± 8.74e+04	+
f18	**4.60e+03 ± 9.74e+02**	4.78e+10 ± 2.05e+10	+
f19	**1.08e+06 ± 1.39e+05**	1.25e+06 ± 1.20e+05	+
f20	**1.91e+03 ± 1.05e+02**	5.52e+10 ± 2.47e+10	+

Table 6
Average fitness ± standard deviation and statistical comparison for DE algorithms against their CBPI variants to solve the Lennard–Jones Potential minimization in 30 dimensions (reference: CBPI-variant).

	CBPI–DE	DE	
LJP_{30D}	**−2.57e+01 ± 1.16e+00**	−2.36e+01 ± 9.68e−01	+
	CBPI–jDE	jDE	
LJP_{30D}	**−2.47e+01 ± 2.00e+00**	−1.74e+01 ± 1.40e+00	+
	CBPI–JADE	JADE	
LJP_{30D}	**−2.58e+01 ± 1.16e+00**	−2.40e+01 ± 8.38e−01	+
	CBPI–SADE	SADE	
LJP_{30D}	**−2.64e+01 ± 1.73e+00**	−2.61e+01 ± 2.34e+00	=
	CBPI–EPSDE	EPSDE	
LJP_{30D}	**−2.68e+01 ± 8.36e−01**	−2.60e+01 ± 1.97e+00	+
	CBPI–MDE-pBX	MDE-pBX	
LJP_{30D}	−2.66e+01 ± 1.61e+00	**−2.70e+01 ± 8.40e−01**	=

3.1. A real world problem: Lennard–Jones Potential minimization

In order to prove the viability of the proposed approach over real-world problems, CBPI has been tested over the Lennard–Jones Potential (LJP) problem. The LJP problem is widely used in physics for representing the interaction energy between non-bonding particles in a fluid. In this case, according to the description given in CEC2011 [21], we refer to the minimization problem of the potential energy of a set of atoms by locating them within the three-dimensional space. In order to determine their position and evaluate the relative LJP 3 parameters, i.e. coordinates along the axes x, y and z, for each atom need to be stored in the fitness function. In this study, an atomic cluster of 10 atoms has been considered, resulting into a 30-dimensional problem. Each algorithm under examination has been run 50 times with a computational budget of 150,000 functional calls. The parameters of the algorithms are the same used above. Numerical results about the 12 algorithms under consideration are given in Table 6.

Table 7
Summary table of the Wilcoxon test.

Optimizer	+	=	−
DE	88	22	19
jDE	62	56	11
JADE	68	39	22
SADE	66	31	32
EPSDE	76	45	8
MDE-pBX	80	27	22

Table 8
Holm–Bonferroni procedure (reference = variant with CBPI component).

Optimizer	z	p	α/i	Hypothesis
MDE-pBX	−5.57e + 00	1.28e−08	5.00e−02	Rejected
SADE	−4.07e + 00	2.39e−05	5.00e−02	Rejected
JADE	−3.18e + 00	7.31e−04	5.00e−02	Rejected
jDE	−5.04e + 00	2.35e−07	5.00e−02	Rejected
EPSDE	−6.45e + 00	5.51e−11	5.00e−02	Rejected
DE	−4.51e + 00	3.28e−06	5.00e−02	Rejected

Table 9
Cluster-based vs opposition-based initialization: Average Fitness ± Standard Deviation and Wilcoxon rank-sum test on the Fitness (reference = CBPI–DE) for CEC2013 in 10D.

	CBPI–DE	OBI–DE	
f1	**0.00e+00 ± 0.00e+00**	**0.00e+00 ± 0.00e+00**	=
f2	**2.17e+06 ± 1.49e+06**	2.43e+06 ± 1.10e+06	=
f3	**1.01e+07 ± 9.14e+06**	1.08e+07 ± 9.89e+06	+
f4	**1.38e+04 ± 4.36e+03**	1.53e+04 ± 3.64e+03	+
f5	**0.00e+00 ± 0.00e+00**	**0.00e+00 ± 0.00e+00**	=
f6	**3.82e+00 ± 3.83e+00**	4.32e+00 ± 3.49e+00	+
f7	**2.39e+01 ± 7.27e+00**	2.85e+01 ± 9.98e+00	+
f8	**2.04e+01 ± 5.71e−02**	2.04e+01 ± 8.10e−02	=
f9	**5.78e+00 ± 7.95e−01**	6.02e+00 ± 6.21e−01	+
f10	**1.19e+00 ± 1.96e−01**	1.28e+00 ± 1.65e−01	+
f11	**0.00e+00 ± 0.00e+00**	3.98e−02 ± 1.95e−01	+
f12	1.96e+01 ± 2.71e+00	**1.82e+01 ± 4.25e+00**	−
f13	**2.17e+01 ± 4.72e+00**	2.42e+01 ± 6.13e+00	+
f14	**2.09e−01 ± 6.57e−01**	2.04e−01 ± 6.46e−01	=
f15	**9.24e+02 ± 1.97e+02**	9.82e+02 ± 1.22e+02	+
f16	**1.03e+00 ± 1.86e−01**	1.09e+00 ± 1.82e−01	+
f17	1.02e+01 ± 7.99e−02	**9.93e+00 ± 1.07e+00**	=
f18	**3.86e+01 ± 5.12e+00**	3.89e+01 ± 4.68e+00	=
f19	**3.94e−01 ± 7.42e−02**	3.99e−01 ± 8.62e−02	+
f20	3.30e+00 ± 2.73e−01	**3.13e+00 ± 2.92e−01**	=
f21	**3.25e+02 ± 8.41e+01**	3.33e+02 ± 7.53e+01	+
f22	2.07e+01 ± 6.48e+00	**2.00e+01 ± 8.66e+00**	=
f23	**1.24e+03 ± 1.74e+02**	1.25e+03 ± 2.39e+02	=
f24	**1.50e+02 ± 2.35e+01**	1.73e+02 ± 2.68e+01	+
f25	2.05e+02 ± 2.22e+01	**1.93e+02 ± 2.63e+01**	=
f26	**1.32e+02 ± 7.54e+00**	1.41e+02 ± 7.23e+00	+
f27	**3.91e+02 ± 3.16e+01**	5.70e+02 ± 8.19e+01	+
f28	**2.79e+02 ± 5.30e+01**	2.87e+02 ± 5.51e+01	=
	15+, 12=, 1−		

3.2. Summary of the results

In order to summarize the results listed above, we reported in Table 7 the sum of significant and non-significant outperformance items, i.e. the sum, for each algorithm, of $+$, $-$, and $=$.

Table 8 displays the result of the Holm–Bonferroni procedure, see [31], for the CBPI variants of DE based algorithms under consideration over all the 129 problems studied in this article. For each variant, the Holm–Bonferroni procedure has been used to pair-wise compare the outperformance of the algorithm when undergone CBPI. The procedure is performed with confidence level of 0.05. CBPI-version is ranked first in each comparison and the null-hypothesis has been rejected in all cases, i.e. the CBPI-variant significantly outperforms the original algorithm in the totality of cases.

3.3. Cluster-based vs opposition-based initialization

In order to further prove the efficiency of the proposed CBPI component we have compared the effect of CBPI against the opposition-based initialization proposed in [60]. More specifically, we have implemented the opposition-based initialization described in [60] and then run the standard DE with the population initialized by the opposition-based technique. This variant of DE is here indicated as Opposition Based Initialization Differential Evolution (OBI–DE). Table 9 shows the comparison results between CBPI–DE and OBI–DE for the testbed CEC2013 in 10 dimensions. The same experimental conditions described above are replicated also for this experiment.

As highlighted, the CBPI component is as successful or more successful than the opposition-based initialization in the vast majority of the cases considered in this section.

4. Conclusion

This article proposes a software module that processes a population randomly sampled within a decision space and performs an intelligent sampling to detect the most interesting/promising areas of the domain. This software module is composed of three sub-modules that consecutively act on the sampled points. At first, two local search algorithms characterized by different search logics are applied to each solution with a limited budget. During the second stage, the improved solutions are clustered to identify the basins of attraction. During the third and last stage, the solutions are sampled from the clusters in order to compose a population belonging to the most promising basins of attraction. All in all, this software implicitly performs an analysis of the problem and extract an estimate of its multi-modality features. This intelligent sampling has been designed in order to enhance the performance of DE frameworks. The proposed software module has been tested on six different DE based algorithms, including a standard DE and five variants, some of them being popular algorithms and some others being recently proposed DE variants that represent the-state-of-the-art in optimization. Numerical experiments have been carried out on 128 test problems and a real-world application. Results show that the proposed preprocessing algorithm is an efficient software component that is able to consistently improve upon the performance of DE frameworks. The success of this intelligent sampling is, according to our interpretation, due to a balance between intra-cluster and inter-cluster mutation strategies. The first type of mutation promotes the sampling of points participating to the mutation from the same promising basins of attraction. Although this mechanism appears beneficial to DE schemes, in order to obtain a proper algorithmic functioning, it must be combined with another mechanism that, on the contrary, explores the entire decision space. This second mechanism is achieved during the inter-cluster mutations, i.e. those mutation that combine points belonging to different clusters. Finally, this study must be seen as a step towards the definition of an automatic problem analyser. In particular, the proposed software module implicitly performs an analysis on multi-modality and estimates the location of the strongest basins of attraction. Then, the module exploits this piece of information to "suggest" DE where to search for the global optimum. Future work will exploit more explicitly the results of the analysis phase to design the optimizer on the basis of the results obtained by the multi-modality test.

Acknowledgement

This research is supported by the Academy of Finland, Akatemiatutkija 130600, "Algorithmic design issues in Memetic Computing".

References

[1] G. Acampora, J.M. Cadenas, V. Loia, E.M. Ballester, A multi-agent memetic system for human-based knowledge selection, IEEE Trans. Syst. Man Cybern. – Part A 41 (5) (2011) 946–960.
[2] G. Acampora, M. Gaeta, V. Loia, Hierarchical optimization of personalized experiences for e-learning systems through evolutionary models, Neural Comput. Appl. 20 (5) (2011) 641–657.
[3] A. Auger, O. Teytaud, Continuous lunches are free!, in: Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation, ACM, 2007, pp 916–922.
[4] M.S. Bazaraa, H.D. Sherali, C.M. Shetty, Nonlinear Programming: Theory And Algorithms, Wiley-Interscience, 2006.
[5] M.F. Bramlette, Initialization, mutation and selection methods in genetic algorithms for function optimization, in: ICGA, 1991, pp. 100–107.
[6] J. Brest, S. Greiner, B. Bošković, M. Mernik, V. Žumer, Self-adapting control parameters in differential evolution: a comparative study on numerical benchmark problems, IEEE Trans. Evol. Comput. 10 (6) (2006) 646–657.
[7] J. Brest, P. Korosec, J. Silc, A. Zamuda, B. Boskovic, M.S. Maucec, Differential evolution and differential ant-stigmergy on dynamic optimisation problems, Int. J. Syst. Sci. 44 (4) (2013) 663–679.
[8] J. Brest, M.S. Maucec, Self-adaptive differential evolution algorithm using population size reduction and three strategies, Soft Comput. 15 (11) (2011) 2157–2174.
[9] J. Brest, M.S. Maučec, Population size reduction for the differential evolution algorithm, Appl. Intell. 29 (3) (2008) 228–247.
[10] E.K. Burke, Y. Bykov, in: PATAT '08 Proceedings of the 7th International Conference on the Practice and Theory of Automated Timetabling, 2008. <http://w1.cirrelt.ca/~patat2008/PATAT_7_PROCEEDINGS/Papers/Bykov-HC2a.pdf>.
[11] E.K. Burke, G. Kendall, E. Soubeiga, A tabu search hyperheuristic for timetabling and rostering, J. Heuristics 9 (6) (2003) 451–470.
[12] E.K. Burke, B. McCollum, A. Meisels, S. Petrovic, R. Qu, A graph-based hyperheuristic for educational timetabling problems, Eur. J. Oper. Res. 176 (2007) 177–192.
[13] Z. Cai, W. Gong, C.X. Ling, H. Zhang, A clustering-based differential evolution for global optimization, Appl. Soft Comput. 11 (1) (2011) 1363–1379.
[14] 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 Trans. Syst. Man Cybern. – Part B 37 (1) (2007) 28–41.
[15] A. Caponio, F. Neri, V. Tirronen, Super-fit control adaptation in memetic differential evolution frameworks, Soft Comput. – Fusion Found. Methodol. Appl. 13 (2009) 811–831.
[16] F. Caraffini, G. Iacca, F. Neri, E. Mininno, The importance of being structured: a comparative study on multi stage memetic approaches, in: Proceedings of the UK Workshop on Computational Intelligence, IEEE, 2012.
[17] F. Caraffini, F. Neri, G. Iacca, A. Mol, Parallel memetic structures, Inf. Sci. 227 (0) (2013) 60–82.
[18] F. Caraffini, F. Neri, L. Picinali, An analysis on separability for memetic computing automatic design, Inf. Sci. 265 (2014) 1–22.
[19] F. Caraffini, F. Neri, I. Poikolainen, Micro-differential evolution with extra moves along the axes. In: Proceedings of the IEEE Symposium Series on Computational Intelligence, 2013b, pp. 46–53.
[20] P. Cowling, G. Kendall, E. Soubeiga, A hyperheuristic approach to scheduling a sales summit, in: Proceedings of the Third International Conference on Practice and Theory of Automated Timetabling, Lect. Notes Comput. Sci., vol. 2079, Springer, 2000, pp. 176–190.
[21] S. Das, P. Suganthan, Problem Definitions and Evaluation Criteria for CEC 2011 Competition on Testing Evolutionary Algorithms on Real World Optimization Problems, Jadavpur Univ., Nanyang Technol. Univ., Kolkata, India, 2010.
[22] S. Das, P. Suganthan, Differential evolution: a survey of the state-of-the-art, IEEE Trans. Evol. Comput. 15 (1) (2011) 4–31.
[23] K.A. Dowsland, E. Soubeiga, E. Burke, A simulated annealing based hyperheuristic for determining shipper sizes for storage and transportation, Eur. J. Oper. Res. 179 (3) (2007) 759–774.

[24] J. Du, R. Rada, Memetic algorithms, domain knowledge, and financial investing, Memetic Comput. 4 (2) (2012) 109–125.
[25] R.C. Eberhart, J. Kennedy, A new optimizer using particle swarm theory. In: Proceedings of the Sixth International Symposium on Micromachine and Human Science, 1995, pp. 39–43.
[26] A.E. Eiben, J.E. Smith, Introduction to Evolutionary Computation, Springer Verlag, Berlin, 2003. pp. 175–188.
[27] M. Epitropakis, D. Tasoulis, N. Pavlidis, V. Plagianakos, M. Vrahatis, Enhancing differential evolution utilizing proximity-based mutation operators, IEEE Trans. Evol. Comput. 15 (1) (2011) 99–119.
[28] W. Gao, G. Yen, S. Liu, A cluster-based differential evolution with self-adaptive strategy for multimodal optimization, IEEE Trans. Cybern. 44 (8) (2014) 1314–1327.
[29] U. Halder, S. Das, D. Maity, A cluster-based differential evolution algorithm with external archive for optimization in dynamic environments, IEEE Trans. Cybern. 43 (3) (2013) 881–897.
[30] W.E. Hart, N. Krasnogor, J.E. Smith, Memetic evolutionary algorithms, in: W.E. Hart, N. Krasnogor, J.E. Smith (Eds.), Recent Advances in Memetic Algorithms, Springer, Berlin, Germany, 2004, pp. 3–27.
[31] S. Holm, A simple sequentially rejective multiple test procedure, Scand. J. Stat. 6 (2) (1979) 65–70.
[32] H.H. Hoos, Programming by optimization, Commun. ACM 55 (2) (2012) 70–80.
[33] F. Hutter, H.H. Hoos, K. Leyton-Brown, Tradeoffs in the empirical evaluation of competing algorithm designs, Ann. Math. Artif. Intell. 60 (1–2) (2010) 65–89.
[34] F. Hutter, L. Xu, H.H. Hoos, K. Leyton-Brown, Algorithm runtime prediction: methods and evaluation, Artif. Intell. 206 (2014) 79–111.
[35] 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.
[36] S. Islam, S. Das, S. Ghosh, S. Roy, P. Suganthan, An adaptive differential evolution algorithm with novel mutation and crossover strategies for global numerical optimization, IEEE Trans. Syst. Man Cybern. – Part B: Cybern. 42 (2) (2012) 482–500.
[37] X. Li, X. Yao, Cooperatively coevolving particle swarms for large scale optimization, IEEE Trans. Evol. Comput. 16 (2) (2012) 210–224.
[38] J.J. Liang, B.Y. Qu, P.N. Suganthan, A.G. Hernndez-Daz, Problem Definitions and Evaluation Criteria for the CEC 2013 Special Session on Real-Parameter Optimization, Tech. Rep. 201212, Zhengzhou University and Nanyang Technological University, Zhengzhou China and Singapore, 2013.
[39] G. Liu, Y. Li, X. Nie, H. Zheng, A novel clustering-based differential evolution with 2 multi-parent crossovers for global optimization, Appl. Soft Comput. 12 (2) (2012) 663–681.
[40] S. Lloyd, Least squares quantization in PCM, IEEE Trans. Inf. Theory 28 (2) (1982) 129–137.
[41] D. Loughlin, S. Ranjithan, Chance-Constrained Optimization Using Genetic Algorithms: An Application in Air Quality Management, 1999, pp. 1–9 (Chapter 49).
[42] H. Maaranen, K. Miettinen, M.M. Mäkelä, Quasi-random initial population for genetic algorithms, Comput. Math. Appl. 47 (12) (2004) 1885–1895.
[43] R. Mallipeddi, S. Mallipeddi, P.N. Suganthan, Ensemble strategies with adaptive evolutionary programming, Inf. Sci. 180 (9) (2010) 1571–1581.
[44] R. Mallipeddi, P.N. Suganthan, Q.K. Pan, M.F. Tasgetiren, Differential evolution algorithm with ensemble of parameters and mutation strategies, Appl. Soft Comput. 11 (2) (2011) 1679–1696 (the Impact of Soft Computing for the Progress of Artificial Intelligenc).
[45] M.A. Montes de Oca, T. Stutzle, M. Birattari, M. Dorigo, Frankenstein's PSO: a composite particle swarm optimization algorithm, IEEE Trans. Evol. Comput. 13 (5) (2009) 1120–1132.
[46] P. Moscato, M. Norman, A Competitive and Cooperative Approach to Complex Combinatorial Search, Tech. Rep. 790, 1989.
[47] R. Mukherjee, G.R. Patra, R. Kundu, S. Das, Cluster-based differential evolution with crowding archive for niching in dynamic environments, Inf. Sci. 267 (0) (2014) 58–82.
[48] N. Hansen, A. Auger, S. Finck, R. Ros, et al., Real-Parameter Black-Box Optimization Benchmarking 2010: Noiseless Functions Definitions, Tech. Rep. RR-6829, INRIA, 2010.
[49] F. Neri, C. Cotta, Memetic algorithms and memetic computing optimization: a literature review, Swarm Evol. Comput. 2 (2012) 1–14.
[50] F. Neri, C. Cotta, P. Moscato, Handbook of Memetic Algorithms, Studies in Computational Intelligence, vol. 379, Springer, 2011.
[51] F. Neri, E. Mininno, G. Iacca, Compact particle swarm optimization, Inf. Sci. 239 (2013) 96–121.
[52] F. Neri, V. Tirronen, Recent advances in differential evolution: a review and experimental analysis, Artif. Intell. Rev. 33 (1–2) (2010) 61–106.
[53] F. Neri, J.I. Toivanen, G.L. Cascella, Y.S. Ong, An adaptive multimeme algorithm for designing HIV multidrug therapies, IEEE/ACM Trans. Comput. Biol. Bioinf. 4 (2) (2007) 264–278.
[54] Y.-S. Ong, M.-H. Lim, X. Chen, Research Frontier: Towards Memetic Computing, Tech. Rep. C2i-1209, School of Computer Engineering, Nanyang Technological University, Singapore, 2009.
[55] Y.-S. Ong, M.-H. Lim, X. Chen, Memetic computation-past, present and future, IEEE Comput. Intell. Mag. 5 (2) (2010) 24–31.
[56] J.R. Palmer, An improved procedure for orthogonalising the search vectors in Rosenbrock's and Swann's direct search optimisation methods, Comput. J. 12 (1) (1969) 69–71.
[57] F. Peng, K. Tang, G. Chen, X. Yao, Population-based algorithm portfolios for numerical optimization, IEEE Trans. Evol. Comput. 14 (5) (2010) 782–800.
[58] R. Poli, M. Graff, There is a free lunch for hyper-heuristics, genetic programming and computer scientists. In: EuroGP, 2009, pp. 195–207.
[59] A.K. Qin, V.L. Huang, P.N. Suganthan, Differential evolution algorithm with strategy adaptation for global numerical optimization, IEEE Trans. Evol. Comput. 13 (2) (2009) 398–417.
[60] S. Rahnamayan, H.R. Tizhoosh, M.M.A. Salama, Quasi-oppositional differential evolution. In: Proceedings of the IEEE Congress on Evolutionary Computation, 2007, pp. 2229–2236.
[61] A. Reese, Random number generators in genetic algorithms for unconstrained and constrained optimization, Nonlinear Anal.: Theory Methods Appl. 71 (12) (2009) e679–e692.
[62] H.H. Rosenbrock, An automatic method for finding the greatest or least value of a function, Comput. J. 3 (3) (1960) 175–184.
[63] P.J. Rousseeuw, Silhouettes: a graphical aid to the interpretation and validation of cluster analysis, J. Comput. Appl. Math. 20 (1987) 53–65.
[64] J. Tang, M.H. Lim, Y.S. Ong, Diversity-adaptive parallel memetic algorithm for solving large scale combinatorial optimization problems, Soft Comput. – Fusion Found. Methodol. Appl. 11 (9) (2007) 873–888.
[65] K. Tang, X. Li, P.N. Suganthan, Z. Yang, T. Weise, Benchmark Functions for the CEC'2010 Special Session and Competition on Large-Scale Global Optimization, Tech. rep., University of Science and Technology of China (USTC), School of Computer Science and Technology, Nature Inspired Computation and Applications Laboratory (NICAL), Hefei, Anhui, China, 2010.
[66] V. Tirronen, F. Neri, T. Kärkkäinen, K. Majava, T. Rossi, A memetic differential evolution in filter design for defect detection in paper production, in: Applications of Evolutionary Computing, Lectures Notes in Computer Science, vol. 4448, Springer, Berlin, Germany, 2007, pp. 320–329.
[67] L.-Y. Tseng, C. Chen, Multiple trajectory search for large scale global optimization, in: Proceedings of the IEEE Congress on Evolutionary Computation, 2008, pp. 3052–3059.
[68] J.A. Vrugt, B.A. Robinson, J.M. Hyman, Self-adaptive multimethod search for global optimization in real-parameter spaces, IEEE Trans. Evol. Comput. 13 (2) (2009) 243–259.
[69] H. Wang, I. Moon, S. Yang, D. Wang, A memetic particle swarm optimization algorithm for multimodal optimization problems, Inf. Sci. 197 (0) (2012) 38–52.
[70] Y.-J. Wang, J.-S. Zhang, G.-Y. Zhang, A dynamic clustering based differential evolution algorithm for global optimization, Eur. J. Oper. Res. 183 (1) (2007) 56–73.
[71] D.H. Wolpert, W.G. Macready, No free lunch theorems for optimization, IEEE Trans. Evol. Comput. 1 (1) (1997) 67–82.
[72] L. Xu, F. Hutter, H. Hoos, K. Leyton-Brown, SATzilla: portfolio-based algorithm selection for SAT, J. Artif. Intell. Res. 32 (2008) 565–606.
[73] J. Zhang, A. Sanderson, Jade: adaptive differential evolution with optional external archive, IEEE Trans. Evol. Comput. 13 (t5) (2009) 945–958.