Noname manuscript No.
(will be inserted by the editor)

Review and Analysis of Three Components of Differential Evolution Mutation Operator in MOEA/D-DE

Ryoji Tanabe · Hisao Ishibuchi

Received: date / Accepted: date

Abstract Abstract A decomposition-based multi-objective evolutionary algorithm with a differential evolution variation operator (MOEA/D-DE) shows high performance on challenging multi-objective problems (MOPs). The DE mutation consists of three key components: a mutation strategy, an index selection method for parent individuals, and a bound-handling method. However, the configuration of the DE mutation operator that should be used for MOEA/D-DE has not been thoroughly investigated in the literature. This configuration choice confuses researchers and users of MOEA/D-DE. To address this issue, we present a review of the existing configurations of the DE mutation operator in MOEA/D-DE and systematically examine the influence of each component on the performance of MOEA/D-DE. Our review reveals that the configuration of the DE mutation operator differs depending on the source code of MOEA/D-DE. In our analysis, a total of 30 configurations (three index selection methods, two mutation strategies, and five bound handling methods) are investigated on 16 MOPs with up to five objectives. Results show that each component significantly affects the performance of MOEA/D-DE. We also present the most suitable configuration of the DE mutation operator, which maximizes the effectiveness of MOEA/D-DE.

Keywords Multi-objective optimization · Decomposition based evolutionary algorithms · Differential evolution operators · Implementation of algorithms

R. Tanabe and H. Ishibuchi
Shenzhen Key Laboratory of Computational Intelligence, University Key Laboratory of Evolving Intelligent Systems of Guangdong Province, Department of Computer Science and Engineering, Southern University of Science and Technology, Shenzhen 518055, China
E-mail: rt.ryoji.tanabe@gmail.com, hisao@sustc.edu.cn (Corresponding author: Hisao Ishibuchi)

Keywords Multi-objective optimization · Decomposition based evolutionary algorithms · Differential evolution operators · Implementation of algorithms

1 Introduction

A multi-objective evolutionary algorithm (MOEA) is an efficient approach for solving multi-objective optimization problems (MOPs) [10]. Since MOEAs use a set of individuals for the search, it is expected that well distributed nondominated solutions can be found by a single run. MOEAs have been successfully applied to real-world problems, such as aerodynamic wing design problems [30], vehicle design problems [28], oil well problems [24], and groundwater monitoring design problems [22].

A variation operator is used to generate a child in MOEAs. Although genetic algorithm (GA) operators are frequently used in the evolutionary multi-objective optimization community, some recent studies report the superiority of differential evolution (DE) operators [35]. For example, the performance of NSGA-II [11], SPEA2 [46], and IBEA [45] with GA and DE operators is investigated in [37]. The results reported in [37] show that better nondominated solutions can be found by using the DE operator. A further examination of the impact of both of these operators on MOEAs is presented in [5]. In [42], the performance of NSGA-III [12] with GA and DE operators is investigated on MOPs with more than four objectives. A comparison of SMS-EMOA [4] with various operators is performed in [2]. The results presented in [2] show that the DE operator is more suitable than the GA operator for SMS-EMOA in most cases.

MOEA/D-DE [25] is one of the most successful examples of the DE variation operator. MOEA/D [43, 36] is an MOEA that decomposes a given MOP into

multiple single-objective sub-problems and solves them simultaneously. MOEA/D-DE is a variant of MOEA/D in which the GA operator is replaced by the DE operator[1]. The results reported in [25] show that MOEA/D-DE is capable of handling complicated Pareto solution sets. Some improved MOEA/D-DE algorithms have been proposed in the literature, and representative examples include MOEA/D-DRA [44], MOEA/D-MAB [26], MOEA/D-STM [27], MOEA/D-GRA [40], and MOEA/D-TPN [21].

In general, the term "DE operator" denotes a complex differential mutation operator and a crossover method [35]. In the procedure of the DE operator, a mutant vector is first generated by applying the DE mutation operator to some individuals in the population. Then, a child is generated by recombining the mutant vector and a parent individual. Binomial crossover is widely used in MOEA/D-DE-type algorithms. However, to handle nonseparability, the procedure of the crossover operator is not actually performed in most MOEA/D-DE-type algorithms by setting the crossover rate $C \in [0, 1]$ to 1 [25].

In contrast to the crossover method, there is large flexibility in implementing the DE mutation operator. The DE mutation operator consists of the following three key components:

- A type of mutation strategy (e.g., the rand/1, rand/2, and current-to-rand/1 strategies),
- An index selection method for parent individuals (i.e., how to select indices r_1, r_2, and r_3), and
- A bound-handling method to repair an infeasible solution.

On the one hand, previous studies [1,29,32,39] reveal that the performance of DE algorithms for single-objective optimization is significantly influenced by the above-listed three components. For example, the results presented in [1] show that the choice of bound-handling methods has a large impact on the performance of the basic DE [35]. For this reason, most researchers in the DE community carefully select each component of the DE mutation operator to maximize the performance of DE.

On the other hand, in contrast to the DE community, the details of the DE mutation operator have not received considerable attention in the evolutionary multi-objective optimization community. Thus, the influence of configurations of the DE mutation operator on MOEA/D-DE has not been thoroughly investigated. Since the configuration of the DE mutation operator that should be used is unclear, algorithm designers and users of MOEA/D-DE face confusion regarding the configuration choice. In fact, different implementations are used in the uploaded source code of MOEA/D-DE. For example, while the current/1 strategy is incorporated into MOEA/D-DE in the jMetal source code[2], the rand/1 strategy is used in the MOEA Framework source code[3]. Different comparison results could be obtained simply depending on the choice of a library if MOEA/D-DE in an MOEA library performs significantly better than that in another MOEA library. This situation is undesirable for researchers. Although users of MOEAs generally want to apply a well-performing algorithm to their MOPs, the best configuration of the DE mutation operator for MOEA/D-DE is unknown.

To address these issues, we present a review of the existing configurations of the DE mutation in MOEA/D-DE. We also analyze the influence of each element on the effectiveness of MOEA/D-DE. The purpose of this review is to clarify the current situation. We examine the three components that can be found in the literature, source code, and MOEA libraries. Then, we present a large-scale experimental study to investigate the effect of the three components on MOEA/D-DE. A total of 30 configurations (three index selection methods, two mutation strategies, and five bound-handling methods) are investigated in our study. The influence of each component on the performance of MOEA/D-DE is carefully examined in a component-wise manner. We also suggest the best configuration of the DE mutation operator, which maximizes the performance of MOEA/D-DE.

The reminder of this paper is organized as follows: Section 2 describes the background of MOPs and MOEA/D-DE. Section 3 reviews existing configurations of the DE mutation operator in MOEA/D-DE. Section 4 describes experimental settings. Section 5 shows results of MOEA/D-DE with various configurations of the DE mutation operator. Finally, Section 6 concludes this paper.

2 Preliminaries

2.1 Definition of MOPs

A bound-constrained MOP, which is addressed in this paper, can be formulated as follows:

$$\text{minimize} \quad \boldsymbol{f}(\boldsymbol{x}) = \big(f_1(\boldsymbol{x}), ..., f_M(\boldsymbol{x})\big)^{\mathrm{T}}, \tag{1}$$

[1] Strictly speaking, the differences between MOEA/D-DE [25] and the original MOEA/D [43] are as follows: (i) the parent individuals are selected from the whole population with some probability, (ii) the number of individuals replaced by a child is restricted, and (iii) the DE variation operator is used in [25].

[2] `http://jmetal.sourceforge.net/`

[3] `http://moeaframework.org/index.html`

where $\mathbb{S}$ is the solution space. $\boldsymbol{f} : \mathbb{S} \to \mathbb{R}^M$ is an objective function vector that consists of M potentially conflicting objective functions, and $\mathbb{R}^M$ is the objective space. In (1), $\boldsymbol{x} = (x_1, ..., x_D)^{\mathrm{T}}$ is a solution vector, and $\mathbb{S} = \prod_{j=1}^{D} [x_j^{\min}, x_j^{\max}]$ is the bound-constrained solution space where $x_j^{\min} \leq x_j \leq x_j^{\max}$ for each index $j \in \{1, ..., D\}$.

We say that $\boldsymbol{x}^1$ dominates $\boldsymbol{x}^2$ if and only if $f_i(\boldsymbol{x}^1) \leq f_i(\boldsymbol{x}^2)$ for all $i \in \{1, ..., M\}$ and $f_i(\boldsymbol{x}^1) < f_i(\boldsymbol{x}^2)$ for at least one index i. Here, $\boldsymbol{x}^*$ is a Pareto-optimal solution if no $\boldsymbol{x} \in \mathbb{S}$ exists such that $\boldsymbol{x}$ dominates $\boldsymbol{x}^*$. In this case, $\boldsymbol{f}(\boldsymbol{x}^*)$ is a Pareto-optimal objective vector. The set of all $\boldsymbol{x}^*$ in $\mathbb{S}$ is the Pareto-optimal solution set, and the set of all $\boldsymbol{f}(\boldsymbol{x}^*)$ in $\mathbb{R}^M$ is the Pareto front. In general, no solution can simultaneously minimize all objective functions $f_1, ..., f_M$ in an MOP. Thus, the goal of multi-objective optimization is to find a set of non-dominated solutions that are well distributed and close to the Pareto front in the objective space.

Algorithm 1: The procedure of MOEA/D-DE

```
1  t ← 1, initialize the population P = {x^1, ..., x^μ};
2  for i ∈ {1, ..., μ} do
3      Set the neighborhood index list B^i = {i_1, ..., i_T};
4  while The termination criteria are not met do
5      for i ∈ {1, ..., μ} do
6          if rand[0, 1] ≤ δ then
7              R ← B^i;
8          else
9              R ← {1, ..., μ};
10         Select parent indices from R with an index
           selection method (Subsection 3.2);
11         Generate the mutant vector v^i using a mutation
           strategy (Subsection 3.1);
12         if v^i ∉ S then
13             Repair v^i using a bound-handling method
               (Subsection 3.3);
14         Generate the child u^i by crossing x^i and v^i;
15         Apply a GA mutation operator to u^i;
16         c ← 1;
17         while c ≤ n^rep and R ≠ ∅ do
18             Randomly select an index j from R, and
               R ← R\{j};
19             if g(u^i|w^j, z^*) ≤ g(x^j|w^j, z^*) then
20                 x^j ← u^i, c ← c + 1;
21     t ← t + 1;
```

2.2 MOEA/D-DE

MOEA/D-type algorithms, including MOEA/D-DE [25], decompose an M-objective MOP defined in (1) into μ single-objective sub-problems $g_1(\boldsymbol{x}|\boldsymbol{w}^1), ..., g_\mu(\boldsymbol{x}|\boldsymbol{w}^\mu)$ using a set of uniformly distributed weight vectors $\boldsymbol{W} = \{\boldsymbol{w}^1, ..., \boldsymbol{w}^\mu\}$ and a scalarizing function $\boldsymbol{g} : \mathbb{R}^M \to \mathbb{R}$, where μ is the population size, $\boldsymbol{w}^i = (w_1^i, ..., w_M^i)^{\mathrm{T}}$ for each $i \in \{1, ..., \mu\}$, and $\sum_{j=1}^{M} w_j^i = 1$. For each $i \in \{1, ..., \mu\}$, an individual $\boldsymbol{x}^i$ is assigned to the i-th sub-problem. MOEA/D-type algorithms attempt to find the optimal solutions of all sub-problems simultaneously.

Algorithm 1 shows the procedure of MOEA/D-DE. After initialization (lines 1–3), the following steps are iteratively performed. For each individual, an index list is selected for use in the mating and replacement selections (lines 5–9). Then, the mating and reproduction operations are performed (lines 10–15). Finally, the replacement selection is applied to the child and the individuals in the population (lines 17–20). We explain each step of MOEA/D-DE in detail below.

At the beginning of the search, all individuals in the population $\boldsymbol{P} = \{\boldsymbol{x}^1, ..., \boldsymbol{x}^\mu\}$ are randomly generated in the solution space (line 1). For each sub-problem index $i \in \{1, ..., \mu\}$, an index list $\boldsymbol{B}^i = \{i_1, ..., i_T\}$ is initialized (lines 2-3): $\boldsymbol{B}^i$ consists of indices of the T closest weight vectors to $\boldsymbol{w}^i$ in the weight vector space, where T is the neighborhood size. After initialization, the following steps (lines 5–20) are repeatedly applied to each sub-problem until a termination condition is satisfied.

For each i, a set of individual indices $\boldsymbol{R}$ is set to $\boldsymbol{B}^i$ with a probability of $\delta \in [0, 1]$ or $\{1, ..., \mu\}$ with a probability of $1 - \delta$ (lines 6–9). The function rand$[0, 1]$ in line 6 is a randomly chosen value in the range $[0, 1]$. After $\boldsymbol{R}$ has been determined, MOEA/D-DE generates a mutant vector $\boldsymbol{v}^i$ (lines 10–13). First, individual indices $\{r_1, r_2, ...\}$ are randomly selected from $\boldsymbol{R}$ using an index selection method (line 10). Next, $\boldsymbol{v}^i$ is generated by applying a mutation strategy to the selected individuals $\{\boldsymbol{x}^{r_1}, \boldsymbol{x}^{r_2}, ...\}$ (line 11). If an element of $\boldsymbol{v}^i$ violates a corresponding bound constraint (i.e., $\boldsymbol{v}^i \notin \mathbb{S}$), then a bound-handling method is applied to it such that $\boldsymbol{v}^i \in \mathbb{S}$ (line 13). The details of each procedure (the index selection, the mutation strategy, and the bound-handling method) are described in Subsections 3.2, 3.1, and 3.3, respectively.

After $\boldsymbol{v}^i$ is repaired by a bound-handling method (if needed), a child $\boldsymbol{u}^i$ is generated by recombining $\boldsymbol{x}^i$ and $\boldsymbol{v}^i$ (line 14). Binomial crossover [35], which is the most basic crossover method in DE, is defined as follows:

$$u_j^i = \begin{cases} v_j^i & \text{if rand}[0, 1] \leq C \text{ or } j = j^{\text{rand}} \\ x_j^i & \text{otherwise} \end{cases}, \tag{2}$$

where the crossover rate $C \in [0, 1]$ in (2) controls the number of inherited variables from $\boldsymbol{x}^i$ to $\boldsymbol{u}^i$. The decision variable index j^{rand} in (2) is randomly selected from $\{1, ..., D\}$. Since binomial crossover only exchanges elements between the parent individual $\boldsymbol{x}^i$ and the mutant vector $\boldsymbol{v}^i$, the child $\boldsymbol{u}^i$ always satisfies the bound constraints as long as $\boldsymbol{x}^i$, $\boldsymbol{v}^i \in \mathbb{S}$.

In MOEA/D-DE, a GA mutation operator is applied to the child $\boldsymbol{u}^i$ (line 15). The following polynomial mutation, which is commonly used in the evolutionary multi-objective optimization community, is performed for each $j \in \{1, ..., D\}$:

$$u_j^i = \begin{cases} u_j^i + \sigma_j \left(x_j^{\max} - x_j^{\min}\right) & \text{if } \mathrm{rand}[0,1] \leq p^{\mathrm{mut}} \\ u_j^i & \text{otherwise} \end{cases}, \tag{3}$$

where $p^{\mathrm{mut}} \in [0, 1]$ is the mutation rate. If $u_j^i \notin [x_j^{\min}, x_j^{\max}]$, then it is replaced by the closest value ($x_j^{\min}$ or $x_j^{\max}$). The amount of perturbation σ_j in (3) is defined as follows:

$$\sigma_j = \begin{cases} (2\,\mathrm{rand}[0,1])^{1/(\eta^{\mathrm{mut}}+1)} - 1 & \text{if } \mathrm{rand}[0,1] \leq 0.5 \\ 1 - (2 - 2\,\mathrm{rand}[0,1])^{1/(\eta^{\mathrm{mut}}+1)} & \text{otherwise} \end{cases}, \tag{4}$$

where $\eta^{\mathrm{mut}} > 0$ in (4) is the distribution index.

After $\boldsymbol{u}^i$ is generated, the replacement procedure is performed using a predefined scalarizing function g (lines 17–20). First, an index j is randomly selected from $\boldsymbol{R}$, and j is removed from $\boldsymbol{R}$ (line 18). Then, the individual $\boldsymbol{x}^j$ is compared with the child $\boldsymbol{u}^i$ based on g and the weight vector $\boldsymbol{w}^j$ (line 19). If $\boldsymbol{u}^i$ is better than $\boldsymbol{x}^j$ according to their scalarizing function values, then $\boldsymbol{x}^j$ is replaced with $\boldsymbol{u}^i$ (line 20). Unlike the original MOEA/D proposed in 2007 [43], the number of individuals replaced by the child is limited to $n^{\mathrm{rep}} > 0$ in MOEA/D-DE (line 17). c is used to count the number of individuals replaced by $\boldsymbol{u}^i$. When c reaches the maximum number of replacements n^{rep}, the replacement procedure terminates (line 17).

Since the replacement criterion is based on the scalarizing function g (line 19), the performance of MOEA/D-type algorithms significantly depends on g [18]. Although there are a number of scalarizing functions, as reviewed in [31], the following Tchebycheff function (g^{tch}) [43] is used in MOEA/D-DE:

$$g^{\mathrm{tch}}(\boldsymbol{x}|\boldsymbol{w}, \boldsymbol{z}^*) = \max_{i \in \{1,...,M\}} \{w_i |f_i(\boldsymbol{x}) - z_i^*|\}, \tag{5}$$

where g^{tch} in (5) should be minimized. The $\boldsymbol{z}^* = (z_1^*, ..., z_M^*)^{\mathrm{T}}$ is the ideal point. Since obtaining the true ideal point $\boldsymbol{z}^*$ for a given MOP is difficult, its approximated point, which consists of the minimum function value for each objective f_i ($i \in \{1, ..., M\}$) found during the search process, is typically used.

3 A review of existing configurations of the DE mutation operator in MOEA/D-DE

Here, we review the existing configurations of the DE mutation operator in MOEA/D-DE. We also explain why many configurations exist in the source code of MOEA/D-DE. Table 1 shows 15 configurations of the DE mutation operator in the source code of MOEA/D-DE and its variants. We downloaded the source code from each website in Table 1 and carefully checked how the differential mutation was implemented. The components of the DE mutation described here include two mutation strategies, three index selection methods for parent individuals, and five bound-handling methods. These components are explained in Subsections 3.1, 3.2, and 3.3, respectively.

3.1 Two DE mutation strategies in MOEA/D-DE

For each target individual $\boldsymbol{x}^i$ ($i \in \{1, ..., \mu\}$) in the population $\boldsymbol{P}$, MOEA/D-DE performs the differential mutation to generate a mutant vector $\boldsymbol{v}^i$ (line 11 in Algorithm 1). A number of mutation strategies exist in DE [8,9]. Among these strategies, the following rand/1 strategy is the most basic mutation strategy:

$$\boldsymbol{v}^i = \boldsymbol{x}^{r_1} + F\left(\boldsymbol{x}^{r_2} - \boldsymbol{x}^{r_3}\right), \tag{6}$$

where r_1, r_2, and r_3 are randomly selected from a set of individual indices $\boldsymbol{R}$ (see lines 6–9 in Algorithm 1) according to the index selection method, which is explained in Subsection 3.2 later. The rand/1 mutation strategy is widely used in DE algorithms for single-objective optimization and multi-objective optimization. For example, the rand/1 operator is incorporated into representative multi-objective DE algorithms, such as GDE3 [23] and DEMO [33].

Although the rand/1 operator is commonly used in single- and multi-objective DE, the current/1 mutation strategy is incorporated into MOEA/D-DE. Note that the term "current/1" is introduced in [14] and is not used in the original MOEA/D-DE paper [25]. The current/1 mutation strategy is defined as follows:

$$\boldsymbol{v}^i = \boldsymbol{x}^i + F\left(\boldsymbol{x}^{r_1} - \boldsymbol{x}^{r_2}\right), \tag{7}$$

where r_1 and r_2 are randomly selected from $\boldsymbol{R}$. Although the base vector is a randomly selected individual from $\boldsymbol{R}$ in the rand/1 strategy, it is always identical to the target individual $\boldsymbol{x}^i$ in the current/1 strategy. Compared to the rand/1 strategy, the mutant vector $\boldsymbol{v}^i$ generated by the current/1 strategy is generally close to $\boldsymbol{x}^i$ [14].

As mentioned above, MOEA/D-DE uses the current/1 strategy for the mutant vector generation. However, in the MOEA/D-DE paper [25], it is not explicitly stated that the current/1 mutation strategy in (7) is incorporated into MOEA/D-DE. Instead, in [25], it is

Table 1: Configurations of the DE mutation operator in the source code of MOEA/D-DE and its variants. Eight configuration numbers from #A to #H are used in Subsection 5.5.

MOEAs	Languages	Web sites	Mutation strategies	Index selection methods	Bound-handling methods	Configuration number #
MOEA/D-DE described in [25]			current/1	WR	reinitialization	#A
MOEA/D-DE	C++	MOEA/D homepage[a]	current/1	WPR	replacement	#B
MOEA/D-DE	MATLAB	MOEA/D homepage[a]	rand/1	WOR	replacement	#C
MOEA/D-DE	Java	MOEA/D homepage[a]	rand/1	WOR	replacement	#C
MOEA/D-DE MOEA/D-DRA	Java	jMetal 4.5[b]	current/1	WPR	replacement	#B
MOEA/D-DRA	Java	MOEA Framework[c]	rand/1	WPR	replacement	#D
MOEA/D-DE	C++	PaGMO[d]	current/1	WOR	r-reflection	#E
MOEA/D-DE	MATLAB	PlatEMO[e]	current/1	WOR	no method	-
MOEA/D-STM	Java	Li's website[f]	current/1	WPR	replacement	#B
MOEA/D-STM	MATLAB	Li's website[f]	current/1	WOR	replacement	#F
MOEA/D-DE	C	COCO website[g]	rand/1	WOR	r-reflection	#G
MOEA/D-DRA	C++	MOEA/D homepage[a]	current/1	WPR	r-reflection	#H
ENS-MOEA/D	MATLAB	Suganthan's website[h]	current/1	WPR	r-reflection	#H
MOEA/D-TPN	C	Yang's website[i]	current/1	WPR	r-reflection	#H

[a] `http://dces.essex.ac.uk/staff/zhang/webofmoead.htm`
[b] `http://jmetal.sourceforge.net/`
[c] `http://moeaframework.org/index.html`
[d] `https://esa.github.io/pagmo2/index.html`
[e] `http://bimk.ahu.edu.cn/index.php?s=/Index/Software/index.html`
[f] `http://www.cs.bham.ac.uk/~likw/publications.html`
[g] `http://coco.gforge.inria.fr/doku.php?id=mo-gecco2015`
[h] `http://www3.ntu.edu.sg/home/EPNSugan/index_files/`
[i] `http://www.tech.dmu.ac.uk/~syang/publications.html`

reported that the rand/1 strategy is used in MOEA/D-DE (equation (6) on page 9 in [25]), but the randomly chosen index r_1 is replaced by the target individual index i (Step 2.2 on page 9 in [25]). The modified rand/1 strategy is identical to the current/1 strategy.

The complicated description in [25] is somewhat difficult to understand. At first glance, one may think that the well-known rand/1 strategy is used in MOEA/D-DE. Unless readers have carefully read all the descriptions of the DE variation operator in MOEA/D-DE, then they will not notice that the current/1 strategy is actually used in MOEA/D-DE. In fact, the descriptions in [25] have confused some researchers. Consequently, the rand/1 strategy is incorporated into MOEA/D-DE in some source code. For example, as shown in Table 1, MOEA/D-DE in MOEA Framework and the source code of [6] use the rand/1 strategy, rather than the current/1 strategy. Additionally, some authors describe that the rand/1 mutation strategy was used in MOEA/D-DE-type algorithms in their articles, but the current/1 strategy was actually used in their source code. MOEA/D-TPN [21] in Table 1 is such an example. Since the rand/1 and current/1 strategies are essentially different, it is expected that the performance of MOEA/D-DE with the two strategies is different.

3.2 Index selection methods for the parent individuals

Here, we explain three index selection methods (WOR, WR, and WPR methods) for the parent individuals in the DE mutation operator. Algorithms 2, 3, and 4 show the procedures of the WOR, WR, and WPR methods for selecting two individual indices r_1 and r_2 for the current/1 mutation strategy. In Algorithm 2 (WOR), r_1 and r_2 are randomly selected such that $r_1 \neq r_2$, $r_1 \neq i$, and $r_2 \neq i$. In contrast, i, r_1, and r_2 can be the same index (i.e., $r_1 = r_2 = i$) in Algorithm 3 (WR). In Algorithm 4 (WPR), r_1 and r_2 differ from each other,

```
1  void CMOEAD::matingselection(vector<int> &list, int cid, int size, int type){
2    // list : the set of the indexes of selected mating parents
3    // cid  : the id of current subproblem
4    // size : the number of selected mating parents
5    // type : 1 - neighborhood; otherwise - whole population
6    int ss   = population[cid].table.size(), r, p;
7    while(list.size()<size)
8      {
9        if(type==1){
10         r = int(ss*rnd_uni(&rnd_uni_init));
11         p = population[cid].table[r];
12       }
13       else
14         p = int(population.size()*rnd_uni(&rnd_uni_init));
15
16       bool flag = true;
17       for(int i=0; i<list.size(); i++)
18         {
19           if(list[i]==p) // p is in the list
20             {
21               flag = false;
22               break;
23             }
24         }
25
26       if(flag) list.push_back(p);
27     }
28 }
```

Fig. 1: The "matingselection" function in the original C++ code of MOEA/D-DE, which is available on the authors' website. To implement the WOR selection method, line 19 should have been "if(list[i]==p && cid==p)".

Algorithm 2: The WOR selection method.

do
  Randomly select r_1 from $\boldsymbol{R}$;
while $r_1 = i$;
do
  Randomly select r_2 from $\boldsymbol{R}$;
while $r_2 = i$ and $r_2 = r_1$;

Algorithm 3: The WR selection method.

Randomly select r_1 from $\boldsymbol{R}$;
Randomly select r_2 from $\boldsymbol{R}$;

Algorithm 4: The WPR selection method.

Randomly select r_1 from $\boldsymbol{R}$;
do
  Randomly select r_2 from $\boldsymbol{R}$;
while $r_2 = r_1$;

but either of them can be equal to i. In the following, the three index selection methods are described in detail.

In the original DE paper [35], the parent individual indices r_1, r_2, and r_3 for the rand/1 strategy are randomly selected from $\{1, ..., \mu\}\backslash\{i\}$ such that they differ from each other (i.e., $r_1 \neq r_2$, $r_1 \neq r_3$, $r_2 \neq r_3$, and $r_j \neq i$ for $j \in \{1, 2, 3\}$). That is, individual indices are selected without replacement. In this paper, the traditional index selection method in DE is called a WOR method (Algorithm 2). Although the WOR selection method is generally used in DE, such a restriction is not described in the original MOEA/D-DE paper [25]. In this case, the selected indices are possibly equal to each other and also equal to i. The method that selects individual indices with replacement is called a WR method (Algorithm 3).

However, in the original C++ code of MOEA/D-DE, r_1 and r_2 for the current/1 strategy are selected from a set of individual indices $\boldsymbol{R}$ (see Subsection 2.2) such that they differ from each other, but there is no restriction about i. Figure 1 shows the `matingselection` function in the original C++ source code of MOEA/D-DE, which is available on the authors' website. The `matingselection` function is for selecting indices for the differential mutation. We do not explain the procedure of the `matingselection` function in detail, but there is a bug in line 19 in Figure 1. The one-dimensional vector `list` is for storing indices $\{r_1, r_2, ...\}$ that have been selected in the `matingselection` function. In line 19, `p` is a randomly selected candidate that is able to enter `list`. Since it is checked whether `p` exists in `list`, `p` may equal `cid`, where `cid` is the index of the target sub-problem and identical to i in Algorithm 1. Thus, while it is ensured that r_1 differs from r_2, either of the two indices may equal i. The selection method with partial replacement used in the original MOEA/D-DE code is called a WPR method (Algorithm 4) in this pa-

per. The WPR selection method has never been found in the DE literature.

We carefully checked the index selection method in each source code in Table 1. As shown in Table 1, it appears that the WR method has hardly been used in MOEA/D-DE. The WPR selection method is used for MOEA/D-DE in some MOEA packages (e.g., jMetal and MOEA Framework), which may be because they were based on the original MOEA/D-DE source code. The WOR method is used in other source code (e.g., PaGMO and PlatEMO), which is because the WOR is the most standard index selection method in the DE community. If researchers attempt to implement MOEA/D-DE without referring to its original C++ source code, most of them are likely to incorporate the WOR method into MOEA/D-DE for this reason.

Wang et al. [39] investigate the impact of the index selection method on the performance of DE algorithms for single-objective optimization. Their results show that the performance of some DE algorithms can be significantly improved or degraded by replacing the traditional WOR method with the WR method. Since removing the restriction $r_1 \neq r_2$ increases the probability that a child is generated near the base vector, the use of the WR method makes the search more greedy [39]. Thus, it is expected that the performance of MOEA/D-DE is affected by the choice of index selection method.

3.3 Bound constraint-handling methods

It is not always ensured that the mutant vector $\boldsymbol{v}$ generated by the differential mutation is in the solution space $\mathbb{S} = \Pi_{j=1}^{D}[x_j^{\min}, x_j^{\max}]$. When an element of the mutant vector v_j is out of the bound ($j \in \{1, ..., D\}$), a bound-handling method must be applied to $\boldsymbol{v}$ such that $\boldsymbol{v} \in \mathbb{S}$ (line 13 in Algorithm 1). In our study, we consider the following five bound-handling methods (resampling, replacement, reinitialization, reflection, and r-reflection methods), which are commonly used in the DE literature:

• **Resampling**: The mutant vector $\boldsymbol{v}$ is repeatedly generated using the differential mutation with different parent individuals until all elements of $\boldsymbol{v}$ satisfy the bound constraints.

• **Replacement**: In the case where v_j is out of the bound, it is replaced with the corresponding minimum/-maximum value ($x_j^{\min}$ or $x_j^{\max}$) as (8):

$$v_j = \begin{cases} x_j^{\min} & \text{if } v_j < x_j^{\min} \\ x_j^{\max} & \text{if } v_j > x_j^{\max} \\ v_j & \text{otherwise} \end{cases}. \tag{8}$$

• **Reinitialization**: When $v_j \notin [x_j^{\min}, x_j^{\max}]$, it is reinitialized in the range $[x_j^{\min}, x_j^{\max}]$ as (9):

$$v_j = \begin{cases} (x_j^{\max} - x_j^{\min})\,\mathrm{rand}[0,1] + x_j^{\min} & \text{if } v_j < x_j^{\min} \\ & \text{or } v_j > x_j^{\max} \\ v_j & \text{otherwise} \end{cases}. \tag{9}$$

• **Reflection**: If v_j violates the j-th bound constraint, it is reflected as much as it exceeded the minimum/-maximum value as (11):

$$v_j = \begin{cases} x_j^{\min} + (x_j^{\min} - v_j) & \text{if } v_j < x_j^{\min} \\ x_j^{\max} + (x_j^{\max} - v_j) & \text{if } v_j > x_j^{\max} \\ v_j & \text{otherwise} \end{cases}. \tag{10}$$

• **Randomized reflection (r-reflection)**: This method is an alternative version of the reflection method. Unlike the reflection method, the amount of reflection is randomly determined as (11):

$$v_j = \begin{cases} x_j^{\min} + \mathrm{rand}[0,1]\,(x_j^{\min} - v_j) & \text{if } v_j < x_j^{\min} \\ x_j^{\max} + \mathrm{rand}[0,1]\,(x_j^{\max} - v_j) & \text{if } v_j > x_j^{\max} \\ v_j & \text{otherwise} \end{cases}. \tag{11}$$

Table 1 shows that the replacement method is used in the original C++ source code and most source code for MOEA/D-DE. In addition to the replacement method, the r-reflection method is used in some implementations, as shown in Table 1. Table 1 indicates that the reinitialization, reflection, and resampling methods have not been incorporated into MOEA/D-DE. Although it is described in [25] that the reinitialization method is used as the repair method (Step 2.3 on page 9 in [25]), the replacement strategy is actually incorporated into the original C++ source code. While the reflection method is one of the most popular repair methods in DE algorithms for single-objective optimization, it has not been used in MOEA/D-DE. It was shown in [1] that the resampling method is most suitable for DE for single-objective optimization. However, the resampling method has not received much attention in the evolutionary multi-objective optimization community.

The use of a repair operator in evolutionary algorithms, including DE algorithms, influences their performance [1,15,16,41]. The impact of the bound-handling method on the basic DE algorithm is investigated in [1]. The results reported in [1] indicate that the choice of repair method significantly influences the performance of the basic DE [35] for single-objective continuous optimization. Thus, it is expected that the performance of MOEA/D-DE also depends on the choice of bound-handling method.

Table 2: Properties of the DTLZ and WFG test problems. "Mult." and "Nonsep." represent the multimodality and the nonseparability, respectively.

Problem	Pareto front	Mult.	Nonsep.	Others
DTLZ1	Linear	✓		
DTLZ2	Nonconvex			
DTLZ3	Nonconvex	✓		
DTLZ4	Nonconvex			Biased
DTLZ5	Partially degenerate			
DTLZ6	Partially degenerate			
DTLZ7	Disconnected	✓		
WFG1	Mixed			Biased
WFG2	Disconnected	✓	✓	
WFG3	Partially degenerate		✓	
WFG4	Nonconvex	✓		
WFG5	Nonconvex			Deceptive
WFG6	Nonconvex		✓	
WFG7	Nonconvex			Biased
WFG8	Nonconvex		✓	Biased
WFG9	Nonconvex	✓	✓	Deceptive Biased

4 Experimental settings

This section describes our experimental settings. Results are reported in Section 5.

4.1 Test problems

The seven DTLZ [13] and nine WFG [17] test problems with $M \in \{2, 3, 4, 5\}$ were used in our study. Table 2 summarizes their properties. Note that the Pareto fronts of the DTLZ5, DTLZ6, and WFG3 problems are partially degenerate [19]. According to [13], for the DTLZ problems, the number of position variables k was set to $k = 5$ for the DTLZ1 problem, $k = 20$ for the DTLZ7 problem, and $k = 10$ for the other DTLZ problems, where the number of variables is $D = M + k - 1$. Additionally, as suggested in [17], for the WFG test problems, the number of position variables k was set to $k = 2\,(M - 1)$, and the number of distance variables l was set to $l = 20$, where $D = k + l$.

4.2 Performance indicators

The hypervolume indicator [47] was used for evaluating the quality of a set of obtained nondominated solutions $\boldsymbol{A}$. Before calculating the hypervolume value, the objective vector $\boldsymbol{f}(\boldsymbol{x})$ of each $\boldsymbol{x} \in \boldsymbol{A}$ was normalized using the ideal point $\boldsymbol{z}^{\text{ideal}} = (z_1^{\text{ideal}}, ..., z_M^{\text{ideal}})^{\text{T}}$ and the nadir point $\boldsymbol{z}^{\text{nadir}} = (z_1^{\text{nadir}}, ..., z_M^{\text{nadir}})^{\text{T}}$ of each problem:

$$f_i^{\text{normalized}}(\boldsymbol{x}) = \frac{f_i(\boldsymbol{x}) - z_i^{\text{ideal}}}{z_i^{\text{nadir}} - z_i^{\text{ideal}}}, \tag{12}$$

where $f_i^{\text{normalized}}(\boldsymbol{x})$ is the i-th normalized objective value ($i \in \{1, ..., M\}$). The i-th element z_i^{ideal} of $\boldsymbol{z}^{\text{ideal}}$ is the minimum value of the i-th objective function of the true Pareto front. Conversely, the i-th element z_i^{nadir} of $\boldsymbol{z}^{\text{nadir}}$ is the maximum value of the i-th objective function of the true Pareto front. That is, the objective space of each test problem is normalized such that the ideal point and the nadir point in the normalized objective space are $(0, 0, ..., 0)^{\text{T}}$ and $(1, 1, ..., 1)^{\text{T}}$, respectively. According to [20], the reference point for the hypervolume calculation was set to $(1.1, ..., 1.1)^{\text{T}}$. In this setting, the hypervolume value is in the range $[0, 1.1^M]$.

The average performance score (APS) [3] was used to aggregate the results on various problems. Suppose that n algorithms $A_1, ..., A_n$ are compared for a given problem instance based on the hypervolume values obtained in multiple runs. For each $i \in \{1, ..., n\}$ and $j \in \{1, ..., n\} \setminus \{i\}$, if A_j significantly outperforms A_i using the Wilcoxon rank-sum test with $p < 0.05$, then $\delta_{i,j} = 1$; otherwise, $\delta_{i,j} = 0$. The performance score $P(A_i)$ is defined as follows: $P(A_i) = \sum_{j \in \{1,...,n\} \setminus \{i\}}^{n} \delta_{i,j}$. The score $P(A_i)$ represents the number of algorithms that outperform A_i. The APS value of A_i is the average of the $P(A_i)$ values for all the considered problem instances. In other words, the APS value of A_i represents how good (relatively) the performance of A_i is among the n algorithms on average over all problem instances. A small APS value indicates that the performance of the target algorithm is better than other compared algorithms.

Table 3: Example of the APS calculation.

Problem	A_1	A_2	A_3
I_1	2	1	0
I_2	0	0	2
I_3	1	0	1
I_4	2	0	1
APS	1.25 (5/4)	0.25 (1/4)	1 (4/4)

Table 3 shows a simple example of the APS calculation. Let us consider the APS values of three algorithms A_1, A_2, and A_3 on four problems I_1, I_2, I_3, and I_4. Each element of Table 3 shows a performance score value of the corresponding algorithm. The performance score value represents the number of algorithms that outperform the corresponding algorithm. For example,

Table 4: Overall performance of MOEA/D-DE with the three index selection methods on the 16 problems. The APS value at the final iteration is shown in the tables (lower is better). The numbers in parentheses indicate the ranks of the three configurations based on their APS values.

(a) current/1

M	WOR	WR	WPR
2	0.88 (3)	0.12 (2)	**0.06** (1)
3	0.56 (3)	**0.06** (1)	0.31 (2)
4	**0.25** (1)	**0.25** (1)	0.31 (3)
5	0.38 (3)	**0.00** (1)	0.12 (2)

(b) rand/1

M	WOR	WR	WPR
2	0.56 (2)	**0.25** (1)	0.62 (3)
3	0.25 (2)	**0.06** (1)	0.44 (3)
4	**0.06** (1)	0.31 (3)	**0.06** (1)
5	**0.06** (1)	0.12 (2)	0.31 (3)

A_1 is outperformed by two algorithms on I_1 and I_4. A_3 is not outperformed by any other algorithms on I_1. The APS value for each algorithm is calculated by dividing the total performance score value (e.g., 5 for A_1) by the number of problems (i.e., 4 for all algorithms). In this example, the APS values of A_1, A_2, and A_3 are 1.25, 0.25, and 1, respectively. It can be concluded that A_2 is the best performer according to the calculated APS values.

4.3 Control parameters of MOEA/D-DE

We used the source code of MOEA/D-DE downloaded from the jMetal website for our experiments. All the control parameters (except for the population size μ) for MOEA/D-DE were set according to the original study [25]. For the DE operators, the scale factor F and the crossover rate C were set to 0.5 and 1, respectively. For the polynomial mutation, p^{mut} and η^{mut} were set to $1/D$ and 20, respectively. The other parameters of MOEA/D-DE were set as follows: $T = 20$, $n^{\text{rep}} = 2$, and $\delta = 0.9$. The population size μ was set to 200, 210, 220, and 210 for $M = 2, 3, 4$, and 5, respectively. The simple normalization strategy described in [43] was introduced into MOEA/D-DE to handle differently scaled objective function values. The maximum number of function evaluations was set to 100 000 for all test problems, and 51 independent runs were performed.

Table 5: Overall performance of MOEA/D-DE with the two mutation strategies on the 16 problems. The APS value at the final iteration is shown in the tables (lower is better). The numbers in parentheses indicate the ranks of the three configurations based on their APS values.

M	current/1	rand/1
2	**0.12** (1)	0.62 (2)
3	**0.00** (1)	0.50 (2)
4	**0.06** (1)	0.38 (2)
5	**0.06** (1)	0.19 (2)

5 Experimental results

This section describes results of MOEA/D-DE with various configurations of the DE mutation operator. First, Subsection 5.1 analyzes the effect of the index selection method in MOEA/D-DE. Next, Subsection 5.2 investigates the influence of the mutation strategy on MOEA/D-DE. Subsection 5.3 presents a comparison of the five bound-handling methods. Subsection 5.4 examines which configuration of the DE mutation operator is the most suitable for MOEA/D-DE. We examine the effectiveness of a total of 30 configurations of three index selection methods (WOR, WR, and WPR), two mutation strategies (rand/1 and current/1), and five bound-handling methods (resampling, replacement, reinitialization, reflection, and r-reflection). Finally, Subsection 5.5 compares the eight existing configurations (#A, ..., #H) shown in Table 1.

5.1 The impact of the index selection method on the performance of MOEA/D-DE

Table 4 shows the overall performance of MOEA/D-DE with the three index selection methods (the WOR, WR, and WPR methods) on the 16 problems with $M \in \{2, 3, 4, 5\}$. Tables 4(a) and (b) show the results of MOEA/D-DE with the current/1 and rand/1 strategies, respectively. We do not present the comparison results of each problem here due to space constraints, but they can be found in Tables S.1 and S.2 in the supplementary file of this paper.

Table 4(a) shows that the best performance of MOEA/D-DE with the current/1 strategy is obtained by using the WPR selection method for $M = 2$. Although MOEA/D-DE with the WOR and WR methods achieve the same APS value for $M = 4$, the WR method is the best selection method for $M \in \{3, 4, 5\}$.

Table 4(b) indicates that the WR selection method is the most suitable for MOEA/D-DE using the rand/1 strategy for $M \in \{2, 3\}$. In contrast to the results for $M \in \{2, 3\}$, MOEA/D-DE with the WR method per-

Table 6: Overall performance of MOEA/D-DE with the five bound-handling methods on the 16 problems. The APS value at the final iteration is shown in the tables (lower is better). The numbers in parentheses indicate the ranks of the five configurations.

M	replacement	reinitialization	reflection	r-reflection	resampling
2	**0.69** (1)	2.12 (5)	1.38 (3)	1.19 (2)	1.69 (4)
3	**0.62** (1)	1.56 (3)	2.31 (5)	1.88 (4)	0.81 (2)
4	**0.62** (1)	1.69 (3)	2.12 (5)	1.75 (4)	1.06 (2)
5	**0.81** (1)	1.62 (3)	2.31 (5)	1.94 (4)	1.19 (2)

forms poorly for $M \in \{4, 5\}$. Although MOEA/D-DE with the WOR and WPR methods show the same APS value for $M = 4$, the WOR method is the best selection method for $M = 5$.

In summary, our results indicate that the choice of index selection method significantly affects the performance of MOEA/D-DE. As described above, the best index selection method depends on the type of mutation strategies used in MOEA/D-DE and on the number of objectives M. In addition, the most suitable index selection method also depends on the type of problems (Tables S.1 and S.2). However, on most problems, the WR and WOR methods are likely to be suitable for the current/1 and rand/1 mutation strategies, respectively.

5.2 The rand/1 and current/1 mutation strategies, which is better for MOEA/D-DE?

Table 5 shows the overall performance of MOEA/D-DE with the current/1 and rand/1 mutation strategies on the 16 problems with $M \in \{2, 3, 4, 5\}$. Based on the results in Subsection 5.1, the WR and WOR methods were used for the current/1 and rand/1 mutation strategies, respectively. Results on each problem are presented in Table S.3 in the supplementary file.

Table 5 shows that the current/1 strategy is more suitable for MOEA/D-DE than the rand/1 strategy for all M. As discussed in [14], the mutant vector $\boldsymbol{v}^i$ generated by the current/1 strategy is generally close to the target individual $\boldsymbol{x}^i$, and the neighborhood region of $\boldsymbol{x}^i$ is efficiently exploited, which is the reason why MOEA/D-DE with the current/1 strategy performs well. Although the rand/1 mutation strategy is incorporated into some packages, as reviewed in Subsection 3.1, Table 5 suggests that the current/1 strategy is more appropriate for MOEA/D-DE.

5.3 The influence of the bound-handling method on the performance of MOEA/D-DE

Table 6 shows the overall performance of MOEA/D-DE with the five bound-handling methods on the 16 problems. Results on each MOP can be found in Table S.4 in the supplementary file.

For all M, the best results are obtained by using the replacement method, followed by the resampling method. For $M \geq 3$, MOEA/D-DE with the reflection method achieves the worst APS values. Although the r-reflection method is used in some implementations of MOEA/D-DE, as reviewed in Subsection 3.3, its APS value is poor for $M \geq 3$. In summary, our results show that the replacement method is the most suitable bound-handling method of the DE mutation operator in MOEA/D-DE.

The results reported in [1] show that the resampling method is the most appropriate for the basic DE [35] for single-objective continuous optimization. In contrast to the results presented in [1], MOEA/D-DE with the resampling method performs the second best on MOPs. Thus, the most suitable bound-handling method for the DE mutation operator depends on the problem domain.

5.4 Comparison of all 30 configurations

We separately examined the effect of the three components of the DE mutation operator in Subsections 5.1, 5.2, and 5.3. The results in the three subsections show that the WR method, the current/1 strategy, and the replacement method are suitable in MOEA/D-DE. However, it is still unclear whether MOEA/D-DE with a combination of the three well-performing elements works well. To determine whether it performs well, this section presents comparisons of MOEA/D-DE with a total of 30 configurations (the three index selection methods, the two mutation strategies, and the five bound-handling methods) of the DE mutation operator.

First, we show the results on the 16 test problems in Subsection 5.4.1. Then, we report the performance of the 30 configurations on each problem type in Subsection 5.4.2.

5.4.1 Results on the 16 test problems

Table 7 shows the overall performance of MOEA/D-DE with all 30 configurations on the 16 problems. Below,

Table 7: Overall performance of MOEA/D-DE with all 30 configurations on the 16 problems with $M \in \{2, 3, 4, 5\}$. The rank of each configuration based on its APS values at the final iteration is shown for each M. The average rank for all M is also reported.

Bound-handling methods	Mutation strategies	Index selection methods	Configuration number #	$M = 2$	$M = 3$	$M = 4$	$M = 5$	Avg. rank
replacement	current/1	WOR	#F	5	3	1	4	3.25
		WR		1	1	3	1	1.5
		WPR	#B	2	2	2	2	2.0
	rand/1	WOR	#C	11	6	5	2	6.0
		WR		4	4	6	4	4.5
		WPR	#D	12	7	4	6	7.25
reinitialization	current/1	WOR		20	20	22	16	19.5
		WR	#A	16	8	10	11	11.25
		WPR		15	16	18	14	15.75
	rand/1	WOR		29	26	27	19	25.25
		WR		22	14	16	15	16.75
		WPR		30	27	28	22	26.75
reflection	current/1	WOR		16	21	22	23	20.5
		WR		8	12	13	20	13.25
		WPR		7	18	21	21	16.75
	rand/1	WOR		28	29	29	30	29.0
		WR		19	22	24	24	22.25
		WPR		25	30	30	29	28.5
r-reflection	current/1	WOR	#E	18	17	14	28	19.25
		WR		3	11	11	17	10.5
		WPR	#H	6	15	19	26	16.5
	rand/1	WOR	#G	23	22	26	25	24.0
		WR		13	19	20	18	17.5
		WPR		24	24	25	27	25.0
resampling	current/1	WOR		14	10	8	8	10.0
		WR		10	5	7	7	7.25
		WPR		9	9	9	10	9.25
	rand/1	WOR		26	28	14	12	20.0
		WR		21	13	12	9	13.75
		WPR		27	25	17	13	20.5

(b, m, s)-MOEA/D-DE denotes an MOEA/D-DE using b, m, and s, where b, m, and s represent a bound-handling method, a mutation strategy, and an index selection method, respectively.

Table 7 indicates that the performance rank of MOEA /D-DE significantly depends on the configuration of the DE mutation operator. For example, (replacement, current/1, WPR)-MOEA/D-DE works well for any number of objectives, and its average rank is 2.0. In contrast, the worst configuration is (reflection, rand /1, WOR)-MOEA/D-DE, whose average rank is 29.0. Although this configuration is frequently incorporated into DE for single-objective continuous optimization (e.g., [34, 38]), our results show that it is unsuitable for MOEA /D-DE. As shown in Table 7, the best configuration depends on the number of objectives M. According to the average rank for all M, the best performance is obtained by (replacement, current/1, WR)-MOEA/D-DE. This configuration consists of the three well-performing components, as shown in Subsections 5.1 – 5.3.

5.4.2 Results on each problem type

Next, we discuss the performance of the 30 configurations on each problem type. According to Table 2, we classified the 16 test problems into the following four groups:

Unimodal problems DTLZ2, DTLZ4, DTLZ5, DTLZ6, WFG1, WFG3, WFG5, WFG6, WFG7, and WFG8

Table 8: Performance of MOEA/D-DE with all 30 configurations on the four problem sets (unimodal, multimodal, separable, and nonseparable problems).The average rank values for all $M \in \{2, 3, 4, 5\}$ are shown.

Bound-handling methods	Mutation strategies	Index selection methods	Configuration number #	Unimodal	Multimodal	Separable	Nonseparable
replacement	current/1	WOR	#F	3.25	7.0	3.5	9.5
		WR		1.25	4.25	1.0	12.5
		WPR	#B	2.25	6.5	2.25	9.5
	rand/1	WOR	#C	5.5	10.0	5.75	13.25
		WR		3.5	8.0	3.75	17.0
		WPR	#D	6.25	10.5	5.75	13.25
reinitialization	current/1	WOR		22.5	10.25	24.0	8.25
		WR	#A	13.0	5.75	13.5	2.5
		WPR		16.5	8.0	20.25	3.0
	rand/1	WOR		27.25	16.5	27.5	18.5
		WR		18.75	9.25	18.75	11.75
		WPR		28.75	16.75	29.0	16.75
reflection	current/1	WOR		20.25	21.0	20.75	16.5
		WR		12.5	20.5	11.0	20.75
		WPR		15.5	17.75	16.25	16.5
	rand/1	WOR		27.25	26.0	28.0	24.25
		WR		20.25	24.5	18.5	25.25
		WPR		28.0	26.75	27.5	25.25
r-reflection	current/1	WOR	#E	18.0	21.75	18.5	17.5
		WR		10.0	16.5	8.5	15.75
		WPR	#H	15.5	17.5	14.75	17.25
	rand/1	WOR	#G	22.75	23.0	23.75	21.75
		WR		15.0	23.0	14.0	20.5
		WPR		23.5	25.25	24.25	22.5
resampling	current/1	WOR		9.75	12.5	10.0	8.25
		WR		8.0	6.5	7.5	7.5
		WPR		9.5	10.25	10.0	8.5
	rand/1	WOR		19.75	19.25	19.75	17.5
		WR		13.75	17.0	13.25	17.0
		WPR		21.5	18.0	21.0	17.25

Multimodal problems DTLZ1, DTLZ3, DTLZ7, WFG2, WFG4, and WFG9

Separable problems DTLZ1, DTLZ2, DTLZ3, DTLZ4, DTLZ5, DTLZ6, DTLZ7, WFG1, WFG4, WFG5, and WFG7

Nonseparable problems WFG2, WFG3, WFG6, WFG8, and WFG9

Table 8 shows results of the 30 configurations on the four problem types. Each value in Table 8 is the average rank value of each configuration over all problems ($M \in \{2, 3, 4, 5\}$) in each problem type. On the one hand, Table 8 indicates that the performance rank of each configuration for the unimodal, multimodal, and separable problem sets is almost consistent with that for all 16 test problems in Table 7. The best average rank value is obtained by (replacement, current/1, WR)-MOEA/D-DE. In addition, (replacement, current/1, WPR)-MOEA/D-DE performs well. On the other hand, the results for the nonseparable problem set show that the reinitialization repair method is suitable for MOEA/D-DE. (reinitialization, current/1, WR)-MOEA/D-DE and (reinitialization, current/1, WPR)-MOEA/D-DE have good performance according to the average rank values. This result may be because MOEA/D-DE needs to keep the diversity of the population to find good solutions on nonseparable problems. Since the reinitialization method randomly generates an element of a child that is out of the bound, it can introduce diversity in the population. Although the best configuration also depends on the problem type, (replacement, current/1, WR)-MOEA/D-DE works well on a wide variety of problems.

5.5 Comparison between the eight existing configurations

As reviewed in Section 3, the configuration of the DE mutation operator differs depending on the source code. According to Table 1, eight configurations (#A, ..., #H) exist. Our results in Subsections 5.1–5.4 also show that the performance of MOEA/D-DE significantly depends on components of the mutation. Here, we discuss the performance rank of MOEA/D-DE with the eight existing configurations.

Table 7 shows that configuration #B has the best average rank value among the eight configurations. Configuration #F is the second-best performer. Although the average rank value of configuration #A is not good for all 16 problems, it is best for the nonseparable problem set, as discussed in Subsection 5.4.2 (see Table 8). In contrast, configurations #E, #G, and #H show poor performance. In particular, configuration #G performs the worst among the eight configurations (#A, ..., #H).

The above results indicate that an incorrect conclusion could be obtained depending on the source code of MOEA/D-DE used. For example, let us consider the experiment to reproduce the results of MOEA/D-DE in the original paper [25]. As shown in Table 1, configuration #B is used in the original C++ code of MOEA/D-DE. However, configuration #G is incorporated into the PaGMO library. If one uses the PaGMO implementation of MOEA/D-DE (#G) rather than its original C++ source code (#B), she/he is likely to obtain results that are worse than those reported in [25]. That is, the performance of MOEA/D-DE is underestimated in this case. To avoid such an undesirable comparison, the configuration of the DE mutation operator should receive careful attention.

6 Conclusion

We have reviewed the existing configurations of the DE mutation in MOEA/D-DE (Section 3) and examined the influence of each component on the performance of MOEA/D-DE (Section 5). The main contributions of this paper can be summarized as follows:

- The DE mutation consists of three components (the mutation strategy, the index selection method, and the bound-handling method). Our review reveals that the configuration of the DE mutation operator in MOEA/D-DE differs depending on the source code. We also explained why different configurations of the DE mutation operator are implemented in the source code of MOEA/D-DE.
- Although implementations of the DE mutation operator in MOEA/D-DE have not received much attention, our results shows that the performance of MOEA/D-DE significantly depends on the configuration of the DE mutation operator. Thus, it is necessary to carefully select the configuration of the DE mutation operator to maximize the performance of MOEA/D-DE. Our results show that the combination of the current/1 strategy, the WR method (or the WPR method), and the replacement method is most suitable for MOEA/D-DE.

As we discussed in Section 5, some of our results on MOPs are inconsistent with previous studies on single-objective optimization problems. For example, while a combination of the WOR method, the rand/1 strategy, and the reflection method is frequently used in DE algorithms for single-objective optimization, our results indicates that it is inappropriate for MOEA/D-DE. As far as we know, such observations have not been reported in the literature. One future research topic is to investigate why the best configuration of the DE mutation operator depends on the problem domain.

Another future research topic is to analyze the impact of configurations of the DE mutation operator on the performance of other DE-based MOEAs (e.g., DEMO [33] and GDE3[23]). We used the DTLZ and WFG problems to examine the performance of MOEA/D-DE. Comparison of various configurations of the DE mutation in MOEA/D-DE on other test problems (e.g., the CEC2017 problems [7]) remains as future work.

Acknowledgement

This work was supported by the Program for Guangdong Introducing Innovative and Entrepreneurial Teams (Grant No. 2017ZT07X386), Shenzhen Peacock Plan (Grant No. KQTD2016112514355531), the Science and Technology Innovation Committee Foundation of Shenzhen (Grant No. ZDSYS201703031748284), the Program for University Key Laboratory of Guangdong Province (Grant No. 2017KSYS008), and National Natural Science Foundation of China (Grant No. 61876075).

This is a pre-print of an article published in Soft Computing. The final authenticated version is available online at: `https://doi.org/10.1007/s00500-019-03842-6`.

Compliance with ethical standards

Conflict of interest

Ryoji Tanabe and Hisao Ishibuchi declare that they have no conflict of interest.

Ethical approval

This article does not contain any studies with human participants performed by any of the authors.

References

1. J. Arabas, A. Szczepankiewicz, and T. Wroniak. Experimental comparison of methods to handle boundary constraints in differential evolution. In *PPSN*, pages 411–420, 2010.
2. A. Auger, D. Brockhoff, N. Hansen, D. Tusar, T. Tusar, and T. Wagner. The Impact of Variation Operators on the Performance of SMS-EMOA on the Bi-objective BBOB-2016 Test Suite. In *GECCO*, pages 1225–1232, 2016.
3. J. Bader and E. Zitzler. HypE: An Algorithm for Fast Hypervolume-Based Many-Objective Optimization. *Evol. Comput.*, 19(1):45–76, 2011.
4. N. Beume, B. Naujoks, and M. T. M. Emmerich. SMS-EMOA: multiobjective selection based on dominated hypervolume. *EJOR*, 181(3):1653–1669, 2007.
5. L. C. T. Bezerra, M. López-Ibáñez, and T. Stützle. To DE or Not to DE? Multi-objective Differential Evolution Revisited from a Component-Wise Perspective. In *EMO*, pages 48–63, 2015.
6. D. Brockhoff, T. Tran, and N. Hansen. Benchmarking Numerical Multiobjective Optimizers Revisited. In *GECCO*, pages 639–646, 2015.
7. R. Cheng, M. Li, Y. Tian, X. Zhang, S. Yang, Y. Jin, and X. Yao. Benchmark Functions for the CEC'2017 Competition on Many-Objective Optimization. Technical report, Univ. of Birmingham, 2017.
8. S. Das, S. S. Mullick, and P. N. Suganthan. Recent advances in differential evolution - An updated survey. *Swarm and Evol. Comput.*, 27:1–30, 2016.
9. S. Das and P. N. Suganthan. Differential Evolution: A Survey of the State-of-the-Art. *IEEE TEVC*, 15(1):4–31, 2011.
10. K. Deb. *Multi-Objective Optimization Using Evolutionary Algorithms*. John Wiley & Sons, 2001.
11. K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE TEVC*, 6(2):182–197, 2002.
12. K. Deb and H. Jain. An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part I: solving problems with box constraints. *IEEE TEVC*, 18(4):577–601, 2014.
13. K. Deb, L. Thiele, M. Laumanns, and E. Zitzler. Scalable Test Problems for Evolutionary Multi-Objective Optimization. In *Evolutionary Multiobjective Optimization. Theoretical Advances and Applications*, pages 105–145. Springer, 2005.
14. W. Gong, Y. Wang, Z. Cai, and S. Yang. A Weighted Biobjective Transformation Technique for Locating Multiple Optimal Solutions of Nonlinear Equation Systems. *IEEE TEVC*, 21(5):697–713, 2017.
15. N. Hansen, A. S. P. Niederberger, L. Guzzella, and P. Koumoutsakos. A Method for Handling Uncertainty in Evolutionary Optimization With an Application to Feedback Control of Combustion. *IEEE TEVC*, 13(1):180–197, 2009.
16. S. Helwig, J. Branke, and S. Mostaghim. Experimental analysis of bound handling techniques in particle swarm optimization. *IEEE TEVC*, 17(2):259–271, 2013.
17. S. Huband, P. Hingston, L. Barone, and R. L. While. A review of multiobjective test problems and a scalable test problem toolkit. *IEEE TEVC*, 10(5):477–506, 2006.
18. H. Ishibuchi, N. Akedo, and Y. Nojima. Behavior of Multiobjective Evolutionary Algorithms on Many-Objective Knapsack Problems. *IEEE TEVC*, 19(2):264–283, 2015.
19. H. Ishibuchi, H. Masuda, and Y. Nojima. Pareto Fronts of Many-Objective Degenerate Test Problems. *IEEE TEVC*, 20(5):807–813, 2016.
20. H. Ishibuchi, Y. Setoguchi, H. Masuda, and Y. Nojima. Performance of Decomposition-Based Many-Objective Algorithms Strongly Depends on Pareto Front Shapes. *IEEE TEVC*, 21(2):169–190, 2017.
21. S. Jiang and S. Yang. An Improved Multiobjective Optimization Evolutionary Algorithm Based on Decomposition for Complex Pareto Fronts. *IEEE Trans. Cyber.*, 46(2):421–437, 2016.
22. J. B. Kollat, P. M. Reed, and R. M. Maxwell. Many-objective groundwater monitoring network design using bias-aware ensemble Kalman filtering, evolutionary optimization, and visual analytics. *Water Resources Research*, 47(2):1–18, 2011.
23. S. Kukkonen and J. Lampinen. GDE3: the third evolution step of generalized differential evolution. In *IEEE CEC*, pages 443–450, 2005.
24. M. N. Le, Y. Ong, S. Menzel, Y. Jin, and B. Sendhoff. Evolution by adapting surrogates. *Evol. Comput.*, 21(2):313–340, 2013.
25. H. Li and Q. Zhang. Multiobjective Optimization Problems With Complicated Pareto Sets, MOEA/D and NSGA-II. *IEEE TEVC*, 13(2):284–302, 2009.
26. K. Li, Á. Fialho, S. Kwong, and Q. Zhang. Adaptive Operator Selection With Bandits for a Multiobjective Evolutionary Algorithm Based on Decomposition. *IEEE TEVC*, 18(1):114–130, 2014.
27. K. Li, Q. Zhang, S. Kwong, M. Li, and R. Wang. Stable Matching-Based Selection in Evolutionary Multiobjective Optimization. *IEEE TEVC*, 18(6):909–923, 2014.
28. X. Liao, Q. Li, X. Yang, W. Zhang, and W. Li. Multiobjective optimization for crash safety design of vehicles using stepwise regression model. *Struct. Multidiscipl. Optim.*, 35(6):561–569, 2008.
29. E. Mezura-Montes, J. Velázquez-Reyes, and C. A. Coello Coello. A comparative study of differential evolution variants for global optimization. In *GECCO*, pages 485–492, 2006.
30. Y. S. Ong, P. B. Nair, and A. J. Keane. Evolutionary optimization of computationally expensive problems via surrogate modeling. *AIAA journal*, 41(4):687–696, 2003.
31. M. Pescador-Rojas, R. Hernández Gómez, El. Montero, N. Rojas-Morales, M. Cristina Riff, and C. A. Coello Coello. An Overview of Weighted and Unconstrained Scalarizing Functions. In *EMO*, pages 499–513, 2017.
32. A. K. Qin, V. L. Huang, and P. N. Suganthan. Differential evolution algorithm with strategy adaptation for global numerical optimization. *IEEE TEVC*, 13(2):398–417, 2009.
33. T. Robič and B. Filipič. DEMO: differential evolution for multiobjective optimization. In *EMO*, pages 520–533, 2005.
34. J. Rönkkönen, S. Kukkonen, and K. V. Price. Real-Parameter optimization with Differential Evolution. In *IEEE CEC*, pages 506–513, 2005.
35. R. Storn and K. Price. Differential Evolution - A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces. *J. Global Optimiz.*, 11(4):341–359, 1997.

36. A. Trivedi, D. Srinivasan, K. Sanyal, and A. Ghosh. A Survey of Multiobjective Evolutionary Algorithms Based on Decomposition. *IEEE TEVC*, 21(3):440–462, 2017.
37. T. Tusar and B. Filipic. Differential Evolution versus Genetic Algorithms in Multiobjective Optimization. In *EMO*, pages 257–271, 2007.
38. Y. Wang, Z. Cai, and Q. Zhang. Differential Evolution With Composite Trial Vector Generation Strategies and Control Parameters. *IEEE TEVC*, 15(1):55–66, 2011.
39. Y. Wang, Z. Liu, J. Li, H. Li, and J. Wang. On the selection of solutions for mutation in differential evolution. *Frontiers of Computer Science*, pages 1–19, 2016.
40. Z. Wang, Q. Zhang, A. Zhou, M. Gong, and L. Jiao. Adaptive Replacement Strategies for MOEA/D. *IEEE Trans. Cyber.*, 46(2):474–486, 2016.
41. S. Wessing. Repair methods for box constraints revisited. In *EvoApplications*, pages 469–478, 2013.
42. Y. Yuan, H. Xu, and B. Wang. An Experimental Investigation of Variation Operators in Reference-Point Based Many-Objective Optimization. In *GECCO*, pages 775–782, 2015.
43. Q. Zhang and H. Li. MOEA/D: A multiobjective evolutionary algorithm based on decomposition. *IEEE TEVC*, 11(6):712–731, 2007.
44. Q. Zhang, W. Liu, and H. Li. The performance of a new version of MOEA/D on CEC09 unconstrained MOP test instances. In *IEEE CEC*, pages 203–208, 2009.
45. E. Zitzler and S. Künzli. Indicator-based selection in multiobjective search. In *Proc. of PPSN*, pages 832–842, 2004.
46. E. Zitzler, M. Laumanns, and L. Thiele. SPEA2: Improving the Strength Pareto Evolutionary Algorithm. Technical report, ETHZ, 2001.
47. E. Zitzler, L. Thiele, M. Laumanns, C. M. Fonseca, and V. Grunert da Fonseca. Performance assessment of multiobjective optimizers: an analysis and review. *IEEE TEVC*, 7(2):117–132, 2003.