A review of the recent use of Differential Evolution for Large-Scale Global Optimization: An analysis of selected algorithms on the CEC 2013 LSGO benchmark suite

Mirjam Sepesy Maučec*, Janez Brest

Faculty of Electrical Engineering and Computer Science
University of Maribor
SI-2000 Maribor, Slovenia

Abstract

This paper gives a review of recent extensions of the Differential Evolution (DE) algorithm for use in Large-Scale Global Optimization (LSGO) and presents an empirical analysis of DE-based and some other state-of-the-art algorithms for LSGO on the CEC 2013 LSGO benchmark suite. Since witnessing the first successful applications of DE for a wide variety of optimization problems in the early nineties, researchers have developed several new algorithms in this field. In this paper, we are especially interested in algorithms for solving LSGO. As LSGO is one of the most active research lines, not only in DE, but in many evolutionary and meta-heuristic algorithms, we discuss general approaches for dealing with LSGO first. The main focus of the paper is DE. We review its basic algorithm and discuss several extensions used for coping with large-scale problems. This paper has two main objectives: (1) To propose, from a theoretical point of view, the grouping of DE mechanisms for dealing with LSGO into nine groups, and (2) To evaluate sixteen recently proposed algorithms for LSGO empirically. Many benchmark suites were designed with the aim of providing a suitable evaluation platform for testing and comparing large-scale optimization algorithms. In this paper, the CEC 2013 LSGO benchmark suite was chosen for comparison, because it resembles the following features of real-world problems: Non-uniform subcomponent sizes; imbalance in the contribution of subcomponents; and functions with interdependent overlapping subcomponents. The performances of state-of-the-art algorithms are compared, and the algorithms are ranked using three different metrics, which evaluate the performance from different perspectives. The conducted research shows that DE is among the best algorithms for LSGO on the CEC 2013 LSGO benchmark suite, especially when used with other mechanisms for dealing with large numbers of variables. Finally, the analysis has shown that there is still some room for further improvements in DE towards the solution of LSGO problems.

Keywords: High-dimensional optimization; Differential evolution; Comparative study; Grouping mechanisms; Benchmark functions; Algorithms' comparison.

*Corresponding author
Email addresses: `mirjam.sepesy@um.si` (Mirjam Sepesy Maučec), `janez.brest@um.si` (Janez Brest)

1. Introduction

Finding the global optimum of a complex function is one of the long-standing goals of applied mathematics and numerical analysis. Special challenges are problems characterized by non-linear, non-convex, multi-modal and non-differentiable functions defined in the continuous space. Many meta-heuristic algorithms were defined to solve them. Among them, Evolutionary Algorithms (EAs) have proved to be very robust with good global search capability. A common characteristic of EAs is that they evolve chromosomes that are vectors of floating point numbers. Chromosomes represent problem solutions directly. Hence, they may be called real-coded EAs. The most prominent representatives of EAs are: Real-coded genetic algorithms, evolution strategies, evolutionary programming, real-coded memetic algorithms and Differential Evolution. In recent years, an algorithm based on swarm intelligence emerged as a competitive alternative to EA. The representatives from this group are Particle Swarm Optimization (PSO), ant colony optimization (ACO) and Artificial Bee Colony (ABC) optimization.

In this paper, we are interested in Differential Evolution (DE), which emerged as a very competitive algorithm more than two decades ago. Since then, DE, in its original form as well as its variants, have been applied successfully to solve numerous real-world problems from diverse domains of science and technology.

Most real-world optimization problems include a large number of decision variables, known as Large-Scale Global Optimization (LSGO) problems. When solving them, the standard meta-heuristic algorithms suffer from one main deficiency, the curse of dimensionality. The performance of algorithms deteriorates when tackling high dimensional problems. By increasing the dimensionality of the problem its landscape complexity grows. At the same time, the search space increases exponentially. An optimization algorithm must be able to explore the entire search space efficiently, which is not a trivial task.

The motivation of this work is to consider the power of the DE algorithm for tackling LSGO problems. Solving LSGO problems is becoming more and more attractive as the technology is evolving and more computational power is available. DE, it's modifications and hybridizations, have been used in many researches, and the ideas are dispersed among many papers addressing the LSGO problems. The purpose of this paper is to review recent extensions of DE for LSGO and to evaluate their contributions in solving LSGO problems in comparison with other bio-inspired algorithms. The paper includes an empirical comparison of sixteen state-of-the-art algorithms for LSGO problems. There has been a study of ten large-scale global optimizers, reported in [1]. In this paper, we will enlarge the set of algorithms and present the analysis from a different perspective.

The CEC 2013 LSGO benchmark functions were chosen for the comparative analysis. They have three major features that resemble real-world problems [2]: Non-uniform subcomponent sizes; imbalance in the contribution of subcomponents; and functions with interdependent overlapping subcomponents. It should be noted that some features, which also belong to the key features of real-world problems, like the use of rotation matrices, are not present in the benchmark suite. All these features pose a challenge to large-scale optimization algorithms.

The main contributions of this paper are: (1) Review of recent uses and modifications of DE in solving LSGO problems; (2) Grouping of different mechanisms for LSGO into nine categories; (3) An empirical comparison of sixteen state-of-the-art algorithms for LSGO problems on CEC 2013 LSGO benchmark functions; (4) Comparisons based on three different evaluation methods.

The paper is organized as follows. Section 2 exposes problems of LSGO, overviews some benchmark suites and describes the CEC 2013 LSGO benchmark suite in some detail. Section 3 outlines common mechanisms for dealing with LSGO. Section 4 reviews the features of the original DE, which are advanced in Section 5 to cope with LSGO problems. Comparative analysis of state-of-the-art algorithms is given in Section 6, where the main empirical research observations are presented. Section 7 concludes the paper and summarizes the objectives addressed.

2. Benchmarks for Large-Scale Global Optimization

The optimization problems we are dealing with in this paper are large-scale, continuous and unconstrained. All of them have a single objective function.

When the number of variables is high, there are several dependencies among the variables. The contribution of the variables to the final result could be independent of the other ones, but that is unusual. These functions are called separable functions and they are easier, but less common, in real problems. In the majority of problems there are sets of variables (i.e., related to the same real-world components, or inter-dependencies' components) with a strong relationship between them. These functions are called non-separable functions, and they are more usual and, unfortunately, more difficult to solve. In non-separable functions, if some variables are related with more than a group of variables, they are called overlapping functions, and they are the most difficult case [2].

The optimization problem is usually stated as a minimization problem:

$$\arg\min_{\vec{x}} f(\vec{x}),$$

where $\vec{x} = (x_1, x_2, ..., x_j, ..., x_D)$ is a vector of D dimensions and $f(\vec{x})$ is a function to be optimized.

Benchmark functions, specially those designed for LSGO, were defined to enable a fair comparison of the performances of several algorithms. The first large-scale benchmark suite was defined for the CEC 2008 Special Session and Competition on Large-Scale Global Optimization [3]. In the CEC 2008 suite the problems were either separable or non-separable. As most of the real-world problems fall somewhere in between these two extreme cases, a new benchmark suite was defined for the CEC 2010 special session [4]. Disjoint subsets of the decision variables are called subcomponents. The benchmark suite contains four major categories of functions: Fully separable functions; partially separable functions with a separable subcomponent; partially separable functions with no separable subcomponent; and fully non-separable functions. There were three major shortcomings of the CEC 2010 suite: The fixed size of all subcomponents; the identical contribution of all subcomponents; and the completely independent subcomponents. Such situations are very unlikely in real-world problems. The CEC 2013 LSGO benchmark suite was defined to approach the real-world environments more closely [5].

At the same time, some other benchmark suites were defined and used by the research community. In the year 2010, the SOCO benchmark suite of 19 functions was defined and used in a Special Issue of the Soft Computing Journal [6]. All functions were completely scalable functions. In 2016, a large-scale version was also defined of the 24 functions from the BBOB-2009 testbed [7, 8]. The authors applied an orthogonal transformation on raw functions that involved only a linear number of operations in order to obtain large-scale optimization benchmark problems. The orthogonal transformation was used as a replacement for the expensive orthogonal (rotation) matrices.

2.1. Definition of functions regarding the separation of variables

Regarding the separation of variables, the functions can be separable, partially separable or non-separable. A function $f(\vec{x})$ is fully separable iff (if and only if):

$$\arg\min_{\vec{x}} f(\vec{x}) = (\arg\min_{x_1} f(\vec{x}), ..., \arg\min_{x_D} f(\vec{x})), \tag{1}$$

where $\vec{x} = (x_1, ..., x_D)$ is a decision vector of D dimensions.

A function $f(\vec{x})$ is partially separable with m independent subcomponents iff:

$$\arg\min_{\vec{x}} f(\vec{x}) = (\arg\min_{\vec{x}_1} f(\vec{x}_1, ...), ..., \arg\min_{\vec{x}_m} f(..., \vec{x}_m)), \tag{2}$$

where $\vec{x}_1, ..., \vec{x}_m$ are disjoint sub-vectors of $\vec{x}$ and $2 \leq m < D$.

A function is fully non-separable if every pair of its decision variables interacts with each other.

A function is partially additively separable iff:

$$f(\vec{x}) = \sum_{i=1}^{m} f_i(\vec{x}_i), \tag{3}$$

where $\vec{x}_i$ are mutually exclusive decision vectors of f_i and m is the number of separable subcomponents.

Partially separable functions can have (fully) separable subcomponents and non-separable subcomponents. If the subcomponent in some degree overlaps with its neighboring subcomponents, the function is called an overlapping function.

Non-separable and overlapping functions are the most difficult to solve.

2.2. *CEC 2013 LSGO benchmark suite*

In this paper, the CEC 2013 LSGO benchmark suite was chosen for the comparison of algorithms. The reason is its wide use in literature. It comprises $N = 15$ functions (see Table 1) that are based on the following original functions:

1. Sphere function,
2. Elliptic function,
3. Rastrigin's function,
4. Ackley's function,
5. Schwefel's Problem 1.2, and
6. Rosenbrock's function.

Table 1: Summary of the CEC 2013 LSGO benchmark functions.

		Function	Properties	Search Range
Fully-separable Functions	1	Elliptic Function	Unimodal; Shifted;	$[-100,100]^D$
	2	Rastrigin Function	Multimodal; Shifted;	$[-5,5]^D$
	3	Ackley Function	Multimodal; Shifted;	$[-32,32]^D$
Functions with a separable subcomponent	4	Elliptic Function	Unimodal; Shifted;	$[-100,100]^D$
	5	Rastrigin Function	Multimodal; Shifted;	$[-5,5]^D$
	6	Ackley Function	Multimodal; Shifted;	$[-32,32]^D$
	7	Schwefels Problem 1.2	Multimodal; Shifted;	$[-100,100]^D$
Functions with no separable subcomponents	8	Elliptic Function	Unimodal; Shifted;	$[-100,100]^D$
	9	Rastrigin Function	Multimodal; Shifted;	$[-5,5]^D$
	10	Ackley Function	Multimodal; Shifted;	$[-32,32]^D$
	11	Schwefels Problem 1.2	Unimodal; Shifted;	$[-100,100]^D$
Overlapping Functions	12	Rosenbrock's Function	Multimodal; Shifted;	$[-100,100]^D$
	13	Schwefels Function with Conforming Overlapping Subcomponents	Unimodal; Shifted;	$[-100,100]^D$
	14	Schwefels Function with Conflicting Overlapping Subcomponents	Unimodal; Shifted;	$[-100,100]^D$
Non-separable Functions	15	Schwefels Problem 1.2	Unimodal; Shifted;	$[-100,100]^D$
$F_i^* = F_i(\vec{x}^*) = 0$; $\vec{x}^*$ denotes the true optimal solution				

For all functions, function value in the global optimum is 0. The functions exhibit the following key features:

- Being fully separable. There are two crucial questions for these functions: How to decompose a function into several smaller subproblems and optimize them independently? What is the relationship between the dimensionality of each subproblem and the subpopulation size of the algorithm?

- Having separable subcomponents. The idea behind these functions is that the problem can be decomposed by having domain knowledge about the problem, or by using an automatic decomposition algorithm, so that each subcomponent is optimized independently.

- Subcomponents are of non-uniform sizes. In the real-world, it is very unlikely that problems have a uniform structure.

- Imbalance in the contribution of subcomponents. It is often the case that the nature of subcomponent functions is different, which results in a different degree of contribution from each subcomponent to the global objective function. The idea is to allocate the computational resources depending on the contribution of each subcomponent.

- Functions with overlapping subcomponents. Subcomponents may interact between each other, and the global optimum of a problem cannot necessarily be found by finding the global optimum of its subcomponent problems.

In addition to the listed features, some functions were also exposed to transformations, such as: Ill-conditioning, and irregularities. The idea behind all these features is a better resemblance with real-world problems [2]. For benchmarking the algorithms, the dimension is set to $D = 1000$. In the continuation of the paper we will discuss the results obtained on the CEC 2013 LSGO benchmark suite.

Optimization benchmark problems are usually stated in terms of minimization. Ref. [9] highlighted a problem that benchmarking heuristics on either minimization or maximization of the same set of functions may lead to very different ranking of tested algorithms. As algorithms are developed in order to be applicable to real-world problems, there is little reason that searching for the minimum of such functions should be more important than searching for their maximum [9]. In [2] different benchmarks are analyzed in some detail, in order to show how similar to the features of real-world problems they are. The authors proposed some general guidelines that can be used to design and construct various benchmark suites to meet different needs.

3. LSGO mechanisms

The dimensionality of the problem has a direct impact on the computational cost of the optimization. Algorithms which search within the neighborhood of a candidate solution might require a very large number of fitness evaluations at each step of the search. Population based algorithms are likely to converge prematurely to suboptimal solutions, or stagnate due to an inability to generate new promising search directions. Approaches that inspect the interaction between pairs of variables can be computationally onerous. For these reasons, there is a clear need to modify the basic principles of algorithms to cope with large-scale problems.

In the last decade, several modern meta-heuristics have been proposed in order to tackle LSGO [10, 11]. Caraffini et al. [12] proposed the division of methods into two main categories: Methods that exploit promising search directions intensively; methods that decompose the search space.

3.1. Exploitation of promising search directions

Using an exhaustive search, the LSGO problem would be extremely hard to solve in a feasible time. For this reason, the first group of algorithms gives up the search for the global optimum, and, instead of that, tries to detect a solution with a high quality, regardless of its optimality. Two ways were proposed to achieve this aim. The first was to use very small populations in population-based algorithms, or to start with a larger population, but shrink it during the run. Refs. [13, 14] and [15] employed cooperative small populations to adapt different subcomponents of high-dimensional optimization problems concurrently in DE, PSO and ABC, respectively. Population size reduction in DE was demonstrated in [16, 17, 18, 19]. The second way to detect a solution with a high quality is by using local search in combination with other algorithms. In [20] an extra search move is incorporated within the DE scheme to attempt to improve the best solution by perturbing it along the axes. Refs. [21, 22] and [23] use local search within a memetic computing algorithm.

The authors in [24] propose an ant-based algorithm to solve the LSGO problem. The idea behind their algorithm is in remembering the move in the search space that improves the current best solution, and directing a further search based on this move. For this purpose, they created a fine-grained discrete form of the continuous domain that represents the search space as a graph. This graph was then used as a means for ants to walk on.

For tackling separable problems, Refs. [25] and [26] propose a simple modification of the Covariance Matrix Adaptation-Evolution Strategy (CMA-ES). The covariance matrix is constrained to be diagonal, and the resulting algorithm samples each coordinate independently. Because the model complexity is reduced, the learning rate for the covariance matrix can be increased. CMA-ES is capable of optimizing unimodal functions efficiently, even when confronted with non-separable problems. MA-SW-Chains [27] is a memetic algorithm that uses the idea of CMA-ES. It assigns a local search intensity to each individual that depends on its features, by chaining different local search applications. To improve the behavior of CMA-ES also on multimodal functions, in [28], CMA-ES was paired with minimum population search, that brought good exploration ability to the algorithm. In [29] the mutation, crossover and selection operators of DE were incorporated into CMA-ES to achieve a better trade-off between its explorative and exploitative tendencies. Premature convergence is a major drawback of PSO. Huang et al. [30] proposed a convergence speed controller for adjusting the convergence speed adaptively.

3.2. Search space decomposition

A natural approach to tackle large-scale optimization problems is to adopt a divide-and-conquer strategy to decompose a large-scale problem into a set of lower dimensional problems which are easier to solve. The main challenge is to find the appropriate decomposition technique, because it depends on the problem. Potter and De Jong [31] proposed a technique, named Cooperative Coevolution (CC), for decomposing a high-dimensional problem, and tackling its subcomponents individually. If the fitness function is separable, then the problem decomposition is trivial, while for non-separable functions, the problem decomposition can turn out to be a very difficult task. Some techniques were proposed for performing the decomposition of non-separable functions. MLCC [32] uses a set of problem decomposers, constructed based on the random grouping strategy with different group sizes. The evolution process is divided into a number of cycles, and, at the start of each cycle, MLCC uses a self-adapted mechanism to select a decomposer according to its historical performance. Since different group sizes capture different interaction levels between the original objective variables, MLCC is able to self adapt among different levels. In [33] random grouping strategy is supplemented with an adaptive weighting in order to strengthen further co-adaptation among decomposed subcomponents when they are interdependent. A weight is applied to each subcomponent after each cycle, and the weight vector is evolved with a certain optimizer. Another technique is differential grouping [34]. It studies the dependencies among variables, comparing solutions with small changes in them. Then, it uses that information for grouping together variables with dependencies among them. Traditional CC tends to converge to the Nash equilibrium rather than the global optimum, due to the information loss accompanied with problem decomposition. In [35] the idea of dynamic optimization and the mechanism of multi-populations is used to achieve information compensation.

Peng and Wu [36] improved the information exchange further by a niching-based multimodal optimization. The idea was to increase the number of informative collaborators when the main optimizing procedure becomes stuck. Additionally, an efficient fast clustering method was used to cluster the collated collaborators, so the number of collaborators that were used to evaluate a given individual was reduced dramatically and, therefore, saved computational resources.

The major problem of CC is that the partial solution to a sub-problem cannot be evaluated naturally with respect to the original objective function. Yang et al. [37] suggest that searching a good solution to a sub-problem can be viewed as a computationally expensive problem that they address with the aid of a meta-model.

Another problem of CC is that decomposition methods are typically computationally expensive. In [38], an incremental CC algorithm was proposed which increased the complexity of the search space incrementally. Sun et al. [39] proposed a recursive differential grouping, which considered the interaction between decision variables based on non-linearity detection. The interaction between a selected decision variable and the remaining variables was examined recursively, without examining all pairwise variable interactions explicitly. Hu et al. [40] proposed a fast interdependency identification algorithm that identified separable and non-separable variables first, and then investigated only non-separable variables further. To make near-optimal decomposition for CC, the proposed algorithm avoids the necessity of obtaining the full interdependency information of non-separable variables. Therefore, a significant number of function evaluations is saved.

Cooperative Coevolution procedures have been integrated successfully within DE [33, 41], PSO [42, 43] and ABC [44, 45] frameworks. In [46], differential grouping and CMA-ES were combined under the Cooperative Coevolution framework. The combination was named CC-CMA-ES.

4. Differential Evolution

The aim of the paper is to analyze how well DE suits in LSGO problem solving. Let us review the general features of DE first.

The Differential Evolution [47] algorithm is a stochastic population based algorithm that was originally designed for numerical optimization. DE belongs to a group of Evolutionary Algorithms, which, besides DE, include genetic algorithms, evolutionary programming, genetic programming, and evolutionary strategies. The original DE was proposed in 1995 as a technical report [48], and published in a scholar journal in 1997 [47].

In recent decades it has shown robustness, efficiency, and it was very competitive when it was applied to solve real-world optimization problems [49, 50, 51].

An optimization process of a population based algorithm can be described as follows: A population consists of Np vectors that are usually initialized uniformly randomly over a search space. Then, the population is evolved throughout several generations, guiding the vectors in a search space toward a global optimum. At the end of the evolutionary process, i.e. after a predefined maximal number of generations, G_{max}, the algorithm stops and returns the best fitted vector as the final solution.

Population $\mathbf{P}$ in the DE algorithm is combined of Np vectors:

$$\mathbf{P}_g = (\vec{x}_{1,g}, \ldots, \vec{x}_{i,g}, \ldots, \vec{x}_{Np,g}), \;\; i = 1, 2, ..., Np,$$

where g denotes a generation index, $g = 1, 2, \ldots, G_{max}$. Each vector consists of D variables or components:

$$\vec{x}_{i,g} = (x_{1,i,g}, ..., x_{j,i,g}, ..., x_{D,i,g}), \;\; j = 1, 2, ..., D.$$

DE employs three operations for each vector during each generation. The operations are mutation, crossover, and selection. Note that similar operations are applied in other EAs, but they are performed in a different order, e.g. a crossover operation is followed by a mutation operation in genetic algorithms.

Mutation

The first DE operation is mutation. During this operation, DE creates a mutant vector $\vec{v}_{i,g}$ using one of the mutation strategies. The "DE/rand/1" mutation strategy was introduced in the original DE algorithm [47] and it is one of the most used mutation strategies when solving optimization problems with DE. This strategy selects two vectors randomly from population $\mathbf{P}$, and their difference, multiplied by scale factor F, is added to a third randomly selected vector. We can express this strategy as follows:

$$\vec{v}_{i,g} = \vec{x}_{r_1,g} + F \cdot (\vec{x}_{r_2,g} - \vec{x}_{r_3,g}),$$

where r_1, r_2, and r_3 are randomly chosen indexes within a set of $\{1, Np\}$ and they are, pairwise, different and also different from index i, i.e., $r_1 \neq r_2 \neq r_3 \neq i$.

There exist many other DE mutation strategies, so let us summarize those that are very often used in practice:

- "DE/rand/1": $\vec{v}_{i,g} = \vec{x}_{r_1,g} + F(\vec{x}_{r_2,g} - \vec{x}_{r_3,g})$,
- "DE/best/1": $\vec{v}_{i,g} = \vec{x}_{best,g} + F(\vec{x}_{r_1,g} - \vec{x}_{r_2,g})$,
- "DE/current-to-best/1":
  $\vec{v}_{i,g} = \vec{x}_{i,g} + F(\vec{x}_{best,g} - \vec{x}_{i,g}) + F(\vec{x}_{r_1,g} - \vec{x}_{r_2,g})$,
- "DE/best/2":
  $\vec{v}_{i,g} = \vec{x}_{best,g} + F(\vec{x}_{r_1,g} - \vec{x}_{r_2,g}) + F(\vec{x}_{r_3,g} - \vec{x}_{r_4,g})$,
- "DE/rand/2":
  $\vec{v}_{i,g} = \vec{x}_{r_1,g} + F(\vec{x}_{r_2,g} - \vec{x}_{r_3,g}) + F(\vec{x}_{r_4,g} - \vec{x}_{r_5,g})$,

where the randomly chosen indexes r_1–r_5 are mutually different integers generated within the range $\{1, Np\}$ and also different from index i. $\vec{x}_{best}$ is the best vector in a current generation. Recently, "DE/current-to-p-best/1" strategy has been introduced, and to get a more detailed overview of DE strategies, a reader is referred to the DE surveys [51, 49, 50]. Each strategy has a different ability to maintain the population diversity, which might increase the algorithm's convergence rate during the evolutionary process, and, therefore, the mutation operation in DE seems to have a great impact on an algorithm's performance. For example, in genetic algorithms, a mutation is used to make small changes/perturbations in a vector's components, while changes of the vector's components in DE are usually larger.

Crossover

The next operation is called crossover. It follows the mutation operation. A mutant vector $\vec{v}_{i,g}$ and parent vector $\vec{x}_{i,g}$ are used in binomial crossover in order to create a trial vector $\vec{u}_{i,g}$ as follows:

$$u_{j,i,g} = \begin{cases} v_{j,i,g}, & \text{if } rand(0,1) \leq Cr \text{ or } j = j_{rand}, \\ x_{j,i,g}, & \text{otherwise,} \end{cases} \tag{4}$$

for $i = 1, 2, ..., Np$ and $j = 1, 2, ..., D$. $Cr \in [0, 1]$ is a crossover parameter and presents the probability of creating components for a trial vector from a mutant vector. If a component was not selected from the mutant vector, then it is taken from the parent vector $\vec{x}_{i,g}$. Randomly chosen index $j_{rand} \in \{1, 2, ..., Np\}$ is responsible for the trial vector to contain at least one component from the mutant vector.

If some variables from the trial vector are out of bounds, a repeat mechanism is applied.

The binomial crossover (Eq. 4) is used widely in DE, the other crossover is the exponential crossover [47, 49].

Selection

Before the last operation takes place, the trial vector is evaluated, i.e., an objective function $f(\vec{u}_{i,g})$ is calculated. Two vectors, trial vector $\vec{u}_{i,g}$, and its corresponding population vector $\vec{x}_{i,g}$, are compared according to their objective function values, and the better one will become a member of the next generation, while the other one is discarded. Given a minimization optimization problem, the selection operation is defined as follows:

$$\vec{x}_{i,g+1} = \begin{cases} \vec{u}_{i,g}, & \text{if } f(\vec{u}_{i,g}) \leq f(\vec{x}_{i,g}), \\ \vec{x}_{i,g}, & \text{otherwise.} \end{cases}$$

This selection operation is greedy and it is known for DE, but rarely applied in other EAs, where tournament, roulette wheel and other selections are used.

5. DE and mechanisms for LSGO

The original DE suffers from slow convergence and stagnation when applied to LSGO. Several modifications of DE in different ways have been proposed to be efficient also in a high-dimensional continuous space. In this section, we will dive more deeply into the details of recently proposed mechanisms used in DE to cope with LSGO. We group the mechanisms in nine groups. Some mechanisms are also reviewed that were not yet used within DE, but are defined independent of the hosting algorithm. Approaches will be analyzed according to the mechanism they use, although it is difficult, since some of the approaches combine multiple mechanisms together.

5.1. Control parameters and their adaptation

The original DE uses control parameters with pre-fixed values, but there has been a growing trend to adopt them based on success histories of the generated trial vectors. jDE [52], JADE [53] and SaDE [54, 55] are DE variants that involve adaptation of control parameters F and Cr. Only the initial parameters' settings are required, and then the parameters are self-adapted gradually, according to the learning experience. When and how often should the parameters be adapted? In jDE, the control parameters change values using a randomization method, where the probabilities of adjusting control parameters take fixed values. Zamuda and Brest [56] studied the frequency of control parameter adjustment (i.e. randomization frequency) in more detail. The research has shown that it might be necessary to tailor it differently with different function types. In [57] the potential of using the extended intervals for self-adapted F and Cr is discussed.

In [32] a DE variant called SaNSDE was defined. It builds up on Differential Evolution with Neighborhood Search (NSDE) [58] to adapt the scale factor F. It is inspired by the neighborhood search strategy in evolutionary programming. It has been found that NSDE is effective in escaping from local optima when searching in environments without prior knowledge about what kind of search step size will be preferred. Three self-adaptive mechanisms are also utilized in SaNSDE: Self-adaptation for two candidate mutation strategies, and self-adaptations for F and Cr, respectively.

Two local search algorithms were also used in [59] to detect a value of F to generate offspring with a high performance during the mutation process.

In [60] the control parameters were adapted by using the Cauchy distribution with the average of the last successful generation's control parameters. In [61] parameter adaptation was improved further using the stability parameter of the α-stable distribution based on some feedbacks from the optimization process.

In [62] parameter F was adapted based on landscape modality detection (unimodal or multimodal).

Yang et al. [63] proposed an algorithm, named GADE, which includes a generalized adaptation framework based on a study of different parameter adaptation techniques for LSGO.

In [64], instead of parameter adaptation for each population member, the F value is switched between 0.5 and 2 in a uniformly randomized way. $F = 2$ is somewhat an unusual choice, since this extreme value has not been reported for DE in literature. However, their experiments indicate that for the large-scale problems this value can indeed enhance the performance of DE. Similarly for each individual, the Cr value is switched between 0 and 1. The same parameter switching was used in [65], where a unit memory success-based mutation scheme and a modified BLX-alpha-beta crossover scheme were also integrated in DE.

Very recently, Ref. [66] investigated the parameter settings of multiple adaptive DE variants. The authors presented an extension for the original taxonomy of metaheuristics' parameter settings.

5.2. *Population size modification*

The utilization of populations is one of the most important features of DE. There have been many studies analyzing the impact of different population sizes on the performance of evolutionary algorithms in general [67]. A large population increases the population diversity, while small populations speed up the convergence.

To cover a D dimensional search space, it is necessary to have $D + 1$ points to define difference vectors that can cover the entire search space. The use of $Np < D$ means that the primary search mechanisms of DE will be trapped within a lower dimensional hyperplane that is a subspace of the overall search space. The experiments in [68] show that this entrapment can affect the convergence properties on sphere – a function where the unshifted global optimum at the origin is within all possible subspaces.

The original DE uses fixed population size, that is set at the beginning. dynNP-DE [16] is an algorithm that extends the adaptive jDE by reducing the population size gradually with generations. Only the initial population size needs to be set. In jDEdynNP-F [69] a new scheme was also added for changing the sign of F. F changes its sign with some probability based on the fitness values of randomly chosen vectors during the mutation operation.

The jDElsgo algorithm [70] introduces two new features to jDEdynNP-F. The first feature acts on two individuals in each generation and perturbs them randomly, since a small population size is used during the evolutionary process. The algorithm tries to improve the current best solution with randomly initialized vectors by the ”DE/rand/1” strategy and a small-scale factor lying in the range $[0.01, 0.09]$. The second feature is the use of exponential crossover alongside the popular binomial with equal probability.

In [17] population size reduction was used in addition to the concept of super-fit individual that is used at the beginning of the optimization process.

In [71, 72] a micro-DE was defined that uses a population of only 5 individuals. A local search procedure was incorporated to improve the accuracy also for high-dimensional problems.

It is known that DE typically converges more quickly with smaller populations than with larger populations, but at the cost of reduced reliability. In embedded systems, where the memory requirement is reduced, only small populations make the use of DE feasible. μJADE [73] is an algorithm that retains the desirable property of small populations, that is, fast convergence, and, at the same time, improves the robustness, which is typically associated with larger populations.

Small populations allow performing more generations, which increases the chances to converge even with a strongly reduced budget of evaluations. Minimum population search, proposed in [28], is a metaheuristic that guarantees full cover of the entire large-scale search space with a small population.

Piotrowski [74] verified the impact of the population size on the performance of DE algorithms. Setting a proper population size for higher-dimensional real-world problems turned out to be a problem and algorithm-dependent. For this reason, the use of adaptive population size is highly recommended. On the contrary, the approach used in [75] did not change population size, but, alternatively, diversified the population if stagnation of convergence was detected by measuring the distribution of the population in each dimension. The population was diversified at the dimensional level and the proposed mechanism was named auto-enhanced population diversity.

5.3. *Strategies and their adaptations*

The original DE uses one mutation strategy from a set of basic mutation strategies (given in Section 4). There has been a growing trend of selecting off-spring generation strategies (mutation and/or crossover strategy) from a pool, instead of using just one throughout the whole optimization process. In [76] two mutation variants were employed, ”DE/rand/1/bin” and ”DE/best/1/bin”, and used a second mutation based on improved non-uniform mutation and opposition-based learning to exploit and explore the search space effectively, respectively, after performing each generation. The same two mutation strategies were also used successfully in [64], and the selection between both of them was done based on the last successful update at the same population index under consideration. In [77], multiple mutation strategies and control parameters are used based on the ranking of target vectors.

In the algorithm jDElscop [18], jDEdynNP-F was improved by using three strategies: ”DE/rand/1/bin”, ”DE/rand/1/exp” and ”DE/best/1/bin”. The first strategy was used on the first half of Np individuals and the second on the remaining individuals. The third strategy was used with a probability of 0.1, after the first half of the maximal number of fitness evaluations. The main idea of using low probability for this strategy was to keep the other two strategies as the leading strategies during the evolutionary process. In jDEsps [19], the set of strategies was extended further by two strategies: Local search strategy applied on the best individual, and the strategy that tries to move forward the best individual using large step movement. In [78], a new mutation strategy was proposed, which generates a mutant vector based on a local best vector and the global best vector.

Islam et al. [79] proposed a new alternative to the ”DE/current-to-best/1” mutation scheme. The current target vector was perturbed with the best member of a group of randomly selected population members. The group was reformed for each vector in a generation. This weakened the attraction toward any fixed point of the fitness landscape, and helped in alleviating the tendency of premature convergence and/or stagnation. The parent selection scheme was also modified by letting each mutant undergo the usual binomial crossover with one of the p top-ranked individuals from the current population, and not with the target vector with the same index as used in general DE. The idea was to update the values of F and Cr guided by the knowledge of their successful values in the previous generation.

The JADE algorithm [53] introduced a mutation strategy, named ”DE/current-to-p-best/1”, where any of the top $100p\%, p \in (0, 1]$ solutions can be chosen randomly to play the role of the p-best solution. The proposed mutation strategy exhibited fast convergence and high diversity for LSGO.

In [80], to solve LSGO, SaDE was enhanced by incorporating the JADE mutation strategy and hybridized with a modified multi-trajectory search. In a crossover operation three options are used: Binomial (uniform) crossover, the exponential crossover, as well as no crossover option. In [81], a modification of the JADE algorithm was proposed. Instead of a fixed p value in the original JADE, an adaptive p setting was used. The idea was to set the p value larger at the beginning of the evolutionary process to enhance the global search ability, and then decrease the value adaptively to strengthen the local search ability. The proposed JADE modification also enhanced the algorithm with a sorting Cr mechanism. Both Cr set and population were sorted according to their values. The smaller Cr value was assigned to an individual with better fitness value; the larger Cr value was then assigned to an individual with poorer fitness value.

In [82] four roles were defined, played by the participating vectors in a regular DE mutation and crossover operation: Receiving, placing, leading, and correcting vectors. The authors proposed the proper selection of vectors for each role in crossover and mutation operations. They used two mechanisms: The role differentiation mechanism, that defines the attributes for which vectors are selected for each role; a malleable mating mechanism, that allows placing vectors to adapt their mating trends to ensure some similarity relations with the leading and correcting vectors.

Binomial and exponential crossover operators have been proposed simultaneously, but binomial crossover has been used more frequently. The original exponential crossover lacked scalability with respect to the dimensionality of the problem. In [83], the commonly used definition of exponential crossover operator was analyzed and an improved version was proposed, called linearly scalable exponential crossover operator. It was based on a number of consecutive dimensions to crossover. When solving complex high dimensional problems, the proposed exponential crossover operator demonstrated superior performance over the binomial operator. Exponential crossover was also used successfully in micro-DE [20].

Guo and Yang [84] proposed an alternative coordinate system employed by individuals during crossover. The standard coordinate system was rotated by utilizing the eigenvector information of the covariance matrix of population. To avoid losing diversity of population, the offspring can be born stochastically from the parents with either the standard coordinate system or the rotated coordinate system.

In [85], the vector generation strategy was improved to increase the number of potential trial vectors within DE. The number of resources that could be saved with the new scheme was significant. This is important, especially when dealing with large-scale problems.

Yang et al. [86] proposed a new mutation strategy where, instead of one parent, multiple top ranked parents were utilized to direct each individual to search the space. Since the failed parents, or trial vectors, may also contain useful information, they maintained an archive to preserve these failed individuals, and utilize a niching method to update the archive during evolution.

Mohamed and Almazyad [87] proposed a new triangular mutation rule based on the convex combination vector of the triplet, defined by the three randomly chosen vectors and the difference vectors between the best, better, and the worst individuals among the three randomly selected vectors. The new mutation rule was combined with the basic mutation strategy ”DE/rand/1/bin”, where the new triangular mutation rule was applied with the probability of $\frac{2}{3}$.

5.4. *Global and local neighborhood search*

In general, DE exhibits good global search performance. The integration of local search operators in DE is a good alternative to improve results in different types of optimization problems [88]. To make the process efficient, it is important to define the interaction between global and local searches.

In [89], three local search methods were used to identify neighborhood structures. Local search based on Cauchy mutation was beneficial in [90]. It was combined with a self-adjusted parameter control mechanism. In [91] an approximate gradient-based local search algorithm was embedded into DE. A diversity-maintained mutation was also used to avoid the stagnation. Huang et al. [92] used the augmented simplex method to speed up the convergence in the local space search, while the opposition-based learning was used to accelerate the convergence in the global space search. Opposition-based learning considers the current estimate and its opposite estimate at the same time. Wang et al. [93] have shown that an opposite candidate solution has a higher chance to be closer to the global optimum solution than a random candidate solution. The algorithm proposed in [94] focused on searching the neighbors of all individuals. First, for each individual, they created two trial individuals by local and global neighborhood search strategies. Second, they selected the fittest one among the current individual and the two created trial individuals as a new current individual. Additionally, sequential DE (DE with one-array) was used as a parent algorithm to accelerate the convergence speed in large-scale search spaces.

In [95] two novel operators, based on the Euclidean neighborhood, were introduced in mutation. The first was the neighbor guided selection scheme for parents involved in mutation, and the second was the direction induced mutation strategy. The direction information prevents an individual from entering an undesired region and moves it to a promising area.

Molina and Herrera [96] combined DE and local search in an iterative manner. Both algorithms share the current best solution for different reasons: The local search method to exploit it, and the DE to use it to guide the search. Two different local search methods were used. In each iteration it was chosen which local search was going to be applied, considering the improvement obtained by the previous one. The local search was not always applied to the best solution. When a solution was detected as a local optima, the local search was applied to a different solution of the population.

Ref. [97] used adaptive social learning strategy to extract the neighborhood relationship information to guide the selection of parents in mutation intelligently. Mutation, enhanced by neighborhood guided search, was also used successfully in [98].

5.5. Population initialization

The initial population plays an important role in optimization [99]. It affects the search for several iterations, and also influences the final solution. Different population initialization techniques were defined.

Refs. [100, 101] and [102] used opposition based learning for generating the initial population, and also during the search process, where it was called the generation jumping stage. An enhanced version of opposition-based learning, called generalized opposition-based learning, was proposed in [93], and used in a parallel DE algorithm [103]. In [104] an extended, shuffled DE was proposed based on the opposition-based population initialization and generation jumping methods. In [105] it was enhanced further by using a bidirectional random optimization concept in its mutation stage to increase the number and diversity of search moves. In [106] an algorithm was proposed which utilized the exact algorithm of opposition-based learning, except for replacing opposite points with center-based individuals. The experiments have shown that the probability of closeness to an unknown solution for the center point is higher than other points and, by increasing the dimension of the problems, this probability value grows drastically. In [107] a CC framework was enhanced with the population initialization strategies, based on the center point, for better handling of non-separable LSGO problems.

In [108] two schemes for generating the initial population were used: The first was based on quadratic interpolation, and the second on the nonlinear simplex method in conjugation with computer generated random numbers.

Ref. [109] used micro-DE with centroid-based population initialization.

Although population initialization plays an important role in finding a better candidate solution, experimental study in [110] indicated that proper values for the control parameters have a greater effect on the optimizer performance than population initialization, when ”DE/rand/1/bin” is used for LSGO. Afterwards, in [111], the experimental evaluation was carried out of a wide range of initialization strategies. In the research, statistically significant differences appeared among the considered strategies, if the best and worst results were taken into account. Based on their observations, if we do not have enough a priori information about the problem being solved, opposition-based learning seems to be a promising scheme.

In evolutionary algorithms there is a risk of premature convergence. Population restart mechanisms were proposed to avoid it. In [112], a comparative study was conducted of 36 population restart strategies for LSGO.

The authors have shown that not all the options selected to construct the population restart strategy are equally important. It seems that the way population size is managed is more important than other options, such as when to restart the population or how much information to reset. They have also shown that the effects of the population restart dilute as the problem size increases.

5.6. External archives

Some algorithms store previous solutions in archives. The purpose of an archive is to diversify the algorithm search and maintain population diversity.

Failed solutions are stored in the following algorithms: JADE [53], SHADE [113], LMDEa [62] and MPGDE [86].

Khanum et al. [114] proposed to use a second archive to store superior solutions to avoid premature convergence towards local optima. At the end of the search process, the best solution is selected from the second archive and the current population.

Ref. [115] stored good search directions in an archive. If a trial vector is better than its associated target vector, their difference is stored in an archive.

5.7. Subpopulations

Another possibility of enhancing the DE is to divide the population into subpopulations, which evolve independently, and exchange pieces of information for performing the global search. The use of multiple populations allows an observation of the decision space from various perspectives, and decreases the risk of stagnation, since each subpopulation imposes a high exploitation pressure. Using subpopulations is suitable for parallelization [116].

A DE with subpopulations is called a distributed DE. In a distributed DE, two important factors need to be considered: Migration strategy and population topology. Migration strategy determines the interaction between subpopulations, which copy, transfer, or exchange individuals. Population topology defines the structure of the population. Ring topology is a common choice. Weber et al. [117] used two families of subpopulations. The subpopulations in the first family were arranged according to a ring topology and had fixed population size. They employed a migration mechanism acting on the individuals with the best performance. The second family was composed of subpopulations with a dynamic population size: The size was reduced progressively. The solutions generated by the second family then migrated to the first family. Using small populations might result in a premature convergence. To overcome this inconvenience, Ref. [118] proposed an algorithm, named shuffle or update parallel DE, which integrated two randomized components. The initial population was distributed with equal probability among some predefined number of subpopulations. At the start, the subpopulation specific scale factor was initialized with values randomly taken from the range $[0.1, 1]$, and mutation took place following the ”DE/rand/1/exp” strategy, where random vectors could be selected from all over the population. The subpopulations quickly exploited some areas of the decision space, thus reducing the fitness value in the highly multi-variate fitness landscape drastically and quickly. New search logics were introduced into the subpopulation functioning in order to avoid a diversity loss and, thus, premature convergence. Two simple mechanisms were integrated in order to pursue this aim. The first, namely, shuffling, consisted of rearranging the individuals randomly over the subpopulations. The second consisted of updating all the scale factors of the subpopulations.

Weber et al. [119] studied scale factors within distributed DE structures. The employment of multiple scale factors was almost always beneficial, although proper choice of a scale factor scheme appeared to be dependent on the distributed structure.

Zhang et al. [120] proposed a memetic distributed DE, where hybridization of DE and local search was used. The population was divided into subpopulations according to von Neuman topology. In [121] it was proposed to mix classical DE and dynamic DE in a heterogeneous framework. In dynamic DE, better individuals update the current generation, and are not reserved for the next generation as in the classical DE. Ref. [121] used four subpopulations, each with its own mutation strategy. Subpopulations evolve independently, but also exchange information amongst others to enhance the efficacy cooperatively.

In [122], the population was structured in two subpopulations running in parallel, with a certain migration strategy. The individuals to be migrated were chosen uniformly randomly by the selection function. The individuals that arrived substituted those local individuals previously determined by the replacement function. Ring topology was used in their version of distributed DE.

Cheng et al. [123] proposed a multicultural migration strategy with two migrant selection approaches, the representative individual and target individual based ones, and an affinity based replacement selection strategy.

Penas et al. [124] proposed asynchronous parallel implementation of a distributed DE.

The algorithm proposed in [125], divided the population into independent subgroups, each with different mutation and update strategies. Adaptive parameter settings were also used. Some individuals were exchanged between the subgroups to facilitate information exchange at a slow rate.

Multiple populations were also used in [126], where the population was divided into different tribes, each of which followed an adjusted mutation strategy. Each of these mutation strategies had different parameter settings to enhance diversity among the population. Each tribe had its own life cycle that controlled its participation ratio for the next generation based on its recorded success. The size of each tribe was also changed in an adaptive success-based manner. The performance of multi-population algorithms is sensitive to the exchanged information involved in the migration process. In [127], information of diversity between subpopulations was utilized to determine the exchanged information. Both diverse and similar exchanged information was involved. Diverse exchanged information helped a lot in maintaining population diversity, and similar exchanged information could accelerate convergence speed.

5.8. Problem decomposition

Problem decomposition is the key issue in CC algorithms. An appropriate decomposition should group interacted variables together so that the interdependencies among different subproblems are minimized. Another key issue in CC is the optimization of the subproblems. Each subproblem is optimized by means of a separate optimizer. The third key issue is cooperative coordination, i.e. how to combine the sub-solutions to obtain an entire solution.

Yang et al. [33] used random grouping strategy to group variables into groups of pre-defined group size. As it is difficult to define the optimal group size, in [128] they improved the approach to use several problem decomposers based on different group sizes. In [129] a variable grouping strategy was designed in which the variables with the interaction between each other were classified into one group, while the variables without interaction were classified into different groups. Then, evolution was conducted in these groups separately. In [130] the fast search operator was proposed to capture the interdependencies among variables. Problem decomposition was then performed based on the obtained interdependencies. A cross-cluster mutation strategy was incorporated into the original DE. The population was divided into several clusters, and the exploitation-biased operators were applied for the individuals within each cluster.The exploration-biased operators were applied for the individuals among different clusters. The exploration-biased operators favored search in the regions among different clusters, and the exploitation-biased operators favored search in the vicinity of each cluster. Ge et al. [131] proposed a circular sliding window strategy to solve the task of the decomposition of the original problem, where the ”window” size represented the size of the group, which could be adjusted according to the separability of the problem. The circular sliding window strategy was used in a self-adapted hybrid Differential Evolution algorithm.

Omidvar et al. [34] also proposed differential grouping for a decomposition of a problem based on the interdependencies of the decision variables. The algorithm was sensitive to the threshold parameter ϵ, which determined its sensitivity to weak interactions between variables. In [132] differential grouping was improved further and named DG2. It does not require the user to specify any external parameter. ϵ value is calculated dynamically to detect the interaction between each pair of variables. DG2 was integrated into Contribution-Based Cooperative Coevolution, defined in [133]. The new algorithm was named CBCC3-DG2. In [134] differential grouping was used to extract the linkage information adaptively between pairs of variables. The linkage information was stored in a linkage matrix, where each element stands for ”linkage strength”, which measured the likelihood of a pair of variables being tightly linked. Groups of tightly interactive variables were constructed. Two group-wise crossover operators were designed to use the identified groups explicitly for guiding the crossover process. The first was a group-wise binomial crossover, and the second a group-wise orthogonal crossover. Both operators were added to the basic binomial crossover. The linkage utilization technique was called hybrid linkage crossover and was incorporated into different DE implementations.

The concept of local fitness function was introduced in [135]. The local selection operator splits a high scale problem into some subcomponents and associated a local fitness function to evaluate each subcomponent. The local and global fitness values were used simultaneously to evolve the population. Subpopulations representing subcomponents are likely to have different contributions to the improvement of the best overall solution of the problem. It would make sense to allocate more computational resources to the subpopulation with greater contributions. Ref. [45] proposed such a framework that tracked the contributions of subpopulations dynamically and allocated computational resources accordingly.

Problem decomposition was also beneficial in [136]. Different subspaces were used in consequent iterations to maintain the search diversity.

Sabar et al. [137] proposed a heterogeneous framework to solve big data optimization problems. The CC was used to split the big problem into subproblems. The subproblems were then solved using various heterogeneous memetic algorithms. For each solution, different DE operators, parameter values and a local search algorithm were assigned adaptively to explore and exploit the search space of the given problem instance efficiently.

Refs. [138, 139] extended the CC by a constructive heuristic to create a good initial feasible solution for the complete problem. This initial solution was then used for the cooperation between the subpopulations at the start of the optimization. When the search stagnated, it restarted to create a new different initial feasible solution. In their research, a variation of jDE was used as the embedded algorithm for the subproblem optimization.

CCFR-IDG2, CCFR-I [45] and SACC [140] are other very recently proposed algorithms that are based on CC. All of them use SaNSDE as the optimizer for subcomponents. CCFR-I used an ideal decomposition, done manually using the prior knowledge of the functions. CCFR-IDG2 allocates computational resources for components according to their dynamic contributions to the improvement of the best overall objective value. In SACC, a sensitivity analysis is performed to allocate computational resources for components. First, the main effect of each variable is computed using the Morris screening method, and then the main effect of each subcomponent is calculated based on the main effect of its variables. In [141] the k-means clustering method was used additionally to construct groups including variables with similar effects on the fitness value.

Peng and Wu [142] proposed a CC algorithm with imbalance multi-modal optimizer. Since CC decomposes a large-scale problem into many subproblems, an imbalanced multi-modal optimizer can help reveal relations between subproblems by locating multiple optima simultaneously.

5.9. *DE hybridization*

In addition to DE, many other algorithms were proposed to solve LSGO, but no single algorithm is better than another on all types of problems. As the algorithms show complementary search characteristics, they can be combined. In Ref. [143] DE, ABC and PSO were combined by means of a Message Passing Interface environment. However, each constituent algorithm, if used as single optimization algorithm, delivered the best performance on some problems, but was not successful on others, while the parallel implementation delivered a robust performance.

2E-Ensemble [144] divides the search procedure into two stages, where, in the second stage, three different sub-optimizers are used: SaDE, Genetic Algorithm (GA), and cooperative GA and DE: SaDE has shown excellent performance on uni-modal problems and many multimodal problems; GA is effective in jumping out of the local optima due to the mutation operator; the cooperative GA and DE has good balance between exploration and exploitation.

Ref. [145] proposed a hybrid of DE and the Variable Mesh Optimization (VMO), defined in [146]. The DE is used to evolve and improve the quality of the initial mesh generated by the VMO operations, before utilizing it in the next iteration of the VMO.

In [147] mutation and crossover operators of DE were integrated into a parameter free version of PSO.

In CADE [148] a hybridization was proposed of Cultural Algorithm (CA) and DE. CA is an evolutionary model that consists of belief and population spaces and a communication channel. CA and DE were executed in parallel. The CADE algorithm managed an overall population, which was shared between CA and DE simultaneously.

Multiple Offspring Sampling (MOS) [149] is a framework that allows the seamless combination of several algorithms (both population-based and local searches) in a dynamic way.

6. Analysis of selected algorithms on the CEC 2013 LSGO benchmark suite

For empirical evaluation of the methods proposed to solve LSGO, the CEC 2013 large-scale benchmark suite was chosen. In this section, we present some analyses of selected algorithms from different perspectives.

6.1. *Results of the original DE algorithm*

Since DE is our main interest, first, the experiments were performed using the original DE. The values of control parameters were set as follows: $F = 0.5$, $Cr = 0.9$, and $Np = 100$.

In Table 2 the experimental results are presented of the original DE algorithm, for binomial and exponential crossover, respectively. The total number of objective function evaluations was set to 3×10^6 as also suggested in the CEC 2013 LSGO benchmark suite. According to the same instructions, 25 runs per function were performed.

Table 2: Experimental results with dimension $D = 1000$ for the original DE algorithm with the binomial (DE/bin) and exponential (DE/exp) crossover, and 100 points that were generated randomly (denoted as $Init.$).

		F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8
	Best	3.28e+11	1.10e+05	2.16e+01	1.46e+13	5.26e+07	1.07e+06	1.17e+15	7.79e+17
$Init.$	Median	3.60e+11	1.18e+05	2.17e+01	3.30e+13	7.11e+07	1.07e+06	6.17e+15	1.74e+18
100	Worst	3.90e+11	1.22e+05	2.17e+01	5.49e+13	8.42e+07	1.07e+06	2.11e+16	3.10e+18
	Mean	3.62e+11	1.18e+05	2.17e+01	3.22e+13	7.01e+07	1.07e+06	7.70e+15	1.85e+18
	Std	1.51e+10	3.53e+03	8.07e-03	9.38e+12	6.61e+06	1.56e+03	5.48e+15	5.91e+17
	Best	2.43e+04	9.13e+03	2.13e+01	3.73e+09	7.96e+05	1.06e+06	2.28e+07	1.11e+13
DE/bin	Median	2.35e+05	1.04e+04	2.14e+01	1.27e+10	7.79e+06	1.06e+06	7.18e+07	4.03e+13
$3.0e+6$	Worst	1.03e+07	1.18e+04	2.15e+01	3.04e+10	8.35e+06	1.06e+06	1.42e+08	1.21e+14
	Mean	9.94e+05	1.04e+04	2.14e+01	**1.38e+10**	**6.47e+06**	1.06e+06	**7.68e+07**	**4.73e+13**
	Std	2.31e+06	7.18e+02	4.32e-02	6.37e+09	2.81e+06	1.15e+03	3.49e+07	2.49e+13
	Best	1.12e-02	1.48e+03	2.03e+01	2.41e+11	7.68e+06	1.05e+06	1.48e+09	7.52e+15
DE/exp	Median	1.39e-02	1.55e+03	2.03e+01	3.36e+11	1.00e+07	1.05e+06	1.96e+09	1.52e+16
$3.0e+6$	Worst	1.85e-02	1.59e+03	2.03e+01	5.17e+11	1.12e+07	1.06e+06	2.52e+09	1.99e+16
	Mean	**1.41e-02**	**1.55e+03**	**2.03e+01**	3.53e+11	9.83e+06	**1.05e+06**	1.98e+09	1.42e+16
	Std	1.68e-03	2.93e+01	3.13e-03	7.50e+10	8.30e+05	1.54e+03	2.73e+08	3.49e+15

		F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}
	Best	1.76e+16	9.56e+07	6.42e+15	7.68e+12	1.74e+17	1.04e+17	2.24e+17
$Init.$	Median	4.17e+16	9.66e+07	1.24e+16	8.33e+12	5.69e+17	8.10e+17	6.80e+17
100	Worst	1.60e+17	9.71e+07	2.37e+16	8.62e+12	1.78e+18	2.70e+18	1.35e+18
	Mean	5.14e+16	9.65e+07	1.25e+16	8.27e+12	6.69e+17	1.03e+18	7.28e+17
	Std	3.22e+16	4.28e+05	4.16e+15	2.33e+11	4.20e+17	7.49e+17	3.01e+17
	Best	5.41e+08	9.32e+07	9.16e+11	1.18e+06	2.84e+09	1.72e+10	2.37e+07
DE/bin	Median	1.21e+09	9.40e+07	9.21e+11	4.41e+06	4.16e+09	5.24e+10	4.08e+07
$3.0e+6$	Worst	2.25e+09	9.43e+07	9.44e+11	5.21e+07	8.22e+09	1.71e+11	5.81e+07
	Mean	**1.23e+09**	9.40e+07	**9.23e+11**	9.12e+06	**4.53e+09**	**5.74e+10**	4.08e+07
	Std	4.91e+08	2.57e+05	6.93e+09	1.21e+07	1.48e+09	3.45e+10	7.75e+06
	Best	1.39e+09	9.26e+07	9.22e+11	1.03e+03	2.18e+10	2.31e+11	8.29e+06
DE/exp	Median	1.68e+09	9.30e+07	9.28e+11	1.03e+03	2.67e+10	3.49e+11	9.96e+06
$3.0e+6$	Worst	1.89e+09	9.34e+07	9.37e+11	1.04e+03	3.06e+10	4.41e+11	1.08e+07
	Mean	1.67e+09	**9.30e+07**	9.29e+11	**1.03e+03**	2.66e+10	3.45e+11	**9.90e+06**
	Std	1.41e+08	2.12e+05	3.65e+09	6.58e-01	2.50e+09	6.31e+10	6.28e+05

In the same table, we also show results of 100 randomly generated solutions in the search space and their evaluations, denoted as $Init$. These solutions were also used as the initial population of original DE. If we compare mean values of the binomial and exponential crossover variants, we can see that the first won in 8 cases while the latter is better in 7 cases. In the continuation of the section, the results in the table will be compared to the results of other, more advanced algorithms.

6.2. *IEEE CEC competitions on LSGO*

The IEEE Congress on Evolutionary Computation (IEEE CEC) is one of the leading events in the field of Evolutionary Computation. It is organized each year. Different competitions are organized in the scope of the congress. Usually, the competition on LSGO takes place every two years. Since 2013, it has been performed on the CEC 2013 LSGO benchmark suite. We were interested in the success of DE-based algorithms. The results are collected in Table 3. We can see that, among the four best algorithms in the years 2013 and 2015, respectively, two of them include the variant of DE. DECC-G is a self-adapted DE with neighborhood search, put into the CC framework. VMODE is the hybrid of Variable Mesh Optimization and DE. IHDELS is an iterative hybrid of DE and two local searches. In 2017, the competition was also planned, but, unfortunately, not carried out. It seems that LSGO is still a quite hard nut to crack.

Table 3: The four best-performing algorithms of the CEC Special Sessions on LSGO.

	CEC 2013		
Place	Algorithm	Reference	uses DE
1.	MOS-CEC2013	[149]	✗
2.	DECC-G	[33]	✓
3.	CC-CMA-ES	[150]	✗
4.	VMO-DE	[145]	✓

	CEC 2015		
Place	Algorithm	Reference	uses DE
1.	MOS-CEC2013	[149]	✗
2.	IHDELS	[96]	✓
3.	CC-CMA-ES	[150]	✗
4.	DECC-G	[33]	✓

6.3. *Algorithms selected for evaluation, and evaluation methods*

Comparative analysis was performed of different algorithms. The set of best performing algorithms from CEC 2013 and 2015 was expanded with 11 other algorithms. Some of them contain DE and some of them do not. The algorithms, considered for comparison are shown in Table 4. The algorithms that were chosen are

Table 4: Algorithms considered for comparison.

No.	Algorithm	Reference	uses DE
1	MA-SW-Chains	[27]	✗
2	2E-Ensemble	[144]	✓
3	MOS-SOCO2011	[151]	✓
4	jDElscop	[18]	✓
5	GADE	[63]	✓
6	MOS-CEC2012	[152]	✗
7	jDEsps	[19]	✓
8	MOS-CEC2013	[149]	✗
9	DECC-G	[33]	✓
10	CC-CMA-ES	[150]	✗
11	LSGOjDE	[57]	✓
12	VMO-DE	[145]	✓
13	IHDELS	[96]	✓
14	CBCC3-DG2	[132]	✓
15	CCFR-IDG2	[153]	✓
16	CCFR-I	[153]	✓

state-of-the-art algorithms for solving LSGO problems. The following algorithms are based on CC: DECC-G, CC-CMA-ES, CBCC3-DG2, CCFR-IDG2 and CCFR-I.

In our study, we used three evaluation methods: SR, $score$ and $Formula\,One\,score$.

The Sum of Ranks (SR) of a given algorithm is defined as follows:

$$SR = \sum_{i=1}^{N} rank_i. \tag{5}$$

N denotes the number of functions, and $rank_i$ is the rank of the algorithm for the ith function.

In the year 2017, a new evaluation method was defined to score the algorithms for the CEC competition [154]. Since, in the CEC 2017 competition, benchmark problems were tested on four dimensions ($D = 10$, 30, 50, and 100), the higher weights were given to the results obtained on higher dimensions. While CEC 2013 test functions had only one dimension, i.e. $D = 1000$, the evaluation method was rewritten as follows.

The evaluation method combines two scores, defined in Eq. (7) and (8), to find the final score as follows:

$$score = score_1 + score_2, \tag{6}$$

where

$$score_1 = (1 - \frac{SE - SE_{min}}{SE}) \times 50. \tag{7}$$

Here, SE is the sum of error values for dimension $D = 1000$ for a given algorithm, and SE_{min} is the minimal sum of error values from all the algorithms.

SE is defined as follows:

$$SE = \sum_{i=1}^{N} ef_i,$$

where ef_i is the error value of the algorithm for the ith function.

The second score is:

$$score_2 = (1 - \frac{SR - SR_{min}}{SR}) \times 50, \quad (8)$$

where SR is the Sum of Ranks (defined in Eq. 5), and SR_{min} is the minimal Sum of Ranks from all the algorithms.

$Formula\,One\,score$ is a metric that scores the ten best performing algorithms according to their places. They get the following scores 25, 18, 15, 12, 10, 8, 6, 4, 2, 1, respectively, i.e. 1^{st} gets 25 scores, 2^{nd} gets 18, etc.

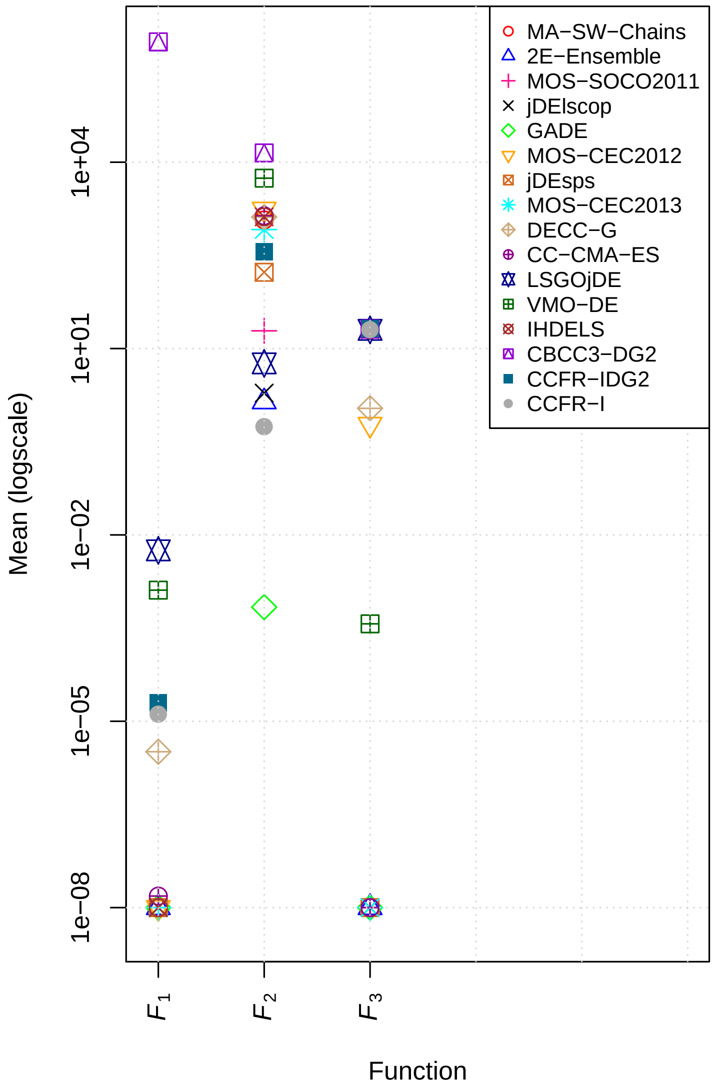
6.4. Analysis of selected algorithms

We collected the results of all the selected algorithms. The total number of objective function evaluations was set to 3×10^6. 25 runs per function were performed. Mean values are given in Tables 5 and 6. Values less than 1×10^{-8} are treated as 0. Bold font represents the best result for each function. The efficiency of each algorithm depends strongly on the landscape to optimize. For this reason, almost every function has its own winner. MOS [149] is an exception, as it is composed of very different algorithms, except that, for each function, the algorithm with the best behavior could lead the search.

In Table 7, for each function, ranks of algorithms are calculated according to the mean values. Finally, values of all three metrics were assigned to each algorithm. The results are collected in Table 8. We can see that SR and $Formula\,One\,score$ almost agree in their ordering of algorithms. DE-based algorithms are placed high: 2^{nd}, 3^{rd}, 4^{th} places, etc. The ordering of metric $score$ is quite different. Its winner is the DE-based algorithm CCFR-I, that is in the middle by both other metrics. Metric $score$ uses error values in the formula and rewards/penalizes the outstanding results of algorithms accordingly. The second highly placed DE-based algorithm based on this metric is in 4^{th} place.

To see how good the algorithms are in general in solving different functions, we plotted the results in Figs. 1 and 2. Fig. 1 shows comparison of the algorithms on the CEC 2013 separable functions (F_1–F_3). It can be noticed that most of the algorithms can solve functions F_1 and F_3, while function F_2 seems to be more difficult.

Fig. 2 presents comparison of the algorithms on the CEC 2013 non-separable functions (F_4–F_{15}). The figure shows that most of the algorithms fail to find a solution close to the global optimum value, which is zero for all the benchmarks. The values given in the tables and depicted in figures, can also be interpreted as error values: $f(\vec{x}) - f(\vec{x}^*)$. It is clear that there is still significant room for further progress.



1

Figure 1: Comparison of the mean values of the algorithms on the CEC 2013 LSGO functions F_1–F_3. Note that small value $1e-8$ is added to each mean in order to plot the graph with a log-scaled y-coordinate.

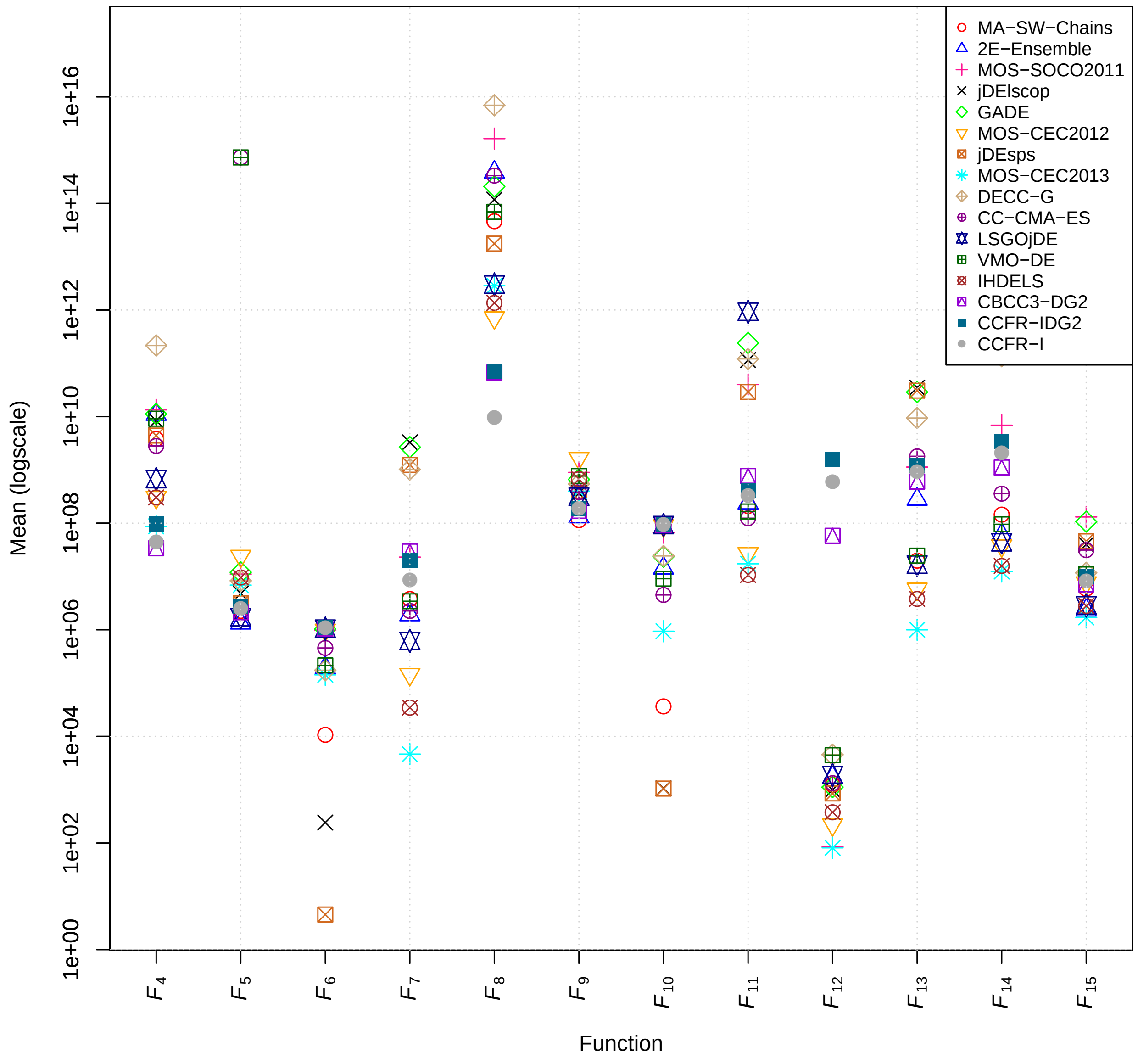


Figure 2: Comparison of mean values of algorithms on the CEC 2013 LSGO functions F_4–F_{15}.

Table 5: Mean value and Standard Deviation (in brackets) for the algorithms on the CEC 2013 LSGO functions.

	Algorithm	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8
1	MA-SW-Chains	**1.14e-12**	1.18e+03	**6.78e-13**	3.80e+09	2.26e+06	1.07e+04	3.78e+06	4.63e+13
		(1.28e-12)	(1.84e+02)	(2.28e-13)	(2.70e-13)	(1.36e+06)	(2.09e+04)	(8.46e+05)	(9.18e+12)
2	2E-Ensemble	**2.25e-24**	1.39e+00	**4.20e-13**	1.10e+10	**1.33e+06**	1.94e+05	1.90e+06	3.85e+14
		(9.16e-24)	(1.62e+00)	(7.82e-14)	(3.44e+09)	(2.50e+05)	(1.64e+04)	(1.14e+06)	(1.39e+14)
3	MOS-SOCO2011	**0.00e+00**	1.93e+01	**0.00e+00**	1.34e+10	1.11e+07	9.85e+05	2.31e+07	1.64e+15
		(0.00e+00)	(4.16e+00)	(0.00e+00)	(7.69e+09)	(1.79e+06)	(3.22e+03)	(4.42e+07)	(1.66e+15)
4	jDElscop	**0.00e+00**	1.92e+00	**2.78e-13**	8.37e+09	4.78e+06	2.42e+02	3.29e+09	1.18e+14
		(0.00e+00)	(2.67e00)	(8.41e-15)	(4.15e+09)	(7.60e+05)	(1.13e+03)	(1.24e+09)	(5.28e+13)
5	GADE	**0.00e+00**	**6.88e-04**	**2.61e-13**	1.12e+10	1.20e+07	1.03e+06	2.67e+09	2.08e+14
		(0.00e+00)	(3.44e-03)	(2.95e-14)	(5.17e+09)	(9.06e+05)	(2.03e+03)	(3.90e+08)	(4.59e+06)
6	MOS-CEC2012	**0.00e+00**	1.75e+03	6.02e-01	3.07e+08	2.41e+07	9.89e+05	1.46e+05	7.18e+11
		(0.00e+00)	(1.06e+02)	(8.77e-01)	(1.47e+08)	(3.93e+06)	(2.67e+03)	(1.12e+05)	(5.42e+11)
7	jDEsps	**9.17e-23**	1.69e+02	**1.73e-13**	4.41e+09	3.14e+06	**4.55e+00**	1.24e+09	1.76e+13
		(1.37e-22)	(1.83e+02)	(2.55e-13)	(5.63e+09)	(6.38e+05)	(7.90e+00)	(4.06e+08)	(4.44e+13)
8	MOS-CEC2013	**0.00e+00**	8.23e+02	**1.69e-12**	8.73e+07	6.89e+06	1.43e+05	**4.65e+03**	2.85e+12
		(0.00e+00)	(4.69e+01)	(9.16e-14)	(3.11e+07)	(9.16e+05)	(6.86e+04)	(1.06e+04)	(1.44e+12)
9	DECC-G	3.22e-06	1.31e+03	1.09e+00	2.16e+11	8.30e+06	1.74e+05	1.02e+09	6.94e+15
		(2.83e-06)	(3.42e+01)	(3.54e-01)	(7.76e+10)	(1.60e+06)	(2.09e+04)	(4.89e+08)	(3.37e+15)
10	CC-CMA-ES	**5.36e-09**	1.37e+03	**1.50e-13**	2.82e+09	7.28e+14	4.56e+05	2.26e+06	3.32e+14
		(1.32e-09)	(1.38e+02)	(6.87e-15)	(1.84e+09)	(5.18e+06)	(4.56e+05)	(2.87e+06)	(1.74e+14)
11	LSGOjDE	5.64e-03	5.77e+00	2.00e+01	6.71e+08	1.72e+06	1.05e+06	6.18e+05	3.00e+12
		(1.15e-02)	(6.11e+00)	(4.79e-03)	(1.19e+08)	(2.99e+05)	(5.33e+03)	(1.48e+05)	(1.67e+12)
12	VMO-DE	1.29e-03	5.53e+03	3.70e-04	9.13e+09	7.28e+14	2.15e+05	3.43e+06	6.94e+13
		(1.27e-03)	(3.96e+02)	(1.37e-04)	(3.73e+09)	(0.00e+00)	(4.33e+04)	(2.71e+05)	(7.22e+13)
13	IHDELS	**4.34e-28**	1.32e+03	2.01e+01	3.04e+08	9.59e+06	1.03e+06	3.46e+04	1.36e+12
		(1.23e-27)	(6.98e+01)	(1.36e-01)	(1.07e+08)	(2.03e+06)	(1.95e+04)	(1.33e+04)	(6.85e+11)
14	CBCC3-DG2	8.65e+05	1.41e+04	2.06e+01	**3.39e+07**	2.14e+06	1.05e+06	2.95e+07	6.74e+10
		NA	NA	NA	(1.77e+07)	(4.24e+05)	(3.37e+03)	(2.78e+07)	(8.86e+10)
15	CCFR-IDG2	2.00e-05	3.60e+02	2.10e+01	9.60e+07	2.80e+06	1.10e+06	2.00e+07	7.00e+10
		(5e-06)	(2e+01)	(1e-02)	(4e+07)	(3e+05)	(1e+03)	(3e+07)	(1e+11)
16	CCFR-I	1.30e-05	5.50e-01	2.00e+01	4.50e+07	2.50e+06	1.10e+06	8.60e+06	**9.60e+09**
		(3.2e-06)	(1.5e+00)	(3.1e-07)	(1.7e+07)	(2.7e+05)	(1.2e+03)	(1.9e+07)	(1.6e+10)

Table 6: Mean value and Standard Deviation (in brackets) for the algorithms on the CEC 2013 LSGO functions.

	Algorithm	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}
1	MA-SW-Chains	**1.14e+08**	3.66e+04	2.10e+08	1.23e+03	1.98e+07	1.45e+08	5.90e+06
		(2.05e+07)	(6.17e+04)	(2.43e+07)	(8.32e+01)	(2.30e+06)	(1.69e+07)	(1.36e+06)
2	2E-Ensemble	1.31e+08	1.43e+07	2.38e+08	1.70e+03	2.80e+08	6.09e+07	2.33e+06
		(1.51e+07)	(2.87e+06)	(6.45e+07)	(2.20e+02)	(1.75e+08)	(1.75e+07)	(1.74e+05)
3	MOS-SOCO2011	8.97e+08	6.65e+07	4.01e+10	8.63e+01	1.13e+09	6.89e+09	1.31e+08
		(1.39e+08)	(2.91e+07)	(1.23e+11)	(7.74e+01)	(7.74e+08)	(1.41e+10)	(6.02e+07)
4	jDElscop	3.66e+08	1.06e+03	1.15e+11	9.91e+02	3.49e+10	4.59e+11	3.98e+07
		(5.57e+07)	(1.72e+02)	(5.86e+10)	(2.70e+01)	(5.58e+09)	(1.16e+11)	(1.50e+07)
5	GADE	6.62e+08	2.35e+07	2.40e+11	1.12e+03	2.88e+10	4.21e+11	1.07e+08
		(7.44e+07)	(4.59e+06)	(5.96e+10)	(6.90e+01)	(3.41e+09)	(7.13e+10)	(1.43e+07)
6	MOS-CEC2012	1.65e+09	9.00e+07	2.71e+07	2.21e+02	5.84e+06	3.78e+07	7.63e+06
		(3.47e+08)	(5.04e+05)	(5.19e+06)	(2.29e+02)	(6.31e+05)	(2.01e+07)	(1.52e+06)
7	jDEsps	2.71e+08	**1.05e+03**	2.90e+10	8.51e+02	3.07e+10	1.93e+11	4.58e+07
		(2.94e+07)	(2.43e+02)	(1.13e+10)	(3.64e+02)	(4.16e+09)	(8.24e+10)	(1.12e+07)
8	MOS-CEC2013	3.99e+08	9.38e+05	1.73e+07	**8.13e+01**	**1.00e+06**	**1.24e+07**	**1.71e+06**
		(6.26e+07)	(4.79e+05)	(5.04e+06)	(1.57e+02)	(5.53e+05)	(2.86e+06)	(1.44e+05)
9	DECC-G	5.47e+08	2.43e+07	1.21e+11	4.53e+03	9.40e+09	1.36e+11	1.17e+07
		(8.94e+07)	(9.75e+06)	(6.03e+10)	(9.48e+02)	(3.15e+09)	(4.79e+10)	(1.18e+06)
10	CC-CMA-ES	3.82e+08	4.51e+06	1.24e+08	1.33e+03	1.80e+09	3.58e+08	3.13e+07
		(1.48e+08)	(1.79e+07)	(9.88e+07)	(4.78e+02)	(4.02e+09)	(1.15e+09)	(5.30e+06)
11	LSGOjDE	3.14e+08	9.24e+07	9.27e+11	1.88e+03	1.67e+07	4.37e+07	2.87e+06
		(4.87e+07)	(4.26e+05)	(1.77e+10)	(1.26e+02)	(2.29e+06)	(4.87e+06)	(4.26e+06)
12	VMO-DE	7.68e+08	9.10e+06	1.66e+08	4.45e+03	2.46e+07	9.54e+07	1.10e+07
		(1.03e+08)	(3.05e+06)	(3.80e+07)	(4.61e+03)	(3.28e+06)	(1.13e+07)	(1.65e+06)
13	IHDELS	6.74e+08	9.16e+07	**1.07e+07**	3.77e+02	3.80e+06	1.58e+07	2.81e+06
		(1.30e+08)	(9.17e+05)	(4.12e+06)	(3.30e+02)	(9.72e+05)	(5.11e+06)	(1.01e+06)
14	CBCC3-DG2	1.70e+08	9.28e+07	7.70e+08	5.81e+07	6.03e+08	1.11e+09	7.11e+06
		(3.16e+07)	(7.16e+05)	(2.80e+08)	NA	NA	NA	NA
15	CCFR-IDG2	1.90e+08	9.50e+07	4.00e+08	1.60e+09	1.20e+09	3.40e+09	9.80e+06
		(3e+07)	(2e+05)	(3e+08)	(2e+09)	(6e+08)	(3e+09)	(4e+06)
16	CCFR-I	1.90e+08	9.50e+07	3.30e+08	6.00e+08	9.30e+08	2.10e+09	8.20e+06
		(2.8e+07)	(1.9e+05)	(3.2e+08)	(7.1e+08)	(5.3e+08)	(2.1e+09)	(3.3e+06)

Table 7: Ranks of the algorithms on the CEC 2013 LSGO functions.

	Algorithm	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}
1	MA-SW-Chains	5.5	10.0	4.5	9.0	4.0	3.0	8.0	9.0	1.0	3.0	6.0	8.0	5.0	7.0	5.0
2	2E-Ensemble	5.5	3.0	4.5	13.0	1.0	6.0	5.0	14.0	2.0	7.0	7.0	10.0	7.0	5.0	2.0
3	MOS-SOCO2011	5.5	6.0	4.5	15.0	12.0	9.0	11.0	15.0	15.0	10.0	12.0	2.0	10.0	12.0	16.0
4	jDElscop	5.5	4.0	4.5	11.0	8.0	2.0	16.0	11.0	8.0	2.0	13.0	6.0	16.0	16.0	13.0
5	GADE	5.5	1.0	4.5	14.0	13.0	11.5	15.0	12.0	12.0	8.0	15.0	7.0	14.0	15.0	15.0
6	MOS-CEC2012	5.5	14.0	10.0	6.0	14.0	10.0	3.0	4.0	16.0	11.0	3.0	3.0	3.0	3.0	7.0
7	jDEsps	5.5	7.0	4.5	10.0	7.0	1.0	14.0	8.0	6.0	1.0	11.0	5.0	15.0	14.0	14.0
8	MOS-CEC2013	5.5	9.0	4.5	3.0	9.0	4.0	1.0	6.0	10.0	4.0	2.0	1.0	1.0	1.0	1.0
9	DECC-G	11.0	11.0	11.0	16.0	10.0	5.0	13.0	16.0	11.0	9.0	14.0	13.0	13.0	13.0	11.0
10	CC-CMA-ES	5.5	13.0	4.5	8.0	15.5	8.0	6.0	13.0	9.0	5.0	4.0	9.0	12.0	8.0	12.0
11	LSGOjDE	15.0	5.0	12.5	7.0	2.0	13.5	4.0	7.0	7.0	13.0	16.0	11.0	4.0	4.0	4.0
12	VMO-DE	14.0	15.0	9.0	12.0	15.5	7.0	7.0	10.0	14.0	6.0	5.0	12.0	6.0	6.0	10.0
13	IHDELS	5.5	12.0	14.0	5.0	11.0	11.5	2.0	5.0	13.0	12.0	1.0	4.0	2.0	2.0	3.0
14	CBCC3-DG2	16.0	16.0	15.0	1.0	3.0	13.5	12.0	2.0	3.0	14.0	10.0	14.0	8.0	9.0	6.0
15	CCFR-IDG2	13.0	8.0	16.0	4.0	6.0	15.5	10.0	3.0	4.5	15.5	9.0	16.0	11.0	11.0	9.0
16	CCFR-I	12.0	2.0	12.5	2.0	5.0	15.5	9.0	1.0	4.5	15.5	8.0	15.0	9.0	10.0	8.0

Table 8: Scores of the algorithms on the CEC 2013 LSGO functions.

	Algorithm	SR	Place	$score_1$	$score_2$	$score$	Place	$Formula\,One\,score$	Place
1	MA-SW-Chains	88	(2)	0.02	35.23	35.24	(3)	136	(3.5)
2	2E-Ensemble	92	(3)	0.00	33.70	33.70	(4)	145	(2)
3	MOS-SOCO2011	155	(14)	0.00	20.00	20.00	(14)	52	(14)
4	jDElscop	136	(10)	0.01	22.79	22.80	(12)	86	(10)
5	GADE	162.5	(15)	0.00	19.08	19.08	(15)	57	(12)
6	MOS-CEC2012	112.5	(5)	0.97	27.56	28.52	(8)	113	(5)
7	jDEsps	123	(6)	0.04	25.20	25.24	(9)	107	(6)
8	MOS-CEC2013	62	(1)	0.24	50.00	50.24	(2)	217	(1)
9	DECC-G	177	(16)	0.00	17.51	17.51	(16)	13	(16)
10	CC-CMA-ES	132.5	(9)	0.00	23.40	23.40	(11)	68	(11)
11	LSGOjDE	125	(7)	0.18	24.80	24.98	(10)	94	(8)
12	VMO-DE	148.5	(12)	0.00	20.88	20.88	(13)	50	(15)
13	IHDELS	103	(4)	0.51	30.10	30.61	(6)	136	(3.5)
14	CBCC3-DG2	142.5	(11)	9.90	21.75	31.65	(5)	88	(9)
15	CCFR-IDG2	151.5	(13)	9.03	20.46	29.49	(7)	56	(13)
16	CCFR-I	129	(8)	50.00	24.03	74.03	(1)	96	(7)

Table 9: Best values of the algorithms on the CEC 2013 LSGO functions.

	Algorithm	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8
1	MA-SW-Chains	**3.03e-14**	9.52e+02	**2.63e-13**	7.20e+08	1.07e+06	9.54e-01	3.36e+05	3.14e+13
2	2E-Ensemble	**0.00e+00**	**1.04e-26**	**2.81e-13**	4.18e+09	**5.89e+05**	1.59e+05	6.88e+05	1.21e+14
3	MOS-SOCO2011	**0.00e+00**	1.10e+01	**0.00e+00**	3.14e+09	8.51e+06	9.80e+05	9.05e+05	3.92e+14
4	jDElscop	**0.00e+00**	**0.00e+00**	**2.56e-13**	1.80e+09	3.24e+06	**3.00e-07**	9.32e+08	3.51e+13
5	GADE	**0.00e+00**	**0.00e+00**	**2.17e-13**	4.68e+09	8.72e+06	1.02e+06	1.97e+09	4.87e+13
6	MOS-CEC2012	**0.00e+00**	1.57e+03	**2.09e-12**	9.45e+07	1.69e+07	9.85e+05	2.50e+03	1.00e+11
7	jDEsps	**1.95e-27**	4.26e+01	**4.97e-14**	8.78e+08	2.05e+06	3.37e-01	6.98e+08	3.92e+12
8	MOS-CEC2013	**0.00e+00**	7.06e+02	**1.54e-12**	**4.77e+07**	5.43e+06	6.16e+00	**6.23e+01**	5.49e+11
9	DECC-G	8.47e-07	1.24e+03	1.05e-04	8.22e+10	5.73e+06	1.43e+05	4.55e+08	1.44e+15
10	CC-CMA-ES	**1.72e-09**	1.15e+03	**1.39e-13**	9.86e+08	7.28e+14	1.18e+03	8.14e+04	6.83e+13
11	LSGOjDE	1.00e-04	1.15e-04	2.00e+01	4.29e+08	1.19e+06	1.02e+06	3.57e+05	4.89e+11
12	VMO-DE	1.02e-04	4.79e+03	1.06e-04	3.23e+09	7.28e+14	1.71e+05	2.81e+06	8.45e+12
13	IHDELS	**0.00e+00**	1.27e+03	2.00e+01	1.03e+08	5.88e+06	1.00e+06	1.33e+04	**5.36e+10**

There is no single best algorithm which will outperform all others. The algorithm CCFR-I is an outstanding winner on function F_8, while MOS-CEC2013 is the best in most other cases.

Analyzing the mean value of the algorithm gives us the information about its robustness. When solving real-world problems, the best-known solution, along with the best-known solution value, are usually highly important. For example, in finance, we are looking for the values of decision variables which maximize the profit. Therefore, in the second analysis, we were interested in the best values obtained by the selected algorithms. The obtained best values for algorithms are presented in Tables 9 and 10. Once again, values of all three metrics were assigned to each algorithm. The results are collected in Table 11. We can see that, for all three metrics, the algorithms are placed into a somewhat different order, whereas MOS-CEC2013 remains the winner for SR and $Formula\,One\,score$ metrics. IHDELS is the best performing algorithm using the $score$ metric. It should be noted that, for CBCC3-DG2, CCFR-IDG2 and CCFR-I, the best results were not available.

Finally, we made the cumulative analysis of the mean and best values of the algorithms on the CEC 2013 benchmark functions F_4–F_{15}. The values are depicted jointly in Fig. 3. In this figure, the mean value of function values of 100 randomly initialized solutions ($Init.$) are added. The mean values of the original DE algorithm are also plotted in order to monitor the progress made by the advanced algorithms. It is interesting to see that the original DE was not always outperformed by more sophisticated algorithms, for example, in cases F_6, F_{10} and F_{11}. It is even more surprising that, for F_5, $Init.$ obtained better results than two other algorithms after 3×10^6 function evaluations.

Analyses in this section indicate that there is no single winner among the algorithms on all CEC 2013 LSGO

Table 10: Best values of the algorithms on the CEC 2013 LSGO functions.

	Algorithm	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}
1	MA-SW-Chains	**8.09e+07**	**8.63e+01**	1.59e+08	1.08e+03	1.64e+07	1.11e+08	4.57e+06
2	2E-Ensemble	1.05e+08	9.22e+06	8.26e+07	1.07e+03	6.51e+07	3.61e+07	1.87e+06
3	MOS-SOCO2011	5.02e+08	1.80e+07	3.83e+08	1.91e-02	2.97e+08	1.29e+08	3.40e+07
4	jDElscop	2.02e+08	5.61e+02	1.81e+10	9.70e+02	2.17e+10	2.53e+11	2.58e+07
5	GADE	4.84e+08	1.52e+07	1.32e+11	1.03e+03	2.12e+10	3.13e+11	7.76e+07
6	MOS-CEC2012	1.00e+09	8.81e+07	1.85e+07	1.50e-02	4.70e+06	7.96e+06	4.89e+06
7	jDEsps	2.21e+08	8.71e+02	1.03e+10	3.34e+02	2.29e+10	7.54e+10	2.94e+07
8	MOS-CEC2013	2.33e+08	4.44e+02	8.89e+06	4.40e-03	**3.08e+05**	**7.83e+06**	1.42e+06
9	DECC-G	3.95e+08	7.87e+06	7.96e+09	3.59e+03	5.15e+09	6.77e+10	9.54e+06
10	CC-CMA-ES	1.76e+08	1.32e+05	4.17e+07	9.78e+02	1.79e+07	2.27e+07	2.31e+07
11	LSGOjDE	2.05e+08	9.12e+07	9.14e+11	1.66e+03	1.15e+07	3.61e+07	1.70e+06
12	VMO-DE	6.12e+08	1.24e+06	1.09e+08	1.20e+03	1.88e+07	7.80e+07	8.10e+06
13	IHDELS	3.27e+08	9.05e+07	**5.59e+06**	**2.63e-13**	1.90e+06	9.41e+06	**1.41e+06**

benchmark functions. Even more, almost every function has its own winner. Therefore, LSGO is still quite a hard nut to crack, and we believe that there is still significant room for future improvements.

7. Conclusion

The complexity of real-world optimization problems is increasing, and there is a clear need for a robust and accurate optimizer, capable of coping with problems characterized by a high number of variables. More than two decades ago, DE emerged as a promising scheme for global optimization over continuous spaces. Over the past few years, many researchers have contributed to improve its efficiency further. The improvements have gone in the following directions: Parameter adaptation, population size modification, strategies' adaptation, local search, using subpopulation, problem decomposition, etc. In this paper, we collected the proposed modifications in one place and group the mechanisms in nine groups. The aim of our paper was also to monitor the progress in LSGO made by a variety of state-of-the-art algorithms, and to see how well the algorithms that include DE perform. We have made the analysis based on the CEC 2013 LSGO benchmark suite. The empirical evaluation was based on three evaluation methods, that brought slightly different rankings. Although it is evident that the new mechanisms have improved the results, the analysis has indicated that the original DE still exhibits remarkable performance in optimizing LSGO. On the other hand, the analysis has shown that there is still some room for future improvements. In which direction they will go is difficult to foresee. The following directions seem to be promising for DE: To enhance further the adaptive scheme of the parameters' control, to improve the switching mechanism of the DE strategies, and to learn variable interdependencies more efficiently and, consequently,

Table 11: Scores based on best values of the algorithms on the CEC 2013 LSGO functions.

	Algorithm	SR	Place	$score_1$	$score_2$	$score$	Place	$Formula\,One\,score$	Place
1	MA-SW-Chains	77.5	(2)	0.09	32.58	32.67	(4)	164	(3)
2	2E-Ensemble	91	(5)	0.02	27.75	27.77	(5)	135	(5)
3	MOS-SOCO2011	125.5	(10)	0.01	20.12	20.13	(10)	61	(10)
4	jDElscop	104.5	(7.5)	0.08	24.16	24.24	(9)	117	(6)
5	GADE	145	(12)	0.06	17.41	17.47	(12)	48	(12)
6	MOS-CEC2012	90.5	(4)	26.74	27.90	54.64	(3)	146	(4)
7	jDEsps	104.5	(7.5)	0.67	24.16	24.83	(7)	107	(9)
8	MOS-CEC2013	50.5	(1)	4.93	50.00	54.93	(2)	236	(1)
9	DECC-G	146	(13)	0.00	17.29	17.30	(13)	33	(13)
10	CC-CMA-ES	96	(6)	0.00	26.30	26.31	(6)	116	(7)
11	LSGOjDE	112.5	(9)	1.93	22.44	24.37	(8)	109	(8)
12	VMO-DE	138.5	(11)	0.00	18.23	18.23	(11)	49	(11)
13	IHDELS	83	(3)	50.00	30.42	80.42	(1)	181	(2)

improve variable grouping. In whatever direction future improvements will go, they will be welcome, as there are many large-scale problems in real-life waiting to be solved.

Acknowledgement

The authors acknowledge the financial support from the Slovenian Research Agency (research core fundings No. P2-0069 and No. P2-0041).

References

[1] A. LaTorre, S. Muelas, J.-M. Peña, A comprehensive comparison of large scale global optimizers, Information Sciences 316 (2015) 517–549.

[2] M. N. Omidvar, X. Li, K. Tang, Designing benchmark problems for large-scale continuous optimization, Information Sciences 316 (2015) 419–436.

[3] K. Tang, X. Yáo, P. N. Suganthan, C. MacNish, Y.-P. Chen, C.-M. Chen, Z. Yang, Benchmark functions for the cec2008 special session and competition on large scale global optimization, Tech. rep. (2007).

[4] K. Tang, X. Li, P. N. Suganthan, Z. Yang, T. Weise, Benchmark functions for the cec2010 special session and competition on large scale global optimization, Tech. rep. (2010).

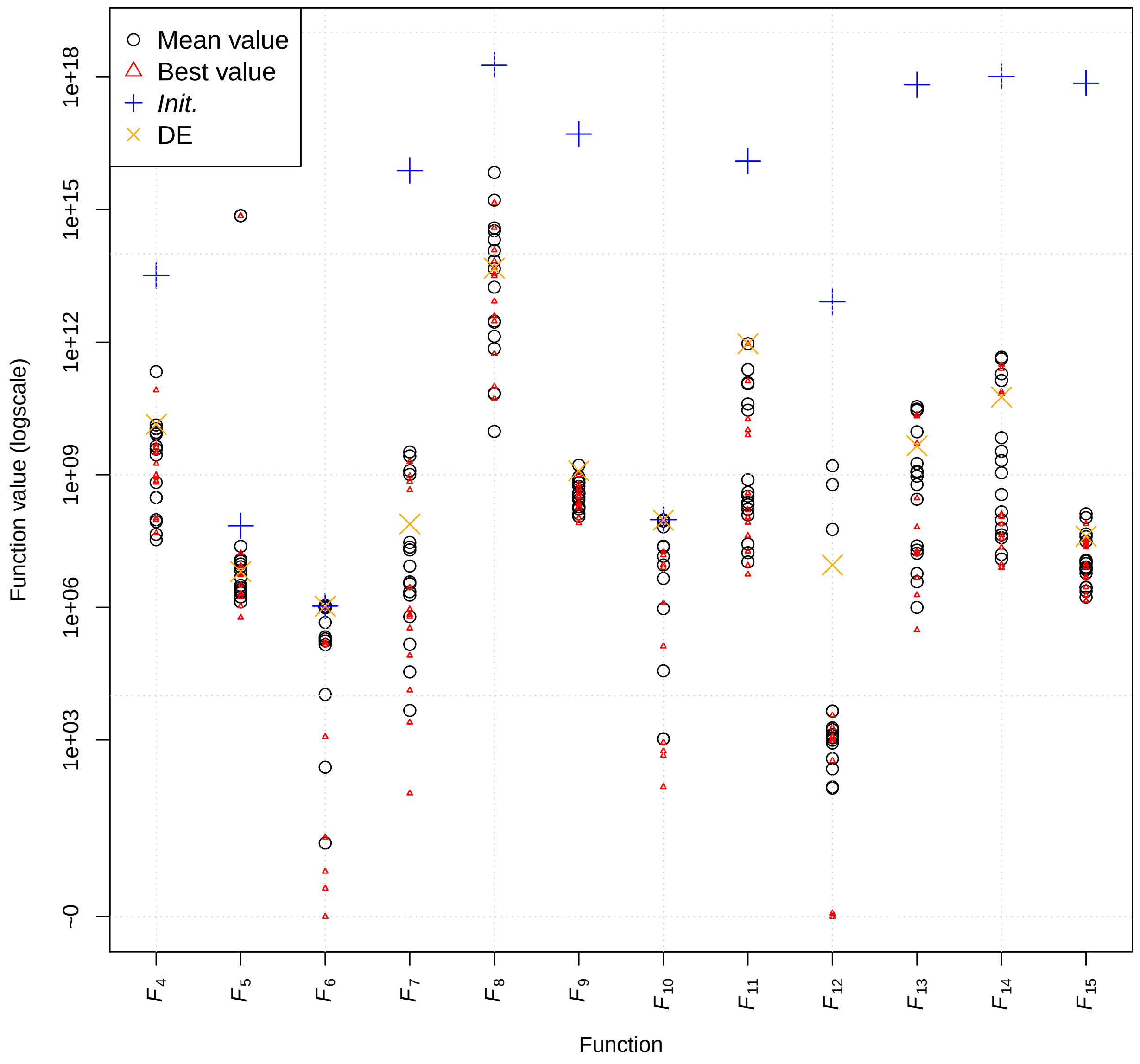


Figure 3: Comparison of mean and best values of algorithms on the CEC 2013 LSGO functions F_4–F_{15}. *Init.* stands for mean of function values of 100 randomly initialized solutions. DE are the solutions of the original DE algorithm with the binomial crossover.

[5] X. Li, K. Tang, M. N. Omidvar, Z. Yang, K. Qin, H. China, Benchmark functions for the cec 2013 special session and competition on large-scale global optimization, Tech., rep., RMIT University, Melbourne, Australia.

[6] M. Lozano, D. Molina, F. Herrera, Test Suite for the Special Issue of Soft Computing on Scalability of Evolutionary Algorithms and other Metaheuristics for Large Scale Continuous Optimization Problems, http://sci2s.ugr.es/sites/default/files/files/TematicWebSites/EAMHCO/functions1-19.pdf, [Online; accessed 4-June-2018] (2010).

[7] N. Hansen, S. Finck, R. Ros, A. Auger, Real-parameter black-box optimization benchmarking 2009: Noiseless functions definitions, Ph.D. thesis, Research Report RR-6829, INRIA (2009).

[8] O. Ait ElHara, A. Auger, N. Hansen, Permuted orthogonal block-diagonal transformation matrices for large scale optimization benchmarking, in: Proceedings of the Genetic and Evolutionary Computation Conference 2016, ACM, 2016, pp. 189–196.

[9] A. P. Piotrowski, Regarding the rankings of optimization heuristics based on artificially-constructed benchmark functions, Information Sciences 297 (2015) 191–201.

[10] S. Mahdavi, M. E. Shiri, S. Rahnamayan, Metaheuristics in large-scale global continues optimization: A survey, Information Sciences 295 (2015) 407–428.

[11] D. M. Cabrera, Evolutionary algorithms for large-scale global optimisation: a snapshot, trends and challenges, Progress in Artificial Intelligence 5 (2) (2016) 85–89.

[12] F. Caraffini, F. Neri, G. Iacca, Large scale problems in practice: The effect of dimensionality on the interaction among variables, in: European Conference on the Applications of Evolutionary Computation, Springer, 2017, pp. 636–652.

[13] K. E. Parsopoulos, Cooperative micro-differential evolution for high-dimensional problems, in: Proceedings of the 11th Annual conference on Genetic and evolutionary computation, ACM, 2009, pp. 531–538.

[14] K. E. Parsopoulos, Cooperative micro-particle swarm optimization, in: Proceedings of the first ACM/SIGEVO Summit on Genetic and Evolutionary Computation, ACM, 2009, pp. 467–474.

[15] A. Rajasekhar, S. Das, Cooperative micro artificial bee colony algorithm for large scale global optimization problems, in: International Conference on Swarm, Evolutionary, and Memetic Computing, Springer, 2013, pp. 469–480.

[16] J. Brest, M. S. Maučec, Population size reduction for the differential evolution algorithm, Applied Intelligence 29 (3) (2008) 228–247.

[17] G. Iacca, R. Mallipeddi, E. Mininno, F. Neri, P. N. Suganthan, Super-fit and population size reduction in compact differential evolution, in: 2011 IEEE Workshop on Memetic Computing (MC), IEEE, 2011, pp. 1–8.

[18] J. Brest, M. S. Maučec, Self-adaptive differential evolution algorithm using population size reduction and three strategies, Soft Computing 15 (11) (2011) 2157–2174.

[19] J. Brest, A. Zamuda, I. Fister, M. S. Maučec, et al., Self-adaptive differential evolution algorithm with a small and varying population size, in: 2012 IEEE Congress on Evolutionary Computation (CEC), IEEE, 2012, pp. 1–8.

[20] F. Caraffini, F. Neri, I. Poikolainen, Micro-differential evolution with extra moves along the axes, in: 2013 IEEE Symposium on Differential Evolution (SDE), IEEE, 2013, pp. 46–53.

[21] G. Iacca, F. Neri, E. Mininno, Y.-S. Ong, M.-H. Lim, Ockhams razor in memetic computing: three stage optimal memetic exploration, Information Sciences 188 (2012) 17–43.

[22] F. Caraffini, F. Neri, G. Iacca, A. Mol, Parallel memetic structures, Information Sciences 227 (2013) 60–82.

[23] F. Caraffini, F. Neri, B. N. Passow, G. Iacca, Re-sampled inheritance search: high performance despite the simplicity, Soft Computing 17 (12) (2013) 2235–2256.

[24] P. Korošec, J. Šilc, B. Filipič, The differential ant-stigmergy algorithm, Information Sciences 192 (2012) 82–97.

[25] R. Ros, N. Hansen, A simple modification in cma-es achieving linear time and space complexity, in: International Conference on Parallel Problem Solving from Nature, Springer, 2008, pp. 296–305.

[26] N. Hansen, The CMA evolution strategy: A tutorial, arXiv preprint arXiv:1604.00772.

[27] D. Molina, M. Lozano, F. Herrera, Ma-sw-chains: Memetic algorithm based on local search chains for large scale continuous global optimization, in: 2010 IEEE Congress on Evolutionary Computation (CEC), IEEE, 2010, pp. 3153–3160.

[28] A. Bolufé-Röhler, S. Fiol-González, S. Chen, A minimum population search hybrid for large scale global optimization, in: 2015 IEEE Congress on Evolutionary Computation (CEC), IEEE, 2015, pp. 1958–1965.

[29] S. Ghosh, S. Das, S. Roy, S. M. Islam, P. N. Suganthan, A differential covariance matrix adaptation evolutionary algorithm for real parameter optimization, Information Sciences 182 (1) (2012) 199–219.

[30] H. Huang, L. Lv, S. Ye, Z. Hao, Particle swarm optimization with convergence speed controller for large-scale numerical optimization, Soft Computing, doi:10.1007/s00500-018-3098-9.

[31] M. A. Potter, K. A. De Jong, A cooperative coevolutionary approach to function optimization, in: International Conference on Parallel Problem Solving from Nature, Springer, 1994, pp. 249–257.

[32] Z. Yang, K. Tang, X. Yao, Self-adaptive differential evolution with neighborhood search, in: IEEE Congress on Evolutionary Computation, CEC 2008 (IEEE World Congress on Computational Intelligence), IEEE, 2008, pp. 1110–1116.

[33] Z. Yang, K. Tang, X. Yao, Large scale evolutionary optimization using cooperative coevolution, Information Sciences 178 (15) (2008) 2985–2999.

[34] M. N. Omidvar, X. Li, Y. Mei, X. Yao, Cooperative co-evolution with differential grouping for large scale optimization, IEEE Transactions on Evolutionary Computation 18 (3) (2014) 378–393.

[35] X. Peng, K. Liu, Y. Jin, A dynamic optimization approach to the design of cooperative co-evolutionary algorithms, Knowledge-Based Systems 109 (2016) 174–186.

[36] X. Peng, Y. Wu, Large-scale cooperative co-evolution using niching-based multi-modal optimization and adaptive fast clustering, Swarm and Evolutionary Computation 35 (2017) 65–77.

[37] P. Yang, K. Tang, X. Yao, Turning high-dimensional optimization into computationally expensive optimization, IEEE Transactions on Evolutionary Computation 22 (1) (2018) 143–156.

[38] S. Mahdavi, S. Rahnamayan, M. E. Shiri, Incremental cooperative coevolution for large-scale global optimization, Soft Computing 22 (6) (2018) 2045–2064.

[39] Y. Sun, M. Kirley, S. K. Halgamuge, A recursive decomposition method for large scale continuous optimization, IEEE Transactions on Evolutionary Computation, doi:10.1109/TEVC.2017.2778089.

[40] X.-M. Hu, F.-L. He, W.-N. Chen, J. Zhang, Cooperation coevolution with fast interdependency identification for large scale optimization, Information Sciences 381 (2017) 142–160.

[41] A. Zamuda, J. Brest, B. Bošković, V. Žumer, Large scale global optimization using differential evolution with self-adaptation and cooperative co-evolution, in: IEEE Congress on Evolutionary Computation, CEC 2008 (IEEE World Congress on Computational Intelligence), 2008, pp. 3718–3725.

[42] X. Li, X. Yao, Cooperatively coevolving particle swarms for large scale optimization, IEEE Transactions on Evolutionary Computation 16 (2) (2012) 210–224.

[43] H. Ge, L. Sun, G. Tan, Z. Chen, C. P. Chen, Cooperative Hierarchical PSO With Two Stage Variable Interaction Reconstruction for Large Scale Optimization, IEEE Transactions on Cybernetics 47 (9) (2017) 2809–2823.

[44] Y. Ren, Y. Wu, An efficient algorithm for high-dimensional function optimization, Soft Computing 17 (6) (2013) 995–1004.

[45] M. Yang, M. N. Omidvar, C. Li, X. Li, Z. Cai, B. Kazimipour, X. Yao, Efficient resource allocation in cooperative co-evolution for large-scale global optimization, IEEE Transactions on Evolutionary Computation 21 (4) (2017) 493–505.

[46] Y. Mei, M. N. Omidvar, X. Li, X. Yao, A competitive divide-and-conquer algorithm for unconstrained large-scale black-box optimization, ACM Transactions on Mathematical Software (TOMS) 42 (2) (2016) 13:1–13:24. doi:10.1145/2791291.

[47] R. Storn, K. Price, Differential Evolution – A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces, Journal of Global Optimization 11 (1997) 341–359.

[48] R. Storn, K. Price, Differential Evolution - a simple and efficient adaptive scheme for global optimization over continuous spaces, Tech. Rep. TR-95-012, Berkeley, CA (1995).

[49] S. Das, P. N. Suganthan, Differential evolution: A survey of the state-of-the-art, IEEE Transactions on Evolutionary Computation 15 (1) (2011) 4–31.

[50] F. Neri, V. Tirronen, Recent advances in differential evolution: a survey and experimental analysis, Artificial Intelligence Review 33 (1-2) (2010) 61–106.

[51] S. Das, S. S. Mullick, P. N. Suganthan, Recent advances in differential evolution–an updated survey, Swarm and Evolutionary Computation 27 (2016) 1–30.

[52] 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 Transactions on Evolutionary Computation 10 (6) (2006) 646–657.

[53] J. Zhang, A. C. Sanderson, JADE: adaptive differential evolution with optional external archive, IEEE Transactions on Evolutionary Computation 13 (5) (2009) 945–958.

[54] A. K. Qin, P. N. Suganthan, Self-adaptive differential evolution algorithm for numerical optimization, in: The 2005 IEEE Congress on Evolutionary Computation, Vol. 2, IEEE, 2005, pp. 1785–1791.

[55] 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 (2) (2009) 398–417.

[56] A. Zamuda, J. Brest, Self-adaptive control parameters' randomization frequency and propagations in differential evolution, Swarm and Evolutionary Computation 25 (2015) 72–99.

[57] M. S. Maučec, J. Brest, B. Bošković, Z. Kačič, Improved differential evolution for large-scale black-box optimization, IEEE Access, doi:10.1109/ACCESS.2018.2842114.

[58] Z. Yang, X. Yao, J. He, Making a difference to differential evolution, in: Advances in metaheuristics for hard optimization, Springer, 2007, pp. 397–414.

[59] F. Neri, V. Tirronen, Scale factor local search in differential evolution, Memetic Computing 1 (2) (2009) 153–171.

[60] T. J. Choi, C. W. Ahn, J. An, An adaptive cauchy differential evolution algorithm for global numerical optimization, The Scientific World Journal 2013. doi:10.1155/2013/969734.

[61] T. J. Choi, C. W. Ahn, Adaptive α-stable differential evolution in numerical optimization, Natural Computing, 16 (4) (2017) 637–657. doi:10.1007/s11047-016-9579-9.

[62] T. Takahama, S. Sakai, Large scale optimization by differential evolution with landscape modality detection and a diversity archive, in: 2012 IEEE Congress on Evolutionary Computation (CEC), IEEE, 2012, pp. 1–8.

[63] Z. Yang, K. Tang, X. Yao, Scalability of generalized adaptive differential evolution for large-scale continuous optimization, Soft Computing 15 (11) (2011) 2141–2155.

[64] S. Das, A. Ghosh, S. S. Mullick, A switched parameter differential evolution for large scale global optimization–simpler may be better, in: Mendel 2015, Springer, 2015, pp. 103–125.

[65] A. Ghosh, S. Das, S. S. Mullick, R. Mallipeddi, A. K. Das, A switched parameter differential evolution with optional blending crossover for scalable numerical optimization, Applied Soft Computing 57 (2017) 329–352. doi:10.1016/j.asoc.2017.03.003.

[66] R. D. Al-Dabbagh, F. Neri, N. Idris, M. S. Baba, Algorithmic design issues in adaptive differential evolution schemes: Review and taxonomy, Swarm and Evolutionary Computation, in press. doi:10.1016/j.swevo.2018.03.008.

[67] T. Chen, K. Tang, G. Chen, X. Yao, A large population size can be unhelpful in evolutionary algorithms, Theoretical Computer Science 436 (2012) 54–70.

[68] S. Chen, J. Montgomery, A. Bolufé-Röhler, Measuring the curse of dimensionality and its effects on particle swarm optimization and differential evolution, Applied Intelligence 42 (3) (2015) 514–526.

[69] J. Brest, A. Zamuda, B. Bošković, M. S. Maučec, V. Žumer, High-dimensional real-parameter optimization using self-adaptive differential evolution algorithm with population size reduction, in: IEEE Congress on Evolutionary Computation, CEC 2008, (IEEE World Congress on Computational Intelligence), IEEE, 2008, pp. 2032–2039.

[70] J. Brest, A. Zamuda, I. Fister, M. S. Maučec, Large scale global optimization using self-adaptive differential evolution algorithm, in: 2010 IEEE Congress on Evolutionary Computation (CEC), IEEE, 2010, pp. 1–8.

[71] M. Olguin-Carbajal, E. Alba, J. Arellano-Verdejo, Micro-differential evolution with local search for high dimensional problems, in: 2013 IEEE Congress on Evolutionary Computation (CEC), IEEE, 2013, pp. 48–54.

[72] M. Olguin-Carbajal, J. C. Herrera-Lozada, J. Arellano-Verdejo, R. Barron-Fernandez, H. Taud, Micro differential evolution performance empirical study for high dimensional optimization problems, in: International Conference on Large-Scale Scientific Computing, Springer, 2013, pp. 281–288.

[73] C. Brown, Y. Jin, M. Leach, M. Hodgson, μJADE: adaptive differential evolution with a small population, Soft Computing 20 (10) (2016) 4111–4120.

[74] A. P. Piotrowski, Review of differential evolution population size, Swarm and Evolutionary Computation 32 (2017) 1–24. doi:10.1016/j.swevo.2016.05.003.

[75] M. Yang, C. Li, Z. Cai, J. Guan, Differential evolution with auto-enhanced population diversity, IEEE Transactions on Cybernetics 45 (2) (2015) 302–315.

[76] Y. Ao, Differential evolution using second mutation for high-dimensional real-parameter optimization, in: Measuring Technology and Mechatronics Automation in Electrical Engineering, Springer, 2012, pp. 191–201.

[77] J.-i. Kushida, A. Hara, T. Takahama, Rank-based differential evolution with multiple mutation strategies for large scale global optimization, in: 2015 IEEE Congress on Evolutionary Computation (CEC), IEEE, 2015, pp. 353–360.

[78] Y. Xuemei, Differential evolution with a new mutation operator for solving high dimensional continuous optimization problems, Journal of Computational Information Systems 6 (9) (2010) 3033–3039.

[79] S. M. Islam, S. Das, S. Ghosh, S. Roy, P. N. Suganthan, An adaptive differential evolution algorithm with novel mutation and crossover strategies for global numerical optimization, IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics) 42 (2) (2012) 482–500.

[80] S.-Z. Zhao, P. N. Suganthan, S. Das, Self-adaptive differential evolution with multi-trajectory search for large-scale optimization, Soft Computing 15 (11) (2011) 2175–2185.

[81] Y.-Z. Zhou, W.-C. Yi, L. Gao, X.-Y. Li, Adaptive differential evolution with sorting crossover rate for continuous optimization problems, IEEE Transactions on Cybernetics 47 (9) (2017) 2742–2753.

[82] C. García-Martínez, F. J. Rodríguez, M. Lozano, Role differentiation and malleable mating for differential evolution: an analysis on large-scale optimisation, Soft Computing 15 (11) (2011) 2109–2126.

[83] S.-Z. Zhao, P. N. Suganthan, Empirical investigations into the exponential crossover of differential evolutions, Swarm and Evolutionary Computation 9 (2013) 27–36.

[84] S.-M. Guo, C.-C. Yang, Enhancing differential evolution utilizing eigenvector-based crossover operator, IEEE Transactions on Evolutionary Computation 19 (1) (2015) 31–49.

[85] C. Segura, C. A. C. Coello, A. G. Hernández-Díaz, Improving the vector generation strategy of differential evolution for large-scale optimization, Information Sciences 323 (2015) 106–129.

[86] Q. Yang, H.-Y. Xie, W.-N. Chen, J. Zhang, Multiple parents guided differential evolution for large scale optimization, in: 2016 IEEE Congress on Evolutionary Computation (CEC), IEEE, 2016, pp. 3549–3556.

[87] A. W. Mohamed, A. S. Almazyad, Differential evolution with novel mutation and adaptive crossover strategies for solving large scale global optimization problems, Applied Computational Intelligence and Soft Computing 2017, article ID 7974218. doi:10.1155/2017/7974218.

[88] M. Locatelli, M. Maischberger, F. Schoen, Differential evolution methods based on local searches, Computers & Operations Research 43 (2014) 169–180.

[89] L.-Y. Tseng, C. Chen, Multiple trajectory search for large scale global optimization, in: 2008 IEEE Congress on Evolutionary Computation, CEC 2008, (IEEE World Congress on Computational Intelligence), IEEE, 2008, pp. 3052–3059.

[90] X. Pan, Y. Zhao, X. Xu, Adaptive differential evolution with local search for solving large-scale optimization problems, Journal of Information & Computational Science 9 (2) (2012) 489–496.

[91] W. Xie, W. Yu, X. Zou, Diversity-maintained differential evolution embedded with gradient-based local search, Soft computing 17 (8) (2013) 1511–1535.

[92] W. Huang, S.-K. Oh, Z. Guo, W. Pedrycz, A space search optimization algorithm with accelerated convergence strategies, Applied Soft Computing 13 (12) (2013) 4659–4675.

[93] H. Wang, Z. Wu, S. Rahnamayan, Enhanced opposition-based differential evolution for solving high-dimensional continuous optimization problems, Soft Computing 15 (11) (2011) 2127–2140.

[94] H. Wang, Z. Wu, S. Rahnamayan, D. Jiang, Sequential de enhanced by neighborhood search for large scale global optimization, in: 2010 IEEE Congress on Evolutionary Computation (CEC), IEEE, 2010, pp. 4056–4062.

[95] Y. Cai, J. Wang, Differential evolution with neighborhood and direction information for numerical optimization, IEEE Transactions on Cybernetics 43 (6) (2013) 2202–2215.

[96] D. Molina, F. Herrera, Iterative hybridization of DE with local search for the CEC'2015 special session on large scale global optimization, in: 2015 IEEE Congress on Evolutionary Computation (CEC), IEEE, 2015, pp. 1974–1978.

[97] Y. Cai, J. Liao, T. Wang, Y. Chen, H. Tian, Social learning differential evolution, Information Sciences 433-434 (2016) 464–509. doi:10.1016/j.ins.2016.10.003.

[98] Y. Cai, M. Zhao, J. Liao, T. Wang, H. Tian, Y. Chen, Neighborhood guided differential evolution, Soft Computing 21 (16) (2017) 4769–4812.

[99] B. Kazimipour, X. Li, A. K. Qin, Initialization methods for large scale global optimization, in: Evolutionary Computation (CEC), 2013 IEEE Congress on, IEEE, 2013, pp. 2750–2757.

[100] M. Ali, M. Pant, A. Abraham, Improving differential evolution algorithm by synergizing different improvement mechanisms, ACM Transactions on Autonomous and Adaptive Systems (TAAS) 7 (2) (2012) 20:1–20:32. doi:10.1145/2240166.2240170.

[101] S. Rahnamayan, G. G. Wang, Solving large scale optimization problems by opposition-based differential evolution (ODE), WSEAS Transactions on Computers 7 (10) (2008) 1792–1804.

[102] S. Rahnamayan, H. R. Tizhoosh, M. M. Salama, Opposition-based differential evolution, IEEE Transactions on Evolutionary computation 12 (1) (2008) 64–79.

[103] H. Wang, S. Rahnamayan, Z. Wu, Parallel differential evolution with self-adapting control parameters and generalized opposition-based learning for solving high-dimensional optimization problems, Journal of Parallel and Distributed Computing 73 (1) (2013) 62–73.

[104] M. A. Ahandani, H. Alavi-Rad, Opposition-based learning in the shuffled differential evolution algorithm, Soft Computing 16 (8) (2012) 1303–1337.

[105] M. A. Ahandani, Opposition-based learning in the shuffled bidirectional differential evolution algorithm, Swarm and Evolutionary Computation 26 (2016) 64–85.

[106] A. Esmailzadeh, S. Rahnamayan, Enhanced differential evolution using center-based sampling, in: 2011 IEEE Congress on Evolutionary Computation (CEC), IEEE, 2011, pp. 2641–2648.

[107] S. Mahdavi, S. Rahnamayan, K. Deb, Center-based initialization of cooperative co-evolutionary algorithm for large-scale optimization, in: Evolutionary Computation (CEC), 2016 IEEE Congress on, IEEE, 2016, pp. 3557–3565.

[108] M. Ali, M. Pant, A. Abraham, Unconventional initialization methods for differential evolution, Applied Mathematics and Computation 219 (9) (2013) 4474–4494.

[109] H. Salehinejad, S. Rahnamayan, Effects of centralized population initialization in differential evolution, in: 2016 IEEE Symposium Series on Computational Intelligence (SSCI), IEEE, 2016, pp. 1–8.

[110] B. Kazimipour, X. Li, A. K. Qin, Effects of population initialization on differential evolution for large scale optimization, in: 2014 IEEE Congress on Evolutionary Computation (CEC), IEEE, 2014, pp. 2404–2411.

[111] E. Segredo, B. Paechter, C. Segura, C. I. González-Vila, On the comparison of initialisation strategies in differential evolution for large scale optimisation, Optimization Letters 12 (1) (2018) 221–234. doi:10.1007/s11590-017-1107-z.

[112] A. LaTorre, J.-M. Peña, On the scalability of population restart mechanisms on large-scale global optimization, in: Evolutionary Computation (CEC), 2017 IEEE Congress on, IEEE, 2017, pp. 1071–1078.

[113] R. Tanabe, A. Fukunaga, Success-history based parameter adaptation for differential evolution, in: 2013 IEEE Congress on Evolutionary Computation (CEC), IEEE, 2013, pp. 71–78.

[114] R. A. Khanum, N. Tairan, M. A. Jan, W. K. Mashwani, A. Salhi, Reflected adaptive differential evolution with two external archives for large-scale global optimization, International Journal of Advanced Computer Science and Applications 7 (2) (2016) 675–683.

[115] X. Zhang, X. Zhang, Improving differential evolution by differential vector archive and hybrid repair method for global optimization, Soft Computing 21 (23) (2017) 7107–7116. doi:10.1007/s00500-016-2253-4.

[116] D. Zaharie, D. Petcu, Parallel implementation of multi-population differential evolution, Concurrent information processing and computing 195 (2005) 223–232.

[117] M. Weber, F. Neri, V. Tirronen, Distributed differential evolution with explorative–exploitative population families, Genetic Programming and Evolvable Machines 10 (4) (2010) 343–371.

[118] M. Weber, F. Neri, V. Tirronen, Shuffle or update parallel differential evolution for large-scale optimization, Soft Computing 15 (11) (2011) 2089–2107.

[119] M. Weber, F. Neri, V. Tirronen, A study on scale factor in distributed differential evolution, Information Sciences 181 (12) (2011) 2488–2511.

[120] C. Zhang, J. Chen, B. Xin, Distributed memetic differential evolution with the synergy of lamarckian and baldwinian learning, Applied Soft Computing 13 (5) (2013) 2947–2959.

[121] G. Jeyakumar, C. S. Velayutham, Distributed heterogeneous mixing of differential and dynamic differential evolution variants for unconstrained global optimization, Soft Computing 18 (10) (2014) 1949–1965.

[122] J. Apolloni, J. García-Nieto, E. Alba, G. Leguizamón, Empirical evaluation of distributed differential evolution on standard benchmarks, Applied Mathematics and Computation 236 (2014) 351–366.

[123] J. Cheng, G. Zhang, F. Neri, Enhancing distributed differential evolution with multicultural migration for global numerical optimization, Information Sciences 247 (2013) 72–93.

[124] D. R. Penas, J. R. Banga, P. González, R. Doallo, Enhanced parallel differential evolution algorithm for problems in computational systems biology, Applied Soft Computing 33 (2015) 86–99.

[125] M. Z. Ali, N. H. Awad, P. N. Suganthan, Multi-population differential evolution with balanced ensemble of mutation strategies for large-scale global optimization, Applied Soft Computing 33 (2015) 304–327.

[126] M. Z. Ali, N. H. Awad, P. N. Suganthan, R. G. Reynolds, An adaptive multipopulation differential evolution with dynamic population reduction, IEEE Transactions on Cybernetics 47 (9) (2017) 2768–2779.

[127] Y.-F. Ge, W.-J. Yu, J. Zhang, Diversity-based multi-population differential evolution for large-scale optimization, in: Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion, ACM, 2016, pp. 31–32.

[128] Z. Yang, K. Tang, X. Yao, Multilevel cooperative coevolution for large scale optimization, in: 2008 IEEE Congress on Evolutionary Computation, CEC 2008, (IEEE World Congress on Computational Intelligence), IEEE, 2008, pp. 1663–1670.

[129] F. Wei, Y. Wang, T. Zong, Variable grouping based differential evolution using an auxiliary function for large scale global optimization, in: 2014 IEEE Congress on Evolutionary Computation (CEC), IEEE, 2014, pp. 1293–1298.

[130] H. Ge, L. Sun, X. Yang, S. Yoshida, Y. Liang, Cooperative differential evolution with fast variable interdependence learning and cross-cluster mutation, Applied Soft Computing 36 (2015) 300–314.

[131] H. Ge, L. Sun, X. Yang, Adaptive hybrid differential evolution with circular sliding window for large scale optimization, in: 2016 12th International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD), IEEE, 2016, pp. 87–94.

[132] M. N. Omidvar, M. Yang, Y. Mei, X. Li, X. Yao, DG2: A faster and more accurate differential grouping for large-scale black-box optimization, IEEE Transactions on Evolutionary Computation 21 (6) (2017) 929–942. doi:10.1109/TEVC.2017.2694221.

[133] M. N. Omidvar, B. Kazimipour, X. Li, X. Yao, Cbcc3a contribution-based cooperative co-evolutionary algorithm with improved exploration/exploitation balance, in: 2016 IEEE Congress on Evolutionary Computation (CEC), 2016, pp. 3541–3548.

[134] Y. Cai, J. Wang, Differential evolution with hybrid linkage crossover, Information Sciences 320 (2015) 244–287.

[135] C. Wang, J.-H. Gao, A differential evolution algorithm with cooperative coevolutionary selection operation for high-dimensional optimization, Optimization Letters 8 (2) (2014) 477–492.

[136] A. F. Ali, N. N. Ahmed, Differential evolution algorithm with space partitioning for large-scale optimization problems, International Journal of Intelligent Systems and Applications 7 (11) (2015) 49–59.

[137] N. Sabar, J. Abawajy, J. Yearwood, Heterogeneous cooperative co-evolution memetic differential evolution algorithms for big data optimisation problems, IEEE Transactions on Evolutionary Computation 21 (2) (2017) 315–327.

[138] E. Glorieux, B. Svensson, F. Danielsson, B. Lennartson, Improved constructive cooperative coevolutionary differential evolution for large-scale optimisation, in: 2015 IEEE Symposium Series on Computational Intelligence, IEEE, 2015, pp. 1703–1710.

[139] E. Glorieux, B. Svensson, F. Danielsson, B. Lennartson, Constructive cooperative coevolution for large-scale global optimisation, Journal of Heuristics 23 (6) (2017) 449–469. doi:10.1007/s10732-017-9351-z.

[140] S. Mahdavi, S. Rahnamayan, M. E. Shiri, Cooperative co-evolution with sensitivity analysis-based budget assignment strategy for large-scale global optimization, Applied Intelligence 47 (3) (2017) 888–913.

[141] S. Mahdavi, S. Rahnamayan, M. E. Shiri, Multilevel framework for large-scale global optimization, Soft Computing 21 (14) (2017) 4111–4140.

[142] X. Peng, Y. Wu, Large-scale cooperative co-evolution with bi-objective selection based imbalanced multi-modal optimization, in: Evolutionary Computation (CEC), 2017 IEEE Congress on, IEEE, 2017, pp. 1527–1532.

[143] R. Akay, A. Basturk, A. Kalinli, X. Yao, Parallel population-based algorithm portfolios: An empirical study, Neurocomputing 247 (2017) 115–125. doi:10.1016/j.neucom.2017.03.061.

[144] Y. Wang, B. Li, Two-stage based ensemble optimization for large-scale global optimization, in: 2010 IEEE Congress on Evolutionary Computation (CEC), IEEE, 2010, pp. 1–8.

[145] E. D. López, A. Puris, R. R. Bello, VMODE: a hybrid metaheuristic for the solution of large scale optimization problems, Investigación Operacional 36 (3) (2015) 232–240.

[146] A. Puris, R. Bello, D. Molina, F. Herrera, Variable mesh optimization for continuous optimization problems, Soft Computing 16 (3) (2012) 511–525.

[147] S. Yang, Y. Sato, Modified bare bones particle swarm optimization with differential evolution for large scale problem, in: 2016 IEEE Congress on Evolutionary Computation (CEC), IEEE, 2016, pp. 2760–2767.

[148] N. H. Awad, M. Z. Ali, P. N. Suganthan, R. G. Reynolds, CADE: A hybridization of Cultural Algorithm and Differential Evolution for numerical optimization, Information Sciences 378 (2017) 215–241.

[149] A. LaTorre, S. Muelas, J.-M. Peña, Large scale global optimization: Experimental results with mos-based hybrid algorithms, in: 2013 IEEE Congress on Evolutionary Computation (CEC), IEEE, 2013, pp. 2742–2749.

[150] J. Liu, K. Tang, Scaling up covariance matrix adaptation evolution strategy using cooperative coevolution, in: International Conference on Intelligent Data Engineering and Automated Learning, Springer, 2013, pp. 350–357.

[151] A. LaTorre, S. Muelas, J.-M. Peña, A mos-based dynamic memetic differential evolution algorithm for continuous optimization: a scalability test, Soft Computing 15 (11) (2011) 2187–2199.

[152] A. LaTorre, S. Muelas, J.-M. Peña, Multiple offspring sampling in large scale global optimization, in: 2012 IEEE Congress on Evolutionary Computation (CEC), IEEE, 2012, pp. 1–8.

[153] M. Yang, M. N. Omidvar, C. Li, X. Li, Z. Cai, B. Kazimipour, X. Yao, Efficient resource allocation in cooperative co-evolution for large-scale global optimization, IEEE Transactions on Evolutionary Computation 21 (4) (2017) 493–505.

[154] N. H. Awad, M. Z. Ali, B. Y. Q. J. J. Liang, P. N. Suganthan, Problem Definitions and Evaluation Criteria for the CEC 2017 Special Session and Competition on Single Objective Bound Constrained Real-Parameter Numerical Optimization, Tech. rep., Nanyang Technological University, Singapore (November 2016).