Journal of Statistical Software

MMMMMM YYYY, Volume VV, Issue II. *doi: 10.18637/jss.v000.i00*

The MOEADr Package – A Component-Based Framework for Multiobjective Evolutionary Algorithms Based on Decomposition

Felipe Campelo
Universidade Federal de Minas Gerais

Lucas S. Batista
Universidade Federal de Minas Gerais

Claus Aranha
University of Tsukuba

Abstract

Multiobjective Evolutionary Algorithms based on Decomposition (MOEA/D) represent a widely used class of population-based metaheuristics for the solution of multicriteria optimization problems. We introduce the **MOEADr** package, which offers many of these variants as instantiations of a component-oriented framework. This approach contributes for easier reproducibility of existing MOEA/D variants from the literature, as well as for faster development and testing of new composite algorithms. The package offers an standardized, modular implementation of MOEA/D based on this framework, which was designed aiming at providing researchers and practitioners with a standard way to discuss and express MOEA/D variants. In this paper we introduce the design principles behind the MOEADr package, as well as its current components. Three case studies are provided to illustrate the main aspects of the package.

Keywords: moead, multi-objective evolutionary algorithms, R, component-oriented design.

1. Introduction

Multiobjective Optimization Problems (MOPs) (Miettinen 1999) are problems in which multiple objective functions must be simultaneously optimized by the same set of parameters . MOPs are often characterized by a set of conflicting objective functions, which results in the existence of a set of optimal compromise solutions, instead of a single globally optimal one. In this way, multiobjective optimization algorithms are frequently characterized by their ability to find (representative samples of) this set of solutions with different compromises between the objective functions.

Multiobjective Evolutionary Algorithms based on Decomposition (MOEA/Ds), originally pro-

posed by Zhang and Li (2007), represent a widely used class of population-based metaheuristics for solving MOPs (Trivedi, Srinivasan, Sanyal, and Ghosh 2016). MOEA/Ds approach the problem by decomposing it into a number of single-objective subproblems, which are then solved in parallel by a set of candidate solutions commonly referred to as a *population*.

Based on this general concept, a number of variations and improvements have been proposed over the past decade (Trivedi *et al.* 2016). As is common in the general literature on evolutionary algorithms, many of these improvements are presented as monolithic entities, in which a fixed composition of operators and adjustments to the original algorithm is presented as a single, novel approach. Opposed to this monolithic approach, Bezerra, López-Ibáñez, and Stützle (2015) have argued towards a more modular design of evolutionary algorithms in general, where an optimizer is seen as a composition of multiple, specialized components. This *component-based* approach allows researchers to clearly identify the contribution of each component, and facilitates the automated generation of new variants based on existing components, e.g., for the solution of specific problem classes. This approach also allows users to more easily test and implement new components, streamlining the development of new ideas and the reproducibility of results.

In this context, we have developed a component-oriented framework for MOEA/Ds, in which modules can be easily added, removed, modified or recombined by either users or automated testing and tuning programs. In particular, we defined a *Variation Stack*, which allows a very flexible way to model many different combinations and use cases of variation, local search, and solution repair operators.

This framework is implemented as the **MOEADr** package, which contains not only the original MOEA/D components, but several components found in more recent variations, all of which were recast as instantiations of the proposed component-based framework. The package is implemented in the R language and environment for statistical computing, and has been published on the Comprehensive R Archive Network (CRAN), at `https://CRAN.R-project.org/package=MOEADr`. A development version is also available at the project repository, `http://github.com/fcampelo/MOEADr`.

This paper describes the package and its underlying framework, and introduces the concepts necessary for a user to sucessfully apply **MOEADr** to their research or application problem. First, we provide a short primer on multiobjective optimization, as well as a short review of the MOEA/D (Section 2). We then review existing implementations of the MOEA/D, as well as existing implementations of MOP solvers in the R ecosystem, and locate our contribution in this context (Section 3).

Following that, we detail the framework of the **MOEADr** package, and present the MOEA/D components that it currently implements (Sections 4 and 5). This serves as a reference to the main features of the package, as well as a starting point for future contributions.

Finally, in Section 6 we describe the basic usage of the package with three case studies: A basic example of solving a benchmark MOP using a traditional MOEA/D algorithm; an example of using the **MOEADr** framework in conjunction with an automated algorithm assembling/tuning method to generate a new algorithmic configuration for a specific problem class; and an example of adding a custom component to the framework. Readers who are not concerned with the theoretical underpinnings of the package can skip straight to this section.

2. Multiobjective Optimization Problems and the MOEA/D

With no loss of generality, we define a continuous MOP, subject to inequality and equality constraints, as

$$\begin{aligned} &\min_{\mathbf{x}} \ \mathbf{f}(\mathbf{x}) = \big(f_1(\mathbf{x}), \ldots, f_{n_f}(\mathbf{x})\big) \\ &\text{subject to: } \mathbf{x} \in \Omega, \end{aligned} \tag{1}$$

where n_f is the number of objective functions, $\mathbf{x} \in \mathbb{R}^{n_v}$ represents a candidate solution, n_v is the number of decision variables, $\mathbf{f}(\cdot) : \mathbb{R}^{n_v} \mapsto \mathbb{R}^{n_f}$ is a vector of objective functions, and $\Omega \subset \mathbb{R}^{n_v}$ is the feasible decision space, such that

$$\Omega = \{\mathbf{x} \in \mathbb{R}^{n_v} \mid g_i(\mathbf{x}) \leq 0 \ \forall i \wedge h_j(\mathbf{x}) = 0 \ \forall j\}, \tag{2}$$

where $g_i(\cdot) : \mathbb{R}^{n_v} \mapsto \mathbb{R}$, $i = 1, \ldots, n_g$ and $h_j(\cdot) : \mathbb{R}^{n_v} \mapsto \mathbb{R}$, $j = 1, \ldots, n_h$ represent the inequality and equality constraint functions, respectively. The image of the set Ω, denoted by $\mathbf{f}(\Omega)$, defines the set of attainable objective values.

Given two feasible solutions $\mathbf{x}_i$, $\mathbf{x}_j \in \Omega$, we say that $\mathbf{x}_i$ *Pareto-dominates* $\mathbf{x}_j$ (written as $\mathbf{f}(\mathbf{x}_i) \prec \mathbf{f}(\mathbf{x}_j)$ or, equivalently, $\mathbf{x}_i \prec \mathbf{x}_j$) iif $f_k(\mathbf{x}_i) \leq f_k(\mathbf{x}_j) \ \forall k \in \{1, \ldots, n_f\}$ and $\mathbf{f}(\mathbf{x}_i) \neq \mathbf{f}(\mathbf{x}_j)$. A solution $\mathbf{x}^* \in \Omega$ is considered *Pareto-optimal* if there exists no other solution $\mathbf{x} \in \Omega$ such that $\mathbf{f}(\mathbf{x}) \prec \mathbf{f}(\mathbf{x}^*)$. The set of all Pareto-optimal solutions is known as the *Pareto-optimal set*, defined as $\mathcal{PS} = \{\mathbf{x}^* \in \Omega \mid \nexists \mathbf{x} \in \Omega : \mathbf{f}(\mathbf{x}) \prec \mathbf{f}(\mathbf{x}^*)\}$. The image of this set is referred to as the *Pareto-optimal front*, $\mathcal{PF} = \{\mathbf{f}(\mathbf{x}^*) \mid \mathbf{x}^* \in \mathcal{PS}\}$.

A widely used way to solve MOP using classical optimization methods is to represent the MOP as an arbitrary number of scalar optimization problems, which are built through aggregation functions such as the Weighted Sum (WS) or Weighted Tchebycheff (WT) (Miettinen 1999) approaches. Each scalar optimization problem, generated by the aggregation function and a given weight vector, leads to problem in which the global optimum coincides with a particular Pareto-optimal solution of the original MOP. In this way, an estimate of the Pareto-optimal set can be obtained by solving a set of scalar aggregation functions. However, simply performing an *independent* optimization of these multiple scalar problems tends to result in difficulties for generating an adequate approximation of the Pareto set (Miettinen 1999; Deb 2001), particularly when a well-spread sample of the Pareto-optimal front is desired.

Deb (2001) and Coello Coello, Lamont, and van Veldhuizen (2007) note that the original MOP described in (1) can be solved through a multiobjective evolutionary algorithm (MOEA), which is a population-based approach that attempts to converge to an approximation of the Pareto-optimal set in a single run. This feature enables a continuous exchange of information between the estimated solutions, which is useful to promote a proper approximation of the Pareto-optimal set. Among the algorithms commonly employed for the solution of continuous MOPs, we focus here on the class of MOEAs based on the explicit decomposition of the multiobjective optimization problem, which are briefly introduced below.

2.1. Multiobjective Evolutionary Algorithms based on Decomposition

Multiobjective evolutionary algorithms based on decomposition (MOEA/Ds), originally proposed by Zhang and Li (2007), combine features of both MOEA approaches and classical scalarization approaches for tackling MOPs. In general, a MOEA/D decomposes a MOP, as

defined in (1), into a finite number of scalar optimization *subproblems*. Each subproblem is defined by a *weight vector*, which is used in an *scalar aggregation function* to calculate the utility value of any solution for that particular subproblem.

This set of subproblems is solved in parallel by iterating over a set of candidate solutions, commonly called the *population*. Aggregation functions and weight vectors are chosen so that (i) an optimal solution to a given subproblem is also Pareto-optimal for the original MOP; and (ii) the optimal solutions to the set of subproblems are well distributed in the space of objectives. It is thus expected that solving the subproblems provides a reasonable approximation of the Pareto-optimal front, regarding both convergence and diversity criteria.

Each subproblem has one *incumbent solution* that is directly associated to it, i.e., the population size is equal to the number of subproblems. At each iteration, one new *candidate solution* is generated for each subproblem, by the application of a sequence of *variation operators* to the existing population. This set of new solutions is compared against each incumbent solution based on their utility values on the respective subproblem. The best solution for each subproblem is maintained as its incumbent solution for the next iteration, following rules defined by a specific *update strategy*.

When generating new candidate solutions or comparing their performance against incumbent ones, the algorithm defines a *neighborhood* for each subproblem which limits the exchange of information between candidate solutions. This neighborhood provides a certain locality to the variation operators and update strategy, aiming to regulate the convergence speed and global exploration abilities of the algorithm.

Among the features that motivate the application of a MOEA/D to the solution of a MOP, Li and Zhang (2009) highlight a few that stand out in terms of their usefulness. First, it is generally simpler to handle objective value comparisons and, to a certain extent, diversity maintenance in a MOEA/D than in other MOEAs. This means that the MOEA/D is frequently able to return a set of reasonably well-spread (in the space of objectives) solutions, even when the number of subproblems is small or the number of objectives is high. Scaling techniques for attenuating the effects of objective functions with vastly different ranges are also easily incorporated into the MOEA/D structure, as are constraint-handling techniques.

Since its introduction, the MOEA/D framework has been the target of several investigations, mainly aimed at: i) improving its performance; ii) overcoming limitations of its components; and iii) adapting it for different classes of problems. For instance, studies on decomposition-based MOEAs have been carried out to investigate new decomposition approaches (Li, Deb, Zhang, and Kwong 2015a; Tan, Jiao, Li, and Wang 2012; Qi, Ma, Liu, Jiao, Sun, and Wu 2014; Giagkiozis, Purshouse, and Fleming 2014), aggregation functions (Wang, Zhang, and Guo 2013; Sato 2014; Ishibuchi, Sakane, Tsukamoto, and Nojima 2010), objective scaling strategies (Deb and Jain 2014; Singh, Isaacs, and Ray 2011), neighborhood assignment methods (Ishibuchi, Akedo, and Nojima 2013; Li, Zhang, Kwong, Li, and Wang 2014b; Li, Kwong, Zhang, and Deb 2015b), variation operators (Li and Zhang 2009; Tan *et al.* 2012; Li, Fialho, Kwong, and Zhang 2014a), and selection operators (Jiang and Yang 2016). Additionally, decomposition-based MOEAs have been investigated for constrained MOPs (Deb and Jain 2014; Cheng, Jin, Olhofer, and Sendhoff 2016), many-objective optimization problems (MaOPs) (Asafuddoula, Ray, and Sarker 2015; Li *et al.* 2015a), and incorporation of decision-maker preferences (Mohammadi, Omidvar, and Li 2012; Gong, Liu, Zhang, Jiao, and Zhang 2011; Pilát and Neruda 2015). A recent, comprehensive survey of MOEAs based on decom-

position has been organized by Trivedi *et al.* (2016).

Despite their specificities, we can characterize these different MOEA/D instantiations by defining the following component classes in the algorithm:

- The *decomposition strategy*, which determines how the weight vectors are calculated and, consequently, how the MOP gets decomposed.
- The *aggregation function*, which uses the weight vectors to generate single-objective subproblems to be solved.
- The *objective scaling strategy*, which defines how differences in the range of values of the objective functions are treated.
- The *neighborhood assignment strategy*, which determines the neighborhood relations among the subproblems. This strategy defines the locality of the exchange of information between candidate solutions when applying variation operators and updating strategies.
- A *Variation Stack*, composed of one or more *variation operators*, which generates new candidate solutions from the existing ones. Our general definition of a variation operator is a function that modifies a candidate solution in the space of decision variables, based on information about the problem structure and/or the current and past states of the population as a whole. Notice that this general definition also encompasses *repair operators* and *local search operators* as special cases of variation.
- The *update strategy*, which determines the candidate solutions to be maintained or discarded after each iteration, based on their utility values for specific subproblems and on their neighborhood relations.
- The *constraint handling method*, which defines how to treat points that violate problem constraints.
- The *termination criteria*, which determines when the algorithm stops the search and returns the set of solutions found.

Based on this decomposition of the algorithm into its individual components, it is possible to define a common framework from which specific MOEA/D variants can be instantiated. This framework is detailed in Section 4.

3. MOEA/D implementations

Several implementations of the original MOEA/D and some of its variants for continuous MOPs are available online, mainly in Java, C++ and Matlab.[1] All implementations mentioned here have their source codes readily available for download from the links listed in the references.

Table 1 summarizes these implementations. It is important to notice that there is no implementation native to R, and that while jMetal (Nebro, Durillo, and Vergne 2015) and the

[1]The *MOEA/D Homepage* conveniently aggregates these resources in a single list, available at https://sites.google.com/view/moead/home.

MOEA Framework (Hadka 2017) represent object-oriented frameworks with some component-oriented design, no MOEA/D program was found to provide the fully modular implementation of the MOEA/D as proposed in the **MOEADr** package.

Table 1: MOEA/D implementations for continuous MOPs.

Algorithm	Language(s)	Framework	Author(s)
Original MOEA/D	C++ Matlab	–	Li and Zhang (2006)
Original MOEA/D	Java	–	Liu (2006)
MOEA/D-DE	C++	–	Li and Zhang (2007)
MOEA/D-DRA	C++ Matlab	–	Zhang, Liu, and Li (2009a)
Original MOEA/D MOEA/D-DE MOEA/D-DRA MOEA/D-STM	Java	jMetal	Nebro *et al.* (2015)
Original MOEA/D	Java	MOEA Fr.	Hadka (2017)

The **MOEADr** package was motivated by a perceived need to facilitate not only the application of existing MOEA/D variants but also the development and investigation of new components, as well as the fast reproduction of newly published innovations based on a library of existing components, with minimal need for reimplementation.

Finally, it is worth mentioning that while there are no specific MOEA/D implementations native to R, a few packages implementing different, general-purpose algorithms for multiobjective optimization are available. Table 2 lists those which are readily available on CRAN.[2]

Table 2: Multiobjective optimization packages native to R.

Package	Algorithm(s)	Author(s)
goalprog	Goal programming	Novomestky (2008)
NSGA2r	NSGA-II	Tsou (2013)
mopsocd	MOPSO	Naval (2013)
mco	NSGA-II	Mersmann (2014)
GPareto	Gaussian Process	Binois and Picheny (2016)
moko	HEGO MEGO VMPF	Passos (2016)
ecr	NSGA-II SMS-EMOA AS-EMOA	Bossek (2017)

Most of those packages offer a closed, individual algorithm, with the exception of Jakob Bossek's **ecr** package, which offers a modular approach for instantiating evolutionary algorithms for both single and multiobjective optimization. For the latter class, the algorithmic

[2]Even though most are not listed under the *Optimization and Mathematical Programming Task View* (`https://CRAN.R-project.org/view=Optimization`) as of the writing of this article.

structure defined by this package lends itself easily for the implementation of dominance and indicator-based approaches, but not necessarily for decomposition-based algorithms such the MOEA/D. This is also the case of another recent initiative to define a common framework for defining multiobjective evolutionary algorithms (Bezerra, López-Ibañez, and Stützle 2016). To the authors' knowledge, there is no R package implementing specific MOEA/D algorithms, nor any component-based implementation that allows an easy instantiation of multiobjective evolutionary algorithms based on decomposition.

4. The MOEADr package

The **MOEADr** package is a R implementation of multiobjective evolutionary algorithms based on decomposition, including many MOEA/D variants found in the literature. Our goals in this package are three-fold:

- To provide a component-wise perspective on the MOEA/D family of algorithms, where each different algorithm exists as a configuration of common components;
- To be easily extensible, so that researchers and users can implement their own components, facilitating both applied uses and scientific inquiries in this field;
- To include a representative section of the existing literature in MOEA/D variants;

To achieve these goals, the implementation was guided by the following design decisions:

1. For the sake of uniformity in the implementation of each component, we define the main variables of the MOEA/D: the solution set, neighborhood set, subproblem weight set, subproblem utility value set and violation value set as matrices;
2. Algorithms in the MOEA/D family are broken down into common components, and each component is recast as an operator on the matrices defined above;
3. Each individual component avoids, to the maximum degree possible, to produce and rely on side effects beyond the explicit manipulation of the main variables above;
4. Each component is implemented as a separate R script file with a fixed naming scheme. The choice of components is specified as a parameter to the main function call;
5. In particular, we define a *Variation Stack*, which is a list of variation components to be used by the MOEA/D in order. The choice of variation operators shows a very large diversity in MOEA/D literature, and using a variation stack allows for a uniform description of the many existing configurations;

The general framework of the MOEA/D, as implemented in our package, is presented in Algorithm 1. The user defines an instance of this framework by choosing a specific component for each of the roles described in section 2.1, and one or more variation components to compose the variation stack $\mathcal{V}$.

In the following sections we detail each of these component classes, presenting a formal definition of their roles as well as relevant examples. The list of components described in this paper (and currently available in the **MOEADr** package) is summarized in Table 3.

Table 3: Components currently available in the `MOEADr` package

Component Role	**Name**	**User Parameters**	**Section**
Decomposition Method	SLD	$h \in \mathbb{Z}_{>0}$	5.1.1
	MSLD	$\mathbf{h} \in \mathbb{Z}_{>0}^{K}$; $\boldsymbol{\tau} \in (0,1]^{K}$	5.1.2
	Uniform	$N \in \mathbb{Z}_{>0}$	5.1.3
Scalar Aggregation Function	WS	–	5.2.1
	WT	–	5.2.2
	AWT	–	5.2.4
	PBI	$\theta^{pbi} \in \mathbb{R}_{>0}$	5.2.3
	iPBI	$\theta^{ipbi} \in \mathbb{R}_{>0}$	5.2.5
Objective Scaling	–	$type \in \{none;\ simple\}$	5.2
Neighborhood Assignment	–	$type \in \left\{by\ \boldsymbol{\lambda_i};\ by\ \mathbf{x}_i^{(t)}\right\}$	5.3
		$\delta_p \in [0,1]$	5.4
Variation Operators	SBX recombination	$\eta_{\mathtt{X}} \in \mathbb{R}_{>0}$; $p_{\mathtt{X}} \in [0,1]$	5.4.1
	Polynomial mutation	$\eta_{\mathtt{M}} \in \mathbb{R}_{>0}$; $p_{\mathtt{M}} \in [0,1]$	5.4.2
	Differential mutation	$\phi \in \mathbb{R}_{>0}$ $basis \in \{rand;\ mean;\ wgi\}$	5.4.3
	Binomial recombination	$\rho \in [0,1]$	5.4.4
	Truncation	–	5.4.5
	Local search	$type \in \{tpqa;\ dvls\}$ $\tau_{ls} \in \mathbb{Z}_{>0}$; $\gamma_{ls} \in [0,1]$ $\epsilon \in \mathbb{R}_{>0}$ (if $type = tpqa$)	5.4.6
Update Strategy	Standard	–	5.5.1
	Restricted	$n_r \in \mathbb{Z}_{>0}$	5.5.2
	Best	$n_r \in \mathbb{Z}_{>0}$; $T_r \in \mathbb{Z}_{>0}$	5.5.3
Constraint Handling	Penalty functions	$\beta_v \in \mathbb{R}_{>0}$	5.6.1
	VBR	$type \in \{ts;\ sr;\ vt\}$ $p_f \in [0,1]$ (if $type = sr$)	5.6.2
Termination Criteria	Evaluations	$max_{eval} \in \mathbb{Z}_{>0}$	5.7
	Iterations	$max_{iter} \in \mathbb{Z}_{>0}$	
	Time	$max_{time} \in \mathbb{R}_{>0}$	

Algorithm 1 Component-wise MOEA/D structure

Require: Objective functions $\mathbf{f}(\cdot)$; Constraint functions $\mathbf{g}(\cdot)$; Component-specific input parameters;

$t \leftarrow 0$
$run \leftarrow$ TRUE
Generate initial population $\mathbf{X}^{(t)}$ by random sampling from Ω.
Generate weights $\boldsymbol{\Lambda}$ ▷ Decomposition strategies (Sec. 5.1)
while *run* **do**
    Define or update neighborhoods $\boldsymbol{B}$ ▷ Neighborhood assignment strategies (Sec. 5.3)
    Copy incumbent solution set $\mathbf{X}^{(t)}$ into $\mathbf{X'}^{(t)}$
    for each variation operator $v \in \mathcal{V}$ **do**
        $\mathbf{X'}^{(t)} \leftarrow v(\mathbf{X'}^{(t)})$ ▷ Variation operators (Sec. 5.4)
    end for
    Evaluate solutions in $\mathbf{X}^{(t)}$ and $\mathbf{X'}^{(t)}$ ▷ Aggregation functions (Sec. 5.2)
        ▷ Constraint handling (Sec. 5.6)
    Define next population $\mathbf{X}^{(t+1)}$ ▷ Update strategies (Sec. 5.5)
    Update *run* flag ▷ Stop criteria (Sec. 5.7)
    $t \leftarrow t + 1$
end while
return $\mathbf{X}^{(\mathbf{t})}$; $\mathbf{f}\left(\mathbf{X}^{(t)}\right)$

5. MOEA/D components available in the MOEADr package

In this section we describe in detail each of the components included in the **MOEADr** package, version 2.1. We present a formal definition of their roles as well as relevant examples, both from the specific MOEA/D literature and the wider body of knowledge on multiobjective evolutionary algorithms in general. The complete list of roles and specific components is summarized in Table 3.

For each component, the different notations of existing works were recast to a standard mathematical notation, to prevent confusion and ambiguities that would inevitably arise if we tried to follow the many different nomenclatures from the literature. Whenever possible, our notation employs vector and matrix operations to describe the modules, in an attempt to highlight the mathematical structure of each component, as well as differences and similarities among variants.

Special care was taken to guarantee the modularity of the definitions provided in the following sections, so that each component is independent from design choices made for the others. This characteristic allows the free exchange of components while guaranteeing the correct flow of the MOEA/D, at the cost of some implementation overhead. This also simplifies the use of automated algorithm assembly and tuning methods, as well as efforts for replicating and testing algorithms from the literature.

To describe the components, let the following common variables be defined: n_f is the number of objective functions that compose the MOP. $N \in \mathbb{Z}_{>0}$ is the number of subproblems, which is either a user-defined parameter, or calculated internally by the decomposition strategy. $\mathbf{X}^{(t)} = \left\{\mathbf{x}_i^{(t)} \mid i = 1, \ldots, N\right\}$ denotes the set of incumbent solutions for the subproblems at

iteration t. With some abuse of notation, $\mathbf{X}^{(t)}$ will also denote a matrix with each row i defined by $\mathbf{x}_i^{(t)}$.[3] $\mathbf{\Lambda} \in \mathbb{R}_{\geq 0}^{N \times n_f}$ is the matrix of subproblem weights calculated by the decomposition strategy (Section 5.1), and $\mathbf{B} \in \mathbb{Z}_{>0}^{N \times T}$ is the matrix of neighborhood relations calculated by the neighborhood assignment strategy (Section 5.3), with T denoting the neighborhood size.

5.1. Decomposition strategies

A decomposition strategy is responsible for generating the weight vectors that characterize the set of scalar subproblems in a MOEA/D. The number of subproblems can be either explicitly defined, or calculated indirectly by the decomposition strategy based on user-defined parameters.

Methods for generating aggregation weight vectors in the structure of MOEA/D are usually borrowed from the theory of design and modeling in experiments with mixtures (Chan 2000). Among the main designs in this area, the simplex-lattice design (Scheffé 1958) has been the most commonly adopted in the context of decomposition-based MOEAs. This approach, as well as a multiple-layer variant are presented below. A third approach, based on uniform designs, is also described.

The following definitions hold for all strategies discussed in this section: let $\mathbf{\Lambda} \in \mathbb{R}_{\geq 0}^{N \times n_f}$ be a matrix of non-negative values. Also, let each row of $\mathbf{\Lambda}$ be a vector summing to unity, $\|\boldsymbol{\lambda}_i\|_1 = 1$. Each row $\boldsymbol{\lambda}_i$ can be interpreted as the *weight vector* defining the i-th subproblem, and the matrix $\mathbf{\Lambda}$ is referred to as the *weight matrix*. Since, given a scalar aggregation function (Sec. 5.2), each weight vector defines a unique subproblem, $\boldsymbol{\lambda}_i$ will also be used to refer to the i-th subproblem in the following sections.

In the **MOEADr** package, a list of available decomposition strategies can be generated using the function `get_decomposition_methods()`.

Simplex-Lattice Design (SLD)

In the simplex-lattice design (Chan 2000; Zhang and Li 2007) the user provides a parameter $h \in \mathbb{Z}_{>0}$ that defines both the number of subproblems and the values of the weights. In this method, elements of the weight matrix can only assume $h + 1$ distinct values

$$\lambda_{i,j} \in \left\{0, \frac{1}{h}, \frac{2}{h}, \ldots, 1\right\}, \ \forall\ i, j. \tag{3}$$

The simplex-lattice design generates N weight vectors by using all combinations (with repetition) of n_f values taken from the set of values defined in (3), which results in

$$N = \binom{h + n_f - 1}{n_f - 1} \tag{4}$$

subproblems defined when this strategy is used. In this way, a general (h, n_f)-simplex-lattice can be used to represent N weight vectors on the objective domain. For instance, for an arbitrary three-objective problem ($n_f = 3$), a value $h = 18$ yields $N = \binom{20}{2} = 190$ distinct weight vectors.

[3] Throughout this paper the distinction between the set and matrix interpretations will always be clear from the context. All other quantities defined as matrices will also be sometimes be treated as sets of vectors.

Multiple-Layer Simplex-Lattice Design (MSLD)

To obtain a reasonable distribution of weight vectors within the n_f-dimensional simplex using the simplex-lattice design it is necessary that $h \geq n_f$ (Deb and Jain 2014). While this condition is generally harmless for MOPs with few objectives, a large number of weight vectors is generated for high-dimensional objective domains, even in the limit condition $h = n_f$. On the other hand, making $h < n_f$ results in weight vectors sparsely distributed only at the boundary of the simplex, which jeopardizes the exploration abilities of the algorithm.

To address this issue, a multiple-layer approach can be devised. This strategy generates k subsets of weight vectors by generalizing the user parameter h to a vector of positive integers $\mathbf{h} = (h_1, \ldots, h_K)\,,\ h_k \in \mathbb{Z}_{>0}\ \forall k$. Each constant h_k is used to calculate N_k weight vectors according to the SLD method and, within each subset, the vectors are scaled down by a factor $\tau_k \in (0, 1]$ using a simple coordinate transformation

$$\boldsymbol{\lambda}'_{k_i} = \tau_k \boldsymbol{\lambda}_{k_i} + (1 - \tau_k)/n_f \tag{5}$$

with $\boldsymbol{\lambda}_{k_i}$ being the i-th vector in the k-th subset. Each subset must be associated with a unique user-defined value of τ_k.[4] The total number of subproblems defined in this method is given by $N = \sum_{k=1}^{K} N_k$, and the final weight matrix $\boldsymbol{\Lambda}$ is composed by the scaled weight vectors from all layers.

This method ultimately amounts to generating weight vectors in layers, with each layer corresponding to a different-sized simplex in the space of objectives. This approach trades the loss of exploration ability by the increased number of user-defined parameters (it requires the definition of $2K + 1$ parameter values). Common sense seems to suggest $\tau_k = k/K$ as a reasonable heuristic for defining the scaling factors, but there is currently neither empirical nor theoretical support for this choice.

Finally, it must be remarked that this approach to generating the weight vectors generalizes the method of Li *et al.* (2015a), which was defined specifically for $K = 2$.

Uniform Design (UD)

The uniform design method represents an alternative approach to generate the weight vectors. Its use was proposed by Tan *et al.* (2012), with the stated objectives of improving the distribution of the weight vectors and providing greater control over the number of subproblems.

The UD method for calculating the weight vectors can be described as follows. First, let H_N be defined as the set

$$H_N = \{h \in \mathbb{Z}_{>0} \mid h < N\ \wedge\ gcd\,(h, N) = 1\}\,, \tag{6}$$

where $gcd\,(\cdot)$ returns the greatest common divisor of two integers, and $N \in \mathbb{Z}_{>0}$ is a user-defined value that determines the number of subproblems. Let $\mathbf{h} = \left(h_1, \ldots, h_{n_f - 1}\right)$ be a vector composed of $n_f - 1$ mutually exclusive elements of H_N. Any such vector can define a matrix $\mathbf{U}_N(\mathbf{h}) \in (\mathbb{Z}_{>0})^{N \times (n_f - 1)}$ with elements calculated as $u_{ij} = (ih_j)\ mod\ N$. Denoting the set of all possible $\mathbf{h}$ vectors defined from H_N as $\mathcal{P}\,(H_N)$, the next step is to identify the

[4] $\tau_k = 1$ keeps the original coordinate system, while $\tau_k \to 0$ performs a maximum contraction of the weight vectors towards $\lambda_{i,j} = 1/n_f,\ \forall\ i, j$.

vector that results in the $\mathbf{U}_N(\mathbf{h})$ matrix with the lowest CD_2 discrepancy

$$\overline{\mathbf{h}} = \underset{\mathbf{h}\in\mathcal{P}(H_N)}{\arg\min}\, CD_2\left(\mathbf{U}_N(\mathbf{h})\right), \tag{7}$$

where $CD_2(\cdot)$ is the centered L_2-discrepancy of a matrix (Fang and Lin 2003; Tan *et al.* 2012), calculated as

$$\begin{aligned} CD_2(\mathbf{U}_N(\mathbf{h})) = \left(\frac{13}{12}\right)^{(n_f-1)} - \frac{2}{N}\sum_{i=1}^{N}\prod_{j=1}^{n_f-1}\left(1+\frac{|u_{i,j}-0.5|-|u_{i,j}-0.5|^2}{2}\right)+ \\ \frac{1}{N^2}\sum_{i=1}^{N}\sum_{k=1}^{N}\prod_{j=1}^{n_f-1}\left(1+\frac{|u_{i,j}-0.5|+|u_{k,j}-0.5|}{2}-\frac{|u_{i,j}-u_{k,j}|}{2}\right). \end{aligned} \tag{8}$$

If we define $\overline{\mathbf{U}}_N = \left(\mathbf{U}_N(\overline{\mathbf{h}}) - 0.5\right)/N$, then the elements of the weight matrix $\mathbf{\Lambda}$ are returned by the transformation

$$\begin{cases} \lambda_{i,j} = \left(1-\overline{u}_{i,j}^{\left(\frac{1}{n_f-j}\right)}\right)\prod_{k=1}^{j-1}\overline{u}_{i,k}^{\left(\frac{1}{n_f-k}\right)} & \text{, if } j < n_f \\ \lambda_{i,n_f} = \prod_{k=1}^{n_f-1}\overline{u}_{i,k}^{\left(\frac{1}{n_f-k}\right)}, \end{cases} \tag{9}$$

with the guarantee that $\mathbf{\Lambda}$ conforms with the properties stated at the beginning of section 5.1.

5.2. Scalar aggregation functions

The scalar aggregation function is used to calculate the utility value of candidate solutions for each subproblem. This is done by specific functions of the objective function values $\mathbf{f}(\mathbf{x})$ and the weight matrix $\mathbf{\Lambda}$.

In the **MOEADr** package, a list of available aggregation functions can be obtained using function `get_scalarization_methods()`.

Weighted Sum (WS)

This technique performs a convex combination of the objective values, resulting in scalar problems of the form

$$\min\ f^{ws}(\mathbf{x}\mid\boldsymbol{\lambda}_i,\mathbf{z}) = \boldsymbol{\lambda}_i\left(\mathbf{f}(\mathbf{x})-\mathbf{z}\right)^{\intercal}, \text{subject to: } \mathbf{x}\in\Omega \tag{10}$$

where $\mathbf{z} = (z_1,\ldots,z_{n_f})$ is a reference vector with the property that $\forall j\ z_j \leq \min\{f_j(\mathbf{x}) \mid \mathbf{x}\in\mathcal{PS}\}$. A diverse set of Pareto-optimal solutions can be achieved by using different weight vectors $\boldsymbol{\lambda}_i$. However, due to the convex nature of this operation, only convex sections of Pareto fronts can be approximated using this strategy (Miettinen 1999).

It is important to highlight here that obtaining a good estimate of $\mathbf{z}$ is by itself a computationally-intensive effort. In many cases, it is common practice to iteratively estimate this reference

point from the set of all points visited up to a given iteration, $\mathcal{X}^{(t')} \triangleq \bigcup_{t=1}^{(t')} \mathbf{X}^{(t)}$. In these cases the elements of the estimated reference vector, $\hat{\mathbf{z}}^{(t)}$, are defined as

$$\hat{z}_j^{(t')} = \min_{\mathbf{x} \in \mathcal{X}^{(t')}} f_j(\mathbf{x}), \quad j = 1, \ldots, n_f.$$

This estimated vector is updated at each iteration and is frequently employed instead of a fixed $\mathbf{z}$ in the scalar aggregation functions used with the MOEA/D.

Weighted Tchebycheff (WT)

In this approach, the scalar optimization problems are defined as

$$\min f^{wt}(\mathbf{x} \mid \boldsymbol{\lambda}_i, \mathbf{z}) = \|\boldsymbol{\lambda}_i \odot (\mathbf{f}(\mathbf{x}) - \mathbf{z})\|_\infty, \text{ subject to: } \mathbf{x} \in \Omega \tag{11}$$

where $\odot$ denotes the Hadamard product, and $\|\cdot\|_\infty$ is the Tchebycheff norm. Unlike the weighted sum approach, the weighted Tchebycheff approach is not sensitive to the convexity of the Pareto front (Miettinen 1999). However, this approach offers poor diversity control, as the solution of (11) for equally-spaced weight vectors do not necessarily translate to a well-spread approximation of the Pareto front (Qi *et al.* 2014; Wang *et al.* 2013).

Penalty-based Boundary Intersection (PBI)

The PBI aggregation function (Zhang and Li 2007) is an extension of an earlier approach known as normal boundary intersection (Das and Dennis 1998). The scalar optimization problems defined by the PBI strategy are given as

$$\min f^{pbi}(\mathbf{x} \mid \boldsymbol{\lambda}_i, \mathbf{z}) = d_{i,1} + \theta^{pbi} d_{i,2}, \text{ subject to: } \mathbf{x} \in \Omega \tag{12}$$

with

$$d_{i,1} = \frac{|(\mathbf{f}(\mathbf{x}) - \mathbf{z})\boldsymbol{\lambda}_i^\intercal|}{\|\boldsymbol{\lambda}_i\|_2}; d_{i,2} = \left\| \mathbf{f}(\mathbf{x}) - \left(\mathbf{z} + \frac{d_{i,1}\boldsymbol{\lambda}_i}{\|\boldsymbol{\lambda}_i\|_2} \right) \right\|_2,$$

where $\|\cdot\|_2$ denotes the Euclidean norm, and $\theta^{pbi} \in \mathbb{R}_{\geq 0}$ is a user-defined penalty parameter. Notice that $d_{i,1}$ is related to the convergence of $\mathbf{f}(\mathbf{x}_i)$ towards the Pareto-optimal front, whereas the minimization of $d_{i,2}$ provides a way to control solution diversity. This aggregation function enables the definition of a trade-off between convergence and diversity (in the space of objectives) by an *a priori* adjustment of θ^{pbi}, which directly influences the performance of the MOEA/D.

Adjusted Weighted Tchebycheff (AWT)

An alternative that attempts to address the poor diversity control of the weighted Tchebycheff approach is the adjusted (or transformed) Tchebycheff scalarization function (Qi *et al.* 2014; Wang *et al.* 2013). This method defines the scalar subproblems as:

$$\begin{aligned} &\min f^{awt}(\mathbf{x} \mid \boldsymbol{\lambda}_i, \mathbf{z}) = \|\boldsymbol{\rho}_i \odot (\mathbf{f}(\mathbf{x}) - \mathbf{z})\|_\infty \\ &\text{subject to: } \mathbf{x} \in \Omega, \end{aligned} \tag{13}$$

with the elements of the vector $\boldsymbol{\rho}_i$ defined as

$$\rho_{i,j} = \frac{(\lambda_{i,j} + \epsilon)^{-1}}{\sum_{j=1}^{n_f} (\lambda_{i,j} + \epsilon)^{-1}}, \quad j = 1, \ldots, n_f,$$

where ϵ is a small constant added to prevent divisions by zero.[5] Notice that the only difference between (11) and (13) is the substitution of the weight vector $\boldsymbol{\lambda}_i$ by its respective "normalized inverse" $\boldsymbol{\rho}_i$. Besides minimizing the distance between $\mathbf{f}(\mathbf{x})$ and $\mathbf{z}$, this transformation promotes the distance minimization between an objective vector $\mathbf{f}(\mathbf{x})$ and its corresponding vector $\boldsymbol{\rho}_i$ (Qi, Ma, Liu, Jiao, Sun, and Wu 2013). In this aspect, the adjusted weighted Tchebycheff strategy presents an idea similar to that of the PBI, but without any additional parameters.

Inverted PBI (iPBI)

While the PBI approach defines its search based on an ideal reference point $\mathbf{z}$ which represents an (estimated) ideal solution, the inverted PBI function uses a *nadir* reference point $\widetilde{\mathbf{z}}$, defined as $\forall k\ \widetilde{z}_k \geq \max\{f_k(\mathbf{x}) \mid \mathbf{x} \in \mathcal{PS}\}$ (Sato 2014, 2015).[6] This method works by defining the scalar minimization[7] problems

$$\begin{aligned} &\min\ f^{ipbi}(\mathbf{x} \mid \boldsymbol{\lambda}_i, \widetilde{\mathbf{z}}) = \theta^{ipbi} d_{i,2} - d_{i,1} \\ &\text{subject to: } \mathbf{x} \in \Omega \end{aligned} \tag{14}$$

with

$$\begin{aligned} d_{i,1} &= \frac{|(\widetilde{\mathbf{z}} - \mathbf{f}(\mathbf{x}))\, \boldsymbol{\lambda}_i^{\intercal}|}{\|\boldsymbol{\lambda}_i\|_2} \\ d_{i,2} &= \left\| \left(\widetilde{\mathbf{z}} - \mathbf{f}(\mathbf{x}) - d_{i,1} \frac{\boldsymbol{\lambda}_i}{\|\boldsymbol{\lambda}_i\|_2} \right) \right\|_2, \end{aligned}$$

where, similarly to the PBI approach, $\theta^{ipbi} \in \mathbb{R}_{\geq 0}$ is a user-defined parameter that controls the balance between $d_{i,1}$ (convergence) and $d_{i,2}$ (diversity).

Scaling of the objective domain

Since the range of objective functions in MOPs can present arbitrarily large differences, an appropriate scaling of the objective domain is sometimes used to improve the performance of the MOEA/D. When performed, this scaling happens prior to the calculation of the scalar aggregation function, so that scaled function values $\bar{f}_i(\cdot)$ replace $f_i(\cdot)$ in the calculations.

Let $\mathbf{z} \in \mathbb{R}^{n_f} : z_i \leq f_i(\mathbf{x}_k^{(t)})\ \forall k = 1, \ldots, N$, and $\widetilde{\mathbf{z}} \in \mathbb{R}^{n_f} : \widetilde{z}_i \geq f_i(\mathbf{x}_k^{(t)})\ \forall k = 1, \ldots, N$ be estimates of the ideal and nadir objective vectors at a given iteration t. A straightforward way to standardize the objective domain (Miettinen 1999; Zhang and Li 2007) is to replace each function value $f_i(\mathbf{x})$ by

$$\bar{f}_i(\mathbf{x}) = \frac{f_i(\mathbf{x}) - z_i}{\widetilde{z}_i - z_i},\ \forall i, \tag{15}$$

which guarantees that $\bar{f}_i(\mathbf{x}) \in [0, 1]\ \forall i$.[8]

[5] Wang *et al.* (2013) set this value as $\epsilon = 0.0001$.

[6] Similarly to the reference point $\mathbf{z}$, $\widetilde{\mathbf{z}}$ can also be iteratively estimated by the algorithm. The procedure is analogous to the one described earlier for the estimation of the ideal point.

[7] The usual expression of the inverted PBI strategy defines a *maximization* problem, but in this paper we express it as the equivalent minimization problem for the sake of standardization.

[8] While alternative standardization strategies can be employed (Deb and Jain 2014; Singh *et al.* 2011), the general idea remains the same.

5.3. Neighborhood assignment function

The neighborhood assignment function generates a matrix $\boldsymbol{B}$ defining the neighborhood relationships between subproblems. In general, neighborhood relations among the subproblems are used for defining restrictions to the exchange of information among candidate solutions when applying the variation operators, as well as for regulating the replacement of points at the end of every iteration (Ishibuchi *et al.* 2013).

Neighborhood relations can be defined either in the space of decision variables, Ω, or in the space of objectives $\mathbf{f}(\Omega)$. The more usual case is the definition of neighborhoods based on the distances between weight vectors in the space of objectives (Zhang and Li 2007; Li and Zhang 2009). Let $\mathbf{M} \in \mathbb{R}_{\geq 0}^{N \times N}$ be the symmetric matrix defined by taking all pairwise Euclidean distances between weight vectors, i.e., with elements given by

$$m_{i,j} = \|\boldsymbol{\lambda}_i - \boldsymbol{\lambda}_j\|_2 . \tag{16}$$

Let $\mathbf{m}_i$ denote the i-th row of $\mathbf{M}$. For the i-th subproblem, its neighborhood vector $\mathbf{b}_i \in \mathbb{Z}_{>0}^T$ consists of the indices of the T smallest elements of $\mathbf{m}_i$, with $T \in \mathbb{Z}_{>0}$ a user-defined parameter.[9] Since weight vectors are usually kept constant throughout the optimization process (with some exceptions, see Qi *et al.* (2014)), the neighborhoods have to be determined only once in this approach, and remain fixed throughout the execution of the algorithm.

An alternative approach uses Euclidean distances between incumbent solutions in the space of decision variables (Chiang and Lai 2011) to define the neighborhood relations among subproblems. In this case the distance matrix $\mathbf{M}$ is defined by the distances between all vector pairs $\mathbf{x}_i^{(t)}, \mathbf{x}_j^{(t)} \in \mathbf{X}^{(t)}$ and

$$m_{i,j} = \left\| \mathbf{x}_i^{(t)} - \mathbf{x}_j^{(t)} \right\|_2 , \tag{17}$$

with $\mathbf{x}_i^{(t)} \in \Omega$ being the incumbent solution to the i-th subproblem at iteration t. A neighborhood vector $\mathbf{b}_i$ is then composed by the indices of the T subproblems whose incumbent solutions are closest, in the space of decision variables, to the one associated with $\boldsymbol{\lambda}_i$. As the incumbent solutions change across iterations, neighborhood relations must be updated whenever a new incumbent solution is determined for any subproblem. While this results in increased computational cost ($N(N-1)/2$ distance calculations per iteration), this neighborhood definition may contribute to the algorithm performance in problems for which solution similarity in Ω does not correspond to small distances in $\mathbf{f}(\Omega)$ (Chiang and Lai 2011).

5.4. Variation stack

As in any evolutionary algorithm, variation operators in the MOEA/D generate new candidate solutions based on information about points visited by the algorithm up to a given iteration. The standard MOEA/D (Zhang and Li 2007) employs two variation operators, namely Simulated Binary recombination (SBX) (Deb and Beyer 2001) followed by Polynomial Mutation (Deb and Agrawal 1999), a combination still widely used in the literature (Asafuddoula *et al.* 2015; Li *et al.* 2015a). Another very successful MOEA/D version known as MOEA/D-DE (Li and Zhang 2009; Tan *et al.* 2012) employs Differential Mutation and Binomial Recombination (Storn and Price 1997), followed by Polynomial Mutation.

[9] Notice that neighborhood vector $\mathbf{b}_i$ will always contain the index i (since $m_{i,i} = 0$) plus additional $T-1$ subproblem indices $j \neq i$.

While these two sets of variation operators are arguably the most widespread in the literature, any combination of variation operators can, at least in principle, be used to drive the search mechanism of the MOEA/D, as long as they are sequentially compatible. This includes the possibility of employing multiple combinations of recombination and mutation variants – applied either sequentially or probabilistically – as well as the use of local search operators. Therefore, Algorithm 1 employs a user-defined set of variation operators which are applied sequentially to a population matrix $\mathbf{X}'^{(t)}$, copied from the incumbent solution matrix $\mathbf{X}^{(t)}$ prior to the application of the variation operators.

In the following descriptions all operators (with the exception of the *repair operators* described later) are defined assuming that the decision variables are contained in the interval $[0, 1]$, which greatly simplifies many calculations. Whenever a MOP is defined with decision variables bound to different limits, it is assumed that the variables are properly scaled prior to the optimization.

Also included in this discussion of variation operators are *Local Search Operators*, which execute a limited search of the solution space focused on the region close to a particular solution. Because they act as an operator that directly modifies the solution set, in this framework we group them together with other variation operators.

The variation operators available in the **MOEADr** package can be listed using functions `get_variation_operators()` and `get_localsearch_methods()`.

Definition of sampling probabilities for variation

Prior to the application of the variation stack, the neighborhood effects (Sec. 5.3) must be determined. For each subproblem, a vector $\mathbf{p}_i = \{p_{i,1}, p_{i,2}, \ldots, p_{i,N}\}$ is defined as containing the probabilities of sampling the candidate solutions associated with each subproblem when variation operators are applied for the i-th subproblem.

In a general form, sampling probabilities are defined by the neighborhood vectors $\mathbf{b}_i$ and by a user-defined parameter $\delta_p \in [0, 1]$, which represents the total probability of sampling from the specific neighborhood $\mathbf{b}_i$. Probabilities $p_{i,j}$ are defined as

$$p_{i,j} = \begin{cases} \delta_p / T & , \text{if } j \in \mathbf{b}_i \\ (1 - \delta_p) \,/\, (N - T) & , \text{otherwise.} \end{cases} \tag{18}$$

The standard MOEA/D (Zhang and Li 2007) samples exclusively from the neighborhood of each subproblem ($\delta_p = 1$ and, consequently, $p_{i,j} = 0 \; \forall j \notin \mathbf{b}_i$). This setting intensifies local exploration, at the cost of a loss of solution diversity, which may compromise the effective exploration of the design space in later iterations (Li and Zhang 2009). Less restrictive approaches have been used in the literature (Li and Zhang 2009; Chiang and Lai 2011), defining $\delta_p < 1$, albeit usually at relatively high values. In these approaches, while each subproblem maintains a strong bias towards using information from subproblems indexed by $\mathbf{b}_i$, access to the other candidate solutions remains possible, enabling the generation of a more diverse set of points by the variation operators.

SBX Recombination

Let $\eta_{\mathtt{X}} \in \mathbb{R}_{>0}$ be a user-defined parameter, and $\mathbf{u}_i \in [0, 1]^{n_v}$ be a vector of uniformly dis-

tributed random values. Also, let $\boldsymbol{\beta}_i \in \mathbb{R}^{n_v}$ be a vector with elements defined as

$$\beta_{i,j} = \begin{cases} (2u_{i,j})^{\frac{1}{\eta_{\mathtt{X}}+1}} & \text{if } u_{i,j} \leq 0.5 \\ [2(1-u_{i,j})]^{\frac{1}{\eta_{\mathtt{X}}+1}} & \text{otherwise.} \end{cases} \tag{19}$$

Let $\mathbf{x}_{a_i}^{\prime(t)} \in \mathbb{R}^{n_v}$ and $\mathbf{x}_{b_i}^{\prime(t)} \in \mathbb{R}^{n_v}$ be two vectors sampled from $\mathbf{X}^{\prime(t)}$ according to the sampling probabilities defined for the i-th subproblem (18). For each subproblem, SBX recombination (Deb and Beyer 2001; Zhang and Li 2007) produces a new candidate solution according to

$$\widetilde{\mathbf{x}}_i^{\prime(t)} = \begin{cases} \dfrac{(1+\boldsymbol{\beta}_i) \odot \mathbf{x}_{a_i}^{\prime(t)} + (1-\boldsymbol{\beta}_i) \odot \mathbf{x}_{b_i}^{\prime(t)}}{2} & , \text{ if } u_i \leq p_{\mathtt{X}} \\ \mathbf{x}_i^{\prime(t)} & , \text{ otherwise,} \end{cases} \tag{20}$$

where $u_i \in [0,1]$ is a uniformly distributed random value, and $p_{\mathtt{X}} \in [0,1]$ is a user-defined parameter. After this operator is applied for all $i \in \{1,\ldots,N\}$, each row of the population matrix $\mathbf{X}^{\prime(t)}$ is updated with its corresponding point $\widetilde{\mathbf{x}}_i^{\prime(t)}$.[10]

Polynomial Mutation

Let $\eta_{\mathtt{M}} \in \mathbb{R}_{>0}$ and $p_m \in [0,1]$ be user-defined parameters, and $\mathbf{u}_i \in [0,1]^{n_v}$ be a vector of uniformly distributed random values. For a given candidate solution $\mathbf{x}_i^{\prime(t)} \in \mathbf{X}^{\prime(t)}$, let $\boldsymbol{\beta}_i \in \mathbb{R}^{n_v}$ be a vector defined by

$$\beta_{i,j} = \left[2u_{i,j} + (1-2u_{i,j})\left(1 - x_{i,j}^{\prime(t)}\right)^{\eta'}\right]^{\frac{1}{\eta'}} - 1 \tag{21}$$

if $u_{i,j} \leq 0.5$, or

$$\beta_{i,j} = 1 - \left[2\left(1-u_{i,j}\right) + (2u_{i,j}-1)\left(x_{i,j}^{\prime(t)}\right)^{\eta'}\right]^{\frac{1}{\eta'}} \tag{22}$$

otherwise, with $\eta\prime = \eta_{\mathtt{M}} + 1$. Let also $\mathbf{v}_i \in \{0,1\}^{n_v}$ be a vector with elements sampled independently from a Bernoulli distribution with probability parameter $p_{\mathtt{M}}$. For each subproblem, the polynomial mutation (Deb and Agrawal 1999; Zhang and Li 2007) generates a candidate solution according to

$$\widetilde{\mathbf{x}}_i^{\prime(t)} = (1-\mathbf{v}_i) \odot \mathbf{x}_i^{\prime(t)} + \mathbf{v}_i \odot \left(\mathbf{x}_i^{\prime(t)} + \boldsymbol{\beta}_i\right). \tag{23}$$

After this operator is applied for all $i \in \{1,\ldots,N\}$, $\mathbf{X}^{\prime(t)}$ is updated with the new points $\widetilde{\mathbf{x}}_i^{\prime(t)}$.

Differential Mutation

Let $\phi \in \mathbb{R}_{\neq 0}$ be a user-defined parameter[11], and $\mathbf{x}_{a_i}^{\prime(t)}$, $\mathbf{x}_{b_i}^{\prime(t)} \in \mathbb{R}^{n_v}$ be two mutually exclusive vectors sampled from $\mathbf{X}^{\prime(t)}$ according to the sampling probabilities defined in (18). For the i-th

[10] Despite being much more compact than the usual definitions of SBX recombination (Deb and Beyer 2001; Zhang and Li 2007), this one is equivalent provided that the stated scaling of the variables to the interval $x_{i,j} \in [0,1]$, $\forall i,j$ is maintained.

[11] ϕ can be set either as a constant value, or defined randomly for each application of this operator. In the latter, it is common to independently sample $\phi \in (0,1]$ for each operation, according to a uniform distribution (Li, Zhou, and Zhang 2014c).

subproblem the differential mutation operator (Price, Storn, and Lampinen 2005) generates a new candidate solution as

$$\widetilde{\mathbf{x}}_i^{\prime(t)} = \mathbf{x}_{i,\text{basis}}^{(t)} + \phi \left(\mathbf{x}_{a_i}^{\prime(t)} - \mathbf{x}_{b_i}^{\prime(t)} \right), \tag{24}$$

where $\mathbf{x}_{i,\text{basis}}^{(t)} \in \mathbb{R}^{n_v}$ is a basis vector, which can be generated in several ways. The most common strategy used with the MOEA/D (Li and Zhang 2009) is to randomly sample a vector from $\mathbf{X}^{\prime(t)}$ (mutually exclusive with $\mathbf{x}_{a_i}^{\prime(t)}$ and $\mathbf{x}_{b_i}^{\prime(t)}$) according to the sampling probabilities defined in (18).

Two possible alternatives for the basis vector in the context of MOEA/D are also suggested here. For a given subproblem i, let the points $\mathbf{x}_{b_{i,k}}^{\prime(t)}$, $b_{i,k} \in \mathbf{b}_i$ be ordered in decreasing order of utility (i.e., in increasing order of aggregation function value). One possibility is to use the mean point of the neighborhood

$$\mathbf{x}_{i,\text{basis}}^{(t)} = \frac{1}{T} \sum_{k=1}^{T} \mathbf{x}_{b_{i,k}}^{\prime(t)}, \tag{25}$$

while the second is to use a weighted global intermediate point, which is an adaptation of the basis vector commonly used in evolution strategies (Arnold 2006)

$$\mathbf{x}_{i,\text{basis}}^{(t)} = \sum_{k=1}^{T} w_k \mathbf{x}_{b_{i,k}}^{\prime(t)} \tag{26}$$

with weights given by

$$\begin{aligned} w_k &= \frac{w_k'}{\sum_{k=1}^{T} w_k'} \\ w_k' &= \ln(T + 0.5) - \ln(k). \end{aligned} \tag{27}$$

Further alternative definitions of the differential mutation operator in the MOEA/D context can be found in the literature (Li *et al.* 2014c). Regardless of the specifics of each definition, this operator is known to be invariant to rotation (Price *et al.* 2005; Li *et al.* 2014c). After its application, $\mathbf{X}^{\prime(t)}$ is updated with the points $\widetilde{\mathbf{x}}_i^{\prime(t)}$.

Binomial Recombination

Let $\rho \in [0, 1]$ be a user-defined parameter, $k_i \in \{1, \ldots, n_v\}, i = 1 \ldots, N$ denote a set of randomly selected integers, and $\mathbf{u}_i \in [0, 1]^{n_v}$ be a vector of uniformly distributed random values. Also, recall that the incumbent solutions at iteration t (unmodified by any variation operators) are stored in the matrix $\mathbf{X}^{(t)}$. This operator can then be expressed by the sequential application of

$$\widetilde{x}_{i,j}^{\prime(t)} = \begin{cases} x_{i,j}^{\prime(t)} & \text{if } u_{i,j} \leq \rho \\ x_{i,j}^{(t)} & \text{otherwise,} \end{cases}; \quad j = 1, \ldots, n_v \tag{28}$$

and (29)

$$\widetilde{x}_{i,k_i}^{(t)} = \begin{cases} x_{i,k_i}^{\prime(t)} & \text{if } \widetilde{\mathbf{x}}_i^{\prime(t)} = \mathbf{x}_i^{(t)} \\ \widetilde{x}_{i,k_i}^{\prime(t)} & \text{otherwise.} \end{cases} \tag{30}$$

As with the previous operators, after this one is applied to all subproblems, each row of $\mathbf{X}'^{(t)}$ is updated with its corresponding point $\widetilde{\mathbf{x}}_i'^{(t)}$.

Repair operators

Repair operators in evolutionary algorithms for continuous optimization usually refer to strategies for ensuring that the *box constraints*, i.e., pairs of constraints defined by $(\mathtt{x}_{j,min} - x_{i,j} \leq 0 \ ; \ x_{i,j} - \mathtt{x}_{j,max} \leq 0) \, \forall j$, are satisfied. While the literature tends to place repair operators in a separate category from the variation ones, we argue that they actually belong to the same class of operations in the MOEA/D framework, as both are used to modify candidate solutions according to specific rules.

While there are a number of repair methods (Hellwig and Arnold 2016), we describe here only the simple truncation repair, which can be defined as

$$\widetilde{x}_{i,j}'^{(t)} = \max\left(\mathtt{x}_{j,min}, \ \min\left(\mathtt{x}_{j,max}, x_{i,j}'^{(t)}\right)\right), \forall i, j, \tag{31}$$

where functions $\max(\cdot,\cdot)$ and $\min(\cdot,\cdot)$ return the largest and the smallest of their arguments, respectively. After this operation is performed, $\mathbf{X}'^{(t)}$ is updated accordingly.

Local search operators

Local search strategies are usually employed in the MOEA/D to accelerate convergence. As with the repair operators, these approaches are commonly classified as distinct from the variation operators, but we argue that they are indeed part of the same block, for the same reasons stated earlier.

Since these operators sometimes result in the need for additional evaluations of candidate solutions, as well as in loss of diversity, their application is usually regulated by a frequency parameter, expressed either as a *period of application* $\tau_{ls} \in \mathbb{Z}_{>0}$ (i.e., the operator is applied to each subproblem once every τ_{ls} iterations); or as a *probability of application* $\gamma_{ls} \in [0, 1]$ (i.e., at every iteration the operator is applied with probability γ_{ls} for a given subproblem) (Deb 2001; Coello Coello *et al.* 2007). Whenever any of these conditions is true, local search is applied on the incumbent solution of a subproblem *instead* of all other variation operators. Notice that both $\tau_{ls} = 1$ and $\gamma_{ls} = 1$ lead to the same behavior, that is, the MOEA/D ignoring all other variation operators and performing only local search at every iteration. Values that are sometimes offered for these parameters in the wider MOEA literature are $\tau_{ls} = 5$ (Wanner, Guimarães, Takahashi, and Fleming 2008) and $\gamma_{ls} \in [0.01, 0.05]$ (Deb 2001; Coello Coello *et al.* 2007), albeit without much theoretical or experimental justification.

As with all other variation methods, after this operation is performed the rows of $\mathbf{X}'^{(t)}$ are updated with the corresponding points $\widetilde{\mathbf{x}}_i'^{(t)}$.

Three-Point Quadratic Approximation (TPQA): Let $f_{i_k}^{agg} \triangleq f^{agg}(\mathbf{x}_{i_k}'^{(t)} \mid \boldsymbol{\lambda}_i, \mathbf{z})$ be the aggregation function value of the kth candidate solution, $k \in \mathbf{b}_i$, for the ith subproblem. Let $\mathbf{x}_{i_1}'^{(t)}$, $\mathbf{x}_{i_2}'^{(t)}$, $\mathbf{x}_{i_3}'^{(t)} \in \mathbb{R}^{n_v}$ be the three candidate solutions with the highest utility for the ith subproblem at iteration t, such that $f_{i_1}^{agg} \leq f_{i_2}^{agg} \leq f_{i_3}^{agg}$. The three-point quadratic approximation method (Tan *et al.* 2012) for performing local search in the MOEA/D can be

expressed as

$$\widetilde{x}_{i,j}^{\prime(t)} = \begin{cases} x_{i_1,j}^{\prime(t)}, & \text{if } q_{i,j} < \epsilon \\ \widehat{x}_{i,j}^{(t)}, & \text{otherwise,} \end{cases} ; \quad j = 1, \ldots, n_v \tag{32}$$

where

$$\begin{aligned} q_{i,j} = &\left(x_{i_2,j}^{\prime(t)} - x_{i_3,j}^{\prime(t)}\right) f_{i_1}^{agg} + \left(x_{i_3,j}^{\prime(t)} - x_{i_1,j}^{\prime(t)}\right) f_{i_2}^{agg} + \\ &\left(x_{i_1,j}^{\prime(t)} - x_{i_2,j}^{\prime(t)}\right) f_{i_3}^{agg} \end{aligned} \tag{33}$$

$$\begin{aligned} \widehat{x}_{i,j}^{(t)} = &\left[\left(x_{i_2,j}^{\prime(t)}\right)^2 - \left(x_{i_3,j}^{\prime(t)}\right)^2\right] \frac{f_{i_1}^{agg}}{2q_{i,j}} + \left[\left(x_{i_3,j}^{\prime(t)}\right)^2 \right. \\ &\left. - \left(x_{i_1,j}^{\prime(t)}\right)^2\right] \frac{f_{i_2}^{agg}}{2q_{i,j}} + \left[\left(x_{i_1,j}^{\prime(t)}\right)^2 - \left(x_{i_2,j}^{\prime(t)}\right)^2\right] \frac{f_{i_3}^{agg}}{2q_{i,j}} \end{aligned} \tag{34}$$

where $\epsilon \in \mathbb{R}_{>0}$ is a small positive constant.[12]

Differential Vector-Based Local Search (DVLS): Although this strategy was originally proposed for non-decomposition MOEAs (Chen, Zeng, Lin, and Zhang 2015), its adaptation to the MOEA/D framework is straightforward. Let $\mathbf{x}_{a_i}^{\prime(t)}$, $\mathbf{x}_{b_i}^{\prime(t)} \in \mathbf{X}^{\prime(t)}$ be two candidate solutions, with $(a_i,\ b_i)$ denoting mutually exclusive indices sampled from the neighborhood $\mathbf{b}_i$. Two new candidate alternatives, $\widehat{\mathbf{x}}_i^+$ and $\widehat{\mathbf{x}}_i^-$, are then generated according to (24), using $\mathbf{x}_{i,\text{basis}}^{(t)} = \mathbf{x}_i^{(t)}$ and $\phi = \pm\phi_{ls}$, with $\phi_{ls} \sim \mathcal{N}(\mu = 0.5, \sigma = 0.1)$ (Chen *et al.* 2015). This local search operator then compares these two alternatives against the incumbent solution, and returns the point with the best performance for the subproblem, i.e.:

$$\widetilde{\mathbf{x}}_i^{\prime(t)} = \underset{\mathbf{x} \in \left\{\mathbf{x}_i^{\prime(t)},\ \widehat{\mathbf{x}}_i^+,\ \widehat{\mathbf{x}}_i^-\right\}}{\arg\min} f^{agg}(\mathbf{x} \mid \boldsymbol{\lambda}_i, \mathbf{z}). \tag{35}$$

5.5. Update strategies

Update strategies in the MOEA/D play the same role as selection operators in general MOEAs, regulating the substitution of incumbent solutions by those resulting from the application of a sequence of variation operators. In the MOEA/D, this substitution is controlled by the replacement strategy (Zhang and Li 2007; Asafuddoula *et al.* 2015; Wang, Zhang, Gong, and Zhou 2014), as well as by the neighborhood relations between subproblems. Three update methods are presented next.

In what follows, let $\mathbf{X}^{\prime(t)}$ denote the matrix containing the candidate solutions generated after the application of the variation stack, and $\mathbf{X}^{(t)}$ be the matrix containing the incumbent solutions at iteration t. Also, let $f^{agg}(\cdot)$ denote the aggregation function (see Sec. 5.2). For each subproblem i, the update strategy will generate a set of candidate solutions C_i (generally composed of the incumbent solution $x_i^{(t)}$ and candidate solutions from a given neighborhood) and select the best solution in this set as the new incumbent solution $x_i^{(t+1)}$.

[12] Tan *et al.* (2012) recommend $\epsilon = 10^{-6}$.

The list of update strategies available in the **MOEADr** package can be generated using `get_update_methods()`.

Standard Neighborhood Replacement

Let the candidate set for the i-th subproblem be defined as

$$\mathbf{C}_i^{(t)} = \mathbf{x}_i^{(t)} \cup \left\{ \mathbf{x}_k'^{(t)} | k \in \mathbf{b}_i \right\}, \tag{36}$$

where $\mathbf{b}_i$ is the neighborhood defined in Section 5.3. This replacement strategy (Zhang and Li 2007) updates the population according to

$$\mathbf{x}_i^{(t+1)} = \left\{ \mathbf{c}_q^{(t)} | \mathbf{c}_q^{(t)} \in \mathbf{C}_i^{(t)} \wedge f^{agg}\left(\mathbf{c}_q^{(t)} \mid \boldsymbol{\lambda}_i, \mathbf{z}\right) \leq \min_{\mathbf{c}_k^{(t)} \in \mathbf{C}_i^{(t)}} f^{agg}\left(\mathbf{c}_k^{(t)} \mid \boldsymbol{\lambda}_i, \mathbf{z}\right) \right\}. \tag{37}$$

That is, the best solution to the i-th subproblem considering the incumbent solution and the candidate solutions indexed in $\mathbf{b}_i$.

Restricted Neighborhood Replacement

In the standard replacement, a single candidate solution can replace up to T (the neighborhood size) incumbent solutions at any given iteration, which can lead to diversity loss and premature stagnation. To overcome this issue, Li and Zhang (2009) proposed a limit to the number of copies that any single point $\mathbf{x}_i'^{(t)}$ can pass on to $\mathbf{X}^{(t+1)}$, defined by a user-defined parameter $n_r \in \mathbb{Z}_{>0} \mid n_r \leq N$. This approach generalizes the standard neighborhood replacement (which happens if $n_r = T$), and provides a relatively simple way to limit the propagation speed of candidate solutions in the population. The definition of good values for n_r, however, may require some tuning: both $n_r = 2$ (Li and Zhang 2009) and $n_r = 0.01N$ (Zhang, Liu, and Li 2009b) are recommended, without much theoretical or empirical support.

Best Subproblem Replacement

In the previous strategies, a candidate solution $\mathbf{x}_i'^{(t)}$ is considered for replacing the incumbent solutions of its neighboring subproblems, i.e., it can replace incumbent solutions $\mathbf{x}_k^{(t)} \mid k \in \mathbf{b}_i$. However, it may happen that the best available candidate solution for a given subproblem i is not among those in the neighborhood $\mathbf{b}_i$. To address this issue, Wang *et al.* (2014) suggested a three-step replacement strategy. The first step is to identify the subproblem k_i for which each new candidate solution $\mathbf{x}_i'^{(t)}$ is most effective, i.e.:

$$k_i = \arg\min_{1 \leq k \leq N} f^{agg}(\mathbf{x}_i'^{(t)} | \boldsymbol{\lambda}_k, \mathbf{z}). \tag{38}$$

The second step is to generate replacement neighborhoods $\mathbf{b}_{k_i}^r$, which are composed of the T_r nearest neighbors to the k_i-th subproblem (calculated as in Sec. 5.3), where $T_r \in \mathbb{Z}_{>0} \mid T_r \leq N$ is a user-defined parameter. Finally, for each subproblem i, the original neighborhood $\mathbf{b}_i$ is replaced by $\mathbf{b}_{k_i}^r$, and then the *Restricted Neighborhood Replacement* is used for updating the population.

5.6. Constraint handling approaches

Some possible approaches to deal with constraints in the MOP formulation are discussed below. While most available implementations of the MOEA/D (see Section 3) provide versions that are only capable of dealing with box constraints (using repair operators such as truncation), dealing with general constraints in the context of the MOEA/D is relatively straightforward, as discussed below.

The constraint handling techniques available in the **MOEADr** package can be consulted using `get_constraint_methods()`.

Penalty functions

The most common approach in the MOEA community to handle constraints is to use penalties (Miettinen 1999; Coello 2002). The idea of penalty functions is to transform a constrained optimization problem to an unconstrained one, by adding a certain value to the aggregation function value of a given point based on the magnitude of constraint violation that it presents.

Assuming that the magnitude of the constraint violation is derived from the sum of violations of all inequality and equality constraints, its value can be defined as[13]

$$v(\mathbf{x}) = \sum_{i=1}^{n_g} \max(g_i(\mathbf{x}), 0) + \sum_{j=1}^{n_h} \max(|h_j(\mathbf{x})| - \epsilon, 0), \tag{39}$$

where $\epsilon \in \mathbb{R}_{\geq 0}$ is a small threshold representing a tolerance for the equality constraints. The new (penalized) aggregation function to be optimized is then obtained as

$$f_{pen}^{agg}(\mathbf{x}|\boldsymbol{\lambda}_k, \mathbf{z}) = f^{agg}(\mathbf{x}|\boldsymbol{\lambda}_k, \mathbf{z}) + \beta_v v(\mathbf{x}) \ , \tag{40}$$

in which $\beta_v \in \mathbb{R}_{>0}$ is a user-defined penalization constant. For each subproblem, the penalized values are then used to select which solution will become (or remain) the incumbent one in the next population, $\mathbf{X}^{(t+1)}$, according to the update strategies (Sec. 5.5).

Violation-based Ranking (VBR)

This constraint handling technique generalizes a few methods available in the literature, such as Tournament Selection (TS) (Deb 2000), Stochastic Ranking (SR) (Runarsson and Yao 2000), and Violation Threshold (VT) (Asafuddoula, Ray, Sarker, and Alam 2012). Like the Penalty Functions approach, Violation-based Ranking uses the total magnitude of constraint violations $v(\mathbf{x})$ (39) to penalize unfeasible solutions, but this penalization takes the form of ranking functions that employ different quantities depending on the feasibility (or not) of the solutions being compared.

In its general form, this constraint handling method uses the following criteria when ranking the solutions:

- If a solution is feasible, use its $f^{agg}(\mathbf{x}|\boldsymbol{\lambda}_k, \mathbf{z})$ value;

[13] Individual violation values can also be subject to scaling, to prevent extreme differences in the scale of constraint functions from dominating the attribution of the penalized utility value. This discussion, however, is not advanced further in the present work.

- If a solution is unfeasible, check a criterion $c(\mathbf{x})$:
  - If $c(\mathbf{x}) = TRUE$, use its $f^{agg}(\mathbf{x}|\boldsymbol{\lambda}_k, \mathbf{z})$ value;
  - If $c(\mathbf{x}) = FALSE$, use its $v(\mathbf{x})$ value;
- Assuming that $n_1 \leq N$ points are to be ranked using their $f^{agg}(\mathbf{x}|\boldsymbol{\lambda}_k, \mathbf{z})$ values, VBR ranks those solutions first (rank values from 1 to n_1), and then attributes ranks for the remaining solutions using their $v(\mathbf{x})$ values, from $n_1 + 1$ to N. Rank values are then used instead of $f^{agg}(\mathbf{x}|\boldsymbol{\lambda}_k, \mathbf{z})$ whenever solutions must be compared.

As mentioned earlier, this criterion generalizes the commonly-used constraint handling techniques Tournament Selection (TS), Stochastic Ranking (SR), and Violation Threshold (VT). These specific methods can be instantiated from the rules defined in Table 4, by choosing an appropriate function $c(\mathbf{x})$, see Table 5. For SR, $p_f \in [0, 1]$ represents a user-defined parameter, and $u \in [0, 1]$ denotes a uniformly distributed random value. For VT, the adaptive threshold ϵ_v is calculated independently for each subproblem, as

$$\epsilon_{v,i} = \frac{[fs]_i}{(T+1)^2} \sum_{k \in \mathbf{b}'_i} v(\mathbf{x}_k), \tag{41}$$

where $\mathbf{b}'_i = \mathbf{x}_i^{(t)} \bigcup \left\{ \mathbf{x}_k'^{(t)} | k \in \mathbf{b}_i \right\}$; $[fs]_i$ represents the number of feasible solutions in $\mathbf{b}'_i$; and $(T+1)$ is the cardinality of $\mathbf{b}'_i$.[14]

Table 4: Ranking criteria for the violation-based ranking

Situation		Rank using
$v(\mathbf{x}) = 0$		$f^{agg}(\mathbf{x}\|\boldsymbol{\lambda}_k, \mathbf{z})$
$v(\mathbf{x}) > 0$	$c(\mathbf{x}) = TRUE$	
	$c(\mathbf{x}) = FALSE$	$v(\mathbf{x})$

Table 5: Secondary criterion functions

Method	Function $c(\mathbf{x}) =$
Tournament Selection (Deb 2000)	$FALSE$
Stochastic Ranking (Runarsson and Yao 2000)	$(u \leq p_f)$
Violation Threshold (Asafuddoula *et al.* 2012)	$(v(\mathbf{x}) \leq \epsilon_v)$

An important point to highlight here is that the VBR method does not necessarily guarantee that feasible solutions will always be maintained in the population. This means that a good feasible solution obtained previously for a scalar subproblem may be eliminated later in the search. To prevent this loss of feasible solutions, an elitist archiving strategy can be adopted in order to preserve the *feasible* solution with the best aggregation value found for each subproblem throughout the iterations. In this case, this elitist archive should be considered as the output population of the algorithm (Ying, He, Huang, Li, and Wu 2016).

[14] If the *Best Subproblem Replacement* is used, then $\mathbf{b}^r_{k_i}$ is used instead of $\mathbf{b}_i$, and T_r instead of T.

Finally, it is worth mentioning that this generalization of known constraint handling techniques may serve as a template for further developments of better methods for dealing with constraints in population-based algorithms, both for single and multi-objective cases. This possibility, however, falls outside the scope of the current paper, and will not be explored further.

5.7. Termination criteria

Termination criteria determine when the algorithm should stop running and return the solutions found. Common criteria in the evolutionary computation literature tend to fall into two broad categories, namely: (i) exhaustion of the algorithmic budget, in terms of execution time, number of points visited, or number of iterations performed; and (ii) attainment of a specific threshold value for some criterion, such as lack of improvement, loss of population diversity, convergence to specified quality values, or some other population statistic.[15]

Time-based criteria stop the process after a user-defined amount of time has passed since the program started running. These criteria can be useful when comparing computationally intensive tasks such as the construction of neighborhood or subproblem weight matrices, but it can be sensitive to background tasks if *wall clock* is used. In general, *process time* tends to be preferable, although it tends to add some complexity to the implementation.

Number of iterations criteria stop the search immediately after the iteration counter becomes higher than a given threshold. This tends to be useful when comparing minor variations of a given algorithm, as long as there is no significant differences in the computational cost per iteration.

Finally, *number of evaluations* criteria terminate the search when the total number of points visited by the algorithm reaches a predefined threshold. This criterion is usually preferred in situations when the larger part of the total computational cost is incurred by utility function evaluations, in which case it is a good practice to keep this value constant when comparing very different algorithms. Notice that because the termination check occurs once per iteration, the total number of evaluations may show some fluctuations from the user-defined value.

The list of available stop criteria in the **MOEADr** package can be consulted using function `get_stop_criteria()`.

6. Usage examples

In this section we present three case studies which illustrate common usage situations for the **MOEADr** package. The first shows how to solve a simple MOP using the package. The second illustrates the application of the component oriented framework with an automatic algorithmic assembling and tuning approach. Finally, the third example demonstrates how to use a user-defined component with the package.

These three case studies are also available as vignettes in the **MOEADr** documentation.

[15] An interesting compilation of works on stop criteria for MOEAs is maintained by Luis Martí at http://lmarti.com/stopping.

6.1. Solving a simple MOP using MOEADr

In this case study, we use **MOEADr** to optimize a simple 2-objective problem composed of the (shifted) Sphere and Rastrigin functions in the domain $\Omega = [-1, 1]^{n_v}$, defined as

$$\begin{aligned} \text{sphere}(\mathbf{x}) &= \sum_{i=1}^{n_v} (x_i + 0.1i)^2 \text{, and} \\ \text{rastrigin}(\mathbf{x}) &= \sum_{i=1}^{n_v} \left[(x_i - 0.1i)^2 - 10\cos\left(2\pi \left(x_i - 0.1i\right)\right) + 10 \right] \end{aligned} \tag{42}$$

The R implementation of the problem can be defined as follows. Note that the **MOEADr** package requires the multiobjective problem to be defined as a function that can receive a population matrix $\mathbf{X}$, and return a matrix of objective function values for each point.

```
R> #Define objective functions
R> sphere <- function(X) {
+    X.shift <- X + seq_along(X) * 0.1
+    sum(X.shift**2)
+  }
R> rastringin <- function(X) {
+    X.shift <- X - seq_along(X) * 0.1
+    sum((X.shift)**2 - 10 * cos(2 * pi * X.shift) + 10)
+  }

R> #wrap objective functions in an evaluator routine
R> problem.sr <- function(X) {
+          t(apply(X, MARGIN = 1,
+          FUN = function(X) { c(sphere(X), rastringin(X)) }
+          ))
+  }

R> #assemble a problem definition list
R> problem.1 <- list(name = "problem.sr", # function that executes the MOP
+                    xmin = rep(-1, 30),  # lower bound for each dimension
+                    xmax = rep(1, 30),   # upper bound for each dimension
+                    m    = 2)            # number of objectives
```

To load the package and run the problem using the original MOEA/D (Zhang and Li 2007), we use the following commands:

```
R> library(MOEADr)
R> results <- moead(problem  = problem.1,
+                   preset   = preset_moead("original"),
+                   seed     = 42)
```

The `moead()` function requires a problem definition, discussed above, an algorithm configuration, logging parameters, and a seed. In this example we used an algorithm preset. The

`preset_moead()` function can output a number of different presets based on combinations found on the literature. These presets can also be modified (either partially or as a whole), as will be shown in another case study. `preset_moead("original")` returns the original MOEA/D configuration, as proposed by Zhang and Li (2007). Running `preset_moead()` without an argument outputs a list of available presets.

The `moead()` function returns a list object of class `moeadoutput`, containing the final solution set, objective values for each solution, and other information about the optimization process. The **MOEADr** package uses S3 to implement versions of `plot()` and `summary()` for this object.

`plot()` will show the estimated Pareto front for the objectives, as in Figure 1. When the number of objectives is greater than 2, a parallel coordinates plot is also produced (see Figure 2). `summary()` displays information about the number of non-dominated and feasible solution points, the estimated ideal and nadir values, and (optionally) the inverted generational distance (IGD) and hypervolume indicators (Zitzler, Thiele, Laumanns, Fonseca, and Fonseca 2003) calculated for the set of feasible, nondominated points returned.

```
R> summary(results)
R> plot(results)

#> Warning: reference point not provided:
#>          using the maximum in each dimension instead.
#> Summary of MOEA/D run
#> #====================================
#> Total function evaluations:  20100
#> Total iterations:  200
#> Population size:  100
#> Feasible points found:  100 (100% of total)
#> Nondominated points found:  100 (100% of total)
#> Estimated ideal point:  31.879 92.925
#> Estimated nadir point:  117.372 450.424
#> Estimated HV:  23972.73
#> Ref point used for HV:  117.3718 450.4242
#> #====================================

## (Plot output shown in Figure 1)
```

A more complex example

Package **smoof** (Bossek 2016) provides the implementation of a large number of single and multiobjective test functions. **MOEADr** provides a wrapper function `make_vectorized_smoof()` to easily convert **smoof** functions to the format required by the `moead()` function, as illustrated below for a 5-objective standard MOP benchmark problem.

```
R> library(smoof)
R> DTLZ2 <- make_vectorized_smoof(prob.name    = "DTLZ2",
+                                 dimensions   = 20,
```

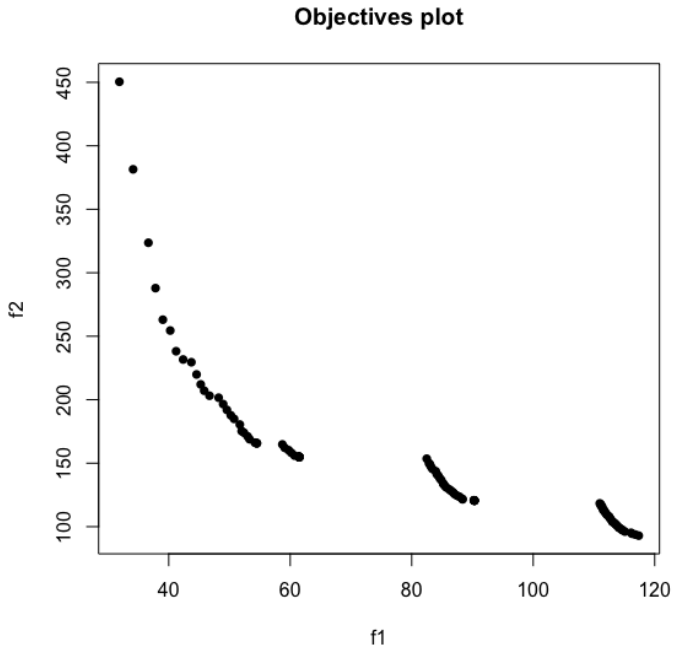

Figure 1: Estimated front produced by running the code in Case Study 1.1

```
+                                  n.objectives = 5)
R> problem.dtlz2  <- list(name = "DTLZ2",
+                         xmin = rep(0, 20),
+                         xmax = rep(1, 20),
+                         m    = 5)
```

The code below shows an example of how to modify an algorithm preset. Because of the higher number of objectives, we want to reduce the value of the parameter H in the SLD decomposition component (see Section 5.1) used by the preset from 100 to 8:

```
R> results.dtlz <- moead(problem   = problem.dtlz2,
+                        preset    = preset_moead("original"),
+                        decomp    = list(name = "SLD", H = 8)
+                        seed      = 42)
```

As before, we see an overview of the results with `summary()` and `plot()`. Notice that the output of `plot()` is different when the number of objectives in a problem is greater than 2, as shown in Figure 2.

```
R> summary(results.dtlz)
R> plot(results.dtlz)

#> Warning: reference point not provided:
#>          using the maximum in each dimension instead.
```

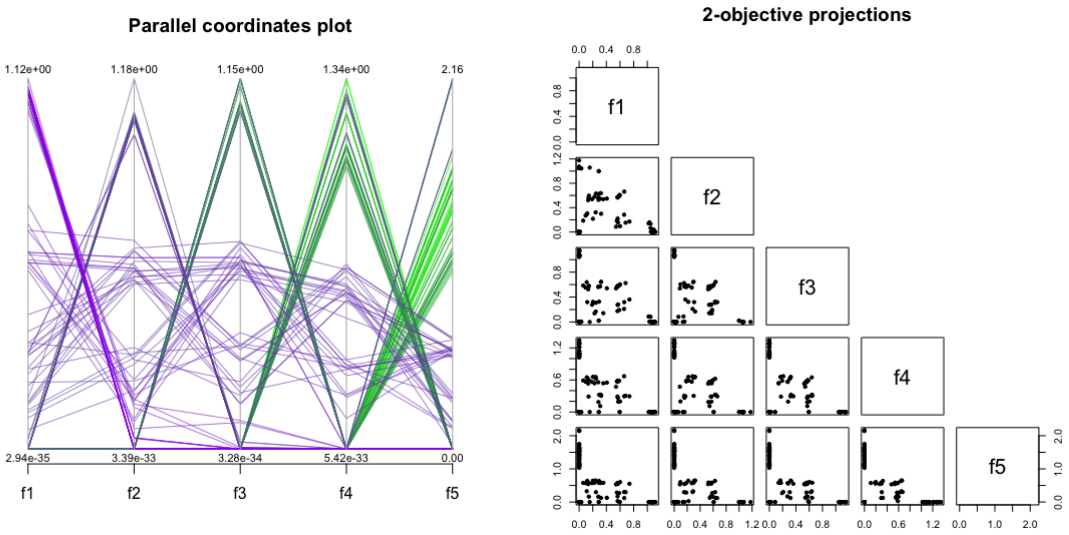

Figure 2: Parallel coordinates plot and 2 objective projections produced by Case Study 1.2

```
#> Summary of MOEA/D run
#> #====================================
#> Total function evaluations:  99495
#> Total iterations:  200
#> Population size:  495
#> Feasible points found:  495 (100% of total)
#> Nondominated points found:  242 (48.9% of total)
#> Estimated ideal point:  0 0 0 0 0
#> Estimated nadir point:  1.252 1.176 1.174 1.571 2.16
#> Estimated HV:  4.974365
#> Ref point used for HV:  1.252253 1.176129 1.174102 1.57124 2.160007
#> #====================================

## (Plot output in Figure 2)
```

6.2. Fine tuning MOEA/D configurations using MOEADr and irace

For this example, we employ the *Iterated Racing* procedure (available in the **irace** package) (López-Ibáñez, Dubois-Lacoste, Cáceres, Birattari, and Stützle 2016), to automatically assemble and fine-tune a MOEA/D configuration based on the components available in the **MOEADr** package.

Ten unconstrained test problems from the CEC2009 competition[16] are used, with dimensions ranging from 20 to 60. Dimensions 30, 40 and 50 were reserved for testing, while all others were used for the training effort. To quantify the quality of the set of solutions returned by a

[16]Functions UF1 to UF10 http://dces.essex.ac.uk/staff/qzhang/moeacompetition09.htm, using the standard implementation available in the **smoof** package.

candidate configuration we use the Inverted Generational Distance (IGD) (Zitzler *et al.* 2003) indicator. The number of subproblems was fixed as 100 for $n_f = 2$ and 150 for $n_f = 3$.[17]

We define a tuning budget of 20,000 runs for this example. The possible configurations are composed from the following choices:

- Decomposition Strategy: SLD or Uniform;
- Scalar Aggregation function: WT, PBI or AWB;
- Type of neighborhood: by weights or by incumbent solutions;
- Type of Update: Standard, Restricted, or Best Subproblem;
- Variation Stack: See below;

For every combination, the parameters of each component (e.g. θ^{pbi} for the PBI aggregation function) were also included as part of the tuning experiment. Objective scaling was employed in all cases. No constraint handling was required, and the stop criterion for each run was set as 100,000 evaluations.

The variation stack was composed of three to five operators, using the following rules: the first and second operators could be any of the traditional operators currently available in the **MOEADr** package: SBX, Polynomial Mutation, Differential Mutation, and Binomial Recombination. The third operator could be any of these, or "none" (i.e., not present). The fourth operator could be either a local search operator or "none". Finally, the variation stack always finished with the truncation repair operator (mainly to avoid errors with the implementation of the test functions). No restrictions were placed on repeats of the variation operators, and the specific conditional parameters for each operator were allowed to be tuned independently for each position in the variation stack.

The code of this case study requires a large amount of boilerplate for the **irace** package and, for the sake of brevity, it is not included in the text. Please refer to the supplementary materials or the "Fine tuning MOEA/D configurations using MOEADr and irace" vignette in the **MOEADr** package for the full replication script.

Figure 3 shows the IGD values achieved by the seven final configurations over the test problems. Based on these final configurations, we assemble the consensus MOEA/D configuration, which is described in Table 6. The third column of this table, together with Fig. 4, provides the consensus value of each component, measured (in the table) as the rate of occurrence of each component in the seven final configurations returned by Iterated Racing procedure. These results suggest that the automated assembling and tuning method reached a reasonably solid consensus, in terms of the components used as well as the values returned for the numeric parameters.

A feature that may seem surprising at first glance is the two sequential applications of Binomial Recombination in the Variation Stack. This means that the results of a Differential Mutation operator are recombined with the incumbent solutions at a (reasonably low) recombination rate $\rho_1 = 0.495$; and then the resulting vectors are again recombined with the incumbent solutions, at a (much higher) rate $\rho_2 = 0.899$. However, a quick review of the Binomial

[17]For the SLD decomposition the actual number was 153 for $n_f = 3$, given the specifics of this method. Please refer to Section 5.1 for details.

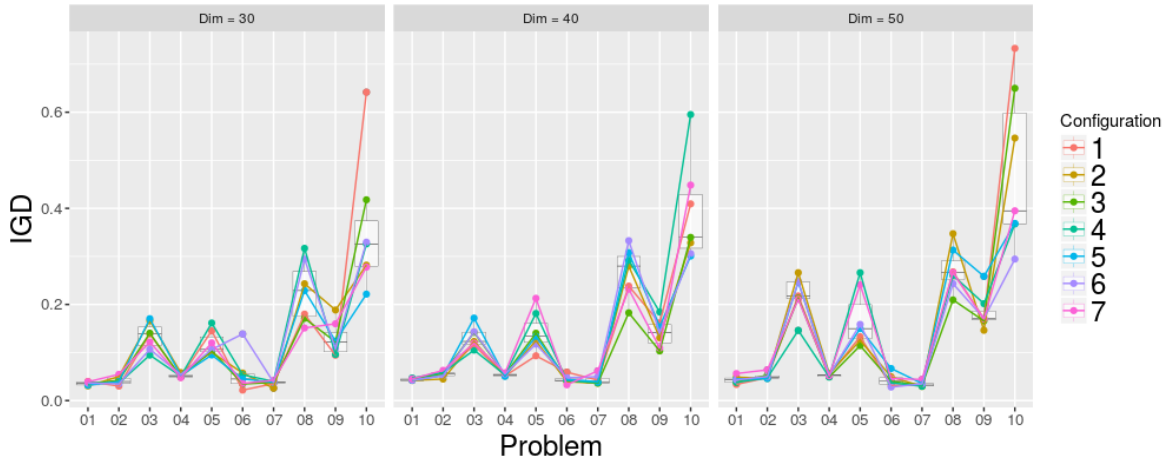

Figure 3: IGD values for the Top 7 configurations generated by **irace** on the testing problems

Recombination (Sec. 5.4) and some elementary probability shows that these two sequential applications of binomial recombination can be expressed as a single application with $\rho = \rho_1\rho_2 = 0.445$. The fact that Iterated Racing converged to two operators instead of a single one can be explained by the fact that these situations are equivalent, coupled with the absence of any pressure towards more parsimonious expressions of the Variation Stack.

Another seemingly counterintuitive aspect of the final configurations reached is the absence of local search. A possible explanation lies in the interaction between the variation operators chosen (differential mutation + binomial recombination) with the type of neighborhood (by $\mathbf{x}$, with a very strong bias towards using points from the neighborhood for the variation operators: $\delta_p = 0.909$). The use of points that are relatively similar in the space of decision variables may result in local exploration in this case, since the magnitude of the differential vectors tends to become relatively small. As the iterations progress, this Variation Stack would tend to perform local search movements, with larger perturbations potentially occurring whenever points are sampled from outside the neighborhoods (i.e., about 10% of the cases, since $\delta_p \approxeq 0.9$). It is therefore possible that this local exploration characteristic may have have resulted in a MOEA/D version that does not benefit from an explicit local search operator.

Besides these considerations, the configurations returned by the Iterated Racing procedure present a few other interesting points. First, the use of a smaller neighborhood than is usually practiced in the literature ($T = 11$), with the neighborhood relations being defined at each iteration by the similarities between the incumbent solutions of each subproblem. Second, the use of a very strict Restricted Neighborhood Update, with $n_r = 1$, which suggests an advantage in trying to maintain diversity instead of accelerating convergence. The use of the variation operators of the MOEA/D-DE *without* Polynomial Mutation (as is usually practiced in the MOEA/D-DE (Li and Zhang 2009)) is also curious, as it may indicate a more parsimonious variation stack than is usually practiced in the literature.

As this case study shows, exploring the space of possible component configurations and parameter values can render improved algorithmic configurations and new insights into the roles of specific components and parameter values outside the standard values from the literature. Thus, we highly recommend that a similar approach is used when developing new compo-

Table 6: Final MOEA/D configuration returned by Iterated Racing. The first two columns show the best configuration, while the third column shows the consensus value

Component	Value	Consensus
Decomposition	SLD	1.00
Aggregation function	AWT	1.00
Objective Scaling	simple	Fixed
Neighborhood	by $\mathbf{x}$	1.00
	$T = 11$	see Fig. 4
	$\delta_p = 0.909$	see Fig. 4
Variation stack	Differential mutation	1.00
	$basis =$ "$rand$"	1.00
	$\phi \sim U(0, 1)$	Fixed
	Binomial recombination	1.00
	$\rho_1 = 0.495$	see Fig. 4
	Binomial recombination	1.00
	$\rho_2 = 0.899$	see Fig. 4
	Truncate	Fixed
Update	Restricted	1.00
	$n_r = 1$	1.00

.

nents, in order to observe not only the individual performance of the novel component, but also its interaction with components which already exist in the MOEA/D environment.

6.3. Adding new components to MOEADr

In addition to a wide variety of components from the literature, the **MOEADr** package also supports the use of user-defined components. For this example, we will create a simple variation operator that does not exist in the package in its current state, and compare it with the original MOEA/D.

Consider the following "Gaussian Mutation" operator: given a set X of solutions $\mathbf{x_i} \in X$ we add, with probability p, a noise $r_{ij} \sim \mathcal{N}(\mu, \sigma)$ to each $x_{ij} \in \mathbf{x_i} \in X$. The R code for this operator is as follows:

```
R> variation_gaussmut <- function(X, mean = 0, sd = 0.1, p = 0.1, ...) {
R>   # You want to do some error checking on the parameters here,
R>   # but for the sake of brevity we are skipping it in this case study.
R>
R>   R <- rnorm(length(X),    # vector of normally distributed values,
R>              mean = mean,  # length(R) = nrow(X) * ncol(X)
R>              sd   = sd)
R>   R <- R * (runif(length(X)) <= p) # Apply binary mask, probability = p
R>   return (X + R)                   # Add mutations to the solution matrix
R> }
```

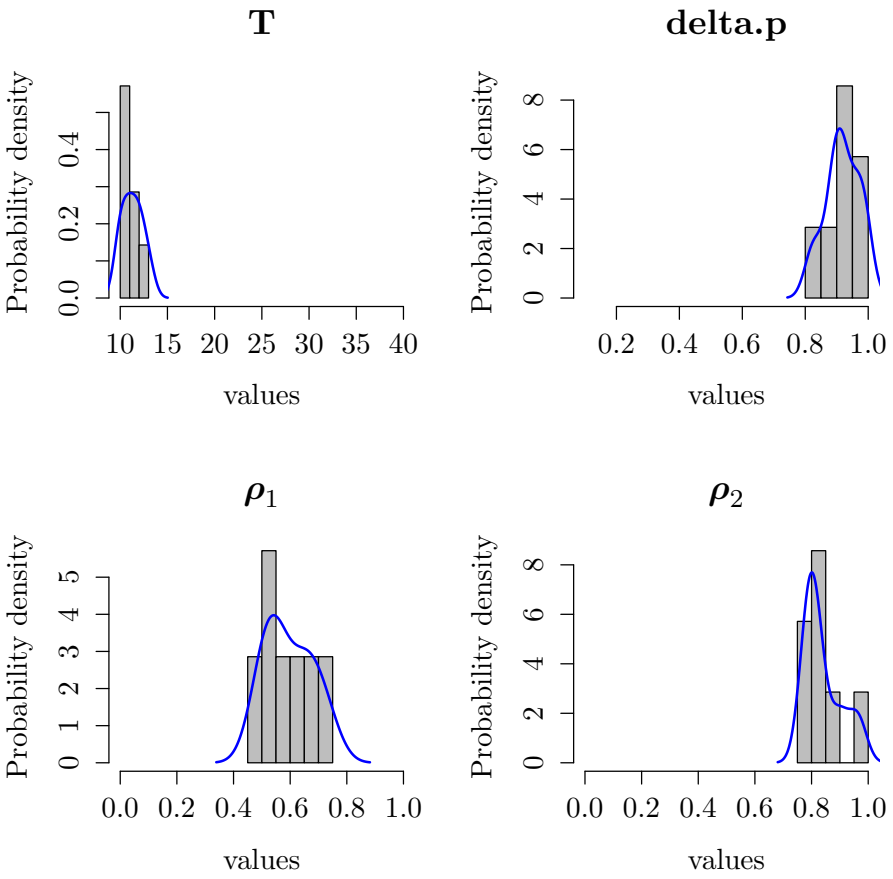

Figure 4: Values of the numeric parameters returned by Iterated Racing.

We would like to highlight a few characteristics of the code sample above. First, the name of the function must be in the form `variation_[functionname]`. The **MOEADr** package uses function name prefixes to perform some automated functions such as listing existing components and error checking. The list of current function prefixes and their meaning is:

- `constraint_`: Constraint handling components
- `decomposition_`: Decomposition functions
- `ls_`: Local search operators
- `scalarization_`: Scalarization functions
- `stop_`: Stop criteria components
- `uptd_`: Update components
- `variation_`: Variation operators

Second, the parameters in the definition of the variation operator function must include: the solution set matrix `X`, any specific parameters for the function, and finally an ellipsis argument

to catch any other inputs passed down by the main `moead()` function, such as objective values and former solution sets. Extensively commented examples of using these parameters are available in the source code for the variation operators included in the package, such as the Binomial Recombination operator (`variation_binrec()`). Other component classes follow similar rules, as documented in the *Writing Extensions for the MOEADr Package* vignette.

If a given function is not available in the **MOEADr** package environment, it will search for it in the base R environment. Therefore, if you have named your component correctly, all you need to do is add it to the appropriate parameter in the `moead()` call.

For example, let us replace the variation stack of the original MOEA/D by our Gaussian Mutation operator, followed by simple truncation, and test it on a standard benchmark function, and use the plotting function to perform a purely qualitative graphical comparison against the original MOEA/D:

```
R> library(MOEADr)
R> library(smoof)
R>
R> ZDT1 <- make_vectorized_smoof(prob.name  = "ZDT1",
+                                dimensions = 30)
R> problem.zdt1 <- list(name = "ZDT1",
+                       xmin = rep(0, 30), xmax = rep(1, 30),
+                       m    = 2)
R>
R> # Variation stack, with our new operator followed by truncate
R> myvar <- list()
R> myvar[[1]] <- list(name = "gaussmut", p = 0.5)
R> myvar[[2]] <- list(name = "truncate")
R>
R> results.orig  <- moead(problem   = problem.zdt1,
+                         preset    = preset_moead("original"),
+                         seed      = 42)
R> results.myvar <- moead(problem   = problem.zdt1,
+                         preset    = preset_moead("original"),
+                         variation = myvar,
+                         seed      = 42)
R> plot(results.orig)
R> plot(results.myvar)
```

Figure 5 shows the estimated Pareto front for both the standard MOEA/D and the MOEA/D with a Gaussian Mutation operator. From these images it seems, rather unsurprisingly, that the example operator is not an adequate replacement for the variation methods used in the standard MOEA/D.

7. Summary and conclusions

In this article we presented the R package **MOEADr**, implementing a new, component-based formulation of the MOEA/D framework. This formulation breaks down the algorithm into

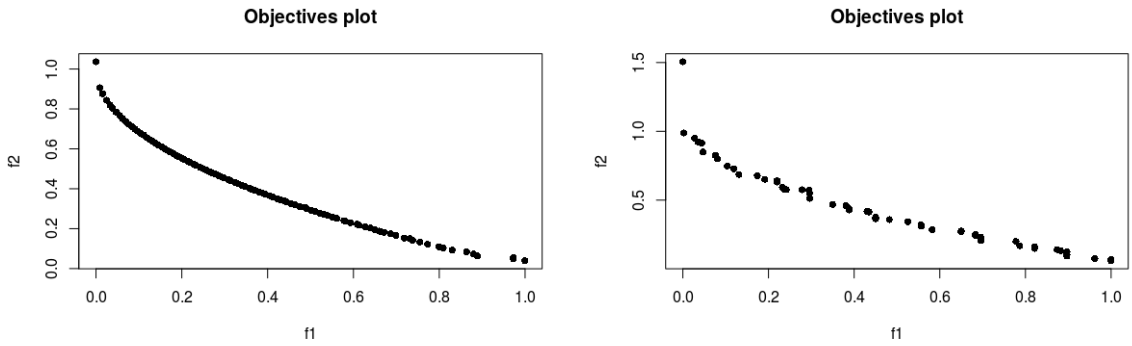

Figure 5: Estimated Pareto front achieved by the "original" MOEA/D composition (left) and the MOEA/D using the Gaussian Mutation variation (right)

independent components that can be separately replaced or configured. We described many recent proposals in the MOEA/D literature in an unified mathematical formulation fitting this framework.

The package allows users to easily instantiate a large variety of works from the MOEA/D literature, which facilitates the reproducibility of published results and the application of known variants to problems of interest. By implementing the components listed in Table 3 and allowing the definition of new functions for any component of the algorithm, the package also allows practitioners to quickly test new proposals and ideas.

Recently, Bezerra *et al.* (2016) introduced a component-wise framework for dominance- and indicator-based MOEAs (a vision that is partially implemented by the **ecr** package), and described how that framework was useful for the automated generation of algorithms. In the present work we introduced a counterpart for decomposition-based approaches, covering the third major branch of multiobjective evolutionary algorithms.

Besides expanding the library of available components, current work for improving the package includes the addition of parameter self-adaptation and of performance-based stop criteria. The possibility of including preference information within the **MOEADr** framework to bias the search towards specific subsets of the Pareto-optimal front (Goulart and Campelo 2016; Goulart, de Souza, Batista, and Campelo 2017b), as well as expanding the package to deal with robust multiobjective optimization problems (Goulart, Borges, Takahashi, and Campelo 2017a) are also among the improvements planned for a future release.

Acknowledgments

The first two authors (FC and LSB) would like to acknowledge the financial support of Brazilian funding agencies CNPq and FAPEMIG, which allowed them to pursue the projects that eventually gave rise to the idea of a component-based framework and of the **MOEADr** package.

Authors FC and CA contributed equally to testing and development, while LS provided expertise regarding the MOEA/D algorithm and its many variants, as well as the initial

formalization of the algorithmic components.

References

Arnold DV (2006). "Weighted multirecombination evolution strategies." *Theoretical Computer Science*, **361**(1), 18–37.

Asafuddoula M, Ray T, Sarker R (2015). "A Decomposition Based Evolutionary Algorithm for Many Objective Optimization." *IEEE Trans. Evolutionary Computation*, **19**(3), 445–460.

Asafuddoula M, Ray T, Sarker R, Alam K (2012). "An Adaptive Constraint Handling Approach Embedded MOEA/D." In *Proc. IEEE Congress on Evolutionary Computation (CEC)*, pp. 1–8. Brisbane, Australia.

Bezerra L, López-Ibañez M, Stützle T (2016). "Automatic Component-Wise Design of Multi-Objective Evolutionary Algorithms." *IEEE Trans. Evolutionary Computation*, **20**(3), 403–417.

Bezerra LCT, López-Ibáñez M, Stützle T (2015). "To DE or Not to DE? Multi-objective Differential Evolution Revisited from a Component-Wise Perspective." In *Lecture Notes in Computer Science*, pp. 48–63. Springer Nature.

Binois M, Picheny V (2016). *GPareto: Gaussian Processes for Pareto Front Estimation and Optimization*. R package version 1.0.3, URL https://CRAN.R-project.org/package=GPareto.

Bossek J (2016). *smoof: Single and Multi-Objective Optimization Test Functions*. R package version 1.4, URL https://CRAN.R-project.org/package=smoof.

Bossek J (2017). "ecr 2.0: A Modular Framework for Evolutionary Computation in R." In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, GECCO '17, pp. 1187–1193.

Chan LY (2000). "Optimal designs for experiments with mixtures: A survey." *Communications in Statistics: Theory and Methods*, **29**(9–10), 2281–2312.

Chen B, Zeng W, Lin Y, Zhang D (2015). "A New Local Search-Based Multiobjective Optimization Algorithm." *IEEE Trans. Evolutionary Computation*, **19**(1), 50–73.

Cheng R, Jin Y, Olhofer M, Sendhoff B (2016). "A reference vector guided evolutionary algorithm for many-objective optimization." *IEEE Trans. Evolutionary Computation*, **20**(5), 773–791.

Chiang T, Lai Y (2011). "MOEA/D-AMS: Improving MOEA/D by an Adaptive Mating Selection Mechanism." In *Proc. IEEE Congress on Evolutionary Computation (CEC)*, pp. 1473–1480.

Coello CAC (2002). "Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: A survey of the state of the art." *Computer Methods in Applied Mechanics and Engineering*, **191**(11–12), 1245–1287.

Coello Coello CA, Lamont GB, van Veldhuizen DA (2007). *Evolutionary Algorithms for Solving Multi-Objective Problems*. 2nd edition. Springer, New York.

Das I, Dennis JE (1998). "Normal-Boundary Intersection: A New Method for Generating the Pareto Surface in Nonlinear Multicriteria Optimization Problems." *SIAM Journal of Optimization*, **8**(3), 631–657.

Deb K (2000). "An Efficient Constraint Handling Method for Genetic Algorithm." *Computer Methods in Applied Mechanics and Engieering*, **186**(2–4), 311–338.

Deb K (2001). *Multi-Objective Optimization Using Evolutionary Algorithms*. John Wiley & Sons, Inc., New York, NY, USA.

Deb K, Agrawal S (1999). "A Niched-Penalty Approach for Constraint Handling in Genetic Algorithms." In *Artificial Neural Nets and Genetic Algorithms*, pp. 235–243. Springer Science + Business Media.

Deb K, Beyer H (2001). "Self-Adaptive Genetic Algorithms with Simulated Binary Crossover." *Evolutionary Computation*, **9**(2), 197–221.

Deb K, Jain H (2014). "An Evolutionary Many-Objective Optimization Algorithm Using Reference-Point-Based Nondominated Sorting Approach, Part I: Solving Problems With Box Constraints." *IEEE Trans. Evolutionary Computation*, **18**(4), 577–601.

Fang KT, Lin DKJ (2003). *Uniform designs and their application in industry*, volume 22 of *Handbook of Statistics*. Elsevier. 131–170.

Giagkiozis I, Purshouse R, Fleming P (2014). "Generalized decomposition and cross entropy methods for many-objective optimization." *Information Sciences*, **282**, 363–387.

Gong M, Liu F, Zhang W, Jiao L, Zhang Q (2011). "Interactive MOEA/D for multi-objective decision making." In *Proc. Genetic and Evolutionary Computation Conference (GECCO)*, pp. 721–728. ACM, New York, NY, USA.

Goulart F, Borges ST, Takahashi FC, Campelo F (2017a). "Robust multiobjective optimization using regression models and linear subproblems." In *Proceedings of the Genetic and Evolutionary Computation Conference*.

Goulart F, Campelo F (2016). "Preference-guided evolutionary algorithms for many-objective optimization." *Information Sciences*, **329**, 236–255.

Goulart F, de Souza DE, Batista LS, Campelo F (2017b). "A Preference-guided Multiobjective Evolutionary Algorithm based on Decomposition." In *Proceedings of the ENIAC - Brazilian National Meeting on Artificial and Computational Intelligence*.

Hadka D (2017). *MOEA Framework*. URL http://moeaframework.org/index.html.

Hellwig M, Arnold DV (2016). "Comparison of Constraint-Handling Mechanisms for the (1, λ)-ES on a Simple Constrained Problem." *Evolutionary Computation*, **24**(1), 1–23.

Ishibuchi H, Akedo N, Nojima Y (2013). "Relation between neighborhood size and MOEA/D performance on many-objective problems." In *Evolutionary Multi-Criterion Optimization (EMO)*, volume 7811 of *Lecture Notes in Computer Science*, pp. 459–474. Springer.

Ishibuchi H, Sakane Y, Tsukamoto N, Nojima Y (2010). "Simultaneous use of different scalarizing functions in MOEA/D." In *Proc. Genetic and Evolutionary Computation Conference (GECCO)*, pp. 519–526. ACM, New York, NY, USA.

Jiang S, Yang S (2016). "An improved multiobjective optimization evolutionary algorithm based on decomposition for complex Pareto fronts." *IEEE Trans. Cybernetics*, **46**(2), 421–437.

Li H, Zhang Q (2006). *MOEA/D for Continuous MOPs*. URL http://dces.essex.ac.uk/staff/qzhang/moead/MOEA-D-Continuous.rar.

Li H, Zhang Q (2007). *C++ Implementation of MOEA/D-DE for Continuous MOPs*. URL http://dces.essex.ac.uk/staff/qzhang/code/MOEADDE/MOEA-D-DE.rar.

Li H, Zhang Q (2009). "Multiobjective Optimization Problems with Complicated Pareto Sets, MOEA/D and NSGA-II." *IEEE Trans. Evolutionary Computation*, **13**(2), 284–302.

Li K, Deb K, Zhang Q, Kwong S (2015a). "An Evolutionary Many-Objective Optimization Algorithm Based on Dominance and Decomposition." *IEEE Trans. Evolutionary Computation*, **19**(5), 694–716.

Li K, Fialho A, Kwong S, Zhang Q (2014a). "Adaptive operator selection with bandits for a multiobjective evolutionary algorithm based on decomposition." *IEEE Trans. Evolutionary Computation*, **18**(1), 114–130.

Li K, Kwong S, Zhang Q, Deb K (2015b). "Interrelationship-based selection for decomposition multiobjective optimization." *IEEE Trans. Cybernetics*, **25**(10), 2076–2088.

Li K, Zhang Q, Kwong S, Li M, Wang R (2014b). "Stable matching-based selection in evolutionary multiobjective optimization." *IEEE Trans. Evolutionary Computation*, **18**(6), 909–923.

Li Y, Zhou A, Zhang G (2014c). "An MOEA/D with Multiple Differential Evolution Mutation Operators." In *Proc. IEEE Congress on Evolutionary Computation (CEC)*, pp. 397–404. Beijing, China.

Liu W (2006). *Java Implementation of MOEA/D for Continuous MOPs*. URL http://dces.essex.ac.uk/staff/qzhang/moead/MOEA-D-Continuous.rar.

López-Ibáñez M, Dubois-Lacoste J, Cáceres LP, Birattari M, Stützle T (2016). "The irace package: Iterated racing for automatic algorithm configuration." *Operations Research Perspectives*, **3**, 43–58.

Mersmann O (2014). *mco: Multiple Criteria Optimization Algorithms and Related Functions*. R package version 1.0-15.1, URL https://CRAN.R-project.org/package=mco.

Miettinen K (1999). *Nonlinear Multiobjective Optimization*. Kluwer Academic Publishers, Boston.

Mohammadi A, Omidvar M, Li X (2012). "Reference point based multi-objective optimization through decomposition." In *Proc. IEEE Congress on Evolutionary Computation (CEC)*, pp. 1–8.

Naval P (2013). *MOPSOCD: Multi-objective Particle Swarm Optimization with Crowding Distance*. R package version 0.5.1, URL https://CRAN.R-project.org/package=mopsocd.

Nebro A, Durillo J, Vergne M (2015). "Redesigning the jMetal Multi-Objective Optimization Framework." In *Proc. Conference on Genetic and Evolutionary Computation (GECCO)*, pp. 1093–1100. URL http://jmetal.github.io/jMetal/.

Novomestky F (2008). *goalprog: Weighted and lexicographical goal programming and optimization*. R package version 1.0-2.

Passos A (2016). *moko: Multi-Objective Kriging Optimization*. R package version 1.0.0, URL https://CRAN.R-project.org/package=moko.

Pilát M, Neruda R (2015). "Incorporating user preferences in MOEA/D through the coevolution of weights." In *Proc. Genetic and Evolutionary Computation Conference (GECCO)*, pp. 727–734. Madrid, Spain.

Price K, Storn R, Lampinen J (2005). *Differential Evolution: A Practical Approach to Global Optimization*. 1st edition. Springer.

Qi Y, Ma X, Liu F, Jiao L, Sun J, Wu J (2013). "MOEA/D with Adaptive Weight Adjustment." *Evolutionary Computation*, **22**(2), 231–264.

Qi Y, Ma X, Liu F, Jiao L, Sun J, Wu J (2014). "MOEA/D with Adaptive Weight Adjustment." *Evolutionary Computation*, **22**(2), 231–264.

Runarsson T, Yao X (2000). "Stochastic ranking for constrained evolutionary optimization." *IEEE Trans. Evolutionary Computation*, **4**(3), 284–294.

Sato H (2014). "Inverted PBI in MOEA/D and its Impact on the Search Performance on Multi and Many-Objective Optimization." In *Proc. Genetic and Evolutionary Computation Conference (GECCO)*, pp. 645–652. Vancouver, BC, Canada.

Sato H (2015). "Analysis of Inverted PBI and Comparison with Other Scalarizing Functions in Decomposition Based MOEAs." *Journal of Heuristics*, **21**(6), 819–849.

Scheffé H (1958). "Experiments with mixtures." *Journal of Royal Statistical Society, Series B*, **20**(2), 344–360.

Singh HK, Isaacs A, Ray T (2011). "A Pareto corner search evolutionary algorithm and dimensionality reduction in many-objective optimization problems." *IEEE Trans. Evolutionary Computation*, **15**(4), 539–556.

Storn R, Price K (1997). "Differential Evolution - A Simple and Efficient Heuristic for global Optimization over Continuous Spaces." *Journal of Global Optimization*, **11**(4), 341–359.

Tan Y, Jiao Y, Li H, Wang X (2012). "A modification to MOEA/D-DE for multiobjective optimization problems with complicated Pareto sets." *Information Sciences*, **213**, 14–38.

Trivedi A, Srinivasan D, Sanyal K, Ghosh A (2016). "A Survey of Multiobjective Evolutionary Algorithms based on Decomposition." *IEEE Transactions on Evolutionary Computation*, **21**(3), 440–462.

Tsou CS (2013). *nsga2R: Elitist Non-dominated Sorting Genetic Algorithm based on R*. R package version 1.0, URL https://CRAN.R-project.org/package=nsga2R.

Wang R, Zhang T, Guo B (2013). "An enhanced MOEA/D using uniform directions and a pre-organization procedure." In *Proc. IEEE Congress on Evolutionary Computation*, pp. 2390–2397. Cancún, Mexico.

Wang Z, Zhang Q, Gong M, Zhou A (2014). "A replacement strategy for balancing convergence and diversity in MOEA/D." In *Proc. IEEE Congress on Evolutionary Computation (CEC)*, pp. 2132–2139. Beijing, China.

Wanner EF, Guimarães FG, Takahashi RHC, Fleming PJ (2008). "Local Search with Quadratic Approximations into Memetic Algorithms for Optimization with Multiple Criteria." *Evolutionary Computation*, **16**(2), 185–224.

Ying WQ, He WP, Huang YX, Li DT, Wu Y (2016). "An Adaptive Stochastic Ranking Mechanism in MOEA/D for Constrained Multiobjective Optimization." In *2016 International Conference on Information System and Artificial Intelligence*, pp. 514–518.

Zhang Q, Li H (2007). "MOEA/D: A Multiobjective Evolutionary Algorithm Based on Decomposition." *IEEE Trans. Evolutionary Computation*, **11**(6), 712–731.

Zhang Q, Liu W, Li H (2009a). *C++ and Matlab Implementation of MOEA/D-DRA for Continuous MOPs*. URL http://dces.essex.ac.uk/staff/qzhang/MOEAcompetition/CEC09final/code/ZhangMOEADcode/.

Zhang Q, Liu W, Li H (2009b). "The performance of a new version of MOEA/D on CEC09 unconstrained MOP test instances." In *Proc. IEEE Congress on Evolutionary Computation (CEC)*. Institute of Electrical and Electronics Engineers (IEEE).

Zitzler E, Thiele L, Laumanns M, Fonseca C, Fonseca V (2003). "Performance assessment of multiobjective optimizers: An analysis and review." *IEEE Trans. Evolutionary Computation*, **7**(2), 117–132.

Affiliation:

Felipe Campelo, Lucas S. Batista
Operations Research and Complex Systems Laboratory,
Department of Electrical Engineering,
Universidade Federal de Minas Gerais,
Belo Horizonte 31270-010, Brazil
E-mail: fcampelo@ufmg.br,lusoba@ufmg.br
URL: http://orcslab.cpdee.ufmg.br/

Claus Aranha
Department of Computer Sciences,
Faculty of Systems Information Engineering,
University of Tsukuba,
Tennodai 1-1-1, Tsukuba City, Ibaraki, Japan, 305-8550
E-mail: caranha@cs.tsukuba.ac.jp
URL: http://conclave.cs.tsukuba.ac.jp

Journal of Statistical Software http://www.jstatsoft.org/
published by the Foundation for Open Access Statistics http://www.foastat.org/
MMMMMM YYYY, Volume VV, Issue II *Submitted:* yyyy-mm-dd
doi:10.18637/jss.v000.i00 *Accepted:* yyyy-mm-dd