SN Computer Science (2020) 1:247
https://doi.org/10.1007/s42979-020-00265-1

SURVEY ARTICLE

On Algorithmic Descriptions and Software Implementations for Multi-objective Optimisation: A Comparative Study

Shahin Rostami[1,2] · **Ferrante Neri**[3] · **Kiril Gyaurski**[2]

Received: 11 June 2020 / Accepted: 21 July 2020 / Published online: 4 August 2020

Abstract
Multi-objective optimisation is a prominent subfield of optimisation with high relevance in real-world problems, such as engineering design. Over the past 2 decades, a multitude of heuristic algorithms for multi-objective optimisation have been introduced and some of them have become extremely popular. Some of the most promising and versatile algorithms have been implemented in software platforms. This article experimentally investigates the process of interpreting and implementing algorithms by examining multiple popular implementations of three well-known algorithms for multi-objective optimisation. We observed that official and broadly employed software platforms interpreted and thus implemented the same heuristic search algorithm differently. These different interpretations affect the algorithmic structure as well as the software implementation. Numerical results show that these differences cause statistically significant differences in performance.

Keywords Multi-objective optimisation · Evolutionary algorithms · Optimisation software platforms

Introduction

Optimisation is a fundamental and multidisciplinary field that most generically aims to detect within a set of potential solution that one that satisfies the most one or more objectives. Within the plethora of possible optimisation methods, a clear distinction can be made between mathematical and heuristic optimisation:

- mathematical optimisation: methods that, under some hypotheses, are guaranteed to rigorously detect the optimum [48];
- heuristic optimisation: methods that, without requiring specific hypotheses on the problem, search for a solution that is close enough to the optimum [1–3, 13, 26].

✉ Ferrante Neri
ferrante.neri@nottingham.ac.uk

Shahin Rostami
shahin@polyra.com; srostami@bournemouth.ac.uk

[1] Data Science Lab, Polyra Limited, Bournemouth BH8 9JN, UK

[2] Department of Computing and Informatics, Bournemouth University, Bournemouth BH12 5BB, UK

[3] COL Laboratory, School of Computer Science, University of Nottingham, Nottingham NG8 1BB, UK

Algorithms of mathematical optimisation are often iterative methods that are rigorously justified and endowed with proofs of convergence [48]. The software implementation of an algorithm of mathematical optimisation is thus a straightforward decodification of mathematical formulas.

Algorithms of heuristic optimisation are software procedures usually described by means of words and formulas and justified by means of metaphors, see [26, 40]. To ensure that a heuristic algorithm can be understood and reproduced by the reader and the community, modern articles provide pseudocode of their proposed algorithms.

However, whilst the pseudocode of modern heuristics can be effective to communicate the general idea of the proposed algorithm, they often do not capture all the implementation details due to their complexity. Although ideas can be generally understood and re-implemented, due to the lack of mathematical rigour, a misinterpretation of an idea can be implemented into an algorithm which still performs reasonably well on an optimisation problem. This is likely to be an experience common to any person who has attempted to implement a program based on the idea of a colleague.

The present article discusses the topic of potential ambiguity in heuristic optimisation and how algorithmic descriptions could be subject to misinterpretation with a specific reference to multi-objective problems. We chose this subfield since multi-objective problems are naturally hard to

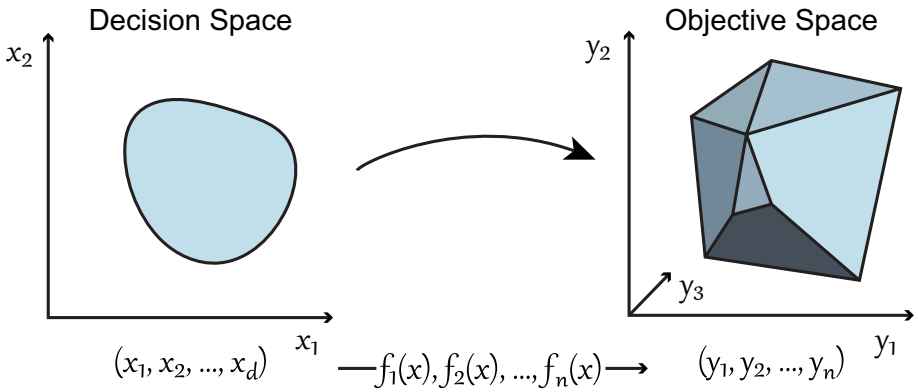


Fig. 1 Illustration of the decision space and the objective space of a multi-objective optimisation problems

solve, e.g. to the difficulty of comparing candidate solutions belonging to the same non-dominated set, and heuristics are often fairly complex. More specifically, we pose the following question

Is the process of implementation of a multi-objective algorithm from its description straightforward and unambiguous?

To address this point we have performed an extensive experimental study. We have focussed on three popular algorithmic frameworks:

- Non-dominated sorting genetic algorithm II (NSGA-II) [11].
- Generalized differential evolution 3 (GDE3) [34].
- Multi-objective evolutionary algorithm based on decomposition with dynamic resource allocation (MOEA/D-DRA) [57].

Each of these frameworks has been extensively implemented by multiple researchers and in various research software platforms such as jMetal [16]. We compared the behaviour and performance of the different implementations/interpretations of each framework. More specifically, five implementations of NSGA-II, five implementations of GDE3, and four implementations of MOEA/D-DRA are evaluated using the test functions provided by the ZDT, DTLZ and WFG test suites, see [12, 25, 59]. Furthermore, the source code of the tested implementations has been analysed to identify the differences in the implementation.

The article is organised as follows. Section "Basic Definitions and Notation" provides the reader with the basic definitions and notation which are used throughout the paper. Section "Algorithmic Frameworks in This Study" provides a description of NSGA-II, GDE3, and MOEA/D-DRA according to their original presentations but with the standardised notation used throughout the paper. Section "Software Implementations of the Algorithmic Frameworks" presents the software platforms where the three algorithmic frameworks under examination have been implemented. Section "Experimental Setup" describes the experimental design, including test problems, parameter settings, and methods used to perform the comparisons. Section "Experimental Results and Discussion" presents the numerical results of this study. Section "Differences in the Implementations" highlights the algorithmic and software engineering differences across the software platforms under consideration. Finally, section "Conclusion" presents the conclusion to this study.

Basic Definitions and Notation

Definition 1 (Multi-objective Optimisation Problem) A multi-objective optimisation problem can be defined as follows:

$$\begin{aligned} &\text{maximise/minimise} && \mathbf{y} = (f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_n(\mathbf{x})) \\ &\text{where} && \mathbf{x} = (x_1, \ldots, x_d) \in \mathbf{X} \subset \mathbb{R}^n \text{ (decision vector)} \\ & && \mathbf{y} = (y_1, \ldots, y_n) \in \mathbf{Y} \subset \mathbb{R}^k \text{ (objective vector).} \end{aligned}$$

The sets $\mathbb{R}^n$ and $\mathbb{R}^k$ represent the decision space and the objective space, respectively (Fig. 1). The set $\mathbf{X} \subseteq \mathbb{R}^n$ is also known as the feasible set and is associated with equality and inequality constraints.

The function $f : \mathbb{R}^n \to \mathbb{R}^k$ represents a transformation from the decision space $\mathbf{X}$ into the objective space $\mathbf{Y}$ that can be used to evaluate the quality of each solution. The image of $\mathbf{X}$ under the function f represents a subset of the function space known as the feasible set in the objective function space. It is denoted by $\mathbf{Y} = f(\mathbf{X})$ [37].

Definition 2 (Pareto dominance) Unlike single-objective optimisation in which the relation $\geq$ can be used to compare the quality of solutions, comparing solutions in multi-objective optimisation problems is not as straightforward (there is no order relation). A relation that is usually adopted to compare solutions to such problems is known as Pareto dominance.[1]

[1] The concept is named after Vilfredo Pareto, who used it in his studies of economic efficiency and income distribution.

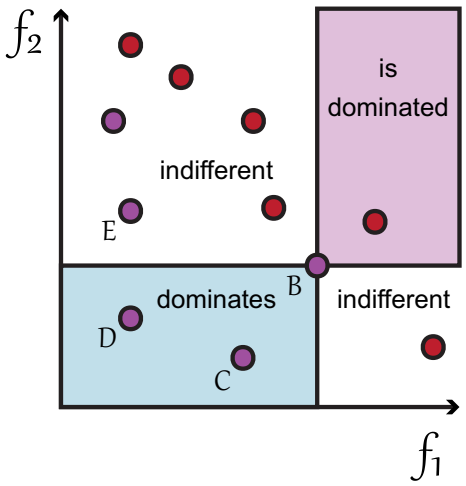


Fig. 2 Illustration of the Pareto dominance relation

Without loss of generality let us assume that all objectives need to be simultaneously maximised. For any two objective vectors

$$\mathbf{y^1} = \left(y_1^1, \ldots, y_n^1\right)$$
$$\mathbf{y^2} = \left(y_1^2, \ldots, y_n^2\right)$$

and the corresponding decision vectors $\mathbf{x^1}$ and $\mathbf{x^2}$, it can be said that $\mathbf{x^1} \succ \mathbf{x^2}$ ($\mathbf{x^1}$ dominates $\mathbf{x^2}$) if no component of $\mathbf{y^2}$ outperforms the corresponding component of $\mathbf{y^1}$, and at least one component of $\mathbf{y^1}$ outperforms the corresponding component of $\mathbf{y^2}$:

$$\forall k = 1 : n, y_k^1 \geq y_k^2$$
$$\exists k \ni y_k^1 > y_k^2.$$

In the event that $\mathbf{x^1}$ does not dominate $\mathbf{x^2}$ and $\mathbf{x^2}$ does not dominate $\mathbf{x^1}$, it can be said that the two vectors are indifferent $\left(\mathbf{x^1} \sim \mathbf{x^2}\right)$.

Figure 2 illustrates the concept of Pareto dominance. The blue rectangle represents the region of the objective space that dominates the objective vector **B**. All the objective vectors in that region have at least one objective value that outperforms the corresponding objective value of vector **B** and the other objective is bigger than or equal to the objective of vector **B**. The red rectangle contains the objective vectors that are dominated by vector **B**, and the vectors that are not in one of the two rectangles are indifferent to vector **B**.

Definition 3 (Pareto optimality) Pareto optimality is a concept that is based on Pareto dominance. A solution $\mathbf{x}^* \in \mathbf{X}$ is Pareto optimal if there is no other solution $\mathbf{x} \in \mathbf{X}$ that dominates it [37]. A Pareto optimal solution denotes that there does not exist another solution that can increase the quality of a given objective without decreasing the quality of at least one other objective [53].

Definition 4 (Pareto optimal set) Instead of having one optimal solution, multi-objective optimisation problems may have a set of Pareto optimal solutions. This set is also known as the Pareto optimal set. It is defined as follows:

$$P^* = \{\mathbf{x} \in \mathbf{X} \mid \nexists\, \mathbf{x}^* \in \mathbf{X} : \mathbf{x}^* \succeq \mathbf{x}\}.$$

Definition 5 (Pareto front) The Pareto front represents the image of the Pareto optimal set in the objective function space (Fig. 3). It can be defined as follows:

$$\mathrm{PF}^* = \{f(\mathbf{x}) | \mathbf{x} \in P^*\}.$$

Definition 6 (Approximation set) The main goal of multi-objective optimisation is to find the Pareto front of a given multi-objective optimisation problem. However, since Pareto fronts usually contain a large number of points, finding all of them might require an undefined amount of time and therefore a more practical solution is to find a good approximation of the Pareto front. This set approximating the Pareto is called approximation set. A good approximation set should be as close as possible to the actual Pareto front and should be uniformly spread over it, otherwise the obtained approximation of the Pareto front will not offer sufficient information for decision making [53].

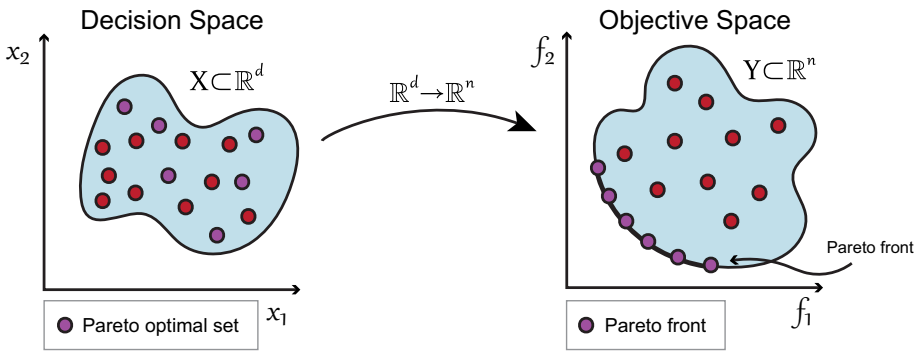


Fig. 3 Illustration of the Pareto optimal set and the Pareto front

Algorithmic Frameworks in This Study

The heuristic algorithms investigated in this study can all be considered under the umbrella name of evolutionary multi-objective optimisation (EMO) algorithms, see [31]. During the past decades, a wide variety of EMO algorithms have been proposed and applied to many different real-world problems.

Although EMO algorithms can differ by major aspects, they can still be considered to be members of the same family that is characterised by the general algorithmic structure outlined in Algorithm 1.

Algorithm 1 Main steps of a typical evolutionary algorithm

```
Initialise population with randomly generated candidate solutions;
Evaluate each solution in the population;
while Termination condition is not met do
    Select parents;
    Breed new individuals with the use of variation operators;
    Evaluate new individuals;
    Select individuals for the next generation;
end while
```

The following subsections introduce the three popular EMO algorithms under examination: the non-dominated sorting genetic algorithm II, the generalized differential evolution 3 algorithm, and the multi-objective evolutionary algorithm based on decomposition with dynamic resource allocation.

Non-dominated Sorting Genetic Algorithm II

The non-dominated sorting genetic algorithm II (NSGA-II) was proposed in [11] as an improvement to NSGA-I [49]. It addresses some of the main issues with the original version such as high-computational complexity of the non-dominated sorting algorithm, lack of elitism, and the need to specify a sharing parameter. Algorithm 2 illustrates the life cycle of NSGA-II.

Algorithm 2 NSGA-II

```
Initialise initial population P of size N with randomly generated solutions;
Sort the solutions using the fast non-dominated sorting algorithm as in Algorithm 3;
Apply selection, recombination and mutation operators to create offspring Q of size N;
while Termination condition is not met do
    Combine P and Q into a combined population R of size 2N;
    Sort the solutions using the fast non-dominated sorting algorithm as in Algorithm 3 to produce a set containing all non-dominated fronts (F_1 to F_m) of R;
    Initialise empty P_new;
    Set i = 1;
    while there is enough space in P_new for all members of F_i; do
        Calculate crowding-distance as in Algorithm 4 for the members of F_i;
        Add members of F_i to P_new;
        i = i + 1;
    end while
    Sort F_i in descending order using crowding-distance calculated as in Algorithm 4;
    Fill the remaining spaces in P_new with the best solutions of F_i;
    Create new offspring Q_new:
    Apply binary tournament selection operator based on the crowding-distance as in Algorithm 4
    Apply recombination operator;
    Apply mutation operator;
    P = P_new; Q = Q_new;
end while
```

NSGA-II starts by building a population of randomly generated candidate solutions. Each solution is then evaluated and assigned a fitness rank equal to its non-domination level (with 1 being the best level) using a fast non-dominated sorting algorithm. Binary tournament selection, recombination, and mutation operators are then applied to create an initial offspring population. In the original paper (as well as in many implementations thereafter), NSGA-II employs the simulated binary crossover (SBX) operator and polynomial mutation, see [19]. Then the generational loop begins. The parent population and the offspring population are combined. Since the best solutions from both the parent and offspring populations are included, elitism is ensured. The new combined population is then partitioned into fronts using the fast non-dominated sorting, see Algorithm 3.

The algorithm then iterates through the set of fronts (from best to worst), and adds their solutions to the population for the next generation until there is not enough space to accommodate all of the solutions of a given front. After the end of the procedure, if there are any places left in the new population, the solutions of the next front that could not be added are sorted in descending order based on their crowding distance and the best ones are added to the population until it is full. The new population is then used for selection, recombination and mutation to create an offspring population for the next generation. The algorithm uses the crowding distance during the selection process in order to maintain diversity in front by ensuring that each member stays a crowding distance apart which supports the algorithm in exploring the objective space [6]. Algorithm 4 shows how

Algorithm 3 Non-dominated sorting algorithm

INPUT population $\mathbf{P}$;
for $i = 1 : N$ **do**
  Initialise empty $\mathbf{S_i}$ set of points dominated by $\mathbf{x^i}$ and $n_i = 0$ points that dominate $\mathbf{x^i}$;
  for $j = 1 : N$ **do**
    if $\mathbf{x^i} \succ \mathbf{x^j}$ **then**
      Update the set $\mathbf{S_i} = \mathbf{S_i} \cup \mathbf{x^j}$;
    else if $\mathbf{x^j} \succ \mathbf{x^i}$ **then**
      Update the counter $n_i = n_i + 1$;
    end if
    if $n_i = 0$ that is $\mathbf{x^i}$ belongs to the first front **then**
      $\mathbf{F_1} = \mathbf{F_1} \cup \mathbf{x^i}$;
    end if
  end for
  Initialise n_j to the size of $\mathbf{S_i}$;
  $k = 1$;
  while $\mathbf{F_k} \neq \emptyset$ **do**
    $\mathbf{S_j} = \emptyset$;
    for $i = 1$: size of $\mathbf{F_k}$ **do**
      for $j = 1$: size of $\mathbf{S_i}$ **do**
        $n_j = n_j - 1$;
        if $n_j = 0$ **then**
          $\mathbf{S_j} = \mathbf{S_j} \cup \mathbf{x^j}$;
        end if
      end for
    end for
    $k = k + 1$;
    $\mathbf{F_k} = \mathbf{S_j}$;
  end while
end for
RETURN $\mathbf{F_1}, \mathbf{F_2}, \ldots \mathbf{F_m}$;

the crowding distance is calculated. The process is repeated until a termination condition is met.

Algorithm 4 Crowding algorithm

INPUT a non-dominated set $\mathbf{I}$;
Calculate the size of $\mathbf{I}$ and assign it to l;
Initialise the distance vector $\mathbf{d} = (d_1, d_2, \ldots, d_l) = (0, 0, \ldots, 0)$;
for $j = 1 : k$ **do**
  Sort $\mathbf{I}$ according to the objective f_j;
  $d_1 = d_l = \infty$ distance associated with the best and worst point;
  for i=2:l-1 **do**
    $d_i = d_i + \frac{\left|f_j(\mathbf{I(i+1)}) - f_j(\mathbf{I(i-1)})\right|}{\left|f_j(\mathbf{I(1)}) - f_j(\mathbf{I(l)})\right|}$;
  end for
end for
RETURN $\mathbf{d}$;

Generalized Differential Evolution 3

The generalized differential evolution 3 (GDE3) algorithm was proposed by Kukkonen and Lampinen in [34]. It is the third version of the GDE algorithm. GDE3 achieves better performance than previous versions by employing a growing offspring population that can accommodate twice as many solutions as there are in the parent population, and non-dominated sorting with pruning of non-dominated solutions to reduce the size of the population back to normal at the end of each generation. This technique improves the diversity of the obtained solution set and makes the algorithm less reliant on the selection of control parameters. The execution life cycle of the algorithm can be seen in Algorithm 5.

Algorithm 5 GDE3

Initialise the population $\mathbf{P}$ of size N with randomly generated solutions and evaluate their fitness;
while Termination condition is not met **do**
  Initialise empty offspring population with size $2N$;
  for $i = 1 : N$ **do**
    Randomly select three distinct parent solutions $\mathbf{x^{r1}}, \mathbf{x^{r2}}, \mathbf{x^{r3}}$ and a random variable index j_{rand};
    {Apply mutation}
    $\mathbf{u} = \mathbf{x^{r3}} + F\left(\mathbf{x^{r1}} - \mathbf{x^{r2}}\right)$ with F scale factor [1];
    {Apply binomial crossover [1]}
    for $j = 1 : d$ **do**
      if $rand[0, 1) < CR$ OR $j = j_{rand}$ (CR crossover rate) **then**
        $u_j^i = u_j^i$;
      else
        $u_j^i = x_j^i$;
      end if
    end for
    Evaluate the fitness $f_1(\mathbf{u}), f_2(\mathbf{u}), \ldots f_k(\mathbf{u})$;
    if the child solution $\mathbf{u}$ and the solution $\mathbf{x^i}$ of the parent population are indifferent to each other **then**
      Add both solutions to offspring population;
    else if $\mathbf{x^i} \prec \mathbf{u}$ **then**
      Add solution $\mathbf{x^i}$ to offspring population;
    else
      Add child solution $\mathbf{u}$ to offspring population;
    end if
  end for
  Sort the offspring population as in Algorithm 3;
  Choose the best N solutions for the next generation;
end while

Similar to other EMO algorithms, GDE3 starts by generating a population with N candidate solutions and evaluating their fitness. The main loop of the algorithm then begins. An empty offspring population with maximum size 2N is initialised. The algorithm then iterates through the initial population. At each position, differential evolution selection and differential evolution

variation operators are used to select parents and generate a child solution. The fitness of the child solution is then evaluated. It is then compared to the solution at the current position of the population. If the two solutions are indifferent to each other, both are added to the offspring solution, otherwise only the dominating solution is added. After the algorithm is finished iterating through the initial population, the offspring population is sorted and pruned to form the population for the next generation.

The GDE3 algorithm has been successfully applied to many real-world problems from various fields such as molecular biology [33] and electronics [22].

Multi-objective Evolutionary Algorithm Based on Decomposition with Dynamic Resource Allocation

The multi-objective evolutionary algorithm based on decomposition (MOEA/D) was proposed in [57]. MOEA/D uses a decomposition method to decompose a multi-objective problem into a number of scalar optimisation sub-problems by means of several weight vectors. A sub-problem is optimised using information from its neighbouring sub-problems, which leads to lower computational complexity than NSGA-II. Different approaches to decomposition exist in the literature. Some of the most popular ones are the weighted sum approach [18, 30], the Tchebycheff approach [38], and the normal-boundary intersection approach [9]. The authors of the algorithm employed the Tchebycheff approach in the paper in which it was introduced, and MOEA/D outperformed or performed similarly to NSGA-II on a number of test functions.

Over the years, many different versions of MOEA/D have been proposed, see [32, 35, 41, 58]. One of the versions that has become very popular is MOEA/D with dynamic resource allocation (MOEA/D-DRA [58]. In the original version of MOEA/D, all of the sub-problems are treated equally and all of them receive equal computational effort. However, different sub-problems might have different computational difficulties and, therefore, they require different amount of computational effort. Because of this, MOEA/D-DRA introduces dynamic resource allocation which allows the algorithm to assign different amounts of computational effort to different sub-problems. The amount of computational resources each sub-problem i gets is based on the computation of a utility value π^i.

The basic principles of MOEA/D-DRA are described in Algorithm 6 which highlights the 5-step structure characterising the framework as highlighted in the paper that originally proposed it, see [58]. It must be observed that MOEA/D-DRA is a relatively complex framework and contains multiple heuristic rules and a problem-specific internal parameter setting as it was designed for entry in the IEEE CEC 2009 competition. Hence, for the sake of brevity, we omitted the details of some of the formulas in Algorithm 6 and refer the original paper for the interested reader.

Algorithm 6 MOEA/D-DRA

{Step 1: Initialisation}
Compute the Euclidean distances between any two weight vectors and find the closest T weight vectors to each weight vector;
Generate an initial population of size N of the type $\mathbf{x^1}, \mathbf{x^2}, \ldots, \mathbf{x^N}$ by uniformly randomly sampling from the search space;
Initialise parameters – generation $gen = 0$; $\pi^i = 1$ for each sub-problem; $\mathbf{z} = (z_1, ..., z_n)$ (the ideal objective vector) where $z_i = \min\{f_i(\mathbf{x^1}), f_i(\mathbf{x^2}), \ldots, f_i(\mathbf{x^N})\}$;
{Step 2: Selection of Subproblems for Search}
while Termination condition is not met **do**
  Select sub-problems for search by using 10-tournament selection based on π^i value [58];
  {Step 3: Selection Reproduction and Update}
  for every selected sub-problem **do**
    Select mating/update range according to a randomised criterion [58];
    Apply differential evolution mutation and crossover operators to generate a child solution, see Algorithm 5;
    Perturb the child solution by means of a mutation;
    Evaluate child solution;
    if the child solution is not within the boundaries of the decision space **then**
      Randomly reset the solution within the boundaries;
    end if
    Update $\mathbf{z}$ with the newly found best objective values;
    Update of solutions according to the update algorithm [58];
  end for
  $gen = gen + 1$;
  {Step 5: Restart of π^i}
  if generation is multiplication of 50 **then**
    Update the utility value π^i of each sub-problem according to a heuristic rule [58];
  end if
end while
{Step 4: Stopping Criteria}

Software Implementations of the Algorithmic Frameworks

In this article, we focus on five popular software platforms and libraries for heuristic optimisation:

- jMetal-Java (version 4.5.2).
- jMetal-.NET (version 0.5).
- MOEA Framework (version 2.12)
- Platypus (GitHub commit 723ad6763abff99b-4f31305191b754c2c26867b0).
- PlatEMO (version 2.8.0).

jMetal-Java jMetal-Java is a java-based framework introduced by Durillo and Nebro in 2006 [16]. The main goal of the framework is to provide an easy to use tool for developing heuristics (and metaheuristics) for solving multi-objective optimisation problems. It also provides implementations of many state-of-the-art EMO algorithms, test problems and performance indicators, and has been used in several experiments described in the literature [44–47, 50].

In 2011, the authors of the framework started a project to implement jMetal using C# and in present days, the C# version provides almost all of the features that can be found in the Java version. The latter version is known as **jMetal.NET**.

MOEA The MOEA Framework library (here referred to also as MOEA for brevity) is a Java-based open-source library for multi-objective optimisation introduced by Hadka in 2011 [23]. Similarly to jMetal, MOEA also provides means to quickly design, develop and execute EMO algorithms. It supports genetic algorithms, differential evolution, genetic programming, and more. A number of state-of-the-art EMO algorithms, test problems, and quality indicators are provided out-of-the-box and have been used in multiple experiment studies detailed in the EMO literature [5, 15, 52].

Platypus Platypus is a python-based framework for evolutionary computing, whose focus lays on EMO algorithms. It was developed by Hadka and was introduced in 2015 [24]. It also provides implementations of several state-of-the-art algorithms, test problems and quality indicators, and also supports parallel processing. The algorithm implementations provided by the library have been used for various experiments in the EMO literature [4, 39].

PlatEMO is a MATLAB platform for evolutionary multi-objective optimisation [54]. It contains implementations of multiple state-of-the-art EMO algorithms as well as numerous test problems.

These algorithmic frameworks under consideration are present in these software platforms according to the following scheme

- NSGA-II → jMetal-Java, jMetal.NET, MOEA, Platypus, PlatEMO.
- GDE3 → jMetal-Java, jMetal.NET, MOEA, Platypus,PlatEMO.
- MOEA/D-DRA → jMetal-Java, jMetal.NET, MOEA, PlatEMO.

Table 1 Parameter configurations for the ZDT test functions

Problem	#Var (n)	#Obj (M)	HV reference vector
ZDT1	30	2	11, 11
ZDT2	30	2	11, 11
ZDT3	30	2	11, 11
ZDT4	10	2	300, 300
ZDT6	10	2	11, 11

Experimental Setup

The three algorithmic frameworks under consideration in the eleven above-mentioned implementations have been extensively tested on the test functions of the ZDT [59], DTLZ [12], and WFG [25] test suites.

The performance of the implementations is measured with the use of the Hypervolume indicator and Inverted Generational Distance+. In order to determine if there is a significant difference between different implementations of the same EMO algorithm, the experimental results have been tested with the Wilcoxon signed-rank test.

This section is organised as follows. The next subsection describes the test problems used in this study, and explicitly provides the reader with all the parameters associated with the problem followed by which the parameters that have been used by each EMO algorithmic framework are displayed. The final subsection describes in detail how the comparisons have been carried out.

Test Problems

ZDT The ZDT test suite was proposed by Zitzler, Deb and Thiele in 2000 [59]. It consists of six bi-objective synthetic test functions, five of which (ZDT1–ZDT4; ZD6) are real-coded and one (ZDT5) which is binary-coded. In this study, only the real-coded test problems were selected. ZDT5 was not selected due to its requirement for binary encoded problem variables. The parameter configuration of the ZDT test functions have been listed in Table 1.

DTLZ The DTLZ test suite was proposed by Deb, Thiele, Laumanns and Zitzler in 2002 [12]. It contains seven scalable multi-objective test functions with diverse characteristics such as non-convex, multimodal, and disconnected Pareto

Table 2 Parameter configurations for the DTLZ test functions

Problem	#Var (n)	#Obj (M)	HV reference vector
DTLZ1	7	2, 3	11, 11 when $M = 2$ 11, 11, 11 when $M = 3$
DTLZ2	12	2, 3	11, 11 when $M = 2$ 11, 11, 11 when $M = 3$
DTLZ3	12	2, 3	11, 11 when $M = 2$ 11, 11, 11 when $M = 3$
DTLZ4	12	2, 3	11, 11 when $M = 2$ 11, 11, 11 when $M = 3$
DTLZ5	12	2, 3	11, 11 when $M = 2$ 11, 11, 11 when $M = 3$
DTLZ6	12	2, 3	15, 15 when $M = 2$ 15, 15, 15 when $M = 3$
DTLZ7	22	2, 3	15, 15 when $M = 2$ 15, 15, 15 when $M = 3$

fronts. In this study, two and three objectives are considered for each problem, and the number of decision variables is set to 7, 12 and 22 for DTLZ1, DTLZ2-6, and DTLZ7, respectively. The parameter configurations for the DTLZ test functions have been listed in Table 2.

WFG The WFG tool-kit was introduced by Huband, Hingston, Barone and While in 2006 [25]. The tool-kit makes it possible to construct synthetic test problems which incorporate common characteristics of real-world problems such as variable linkage and separability. To demonstrate its functionality, the authors constructed nine test problems (WFG1-9) with scalable variables and problem objectives. The proposed test functions are referred to as the WFG test suite. In this study, two and three objectives are considered for each problem. The full parameter configurations for each test problem can be seen in Table 3.

Table 3 Parameter configurations for the WFG test functions

Parameter	Value
Number of objectives, M	2, 3
Number of variable, n	24
Number of position variables, k	$2(M-1)$
Number of distance vairables, l	$n-k$
HV reference vector when $M = 2$	11, 11
HV reference vector when $M = 3$	11, 11, 11

Table 4 Parameter configurations of the chosen NSGA-II implementations

NSGA-II	
Recombination	Simulated binary crossover + polynomial mutation
Crossover probability	0.9
Crossover distribution index	20
Mutation probability	1/number of decision variables
Mutation distribution index	20
Selection operator	Binary tournament
Population size	100
Generation count	500

Parameter Setting

The parameters used to configure the implementations of the selected algorithms can be seen in Tables 4, 5 and 6. The implementations of NSGA-II and MOEA/D-DRA are configured with the parameters proposed by the authors of the algorithms, while GDE3 is configured with the parameters used in the experiment described in [17]. The maximum

Table 5 Parameter configurations of the chosen GDE3 implementations

GDE3	
Recombination	Differential evolution variation
Crossover rate	0.1
Scaling factor	0.5
Selection operator	Differential evolution selection
Population size	100
Generation count	500

Table 6 Parameter configurations of the chosen MOEA/D-DRA implementations

MOEA/D-DRA	
Recombination	Differential evolution variation + polynomial mutation
Crossover rate	1
Scaling factor	0.5
Mutation probability	1/number of decision variables
Mutation distribution index	20
T (size of the neighbourhood)	0.1 × population size
n_r (maximum number of solutions replaced by each child solution)	0.01 × population size
Δ (mating probability)	0.9
Population size	600 when $M = 2$, 1000 when $M = 3$
Generation count	500

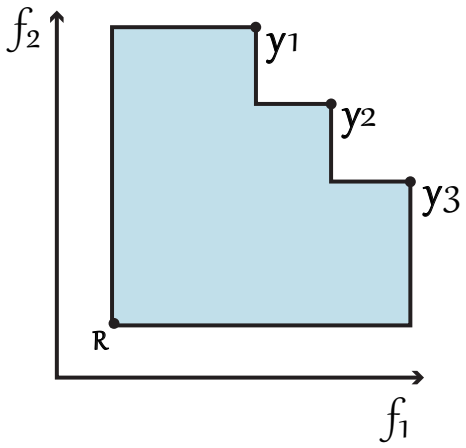


Fig. 4 Illustration of the HV indicator in two-dimensional space with three solutions

number of generations per execution is set to 500, with a sample size of 30 executions per test function.

Comparison Method

In the last decades, various performance indicators have been proposed to assess the performance of EMO algorithms, see [27]. The most popular performance indicator amongst the EMO society in recent years has been the Hypervolume (HV) indicator [42] and the Inverted Generational Distance+ (IGD$^+$), see [27, 29, 51].

The HV indicator is a performance metric proposed by Zitzler and Thiele in [60]. It measures the volume of the objective space dominated by an approximation set (Fig. 4). The HV indicator is a popular choice for researchers as it does not require knowledge about the true Pareto optimal front, which is an important factor when working with multi-objective optimisation problems that are yet to be solved [43].

A reference point is required for the calculation of the HV indicator. When the HV indicator is used to compare EMO algorithms, the reference point must be the same otherwise the results will not be accurate or comparable. Selecting the reference point is an important issue and the difficulty of selecting such a point increases with the number of objectives [36]. The reference points used in the experimental study are selected by constructing a vector of the worst objective values contained in the union of the approximation sets generated by the algorithm implementations chosen for comparison. They have been listed in Tables 1, 2, and 3.

The IGD is a classical metric used to assess the quality of a non-dominated set, see [7] on the basis of the study reported in [8]. Let us consider a set of reference points $Z = \{\mathbf{z^1}, \mathbf{z^2}, \ldots \mathbf{z^r}\}$ that approximate the Pareto set. Let us now consider the set of non-dominated points returned by an algorithm $X = \{\mathbf{x^1}, \mathbf{x^2}, \ldots, \mathbf{x^N}\}$ and the corresponding set of objective vectors $Y = \{\mathbf{y^1}, \mathbf{y^2}, \ldots, \mathbf{y^N}\}$. The IGD of the set Y is calculated as

$$\mathrm{IGD}_Z(Y) = \frac{1}{r}\left(\sum_{j=1}^{r} \min\{d(\mathbf{z^j}, \mathbf{y^k}) | \mathbf{y^k} \in Y\}\right)$$

where $d(\mathbf{z^j}, \mathbf{y^k})$ is the Euclidean distance between $\mathbf{z^j}$ and $\mathbf{y^k}$:

$$d(\mathbf{z^j}, \mathbf{y^k}) = \sqrt{\sum_{i=1}^{n} \left(z_i^j - y_i^k\right)^2},$$

where n is the dimensionality of the objective space.

For the sake of clarity, IGD is obtained from Z and Y. At first, for each element of Z, the Euclidean distances between the element of Z and all the elements of Y are calculated. Then minimal distance is chosen. The average sum of all the minimal distances is the desired IGD, see [28].

The IGD$^+$ corrects the IGD using the modified distance d^+ instead of the Euclidean distance, [29]:

$$d^+(\mathbf{z^j}, \mathbf{y^k}) = \sqrt{\sum_{i=1}^{n} \left(\max\{0, \left(z_i^j - y_i^k\right)\}\right)^2}$$

and then uses in the IGD$^+$ calculation

$$\mathrm{IGD}_Z^+(Y) = \frac{1}{r}\left(\sum_{j=1}^{r} \min\{d^+(\mathbf{z^j}, \mathbf{y^k}) | \mathbf{y^k} \in Y\}\right).$$

To determine whether there is significant difference between the results of different implementations of a given EMO algorithm, it is necessary to carry out a statistical test. In this paper, the mean hypervolume and IGD$^+$ results of the implementations used in the experiment are compared with the use of the Wilcoxon signed [14, 20, 21, 55, 61]to test for such statistical difference. The Wilcoxon test is a non-parametric test used to test for a difference in the mean (or median) of paired observations.

The results in tables are expressed as mean value $\pm$ and standard deviation σ. For all the tables presented in this study, we have used jMetal-Java implementations as the reference for the Wilcoxon test. This choice has been made on the basis of the practical consideration that it is the most complete framework amongst those considered in this article. In each column of the Tables, a "+" indicates that the implementation in that column outperforms the jMetal counterpart, a "−" indicates that the implementation is outperformed by the jMetal-Java implementation, an "=" indicates that the two implementations have a statistically indistinguishable performance.

Table 7 Hypervolume results (mean value and standard deviation) from 30 executions of five versions of NSGA-II on two and three objective test functions after 500 generations

M	Fn	JMetal-Java		JMetal-.NET			MOEA			Platypus			PlatEMO		
		Mean	±σ	Mean	±σ		Mean	±σ		Mean	±σ				
2	ZDT1	0.997190	2.22e−06	0.997186	4.21e−06	−	0.997191	2.95e−06	=	0.997194	2.92e−06	+	**0.997679**	1.78e−06	+
2	ZDT2	0.994437	3.58e−06	0.994427	8.03e−06	−	0.994437	3.32e−06	=	0.994436	1.57e−05	=	**0.995404**	1.96e−06	+
2	ZDT3	1.064254	1.05e−06	1.064247	2.97e−06	−	**1.064255**	1.06e−y06	=	1.064253	2.01e−05	−	0.995381	2.87e−04	−
2	ZDT4	0.999993	2.28e−06	0.001340	1.45e−07	−	0.997939	1.14e−03	−	0.001275	5.13e−05	−	**0.999995**	1.34e−06	+
2	ZDT6	0.971137	3.16e−05	0.971115	2.20e−05	−	0.971058	2.56e−05	−	0.969137	1.64e−05	−	**0.974079**	3.71e−06	+
2	DTLZ1	0.998950	2.26e−06	0.998952	2.35e−06	+	0.998953	2.23e−06	+	0.998943	3.98e−05	=	**0.999133**	1.67e−06	+
2	DTLZ2	0.993464	1.73e−06	0.993465	1.64e−06	+	0.993465	2.44e−06	+	0.993468	1.47e−06	+	**0.994599**	2.00e−06	+
2	DTLZ3	0.993365	5.45e−05	0.992642	3.57e−03	=	0.993399	6.97e−05	+	0.991258	9.54e−03	−	**0.994538**	3.57e−05	+
2	DTLZ4	**0.982215**	2.87e−02	0.970966	3.73e−02	=	0.965341	3.98e−02	=	0.962530	4.07e−02	=	0.974001	3.42e−02	=
2	DTLZ5	0.993464	1.69e−06	0.993465	1.49e−06	+	–	–	–	–	–	–	**0.994600**	1.57e−06	+
2	DTLZ6	0.996376	9.14e−05	0.996419	1.04e−04	+	–	–	–	–	–	–	**0.997095**	9.70e−07	+
2	DTLZ7	0.841893	7.93e−07	0.839215	1.44e−02	=	0.841893	9.25e−07	+	0.833861	2.41e−02	−	**0.856622**	7.39e−07	+
2	WFG1	0.912400	1.94e−02	0.895798	1.70e−02	−	0.829089	1.88e−02	−	0.823324	1.82e−02	−	**0.973273**	5.49e−03	+
2	WFG2	0.931930	2.85e−02	0.932030	2.86e−02	=	0.933659	2.93e−02	=	0.927973	2.71e−02	=	**0.937988**	2.59e−02	=
2	WFG3	0.965088	3.39e−04	0.964833	2.66e−04	−	0.964986	2.99e−04	=	0.964575	4.72e−04	−	**0.971215**	2.91e−04	+
2	WFG4	0.946902	1.83e−04	0.946506	2.55e−04	−	0.946851	1.74e−04	=	0.946569	3.33e−04	−	**0.956092**	1.59e−04	+
2	WFG5	0.934980	2.90e−03	0.932576	2.37e−03	−	0.934178	2.52e−03	=	0.931425	1.86e−03	−	**0.945010**	2.04e−03	+
2	WFG6	0.940206	8.85e−04	0.939823	9.90e−04	=	0.939898	1.21e−03	=	0.940034	1.09e−03	=	**0.949922**	8.98e−04	+
2	WFG7	0.947362	6.84e−05	0.947117	1.42e−04	−	0.947264	7.92e−05	−	0.947104	1.46e−04	−	**0.956580**	4.35e−05	+
2	WFG8	0.930734	9.00e−04	0.929948	7.02e−04	−	0.930606	6.01e−04	=	0.929801	8.25e−04	−	**0.941628**	3.34e−04	+
2	WFG9	0.923818	6.94e−03	0.928121	6.16e−03	=	0.928068	6.56e−03	+	0.926594	7.21e−03	=	**0.936021**	6.63e−03	+
3	DTLZ1	0.999889	4.75e−04	0.999940	1.03e−04	+	0.999971	2.21e−05	+	0.999949	9.40e−05	+	**0.999983**	4.47e−07	+
3	DTLZ2	0.999430	9.26e−05	0.999374	1.09e−04	−	0.999424	1.02e−04	=	0.999418	1.39e−04	=	**0.999648**	3.36e−06	+
3	DTLZ3	0.997873	3.75e−03	0.998144	3.28e−03	=	0.994959	1.15e−02	=	0.998618	2.05e−03	=	**0.999638**	9.88e−06	+
3	DTLZ4	0.999503	4.29e−05	0.996438	1.62e−02	−	0.993477	2.26e−02	=	0.997531	3.01e−03	−	**0.999651**	2.07e−06	+
3	DTLZ5	0.991003	4.56e−06	0.991006	3.19e−06	+	–	–	–	–	–	–	**0.992545**	3.81e−06	+
3	DTLZ6	0.995073	7.24e−05	0.994988	1.72e−04	=	–	–	–	–	–	–	**0.995967**	1.30e−06	+
3	DTLZ7	0.816021	1.88e−04	0.805150	2.70e−02	−	0.813374	1.43e−02	=	0.762949	5.18e−02	−	**0.831159**	1.30e−02	+
3	WFG1	0.812572	1.51e−02	0.794213	1.28e−02	−	0.767795	4.87e−03	−	0.766936	6.01e−03	−	**0.929729**	8.31e−03	+
3	WFG2	**0.961418**	4.82e−02	0.949519	5.09e−02	−	0.940065	5.22e−02	−	0.943264	5.25e−02	−	0.956154	4.73e−02	=
3	WFG3	0.927309	1.09e−03	0.925396	1.79e−03	−	0.927293	1.22e−03	=	0.927133	1.48e−03	=	**0.939482**	1.49e−03	+
3	WFG4	0.967684	3.07e−03	0.958141	5.17e−03	−	0.966589	2.63e−03	=	0.965583	4.16e−03	=	**0.977761**	1.15e−03	+
3	WFG5	0.955707	2.21e−03	0.950895	2.92e−03	−	0.955307	1.89e−03	=	0.956195	1.64e−03	=	**0.964817**	2.30e−03	+
3	WFG6	0.933526	2.89e−03	0.928945	2.47e−03	−	0.933333	2.95e−03	=	0.931800	3.00e−03	−	**0.970124**	2.03e−03	+
3	WFG7	0.969758	3.47e−03	0.965371	3.41e−03	−	0.970401	3.09e−03	=	0.967066	4.62e−03	−	**0.981323**	4.26e−04	+

Table 7 (continued)

M	Fn	JMetal-Java		JMetal-.NET			MOEA			Platypus			PlatEMO		
		Mean	±σ	Mean	±σ		Mean	±σ		Mean	±σ				
3	WFG8	0.941907	1.85e−03	0.935578	2.47e−03	−	0.941613	1.80e−03	=	0.940570	1.91e−03	−	**0.960703**	1.68e−03	+
3	WFG9	0.902004	3.43e−03	0.904241	6.30e−03	=	0.903221	4.86e−03	=	0.901762	5.98e−03	=	**0.941214**	4.92e−03	+

The best results are highlighted in bold

Experimental Results and Discussion

Table 7 displays the HV results for the five implementations of NSGA-II. Table 8 displays the HV results for the five implementations of GDE3. Table 9 displays the HV results for the five implementations of MOEA/D-DRA.

It can be observed that the HV results for DTLZ4 and DTLZ5 are missing in Tables 7, 8, 9. The lack of those results is due to a malfunctioning of the MOEA and Platypus platforms: there is a systematic crash in the system when any of the three algorithms under examination is executed. Hence, we decided to omit these results.

Numerical results in Tables 7, 8, 9 show that the implementations of the same algorithms on different platforms leads to dramatically different results. For example, with reference to Table 7, the NSGA-II implementations on jMetal-Java and MOEA platforms display very different performance, which is statistically different in about half of the problems. A macroscopic difference displayed in Tables 7, 8 and 9 is that PlatEMO significantly outperforms the other platforms on most of the problems, especially for NSGA-II and MOEA/D-DRA.

The results on this extensive data set confirm the intuition shown on the study on NSGA-II implementations published in [56]. It was suggested that different implementations of the same algorithm might perform significantly differently. This means that newly proposed algorithms might perform better than one implementation of the benchmark algorithm but worse than another one.

When proposing a new algorithm, however, researchers typically use only one implementation of the state-of-the-art algorithms chosen for benchmarking. Algorithms are often considered as conceptual paradigms which are associated with their performance. On the contrary, implementations are tasks of secondary importance which consist of communicating the conceptual paradigm expressed in equations and pseudocode to a machine. Furthermore, the use of multiple implementations while running tests is often overlooked since, besides being perceived as a repeated operation, it can be time consuming and onerous. However, the results in this paper show that different implementations of the same algorithm may perform significantly differently. Thus, the choice of the implementation/software platform may likely have an impact on the conclusion of research papers in algorithmics and on the performance of newly proposed algorithms.

To provide the reader with a graphical representation of the differences in HV performance across the platforms under consideration, we plotted the evolution of the HV indicator. Figure 5 depicts the three sets of trends for NSGA-II in Fig. 5a, GDE3 in Fig. 5b, and MOEA/D-DRA in Fig. 5c, respectively.

Table 8 Hypervolume results (mean value and standard deviation) from 30 executions of five versions of GDE3 on two and three objective test functions after 500 generations

M	Fn	JMetal-Java		JMetal-.NET			MOEA			Platypus			PlatEMO		
		Mean	±σ	Mean	±σ		Mean	±σ		Mean	±σ				
2	ZDT1	0.997206	1.72e−07	0.997207	1.56e−07	+	**0.997207**	1.78e−07	+	0.997207	1.87e−07	+	0.995013	1.19e−03	−
2	ZDT2	0.994452	1.75e−07	0.994452	1.16e−07	+	0.994452	1.84e−07	+	**0.994452**	1.57e−07	+	0.979690	3.60e−03	−
2	ZDT3	1.064261	1.32e−06	**1.064262**	9.56e−07	+	1.064261	4.20e−06	=	1.064261	3.31e−06	=	0.995480	1.17e−04	−
2	ZDT4	0.999982	7.89e−05	**0.999996**	2.61e−10	+	0.999754	3.12e−04	=	0.999967	1.10e−04	−	0.948769	1.03e−02	−
2	ZDT6	0.971197	1.01e−07	0.971197	9.88e−08	+	0.971197	7.28e−08	+	0.969243	2.45e−07	−	**0.974086**	3.17e−07	+
2	DTLZ1	0.998956	4.13e−08	**0.998956**	4.28e−08	+	0.998956	3.33e−07	+	0.998956	4.65e−08	+	0.716349	1.21e−01	−
2	DTLZ2	0.993477	2.47e−07	0.993477	2.31e−07	+	0.993477	1.93e−07	+	0.993477	2.15e−07	+	**0.994590**	1.85e−06	+
2	DTLZ3	0.993477	2.89e−07	**0.993477**	1.87e−07	+	0.825374	2.66e−01	−	0.993477	2.61e−07	+	0.103977	2.33e−01	−
2	DTLZ4	0.993477	2.45e−07	0.993477	2.55e−07	+	0.993477	1.90e−07	+	0.993477	2.18e−07	+	**0.994589**	1.93e−06	+
2	DTLZ5	0.993477	2.62e−07	0.993477	1.95e−07	+	–	–	–	–	–	–	**0.994590**	1.63e−06	+
2	DTLZ6	0.996492	1.29e−07	0.996492	1.58e−07	+	–	–	–	–	–	–	**0.997101**	1.02e−07	+
2	DTLZ7	0.841898	1.41e−07	**0.841898**	1.82e−07	+	0.841898	4.02e−07	+	0.841898	3.05e−07	+	0.839794	1.57e−02	=
2	WFG1	0.946891	5.81e−03	**0.947201**	5.30e−03	=	0.864956	4.58e−03	−	0.861957	4.10e−03	−	0.830400	1.03e−03	−
2	WFG2	0.970557	2.75e−05	0.970567	2.29e−05	=	0.970642	2.70e−05	+	0.970561	3.34e−05	=	**0.970710**	1.17e−03	=
2	WFG3	0.966467	2.47e−05	0.966459	1.50e−05	=	0.966479	2.27e−05	=	0.966458	2.36e−05	=	**0.969572**	3.26e−04	+
2	WFG4	0.946184	4.93e−04	0.946191	4.14e−04	=	0.946239	6.05e−04	=	**0.946263**	4.40e−04	=	0.943574	1.04e−03	−
2	WFG5	0.929449	1.95e−03	0.928625	1.89e−03	=	0.929074	1.95e−03	=	0.929105	1.60e−03	=	**0.940135**	1.60e−03	+
2	WFG6	0.942651	8.16e−04	0.942781	8.38e−04	=	0.942224	1.39e−03	=	0.942957	1.14e−03	=	**0.943832**	3.47e−03	=
2	WFG7	0.947792	4.95e−06	0.947791	4.91e−06	=	0.947790	5.44e−06	=	0.947789	4.90e−06	=	**0.954914**	1.88e−04	+
2	WFG8	0.931746	6.92e−04	0.931791	5.49e−04	=	0.931779	6.37e−04	=	0.931740	7.42e−04	=	**0.939153**	1.34e−03	+
2	WFG9	0.919240	3.38e−03	0.919333	3.58e−03	=	0.919280	3.52e−03	=	0.919272	3.49e−03	=	**0.929479**	1.10e−04	+
3	DTLZ1	0.999969	3.72e−05	0.999970	3.20e−05	=	0.999855	4.71e−04	=	**0.999972**	2.89e−05	=	0.973884	1.75e−02	−
3	DTLZ2	0.999540	2.65e−06	0.999541	3.41e−06	=	0.999540	3.84e−06	=	0.999543	3.15e−06	+	**0.999624**	3.05e−06	+
3	DTLZ3	**0.999007**	1.44e−03	0.997113	1.04e−02	=	0.963073	6.14e−02	−	0.996697	1.02e−02	=	0.174285	3.00e−01	−
3	DTLZ4	0.999540	2.60e−06	0.999540	3.91e−06	=	0.999543	3.01e−06	+	0.999543	2.59e−06	+	**0.999626**	3.42e−06	+
3	DTLZ5	0.991026	5.25e−07	0.991026	3.37e−07	+	–	–	–	–	–	–	**0.992530**	3.62e−06	+
3	DTLZ6	0.995138	1.83e−07	0.995139	1.56e−07	+	–	–	–	–	–	–	**0.995974**	1.74e−07	+
3	DTLZ7	**0.816346**	2.05e−04	0.816193	2.04e−04	−	0.816343	1.78e−04	=	0.816224	2.37e−04	=	0.772090	2.44e−02	−
3	WFG1	**0.834692**	1.23e−02	0.829531	1.74e−02	=	0.792034	1.68e−03	−	0.791979	1.17e−03	−	0.779200	1.89e−03	−
3	WFG2	0.995530	2.23e−04	**0.995569**	2.12e−04	=	0.995402	2.70e−04	=	0.995490	2.48e−04	=	0.975030	2.70e−03	−
3	WFG3	0.927005	1.42e−03	0.926749	1.13e−03	=	0.927073	9.81e−04	=	**0.927196**	9.67e−04	=	0.910117	3.93e−03	−
3	WFG4	0.970317	8.96e−04	0.970364	7.92e−04	=	0.970516	8.53e−04	=	**0.970662**	8.49e−04	=	0.949862	2.42e−03	−
3	WFG5	0.953822	1.91e−03	0.952843	2.63e−03	=	**0.954345**	1.37e−03	=	0.952487	1.85e−03	−	0.948643	4.16e−03	−
3	WFG6	0.940310	2.74e−03	0.940090	2.66e−03	=	0.941017	2.44e−03	=	0.940965	2.59e−03	=	**0.959881**	2.36e−03	+
3	WFG7	0.975923	4.02e−04	0.975951	3.30e−04	=	**0.975972**	4.45e−04	=	0.975886	3.84e−04	=	0.943961	3.51e−03	−

Table 8 (continued)

M	Fn	JMetal-Java		JMetal-.NET			MOEA			Platypus			PlatEMO		
		Mean	±σ	Mean	±σ		Mean	±σ		Mean	±σ				
3	WFG8	**0.948081**	1.41e−03	0.947963	1.20e−03	=	0.947649	1.13e−03	=	0.948060	1.43e−03	=	0.916487	5.16e−03	−
3	WFG9	0.901862	3.42e−03	0.901895	1.50e−03	=	0.901471	1.48e−03	=	0.901480	1.56e−03	=	**0.932399**	2.56e−03	+

The best results are highlighted in bold

The HV trends in Fig. 5 show that for each algorithm the various software implementations of the same algorithm may lead to different results. It is worth noting that jMetal-Java and jMetal-.NET lead to similar results. This is overall expected since jMetal-.NET is based on jMetal-Java and thus relies on the same interpretation of the algorithms. However, Fig. 5a shows that for NSGA-II, the two jMetal versions lead to different results. Furthermore, Fig. 5 shows that the performance on PlatEMO implementations are dramatically different.

Tables 10, 11, and 12 display the IGD$^+$ results for NSGA-II, GDE3, and MOEA/D-DRA, respectively. For this set of results, we excluded PlatEMO since it does not have IGD$^+$ among the metrics available. The IGD$^+$ results clearly show that different software implementations can lead to extremely different performance values. For example, Table 10 shows that for ZDT4 the jMetal IGD$^+$ values are about 100 times lower than those achieved by the MOEA and Platypus platforms. We conclude that an expert who analyses these results without knowing that they are allegedly produced by the same algorithm would think that they are produced by conceptually different algorithmic frameworks.

To give a graphical representation of the IGD$^+$ results, Fig. 6 displays the IGD$^+$ trends of the three algorithmic and the four software platforms under investigation in the case of WFG1 with two objectives. Figure 6a shows the results of the four NSGA-II software implementations, Fig. 6b shows the results of the four GDE3 software implementations and Fig. 6c shows the results of the three MOEA/D-DRA software implementations.

The results on IGD$^+$ values confirm what was shown for HV values: the various trends appear distinct with a similarity between the two jMetal platforms.

Differences in the Implementations

To understand how the original papers, [11, 34], and [58] have been interpreted by the authors of the software platforms, we analysed the codes and highlighted the major differences that impacted on the performance of the algorithms. We classified these differences according to their type: (1) algorithmic differences, i.e. different interpretations of the text or pseudocode in the original publication; (2) software engineering differences, i.e. implementation choices that are not part of the algorithmic structure but are still important elements that affect the runs and the results.

Furthermore, there is a difference in programming style between PlatEMO and the other packages which appears to heavily affect the performance for the algorithms and problems considered in this study. In PlatEMO, the software parameters are mostly pre−defined without opportunity

Table 9 Hypervolume results (mean value and standard deviation) from 30 executions of the four versions available of MOEA/D-DRA on two and three objective test functions after 500 generations

M	Fn	JMetal-Java		JMetal-.NET			MOEA			PlatEMO		
		Mean	±σ	Mean	±σ		Mean	±σ				
2	ZDT1	0.996866	1.84e−04	0.996867	1.11e−04	=	0.996832	1.88e−04	=	**0.997717**	9.02e−08	+
2	ZDT2	0.993775	4.08e−04	0.993905	2.79e−04	=	0.994002	3.50e−04	+	**0.995440**	3.23e−07	+
2	ZDT3	1.063422	4.12e−04	1.063270	4.93e−04	=	**1.063735**	2.46e−04	+	0.995342	1.58e−07	−
2	ZDT4	0.999803	3.06e−04	0.999888	2.15e−04	=	0.001215	4.16e−05	−	**0.999997**	1.22e−08	+
2	ZDT6	0.971207	5.91e−07	0.971207	1.25e−06	=	0.971207	4.55e−07	=	**0.974105**	1.81e−09	+
2	DTLZ1	0.998963	1.01e−06	0.998963	1.15e−06	=	0.998953	4.40e−05	−	**0.999145**	3.29e−08	+
2	DTLZ2	0.993485	7.90e−05	0.993453	1.85e−04	=	0.993500	1.50e−07	+	**0.994630**	7.53e−08	+
2	DTLZ3	0.970621	4.70e−02	0.361852	2.39e+00	=	0.976622	5.89e−02	=	**0.994629**	8.91e−07	+
2	DTLZ4	0.993485	6.63e−05	0.993451	1.42e−04	=	0.993500	4.80e−07	+	**0.994630**	3.35e−08	+
2	DTLZ5	0.993496	1.78e−05	0.993478	1.13e−04	=	–	–	–	**0.994630**	9.08e−08	+
2	DTLZ6	0.996504	2.37e−06	0.996503	2.60e−06	=	–	–	–	**0.997112**	1.36e−08	+
2	DTLZ7	0.825755	3.21e−02	0.823066	3.39e−02	=	0.820431	3.55e−02	=	**0.849299**	2.20e−02	+
2	WFG1	0.824744	3.98e−03	0.824837	5.55e−03	=	0.817420	1.18e−03	−	**0.881350**	9.64e−03	+
2	WFG2	0.968088	8.05e−04	0.968307	6.51e−04	=	0.968042	7.81e−04	=	**0.968779**	1.05e−02	+
2	WFG3	0.965243	1.76e−04	0.965217	2.96e−04	=	0.965193	2.01e−04	=	**0.965377**	1.04e−02	+
2	WFG4	0.936578	1.47e−03	0.936554	1.98e−03	=	0.939591	1.28e−03	+	**0.946310**	3.92e−03	+
2	WFG5	0.929594	1.67e−04	0.929791	9.96e−04	=	0.929648	1.18e−04	=	**0.939962**	7.63e−05	+
2	WFG6	0.934807	4.64e−03	0.933698	3.76e−03	=	0.933351	3.31e−03	=	**0.942863**	3.24e−03	+
2	WFG7	0.946899	3.77e−04	0.946981	1.50e−04	=	0.947348	8.87e−05	+	**0.955748**	9.98e−04	+
2	WFG8	0.930136	9.01e−04	0.930263	9.38e−04	=	0.931090	9.09e−04	+	**0.933218**	7.37e−03	+
2	WFG9	0.919549	3.16e−03	0.921628	5.43e−03	=	0.922045	5.44e−03	=	**0.932398**	4.38e−03	+
3	DTLZ1	0.999982	5.68e−08	0.999982	4.68e−08	+	0.999982	3.47e−08	−	**0.999986**	1.85e−08	+
3	DTLZ2	0.999584	4.23e−07	0.999584	3.45e−07	+	0.999584	2.99e−07	+	**0.999688**	2.63e−07	+
3	DTLZ3	0.999575	2.19e−05	0.999089	2.45e−03	=	0.999465	6.24e−04	−	**0.999688**	3.17e−07	+
3	DTLZ4	0.999579	2.21e−05	0.999573	3.02e−05	=	**0.999585**	2.90e−07	+	0.999516	9.08e−04	−
3	DTLZ5	0.991053	8.78e−07	0.991053	4.88e−07	+	–	–	–	**0.992588**	5.95e−08	+
3	DTLZ6	0.995155	2.79e−06	0.995155	1.77e−06	=	–	–	–	**0.995987**	3.69e−09	+
3	DTLZ7	0.789870	4.77e−02	0.797819	3.96e−02	=	0.787326	5.66e−02	=	**0.824620**	2.49e−02	+
3	WFG1	0.759856	7.37e−04	0.759663	6.46e−04	=	0.759832	7.98e−04	=	**0.785441**	5.32e−03	+
3	WFG2	**0.993395**	4.23e−04	0.993382	5.58e−04	=	0.993113	5.06e−04	−	0.990767	1.41e−03	−
3	WFG3	0.930985	6.73e−04	**0.931089**	7.48e−04	=	0.929534	7.99e−04	−	0.923832	4.46e−03	−
3	WFG4	0.961653	1.29e−03	0.961995	1.91e−03	=	0.963593	1.64e−03	+	**0.969399**	2.41e−03	+
3	WFG5	0.959014	5.25e−04	0.958856	5.63e−04	=	–	–	–	**0.967131**	3.42e−04	+
3	WFG6	0.934261	7.18e−03	0.935157	8.01e−03	=	0.933285	6.64e−03	=	**0.961945**	2.17e−03	+
3	WFG7	0.975550	5.11e−04	0.975435	4.95e−04	=	0.976631	4.32e−04	+	**0.979292**	1.61e−03	+
3	WFG8	0.948587	2.31e−03	0.948924	2.58e−03	=	**0.949332**	2.12e−03	=	0.935644	1.37e−02	−
3	WFG9	0.913475	7.02e−03	0.913057	5.60e−03	=	0.913668	6.49e−03	=	**0.946494**	5.80e−03	+

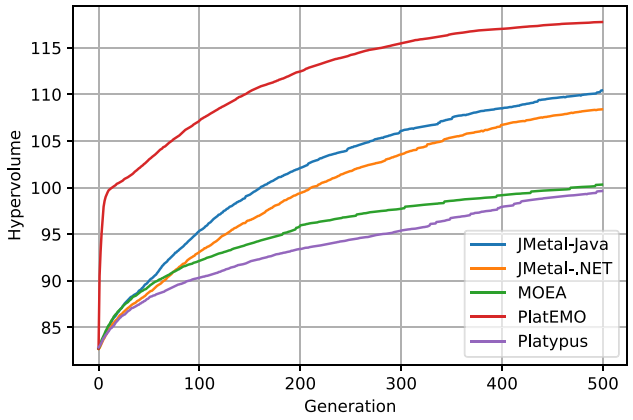
The best results are highlighted in bold

for user-specification outside of modifying the source code directly. Many of the parameters appear to be pre−tuned and orientated towards high performance.

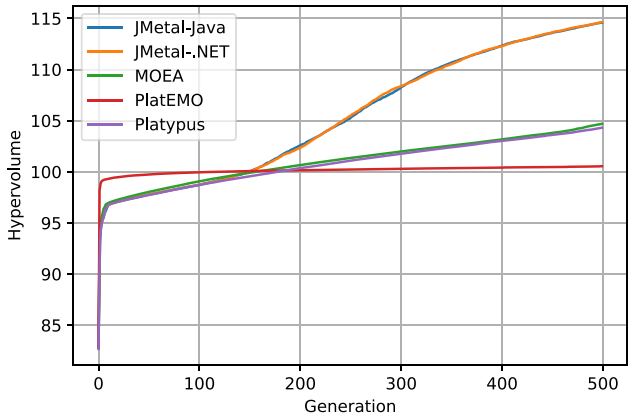
In the following subsections, we highlight some of the software differences that we were able to observe by comparing the five software platforms under consideration.

NSGA-II

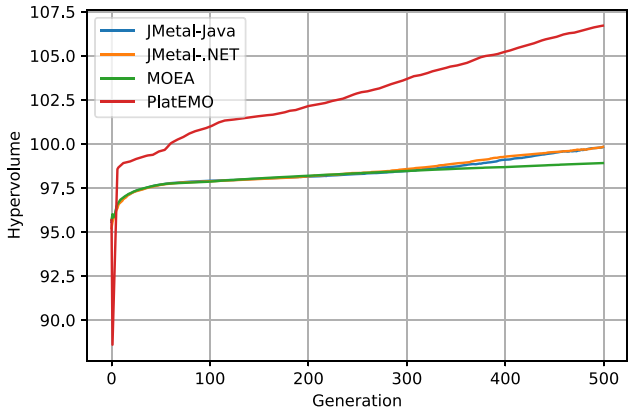
Simulated binary crossover (SBX) [10] is a popular and established recombination operator which was motivated by the binary crossover operators which were popular in early genetic algorithms. SBX enables the single−point crossover of two real-coded parent solutions to produce an offspring solution through exploitation of the existing information,



(a) HV trend for NSGA-II implementations



(b) HV trend for GDE3 implementations



(c) HV trend for MOEA/D-DRA implementations

Fig. 5 HV trends for the three algorithms and five software platforms under consideration for the WFG1 problem with two objectives

with the benefit of using a probability distribution for selecting offspring solutions. In particular, NSGA-II employs the SBX operator when operating on problems consisting of real-coded decision variables, and any difference in its implementation will impact the convergence and search pressure throughout the optimisation process.

The software platforms under considerations give different interpretations of SBX and thus propose slightly different algorithms. In SBX, a random number determines whether SBX performs the recombination according to the crossover probability, but if this criterion is not satisfied then the offspring solution directly inherits the decision variable from the first parent. However, in jMetal, the offspring solution also inherits directly from the first parent if the absolute difference between the decision variable from both parents is greater than a pre−defined precision value. This extra swap mechanism does not occur in the MOEA Framework. The SBX operation in Platypus follows the same logic as MOEA Framework. An empirical comparison using identical decision variables (for two parent solutions) and pre−generated random numbers proved a difference in the output from both jMetal and MOEA. The results are presented in Table 13.

An example of a software engineering difference is in the epsilon (EPS) value used for precision in the frameworks, where jMetal-Java and jMetal-.NET both use $1.0e-14$, MOEAFramework uses $1.0e-1$, and Platypus uses 2.220446049250313*e*-1 (provided by "sys.float_info.epsilon"). Additionally, the jMetal-.NET implementation of NSGA-II had hard-coded the seed used by the random number generators. This meant that every execution on a problem would produce identical results, so this was corrected for the experiments in this paper.

Finally, another difference is in the use random seeds, while jMetal implementations of NSGA-II make use of some hard-coded seeds, MOEA and Platypus generate new random numbers at each occurrence.

GDE3

The four software platform present a subtle different interpretation of the crossover in GDE3. To decide which design variables are copied from the parent solution, jMetal platforms use the condition $\text{rand}[0,1) < CR$ as mentioned in the original paper [34] whereas MOEA and Platypus platforms reinterpret the expressions as $\text{rand}[0,1) \leq CR$. Although this difference does not impact most of the comparisons, it is performed many times in each run. Thus, it is likely to generate some different offspring solutions and modify the behaviour of the algorithm. For example, if we consider

$$\mathbf{x^i} = (0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0)$$
$$\mathbf{x^{r1}} = (1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0)$$
$$\mathbf{x^{r2}} = (10.0, 9.0, 8.0, 7.0, 6.0, 5.0, 4.0, 3.0, 2.0, 1.0)$$
$$\mathbf{x^{r3}} = (2.0, 4.0, 6.0, 8.0, 10.0, 12.0, 14.0, 16.0, 18.0, 20.0)$$

with $F = CR = 0.5$, the mutant vector is

Table 10 Inverted generational distance+ results (mean value and standard deviation) from 30 executions of the four versions available of NSGA-II on two and three objective test functions after 500 generations

M	Fn	jMetal-Java		jMetal-.NET			MOEA			Platypus		
		Mean	$\pm\sigma$	Mean	$\pm\sigma$		Mean	$\pm\sigma$		Mean	$\pm\sigma$	
2	ZDT1	0.003377	1.42e−04	0.003277	1.38e−04	−	0.003218	1.51e−04	−	**0.003051**	1.13e−04	−
2	ZDT2	0.002917	1.10e−04	0.002870	9.52e−05	=	0.002810	1.08e−04	−	**0.002696**	8.46e−05	−
2	ZDT3	0.001878	1.28e−04	0.001852	7.83e−05	=	0.001800	1.10e−04	−	**0.001752**	1.18e−04	−
2	ZDT4	0.003948	5.80e−04	**0.003918**	8.85e−04	=	0.552275	2.96e−01	+	0.473648	3.60e−01	+
2	ZDT6	0.002276	3.10e−04	**0.002068**	1.95e−04	−	0.002539	2.28e−04	+	0.046240	5.22e−05	+
2	DTLZ1	0.002037	3.03e−04	0.001855	3.02e−04	−	**0.001751**	2.80e−04	−	0.002047	3.69e−04	=
2	DTLZ2	0.002636	1.14e−04	0.002577	1.03e−04	−	0.002565	1.10e−04	−	**0.002402**	1.04e−04	−
2	DTLZ3	0.009267	3.84e−03	0.046039	1.80e−01	=	**0.007021**	4.85e−03	−	0.018784	9.75e−03	+
2	DTLZ4	**0.005169**	1.19e−02	0.009836	1.55e−02	=	0.012164	1.65e−02	=	0.013319	1.69e−02	=
2	DTLZ5	0.002632	9.79e−05	**0.002589**	1.12e−04	=	–	–	–	–	–	–
2	DTLZ6	0.016740	1.20e−02	**0.011361**	1.38e−02	=	–	–	–	–	–	–
2	DTLZ7	0.002555	1.28e−04	0.014401	6.43e−02	+	**0.002448**	1.69e−04	−	0.038190	1.07e−01	+
2	WFG1	**0.414873**	1.28e−01	0.543237	1.22e−01	+	0.973705	9.40e−02	+	1.016263	1.11e−01	+
2	WFG2	0.028480	1.41e−02	**0.027585**	1.44e−02	=	0.028867	1.63e−02	=	0.029839	1.37e−02	=
2	WFG3	**0.131865**	1.96e−03	0.132146	1.55e−03	=	0.132060	1.87e−03	=	0.132634	2.45e−03	=
2	WFG4	0.014480	1.29e−03	0.014899	1.24e−03	=	0.013973	1.06e−03	=	**0.013069**	1.02e−03	−
2	WFG5	0.068611	7.36e−04	0.068542	3.43e−04	=	0.068361	3.15e−04	=	**0.067875**	3.63e−04	−
2	WFG6	0.055747	5.97e−03	0.056628	6.35e−03	=	0.057441	8.16e−03	=	**0.054451**	7.75e−03	=
2	WFG7	**0.011921**	6.18e−04	0.012346	1.06e−03	=	0.012152	6.26e−04	=	0.012435	7.72e−04	+
2	WFG8	0.080751	3.70e−03	0.080975	3.05e−03	=	0.081074	2.92e−03	=	**0.068803**	4.90e−03	−
2	WFG9	0.076791	4.67e−02	**0.046155**	4.19e−02	−	0.049713	4.31e−02	−	0.057687	4.67e−02	=
3	DTLZ1	**0.018335**	1.39e−03	0.018857	1.81e−03	=	0.018475	1.95e−03	=	0.018796	2.59e−03	=
3	DTLZ2	0.034503	1.30e−03	0.035556	1.92e−03	=	0.034435	1.70e−03	=	**0.033640**	1.30e−03	−
3	DTLZ3	0.341154	5.79e−01	0.323672	5.51e−01	=	**0.060478**	4.76e−02	−	0.229931	3.36e−01	=
3	DTLZ4	**0.025321**	2.77e−03	0.032272	2.60e−02	+	0.035631	3.63e−02	=	0.042822	3.07e−02	=
3	DTLZ5	0.003060	1.71e−04	**0.002944**	1.35e−04	−	–	–	–	–	–	–
3	DTLZ6	**0.009721**	9.32e−03	0.019904	2.17e−02	=	–	–	–	–	–	–
3	DTLZ7	**0.042005**	2.12e−03	0.067606	6.20e−02	+	0.048011	3.30e−02	=	0.185682	1.77e−01	+
3	WFG1	**1.091738**	9.31e−02	1.187917	8.26e−02	+	1.361926	1.87e−02	+	1.369839	2.93e−02	+
3	WFG2	**0.112497**	1.88e−02	0.118701	2.06e−02	=	0.123861	2.02e−02	+	0.120128	1.93e−02	=
3	WFG3	0.075182	1.07e−02	0.080174	1.46e−02	=	**0.072751**	1.10e−02	=	0.080798	1.27e−02	=
3	WFG4	0.160639	9.79e−03	0.184817	1.25e−02	+	**0.160405**	8.38e−03	=	0.162036	8.88e−03	=
3	WFG5	0.181463	7.31e−03	0.194805	6.91e−03	+	0.181728	6.65e−03	=	**0.179287**	7.06e−03	=
3	WFG6	0.219942	1.42e−02	0.236777	1.09e−02	+	**0.219577**	1.63e−02	=	0.226750	1.52e−02	=
3	WFG7	**0.125801**	4.09e−03	0.138686	6.70e−03	+	0.126355	5.89e−03	=	0.145508	1.86e−02	+
3	WFG8	0.301378	7.83e−03	0.312256	8.48e−03	+	**0.295862**	7.36e−03	−	0.308839	7.92e−03	+
3	WFG9	0.259963	1.54e−02	**0.247956**	3.27e−02	=	0.255812	2.12e−02	=	0.262606	2.39e−02	+
					17 =, **16** ≠			**18** =, **15** ≠			**14** =, **19** ≠	

The best results are highlighted in bold

$$\mathbf{u} = \mathbf{x^{r3}} + F\left(\mathbf{x^{r1}} - \mathbf{x^{r2}}\right)$$
$$= (-2.5, 0.5, 3.5, 6.5, 9.5, 12.5, 15.5, 18.5, 21.5, 24.5).$$

After a crossover where we have synthetically ensured that CR will equal 0.5 in some instances (highlighted) through pre-generated random numbers, jMetal and MOEA platforms produced the following, respectively,

$$\mathbf{u_{jMetal}} = (\mathbf{0.5}, 0.5, 1.5, 6.5, \mathbf{2.5}, 12.5, \mathbf{3.5}, 4.0, 4.5, 24.5)$$
$$\mathbf{u_{MOEA}} = (\mathbf{-2.5}, 0.5, 1.5, 6.5, \mathbf{9.5}, 12.5, \mathbf{15.5}, 4.0, 4.5, 24.5)$$

which then affects the final results.

Table 11 Inverted generational distance+ results (mean value and standard deviation) from 30 executions of the four versions available of GDE3 on two and three objective test functions after 500 generations

M	Fn	jMetal-Java		jMetal-.NET			MOEA			Platypus		
		Mean	$\pm\sigma$	Mean	$\pm\sigma$		Mean	$\pm\sigma$		Mean	$\pm\sigma$	
2	ZDT1	0.002450	1.50e−05	0.002397	1.14e−05	−	**0.002391**	1.72e−05	−	0.002401	1.72e−05	−
2	ZDT2	0.002241	1.03e−05	0.002195	9.74e−06	−	0.002199	9.53e−06	−	**0.002191**	9.17e−06	−
2	ZDT3	0.001432	3.13e−05	0.001424	3.05e−05	=	**0.001410**	3.13e−05	−	0.001421	3.01e−05	=
2	ZDT4	0.006560	2.21e−02	**0.002404**	1.89e−05	−	0.069640	8.62e−02	=	0.010605	3.07e−02	+
2	ZDT6	0.001453	5.00e−05	0.001416	4.65e−05	−	**0.001414**	6.28e−05	−	0.045895	4.01e−06	+
2	DTLZ1	0.001313	6.79e−06	**0.001291**	6.74e−06	−	0.001300	4.22e−05	−	0.001291	6.85e−06	−
2	DTLZ2	0.001887	3.92e−05	**0.001849**	3.40e−05	−	0.001849	2.46e−05	−	0.001852	3.67e−05	−
2	DTLZ3	0.001852	4.14e−05	0.001857	4.27e−05	=	3.012354	3.27e+00	+	**0.001851**	3.66e−05	=
2	DTLZ4	0.000364	4.99e−06	0.000356	3.34e−06	−	0.000357	4.04e−06	−	**0.000355**	4.38e−06	−
2	DTLZ5	0.001891	3.80e−05	**0.001846**	3.28e−05	−	–	–	–	–	–	–
2	DTLZ6	0.001895	3.33e−05	**0.001855**	2.93e−05	−	–	–	–	–	–	–
2	DTLZ7	0.001968	2.90e−05	0.001922	2.34e−05	−	0.001922	2.24e−05	−	**0.001915**	2.70e−05	−
2	WFG1	0.228657	3.76e−02	**0.227381**	3.52e−02	=	0.806974	3.32e−02	+	0.828995	3.25e−02	+
2	WFG2	0.003483	2.17e−04	0.003416	2.06e−04	=	**0.002913**	2.12e−04	−	0.003499	2.61e−04	=
2	WFG3	0.121656	2.20e−04	0.121666	1.66e−04	=	**0.121540**	1.99e−04	=	0.121721	2.38e−04	=
2	WFG4	0.011072	9.31e−04	0.010932	1.03e−03	=	**0.010680**	1.11e−03	=	0.011116	9.81e−04	=
2	WFG5	0.066166	3.53e−05	**0.066120**	2.15e−04	=	0.066184	4.98e−05	=	0.066132	1.80e−04	=
2	WFG6	0.038300	5.51e−03	0.037432	5.64e−03	=	0.041142	9.35e−03	=	**0.036238**	7.66e−03	=
2	WFG7	0.006454	1.35e−04	**0.006419**	1.21e−04	=	0.006426	1.49e−04	=	0.006493	1.03e−04	=
2	WFG8	**0.070646**	1.82e−03	0.071214	2.31e−03	=	0.071809	2.56e−03	=	0.071123	2.14e−03	=
2	WFG9	0.107254	2.34e−02	**0.106928**	2.46e−02	=	0.106981	2.41e−02	−	0.107079	2.40e−02	=
3	DTLZ1	0.034044	6.97e−02	0.032288	6.36e−02	=	0.115323	2.32e−01	=	**0.028852**	5.43e−02	=
3	DTLZ2	0.027628	1.08e−03	0.027207	6.20e−04	=	0.027533	1.06e−03	=	**0.027202**	1.09e−03	=
3	DTLZ3	**0.176975**	3.77e−01	0.263009	7.98e−01	=	2.357332	1.92e+00	+	0.400710	8.32e−01	=
3	DTLZ4	0.025929	2.55e−03	0.026120	3.02e−03	=	0.025413	2.26e−03	=	**0.023907**	2.04e−03	−
3	DTLZ5	0.001992	4.85e−05	**0.001964**	4.07e−05	−	–	–	–	–	–	–
3	DTLZ6	0.001899	6.78e−05	**0.001862**	6.39e−05	−	–	–	–	–	–	–
3	DTLZ7	0.034267	1.70e−03	0.034084	1.97e−03	=	**0.033592**	1.77e−03	=	0.034248	2.78e−03	=
3	WFG1	**0.981482**	7.38e−02	1.016024	1.03e−01	=	1.236083	1.12e−02	+	1.238163	8.73e−03	+
3	WFG2	0.089175	1.90e−02	0.090306	1.29e−02	=	**0.087679**	1.50e−02	=	0.094205	1.74e−02	=
3	WFG3	0.101445	1.87e−02	0.103290	1.41e−02	=	0.099624	1.48e−02	=	**0.095262**	1.29e−02	=
3	WFG4	0.148576	8.73e−03	0.150187	7.13e−03	=	0.149353	8.21e−03	=	**0.143996**	7.07e−03	=
3	WFG5	0.160164	4.38e−03	**0.158253**	2.52e−03	=	0.159672	4.98e−03	=	0.159302	4.45e−03	=
3	WFG6	0.179826	1.56e−02	0.182267	1.28e−02	=	**0.175210**	1.17e−02	=	0.176543	1.61e−02	=
3	WFG7	0.103230	4.77e−03	0.103144	4.34e−03	=	0.102714	4.80e−03	=	**0.102019**	4.03e−03	=
3	WFG8	**0.274296**	6.91e−03	0.274619	5.80e−03	=	0.275527	5.55e−03	=	0.275152	7.30e−03	=
3	WFG9	**0.257143**	1.23e−02	0.258771	6.30e−03	=	0.259092	6.93e−03	=	0.257639	6.93e−03	=
					26 $=, 11 \neq$			**19** $=, 14 \neq$			**22** $=, 11 \neq$	

The best results are highlighted in bold

MOEA/D-DRA

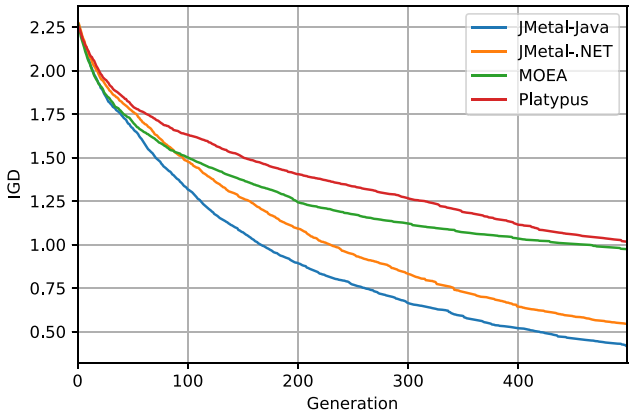
The first algorithmic difference in the MOEA/D-DRA implementations is in the way the decision whether solutions from the neighbourhood or from the entire population should be selected during the mating process. The comparison of a random number against a threshold δ is coded as $\text{rand}(0, 1] < \delta$ in jMetal and $\text{rand}(0, 1] \leq \delta$ in MOEA, respectively.

The main algorithmic difference is in the way sub-problems are selected. In MOEA sub-problem's indices are randomly selected from the entire list of problems. Thus, the MOEA Framework allows the same subproblem to be selected multiple times. The jMetal platforms implement a

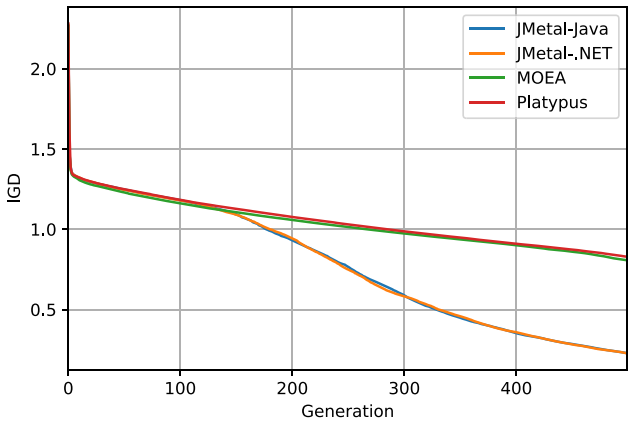
Table 12 Inverted generational distance+ results (mean value and standard deviation) from 30 executions of the three versions available of MOEA/D-DRA on two and three objective test functions after 500 generations

M	Fn	jMetal-Java		jMetal-.NET			MOEA		
		Mean	$\pm\sigma$	Mean	$\pm\sigma$		Mean	$\pm\sigma$	
2	ZDT1	0.006595	1.32e−03	0.006679	1.19e−03	=	**0.004333**	1.48e−03	−
2	ZDT2	0.005752	9.93e−04	0.005409	9.23e−04	=	**0.002561**	1.02e−03	−
2	ZDT3	0.006207	9.71e−04	0.006633	1.27e−03	=	**0.002178**	1.01e−03	−
2	ZDT4	0.038047	6.15e−02	**0.027144**	4.47e−02	=	0.887277	2.91e−01	+
2	ZDT6	0.001626	3.15e−05	0.001616	2.62e−05	−	**0.000204**	1.63e−06	−
2	DTLZ1	0.001850	9.86e−05	0.001862	1.09e−04	=	**0.000288**	2.75e−05	−
2	DTLZ2	0.002931	2.64e−05	0.002929	2.13e−05	=	**0.000479**	1.21e−05	−
2	DTLZ3	0.425098	8.31e−01	1.915119	5.47e+00	=	**0.381288**	1.10e+00	=
2	DTLZ4	0.000568	2.81e−05	0.000572	2.38e−05	=	**0.000123**	1.32e−05	−
2	DTLZ5	0.002929	1.82e−05	**0.002923**	2.69e−05	=	–	–	–
2	DTLZ6	0.002752	3.43e−06	**0.002751**	3.05e−06	=	–	–	–
2	DTLZ7	**0.076527**	1.42e−01	0.088456	1.51e−01	=	0.096806	1.59e−01	=
2	WFG1	**1.103865**	3.96e−02	1.107224	4.25e−02	=	1.181690	1.15e−02	+
2	WFG2	0.024700	4.93e−03	0.025237	3.74e−03	=	**0.019033**	4.75e−03	−
2	WFG3	0.131586	7.77e−04	0.131628	1.09e−03	=	**0.126314**	8.84e−04	−
2	WFG4	0.069671	8.09e−03	0.066713	6.18e−03	=	**0.066275**	5.49e−03	=
2	WFG5	0.068714	2.00e−04	0.068835	3.58e−04	=	**0.065288**	2.81e−04	−
2	WFG6	**0.094490**	3.24e−02	0.102064	2.62e−02	=	0.102580	2.30e−02	=
2	WFG7	0.015902	5.52e−04	0.015563	3.25e−04	−	**0.007938**	7.53e−04	−
2	WFG8	0.082901	5.86e−03	**0.081415**	5.36e−03	=	0.083355	5.57e−03	=
2	WFG9	0.106005	2.53e−02	0.091840	3.75e−02	=	**0.086776**	3.92e−02	−
3	DTLZ1	0.014400	2.23e−04	0.014357	2.00e−04	=	**0.005766**	8.10e−05	−
3	DTLZ2	0.029547	3.84e−04	0.029515	3.63e−04	=	**0.010989**	1.01e−04	−
3	DTLZ3	**0.036820**	9.46e−03	0.116035	3.80e−01	=	0.040343	1.39e−01	+
3	DTLZ4	0.020006	3.72e−03	0.019135	4.57e−03	=	**0.004806**	4.69e−04	−
3	DTLZ5	0.004480	1.22e−04	**0.004475**	8.71e−05	=	–	–	–
3	DTLZ6	0.003606	9.28e−05	**0.003601**	8.72e−05	=	–	–	–
3	DTLZ7	0.131559	1.54e−01	**0.106514**	1.19e−01	=	0.125522	2.12e−01	−
3	WFG1	1.443795	6.85e−03	1.443376	5.39e−03	=	**1.421833**	4.25e−03	−
3	WFG2	0.089475	1.22e−02	0.091613	1.39e−02	=	**0.049289**	7.47e−03	−
3	WFG3	0.052677	4.38e−03	0.052140	6.33e−03	=	**0.047311**	6.11e−03	−
3	WFG4	0.236131	5.75e−03	0.232860	8.93e−03	=	**0.168144**	6.48e−03	−
3	WFG5	0.176035	1.73e−03	0.176957	1.91e−03	=	**0.110753**	1.35e−03	−
3	WFG6	0.230581	2.95e−02	0.226527	3.40e−02	=	**0.181344**	2.94e−02	−
3	WFG7	0.124108	2.44e−03	0.124000	3.14e−03	=	**0.072679**	3.61e−03	−
3	WFG8	0.277606	9.76e−03	0.273707	1.13e−02	=	**0.223948**	1.01e−02	−
3	WFG9	0.237548	3.29e−02	0.238016	2.77e−02	=	**0.176508**	3.22e−02	−
					35 $=, \mathbf{2} \neq$			**5** $=, \mathbf{28} \neq$	

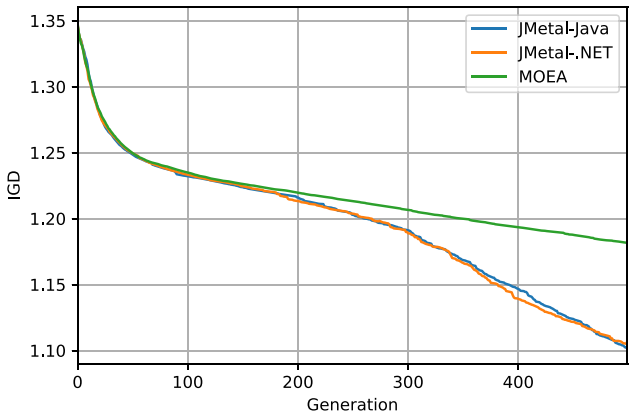
The best results are highlighted in bold



(a) IGD$^+$ trend for NSGA-II implementations



(b) IGD$^+$ trend for GDE3 implementations



(c) IGD$^+$ trend for MOEA/D-DRA implementations

Fig. 6 IGD$^+$ trends for the three algorithms and four software platforms under consideration for the WFG1 problem with two objectives

very different logic: the selected sub-problems are "blacklisted" so that they cannot be selected twice.

Moreover, MOEA shuffles the indices before returning them whereas jMetal does not.

Conclusion

This article performs an experimental study on various interpretations and implementations of well-known algorithms. Numerical results clearly show that three popular heuristic algorithms for multi-objective optimisation, that is, NSGA-II, GDE3, and MOEA/D-DRA, have been subject to various interpretations by four popular software platforms, that is, the two versions of jMetal, MOEA Framework, Platypus, and PlatEMO. As expected, each of these platforms make software engineering decisions which may affect the results, such as the use of different precision values or random number generator. More importantly, the platforms propose different logical implementations of the algorithms. Effectively, these platforms run different algorithms under the same name.

Algorithmic and software engineering differences are evident in the presented Results section. Numerical results show that different implementations of the same algorithm lead to statistically different performance across the experimental setup used in this study. According to our position, the differences are due to two concurrent reasons. The first is that several articles in the field may not be explicit about some important details of an algorithm. This may be due to the complexity of the algorithm and the decision by authors to use references from other articles instead of detailed explanations to enhance the readability of the proposed method. The second is that software platforms may creatively change some of the details because they may be more convenient or for consistency across the platform, e.g. the use of "$\leq$" in MOEA and "$<$" in jMetal.

We believe that the description of complex software and its explanation to a human is a difficult task which can easily lead to misinterpretations. Furthermore, we believe that this will have consequences not only in frameworks contributed by and for the scientific community, but also for

Table 13 The offspring solutions generated by SBX implementations in jMetal.NET and MOEA Framework using identical inputs

Parent 1	1.0	2.0	3.0	4.0	5.0	6.0	7.0	8.0	9.0	10.0
Parent 2	10.0	9.0	8.0	7.0	6.0	5.0	4.0	3.0	2.0	1.0
jMetal offspring 1	1.109694	**9.000000**	50.000000	**7.000000**	**6.000000**	6.057942	3.992455	**3.000000**	8.978759	10.022634
jMetal offspring 2	9.891858	**2.000000**	0.000000	**4.000000**	**5.000000**	4.942058	7.007545	**8.000000**	2.021247	0.979303
MOEA offspring 1	1.109694	**2.000000**	50.000000	**4.000000**	**5.000000**	6.057942	3.992455	**8.000000**	8.978759	10.022634
MOEA offspring 2	9.891858	**9.000000**	0.000000	**7.000000**	**6.000000**	4.942058	7.007545	**3.000000**	2.021247	0.979303

The best results are highlighted in bold

commercial platforms such as Kimeme[2] and ModeFRONTIER[3]. Hence, we feel that the field of heuristic optimisation would greatly benefit from a standardised protocol to present new algorithms.

Funding This study was funded by the institutions indicated in the list of affiliations.

Compliance with Ethical Standards

Conflict of interest The authors declare that they have no conflict of interest.

References

1. Al-Dabbagh RD, Neri F, Idris N, Baba MS. Algorithmic design issues in adaptive differential evolution schemes: review and taxonomy. Swarm Evolut Comput. 2018;43:284–311.
2. Bianchi L, Dorigo M, Gambardella LM, Gutjahr WJ. A survey on metaheuristics for stochastic combinatorial optimization. Nat Comput. 2009;8(2):239–87.
3. Blum C, Roli A. Metaheuristics in combinatorial optimization: overview and conceptual comparison. ACM Comput Surv. 2003;35(3):268–308.
4. Boryssenko AO, Herscovici N. Machine learning for multiobjective evolutionary optimization in python for em problems. In: 2018 IEEE international symposium on antennas and propagation & USNC/URSI national radio science meeting. 2018. p. 541–42
5. Cocańa-Fernández A, Sanchez L, Ranilla J. Improving the eco-efficiency of high performance computing clusters using eecluster. Energies. 2016;9:197–213.
6. Coello CAC, Lamont GB, Veldhuizen DAV. Evolutionary algorithms for solving multi-objective problems. 2nd ed. Berlin: Springer; 2007.
7. Coello Coello CA, Reyes Sierra M. A study of the parallelization of a coevolutionary multi-objective evolutionary algorithm. In: Monroy R, Arroyo-Figueroa G, Sucar LE, Sossa H, editors. MICAI 2004: advances in artificial intelligence. Berlin: Springer; 2004. p. 688–97.
8. Czyźak P, Jaszkiewicz A. Pareto simulated annealing—a metaheuristic technique for multiple-objective combinatorial optimization. J Multi Criteria Decis Anal. 1998;7(1):34–47.
9. Das I, Dennis J. Normal-boundary intersection: a new method for generating the pareto surface in nonlinear multicriteria optimization problems. SIAM J Optim. 1998;8(3):631–57.
10. Deb K, Agrawal RB. Simulated binary crossover for continuous search space. Complex Syst. 1995;9(2):115–48.
11. Deb K, Agrawal S, Pratap A, Meyarivan T. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: Nsga-ii. In: Schoenauer M, Deb K, Rudolph G, Yao X, Lutton E, Merelo JJ, Schwefel HP, editors. Parallel problem solving from nature PPSN VI. Berlin: Springer; 2000. p. 849–58.
12. Deb K, Thiele L, Laumanns M, Zitzler E. Scalable multi-objective optimization test problems. In: Proceedings of the 2002 congress on evolutionary computation, vol. 1. 2002. p. 825–30.
13. Del Ser J, Osaba E, Molina D, Yang XS, Salcedo-Sanz S, Camacho D, Das S, Suganthan PN, Coello Coello CA, Herrera F. Bio-inspired computation: where we stand and what's next. Swarm Evolut Comput. 2019;48:220–50.
14. Derrac J, García S, Molina D, Herrera F. A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. Swarm Evolut Comput. 2011;1(1):3–18.
15. Desjardins B, Falcon R, Abielmona R, Petriu E. A multi-objective optimization approach to reliable robot-assisted sensor relocation. In: 2015 IEEE congress on evolutionary computation. 2015. p. 956–64
16. Durillo JJ, Nebro AJ. jmetal: a java framework for multi-objective optimization. Adv Eng Softw. 2011;42:760–71.
17. Durillo JJ, Nebro AJ, Coello CAC, Garcia-Nieto J, Luna F, Alba E. A study of multiobjective metaheuristics when solving parameter scalable problems. IEEE Trans Evolut Comput. 2010;14:618–35.
18. Fonseca CM, Fleming PJ. Genetic algorithms for multiobjective optimization: formulation, discussion and generalization. Morgan Kaufmann. 1993. p. 416–423
19. Fonseca CM, Fleming PJ. Multiobjective optimization and multiple constraint handling with evolutionary algorithms. I. A unified formulation. IEEE Trans Syst Man Cybern Part A Syst Hum. 1998;28(1):26–37.

[2] Developed by Cyber Dyne Srl, https://cyberdyne.it/.

[3] Developed by ESTECO SpA, https://www.esteco.com/modefrontier.

20. Garcia S, Fernandez A, Luengo J, Herrera F. A study of statistical techniques and performance measures for genetics-based machine learning: accuracy and interpretability. Soft Comput. 2008;13(10):959–77.
21. García S, Molina D, Lozano M, Herrera F. A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour: a case study on the CEC'2005 Special Session on Real Parameter Optimization. J Heuristics. 2008;15(6):617–44.
22. Goudos SK, Sahalos JN. Pareto optimal microwave filter design using multiobjective differential evolution. IEEE Trans Antennas Propag. 2010;58(1):132–44.
23. Hadka D. Moea—a free and open source java framework for multiobjective optimization. 2011. https://github.com/MOEAFramework/MOEAFramework
24. Hadka D. Platypus—multiobjective optimization in python. 2015. https://github.com/Project-Platypus/Platypus
25. Huband S, Hingston P, Barone L, While L. A review of multiobjective test problems and a scalable test problem toolkit. IEEE Trans Evolut Comput. 2006;10:477–506.
26. Hussain K, Mohd Salleh MN, Cheng S, Shi Y. Metaheuristic research: a comprehensive survey. Artif Intell Rev. 2019;52(4):2191–233.
27. Ishibuchi H, Imada R, Masuyama N, Nojima Y. Comparison of hypervolume, igd and igd+ from the viewpoint of optimal distributions of solutions. In: Deb K, Goodman E, Coello Coello CA, Klamroth K, Miettinen K, Mostaghim S, Reed P, editors. Evolutionary multi-criterion optimization. Springer International Publishing; 2019. p. 332–45.
28. Ishibuchi H, Imada R, Setoguchi Y, Nojima Y. Reference point specification in inverted generational distance for triangular linear pareto front. IEEE Trans Evolut Comput. 2018;22(6):961–75.
29. Ishibuchi H, Masuda H, Tanigaki Y, Nojima Y. Modified distance calculation in generational distance and inverted generational distance. In: Gaspar-Cunha A, Henggeler Antunes C, Coello CC, editors. Evolutionary multi-criterion optimization. Springer International Publishing; 2015. p. 110–25.
30. Jakob W, Gorges-Schleuter M, Blume C. Application of genetic algorithms to task planning and learning. In: Nanderick RMB, editor. Parallel problem solving from nature, 2nd workshop, lecture notes in computer science. North-Holland Publishing Company; 1992. p. 291–300.
31. Kalyanmoy D. Multi objective optimization using evolutionary algorithms. Wiley. 2001.
32. Ke L, Zhang Q, Battiti R. Moea/d-aco: a multiobjective evolutionary algorithm using decomposition and antcolony. IEEE Trans Cybern. 2013;43(6):1845–59.
33. Kukkonen S, Jangam SR, Chakraborti N. Solving the molecular sequence alignment problem with generalized differential evolution 3 (gde3). In: 2007 IEEE symposium on computational intelligence in multi-criteria decision-making. New Jersey: Institute of Electrical and Electronics Engineers (IEEE); 2007. p. 302–09.
34. Kukkonen S, Lampinen J. Gde3: the third evolution step of generalized differential evolution. In: 2005 IEEE congress on evolutionary computation, vol. 1. New Jersey: Institute of Electrical and Electronics Engineers (IEEE); 2005. p. 443–450..
35. Li H, Zhang Q. Multiobjective optimization problems with complicated pareto sets, moea/ d and nsga-ii. IEEE Trans Evolut Comput. 2009;13(2):284–302.
36. Li M, Yang S, Liu X. A performance comparison indicator for pareto front approximations in many-objective optimization. In: Proceedings of the 17th annual conference on genetic and evolutionary computation. 2015. p. 703–10.
37. López Jaimes A, Zapotecas-Martínez S, Coello Coello C. An introduction to multiobjective optimization techniques. In: Gaspar-Cunha A, Covas JA, editors. Optimization in polymer processing. New York: Nova Science Publishers; 2011. p. 29–57.
38. Miettinen K. Nonlinear multiobjective optimization, vol. 12. Berlin: Springer; 1999.
39. Mytilinou V, Kolios AJ. A multi-objective optimisation approach applied to offshore wind farm location selection. J Ocean Eng Mar Energy. 2017;3:265–84.
40. Neri F, Cotta C. Memetic algorithms and memetic computing optimization: a literature review. Swarm Evolut Comput. 2012;2:1–14.
41. Qi Y, Ma X, Liu F, Jiao L, Sun J, Wu J. Moea/d with adaptive weight adjustment. Evolut Comput. 2014;22(2):231–64.
42. Riquelme N, Von Lücken C, Baran B. Performance metrics in multi-objective optimization. In: 2015 Latin American computing conference. 2015. p. 1–11.
43. Rostami S. Preference focussed many-objective evolutionary computation. Ph.D. thesis, The Manchester Metropolitan University, UK. 2014
44. Rostami S, Neri F. Covariance matrix adaptation pareto archived evolution strategy with hypervolume-sorted adaptive grid algorithm. Integr Comput Aided Eng. 2016;23(4):313–29.
45. Rostami S, Neri F. A fast hypervolume driven selection mechanism for many-objective optimisation problems. Swarm Evolut Comput. 2017;34:50–67.
46. Rostami S, Neri F, Epitropakis M. Progressive preference articulation for decision making in multi-objective optimisation problems. Integr Comput Aided Eng. 2017;24(4):315–35.
47. de Alcântara dos Santos Neto P, Britto R, de Andrade Lira Rabêo R, de Almeida Cruz JJ, Lira WAL. A hybrid approach to suggest software product line portfolios. Appl Soft Comput. 2016;49:1243–55.
48. Sinha SM. Mathematical programming. Amsterdam: Elsevier; 2005.
49. Srinivas N, Deb K. Muiltiobjective optimization using nondominated sorting in genetic algorithms. Evolut Comput. 1994;2(3):221–48.
50. Strickler A, Lima JAP, Vergilio SR, Pozo AT. Deriving products for variability test of feature models with a hyper-heuristic approach. Appl Soft Comput. 2016;49:1232–42.
51. Sun Y, Yen GG, Yi Z. Igd indicator-based evolutionary algorithm for many-objective optimization problems. IEEE Trans Evolut Comput. 2019;23(2):173–87.
52. Svensson M. Using evolutionary multiobjective optimization algorithms to evolve lacing patterns for bicycle wheels. Master's thesis, NTNU-Trondheim. 2015
53. Talbi EG. Metaheuristics: from design to implementation. Hoboken: Wiley; 2008.
54. Tian Y, Cheng R, Zhang X, Jin Y. Platemo: a matlab platform for evolutionary multi-objective optimization [educational forum]. IEEE Comput Intell Mag. 2017;12(4):73–87.
55. Wilcoxon F. Individual comparisons by ranking methods. Biometr Bull. 1945;1:80–3.
56. Wilson K, Rostami S. On the integrity of performance comparison for evolutionary multi-objective optimisation algorithms. In: Lotfi A, Bouchachia H, Gegov A, Langensiepen C, McGinnity M, editors. Advances in computational intelligence systems. Berlin: Springer; 2019. p. 3–15.
57. Zhang Q, Li H. Moea/d: a multiobjective evolutionary algorithm based on decomposition. IEEE Trans Evolut Comput. 2007;11(6):712–31.
58. Zhang Q, Liu W, Li H. The performance of a new version of moea/d on cec09 unconstrained mop test instances. In: 2009 IEEE congress on evolutionary computation. 2009. p. 203–208.
59. Zitzler E, Deb K, Thiele L. Comparison of multiobjective evolutionary algorithms: empirical results. Evolut Comput. 2000;8(2):173–95.

60. Zitzler E, Thiele L. An evolutionary algorithm for multiobjective optimization: the strength pareto approach. Tech. Rep. 43, Swiss Federal Institute of Technology. 1998.
61. Carrasco J, García S, Rueda MM, Das S, Herrera F. Recent trends in the use of statistical tests for comparing swarm and evolutionary computing algorithms: Practical guidelines and a critical review. Swarm and Evolutionary Computation. 2020;54:100665.