Advances in Engineering Software 42 (2011) 760–771

Contents lists available at ScienceDirect

Advances in Engineering Software

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

# jMetal: A Java framework for multi-objective optimization

Juan J. Durillo, Antonio J. Nebro *

*Departamento de Lenguajes y Ciencias de la Computación, University of Málaga, Spain*

ARTICLE INFO

*Article history:*
Received 30 November 2009
Received in revised form 19 November 2010
Accepted 16 May 2011
Available online 12 June 2011

*Keywords:*
Multi-objective optimization
Metaheuristics
Software tool
Object-oriented architecture
Performance assessment support
Experimentation

ABSTRACT

This paper describes jMetal, an object-oriented Java-based framework aimed at the development, experimentation, and study of metaheuristics for solving multi-objective optimization problems. jMetal includes a number of classic and modern state-of-the-art optimizers, a wide set of benchmark problems, and a set of well-known quality indicators to assess the performance of the algorithms. The framework also provides support to carry out full experimental studies, which can be configured and executed by using jMetal's graphical interface. Other features include the automatic generation of statistical information of the obtained results, and taking advantage of the current availability of multi-core processors to speed-up the running time of the experiments. In this work, we include two case studies to illustrate the use of jMetal in both solving a problem with a metaheuristic and designing and performing an experimental study.



## 1. Introduction

Most of the optimization problems in the real world are multi-objective in nature, which means that solving them requires to optimize two or more contradictory functions or objectives. These problems are known as multi-objective optimization problems (MOPs). The searched optimum of this kind of problems is not a single solution as in the mono-objective case, but a set of solutions known as the Pareto optimal set. Any element in this set is no better than the other ones for all the objectives. This set is given to the decision maker, who has to choose the best trade-off solution according to his/her preferences. As whatever optimization problem, MOPs may present features such as epistasis, deceptiveness, or NP-hard complexity [1], making them difficult to solve. Furthermore, if we consider engineering problems, we find frequently that some of the functions composing them are non-linear and they can be computationally expensive to evaluate.

In those situations, exact techniques are often not applicable, so approximated methods are mandatory. Like in mono-objective optimization, metaheuristics [2,3] are approximated algorithms for solving MOPs. Among them, evolutionary algorithms are very popular [4,5], and some of the most well-known algorithms in this field belong to this class (e.g. NSGA-II [6], PAES [7], SPEA2 [8]). Nevertheless, other metaheuristic techniques are also gaining momentum, as particle swarm optimization [9].

Multi-objective optimization using metaheuristics is an active research field where new techniques are continuously being proposed to cope with real settings (such as uncertainty and noise, many-objective optimization, and convergence speed). In this context, the use of software frameworks is a valuable tool for helping in the implementation of new ideas, facilitated by reusing code of existing algorithms, and for understanding the behavior of existing techniques. Desirable features of these kinds of software tools comprise:

- to include state-of-the-art algorithms,
- to contain commonly accepted benchmark MOPs,
- to provide quality indicators for performance assessment, and
- to assist users in carrying out research studies.

Some years ago, it was difficult to find any software package satisfying those requirements. The implementation in C of NSGA-II, the most used multi-objective metaheuristic algorithm, was publicly available,[1] but it was difficult to be used as the basis of new algorithms, in part due to its lack of an object-oriented design. An interesting choice was (and still is) PISA [10], a C-based framework for multi-objective optimization which is based on separating the algorithm specific part of an optimizer from the application-specific part. This is carried out using a shared-file mechanism to communicate the module executing the application with the module running the metaheuristic. However, a drawback of PISA is that their internal design hinders to reuse code.

Those reasons leaded us to develop a new software toolbox starting from scratch, and the result was jMetal, a Java-based framework designed to multi-objective optimization using metaheuristics. We imposed ourselves as design goals that jMetal

* Corresponding author.
*E-mail addresses:* durillo@lcc.uma.es (J.J. Durillo), antonio@lcc.uma.es (A.J. Nebro).

[1] NSGA-II: http://www.iitk.ac.in/kangal/codes.shtml.

0965-9978/$ - see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.advengsoft.2011.05.014

should be simple and easy to use, portable (hence the choice of Java), flexible, and extensible [11]. Nowadays, jMetal is publicly available to the community of people interested in multi-objective optimization. It is licensed under the GNU Lesser General Public License,[2] and it can be obtained freely from http://jmetal.sourceforge.net.

During the development of jMetal, other Java-based software tools have emerged, e.g., EVA2,[3] OPT4j,[4] or ECJ.[5] Other similar tools based in other programming languages have also come into light, as MOEAT [12], that is implemented in C#, what restricts them only to Windows-based PCs. All these toolboxes can be useful enough for many researchers. However, while jMetal is specifically oriented to multi-objective optimization with metaheuristics, most of those frameworks are mainly focused on evolutionary algorithms, and many of them are centered in single-objective optimization, offering only extensions to the multi-objective domain.

jMetal has the following features that, all together, make our framework a unique system compared to existing proposals:

- Implementation of a number of modern multi-objective optimization algorithms: NSGA-II [6], SPEA2 [8], PAES [7], PESA-II [13], OMOPSO [14], MOCell [15], AbYSS [16], MOEA/D [17], Densea [18], CellDE [19], GDE3 [20], FastPGA [21], IBEA [22], SMPSO [23], MOCHC [24], and SMS-EMOA [25].
- Validation of the implementation: we compared our implementations of NSGA-II and SPEA2 with the original versions, achieving competitive results [11].
- A rich set of test problems including:
  - Problem families: Zitzler–Deb–Thiele (ZDT) [26], Deb–Thiele–Laumanns–Zitzler (DTLZ) [27], Walking-Fish-Group (WFG) test problems [28]), CEC2009 (unconstrained problems) [29], and the Li-Zhang benchmark [17].
  - Classical problems: Kursawe [30], Fonseca and Flemming [31], Schaffer [32].
  - Constrained problems: Srinivas [33], Tanaka [34], Osyczka2 [35], Constr_Ex [6], Golinski [36], Water [37].
- Implementation of the most widely used quality indicators: Hypervolume [38], Spread [6], Generational Distance [39], Inverted Generational Distance [39], Epsilon [40].
- Different variable representations: binary, real, binary-coded real, integer, permutation.
- Support for performing experimental studies, including the automatic generation of LaTeX tables with the results after applying quality indicators, statistical pairwise comparison by using the Wilcoxon test to the obtained results, and R (http://www.r-project.org/) boxplots summarizing those results. In addition, jMetal includes the possibility of using several threads for performing these kinds of experiments in such a way that several independent runs can be executed in parallel using modern multi-core CPUs.
- A Graphical User Interface (GUI) for giving support in solving problems and performing experimental studies.
- A Web site (http://jmetal.sourceforge.net) containing the source codes, the user manual and, among other information, the Pareto fronts of the included MOPs, references to the implemented algorithms, and references to papers using jMetal.

We have used jMetal in a number of works: we have proposed new multi-objective techniques (AbYSS [16], a scatter search algorithm; SMPSO, a particle swarm optimization technique [23]; or CellDE [19], a cellular genetic algorithm hybridized with differential evolution) and we have investigated properties of state-of-the-art multi-objective optimizers (convergence speed [41], behavior when solving parameter scalable problems [42,43], or the influence of using a steady-state selection scheme in multi-objective genetic algorithms [44]). Besides our own work, jMetal has been used by other groups, as it can be seen in the publication section of jMetal's Web site.

In this paper, our purpose is to describe jMetal (version 3.1) and how it can be used by people interested in using metaheuristics for solving MOPs. Since many researches in the multi-objective field are not only interested in solving a given problem but also in assessing the performance of different algorithms and comparing them when solving a given benchmark, we also describe here how to use our tool in that sense. In particular, to cope with this last issue, we have considered a case study consisting in solving a benchmark composed of four constrained problems (Osyczka2, Srinivas, Golinski, and Tanaka) by using four different multi-objective algorithms (NSGA-II, SPEA2, MOCell and AbYSS).

The content of this paper can be summarized as follows. In the next section we include some background on multi-objective optimization. Section 3 is aimed at presenting the jMetal architecture and its main components. The quality indicators included in jMetal are described in Section 4. We describe in Section 5 how to extend the framework with a new problem and solving it. After that, we explain hot to develop an algorithm using jMetal. Section 7 is devoted to describing the case study and how the framework can be used to carry out a full comparison of experiments. Finally, Section 8 presents the conclusions and further work.

## 2. Multi-objective background

In this section, we include some background on multi-objective optimization. We define formally the concept of MOP, Pareto optimality, Pareto dominance, Pareto optimal set, and Pareto front. In these definitions we are assuming, without loss of generality, that minimization is the goal for all the objectives. A general MOP can be formally defined as follows:

**Definition 1** (*MOP*). Find a vector $\vec{x}^* = [x_1^*, x_2^*, \ldots, x_n^*]$ which satisfies the $m$ inequality constraints $g_i(\vec{x}) \geqslant 0, i = 1, 2, \ldots, m$, the $p$ equality constraints $h_i(\vec{x}) = 0,\ i = 1, 2, \ldots, p$, and minimizes the vector function $\vec{f}(\vec{x}) = [f_1(\vec{x}), f_2(\vec{x}), \ldots, f_k(\vec{x})]^T$, where $\vec{x} = [x_1, x_2, \ldots, x_n]^T$ is the vector of decision variables.

The set of all values satisfying the constraints defines the *feasible region* $\Omega$ and any point $\vec{x} \in \Omega$ is a *feasible solution*. As mentioned before, we seek the *Pareto optima*. More formally:

**Definition 2** (*Pareto optimality*). A point $\vec{x}^* \in \Omega$ is Pareto Optimal if for every $\vec{x} \in \Omega$ and $I = \{1,2,\ldots,k\}$ either $\forall_{i \in I}(f_i(\vec{x}) = f_i(\vec{x}^*))$ or there is at least one $i \in I$ such that $f_i(\vec{x}) > f_i(\vec{x}^*)$.

This definition states that $\vec{x}^*$ is Pareto optimal if no other feasible vector $\vec{x}$ exists which would improve some criteria without causing a simultaneous worsening in at least one other criterion. Other important nomenclature associated with Pareto optimality are defined below:

**Definition 3** (*Pareto dominance*). A vector $\vec{u} = (u_1, \ldots, u_k)$ is said to dominate $\vec{v} = (v_1, \ldots, v_k)$ (denoted by $\vec{u} \preccurlyeq \vec{v}$) if and only if $\vec{u}$ is partially less than $\vec{v}$, i.e., $\forall i \in \{1,\ldots,k\}, u_i \leqslant v_i \wedge\ \exists i \in \{1,\ldots,k\}: u_i < v_i$.

**Definition 4** (*Pareto optimal set*). For a given MOP $\vec{f}(\vec{x})$, the Pareto optimal set is defined as $\mathcal{P}^* = \{\vec{x} \in \Omega | \neg \exists \vec{x}' \in \Omega, \vec{f}(\vec{x}') \preccurlyeq \vec{f}(\vec{x})\}$.

**Definition 5** (*Pareto front*). For a given MOP $\vec{f}(\vec{x})$ and its Pareto optimal set $\mathcal{P}^*$, the Pareto front is defined as $\mathcal{PF}^* = \{\vec{f}(\vec{x}), \vec{x} \in \mathcal{P}^*\}$.

[2] LGPL License: http://creativecommons.org/licenses/LGPL/2.1/.
[3] EVA2: http://www.ra.cs.uni-tuebingen.de/software/EvA2/.
[4] OPT4j: http://opt4j.sourceforge.net/.
[5] ECJ: http://cs.gmu.edu/eclab/projects/ecj/.

Obtaining the Pareto front of a MOP is the main goal of multi-objective optimization. In theory, a Pareto front could contain a large number (or even infinitely many) points. In practice, a usable approximate solution will only contain a limited number of them; thus, an important goal is that they should be as close as possible to the exact Pareto front (*convergence*) and uniformly spread (*diversity*), otherwise, they would not be very useful to the decision maker. Closeness to the Pareto front ensures that we are dealing with optimal solutions, while a uniform spread of the solutions means that we have made a good exploration of the search space and no regions are left unexplored.

## 3. Architecture of jMetal

jMetal has been designed following an object-oriented architecture which facilitates the inclusion of new components and the reuse of existing ones. It provides the user with a set of base classes which can be used to design new algorithms, and to implement problems or complex operators, etc. This section is aimed at describing the classes composing the core of jMetal.

Fig. 1 depicts an UML diagram including the base classes within jMetal. The working principle jMetal is based in relies in that an *Algorithm* solves a *Problem* using one (and possibly more) *SolutionSet* and a set of *Operator* objects. As we can see in the figure, our tool contains classes for representing those and some other elements.

A generic terminology to name the classes within jMetal has been employed in order to make them general enough to be used in any metaheuristic. This way, in the context of evolutionary algorithms, populations and individuals correspond to *SolutionSet* and *Solution* jMetal classes, respectively; the same can be applied to particle swarm optimization algorithms concerning the concepts of swarm and particles.

Class *Algorithm* represents the superclass for all the optimizers: whatever metaheuristic included in jMetal has to inherit from it. An instance object of *Algorithm* may require some application-specific parameters, that can be added and accessed by using the methods *addParameter ()* and *getParameter ()*, respectively. Similarly, an algorithm may also make use of some operators. For example, genetics algorithms typically use operators for recombination and selection of individuals. There are also methods for incorporating operators (*addOperator ()*) and to get them (*getOperator ()*). The main method in *Algorithm* is *execute ()*, which starts the execution of the algorithm.

As its name suggests, class *SolutionSet* represents a set of *Solution* objects. Each *Solution* object has associated a type (*SolutionType*) and it is composed of an array of *Variable* objects. This last is a superclass aimed at describing different kinds of representations for solutions. The proposed scheme by jMetal is very flexible because a *Solution* object is not restricted to contain variables of the same representation; instead, it can be composed of an array of mixed variable types. Furthermore, it is also

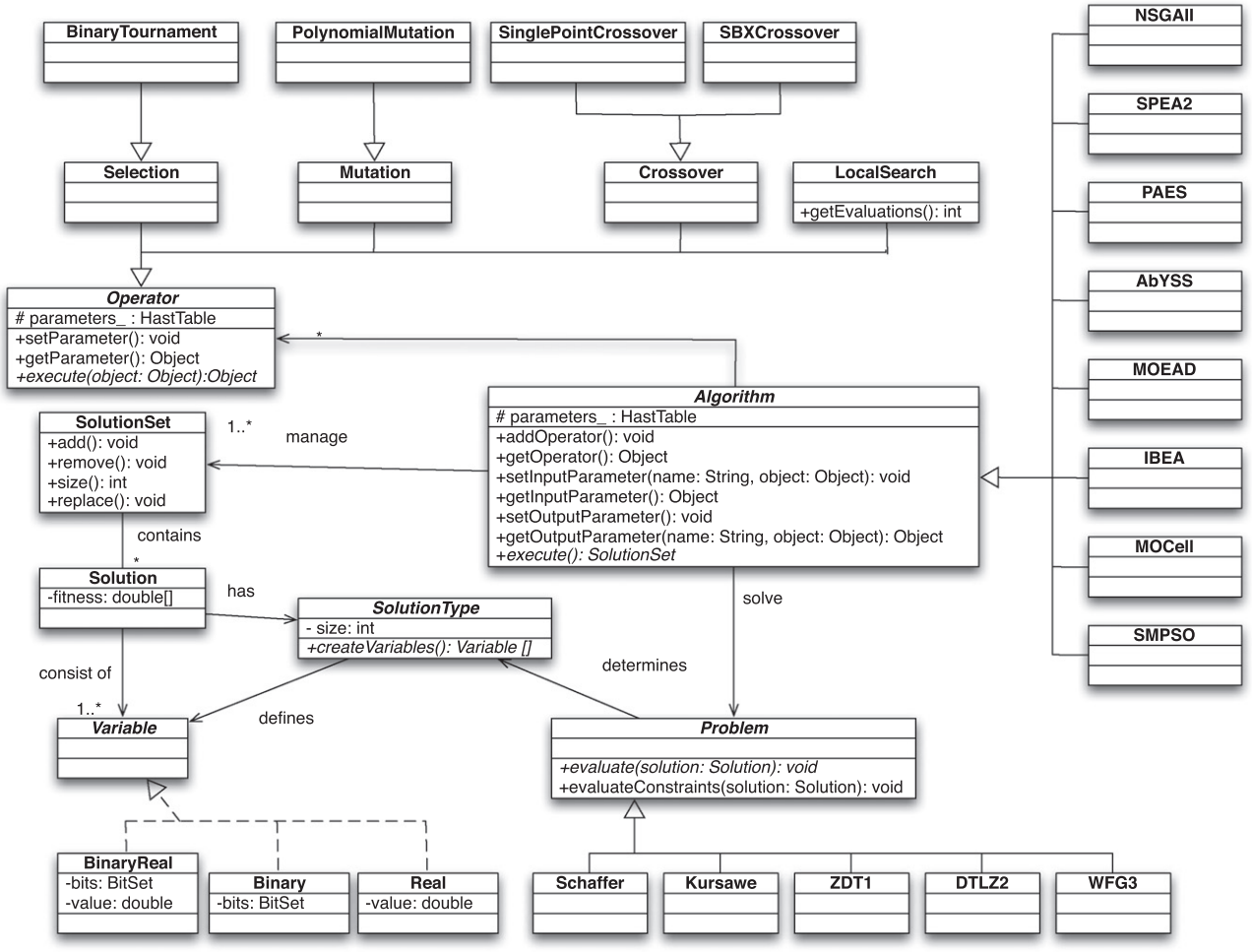


**Fig. 1.** General architecture of jMetal.

extensible, because new representations can be easily incorporated into the framework, just by inheriting from *Variable*.

In jMetal, all the problems have to inherit from class *Problem*. This class contains two basic methods: *evaluate ()* and *evaluateConstraints ()*. Both methods receive a *Solution* representing a candidate solution to the problem; the first one evaluates it, and the second one determines the overall constraint violation of that solution. All the problems have to define the *evaluate ()* method, while only problems having side constraints need to define *evaluateConstraints ()*. The constraint handling mechanism implemented by default is the one proposed in [6].

*Operator* is a superclass aimed at representing generic operators to be used by the different algorithms (e.g., crossover, mutation, or selection are typical operators in the field of evolutionary algorithms). As *Algorithm*, it contains the *getParameter ()* and *setParameter ()* methods, which are used for adding and accessing to operator specific parameters. For example, the simulated binary (SBX) crossover [4], employed by many evolutionary algorithms, requires two parameters: a crossover probability (as most crossover operators) plus a value for the distribution index (specific of the operator). This operator receives as arguments two parent individuals and, as a result, it returns the offspring resulting of recombining both parents. The use of the generic Java class `Object` allows a high degree of flexibility. Thus, a *BinaryTournament* operator, which is a selection operator employed by many genetic algorithms, will receive as an argument a *SolutionSet* object and it will return a *Solution* object, while a *PolynomialMutation* operator, also a mutation operator typically used in evolutionary algorithms, will receive a *Solution* object, will modify this object, and it will return a null value.

A more detailed description of the jMetal architecture can be found in [45] and in the jMetal user manual [46].

## 4. Performance assessment with jMetal

Contrary to single-objective optimization, where assessing the performance of a metaheuristic mainly requires to observe the best value yielded by an algorithm (i.e., the lower the better, in case of minimization problems), in multi-objective optimization this is not applicable. Instead, an approximation set to the optimal Pareto front of the problem is computed. As we stated in Section 2, two properties are usually required: convergence and a uniform diversity. A number of quality indicators for measuring these two criteria have been proposed in the literature: Generational Distance (GD) [39], Inverse Generational Distance (IGD), Hypervolume (HV) [38], Epsilon [40], Spread or $\Delta$ [4], Generalized Spread indicators, and others. Some of them are intended to measuring only the convergence or diversity, and others take into account both criteria. Fig. 2 depicts a classification of the indicators based which aspect they measure. All these indicators are included in jMetal and they can be defined as follows.

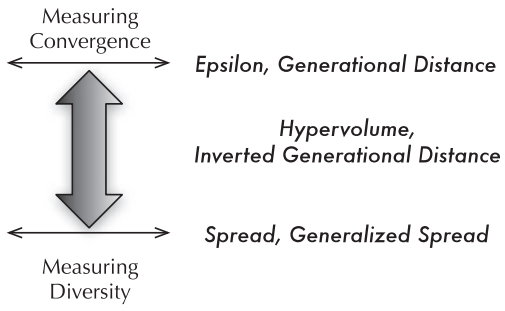


**Fig. 2.** A classification of quality indicators.

- *GD*. This indicator was introduced by Van Veldhuizen and Lamont [39] for measuring how far are the elements in the computed approximation from those in the optimal Pareto front and it is defined as:

$$GD = \frac{\sqrt{\sum_{i=1}^{n} d_i^2}}{n}, \tag{1}$$

where $n$ is the number of solutions in the approximation and $d_i$ is the Euclidean distance (measured in objective space) between each of these solutions and the nearest member into the optimal Pareto front. A value of $GD = 0$ indicates that all the generated elements are in the Pareto front.

- *IGD*. It is a variant of the Generational Distance. It measures the distances between each solution composing the optimal Pareto front and the computed approximation. It can be defined as follows:

$$IGD = \frac{\sqrt{\sum_{i=1}^{n} d_i^2}}{n}, \tag{2}$$

being $n$ the number of solutions in the optimal Pareto front and $d_i$ is the Euclidean distance (measured in objective space) between each point of that front and the nearest member of approximation.

- *HV*. This indicator calculates the volume, in the objective space, covered by members of a non-dominated set of solutions $Q$, e.g., the region enclosed into the discontinuous line in Fig. 3, $Q = \{A, B, C\}$, for problems where all objectives are to be minimized [38]. Mathematically, for each solution $i \in Q$, a hypercube $v_i$ is constructed with a reference point $W$ and the solution $i$ as the diagonal corners of the hypercube. The reference point can simply be found by constructing a vector of worst objective function values. Thereafter, a union of all hypercubes is found and its hypervolume ($HV$) is calculated:

$$HV = \text{volume}\left(\bigcup_{i=1}^{|Q|} v_i\right). \tag{3}$$

Algorithms with larger values of $HV$ are desirable.

- *Epsilon*. Given an computed front for a problem, A, this indicator is a measure of the smallest distance one would need to translate every solution in A so that it dominates the optimal Pareto front of this problem. More formally, given $\vec{z^1} = (z_1^1, \ldots, z_n^1)$ and $\vec{z^2} = (z_1^2, \ldots, z_n^2)$, where $n$ is the number of objectives:

$$I_{\epsilon+}^1(A) = inf_{\epsilon \in \mathbb{R}} \left\{ \forall \vec{z^2} \in \mathcal{PF}^* \exists \vec{z^1} \in A : \vec{z^1} \prec_\epsilon \vec{z^2} \right\} \tag{4}$$

where, $\vec{z^1} \prec_\epsilon \vec{z^2}$ if and only if $\forall 1 \leqslant i \leqslant n : z_i^1 < \epsilon + z_i^2$.

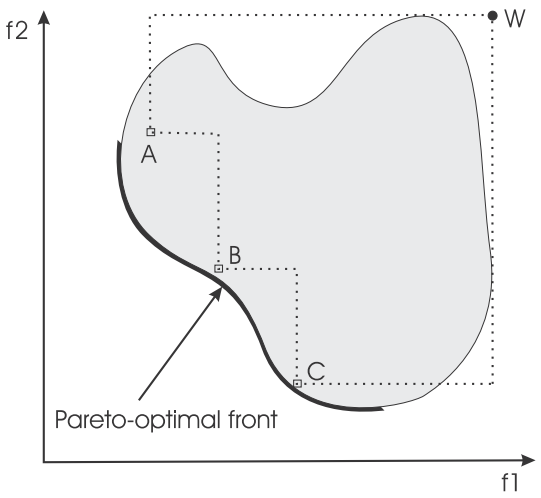


**Fig. 3.** Hypervolume enclosed by the non-dominated solutions *A*,*B*, and *C*.

- *Spread or* $\Delta$. This indicator [4] measures the extent of spread by the set of computed solutions. It is defined as:

$$\Delta = \frac{d_f + d_l + \sum_{i=1}^{N-1} |d_i - \bar{d}|}{d_f + d_l + (N-1)\bar{d}}, \tag{5}$$

where $d_i$ is the Euclidean distance between consecutive solutions, $\bar{d}$ is the mean of these distances, and $d_f$ and $d_l$ are the Euclidean distances to the *extreme* solutions of the optimal Pareto front in the objective space, e.g., see Fig. 4. This indicator takes a zero value for an ideal distribution, pointing out a perfect spread of the solutions in the Pareto front.

- *Generealized Spread*. The previous indicator is based on calculating the distance between two consecutive solutions, which works only for 2-objective problems. We extended this metric by computing the distance from a given point to its nearest neighbor in [16]. This extension was based on the metric proposed in [47]:

$$\Delta = \frac{\sum_{i=1}^{m} d(e_i, S) + \sum_{X \in S} |d(X,S) - \bar{d}|}{\sum_{i=1}^{m} d(e_i, S) + |S| * \bar{d}} \tag{6}$$

where $S$ is a set of solutions, $S^*$ is the set of Pareto optimal solutions, $(e_1,\ldots,e_m)$ are $m$ extreme solutions in $S^*$, $m$ is the number of objectives and

$$d(X,S) = \min_{Y \in S, Y \neq X} ||F(X) - F(Y)||^2, \tag{7}$$

$$\bar{d} = \frac{1}{|S^*|} \sum_{X \in S^*} d(X,S). \tag{8}$$

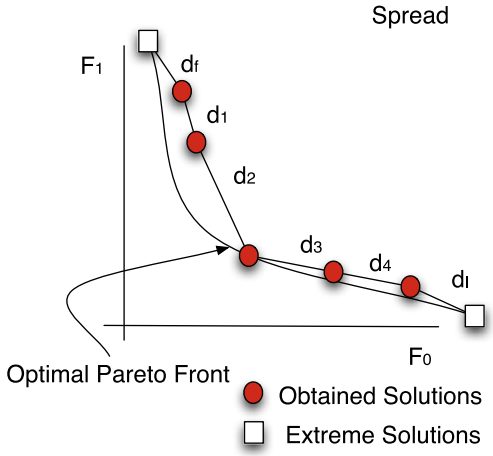


**Fig. 4.** Distances from the extreme solutions.

Since those indicators are not free from arbitrary scaling of objectives, jMetal always applies them after normalizing the objective function values.

## 5. Solving MOPS with jMetal

In this section, we detail how jMetal can be used for computing and visualizing the Pareto front of a problem. Taken as starting point a well-known problem in the multi-objective literature (the one proposed by Schaffer [32]), we will follow a bottom-up approach for describing how our tool can be used in that sense: first,

**Table 1**
Definition of problem Schaffer.

| Problem | Objective functions | Variable bounds | $n$ |
|---|---|---|---|
| Schaffer | $f_1(x) = x^2$<br>$f_2(x) = (x-2)^2$ | $-10^5 \leqslant x \leqslant 10^5$ | 1 |

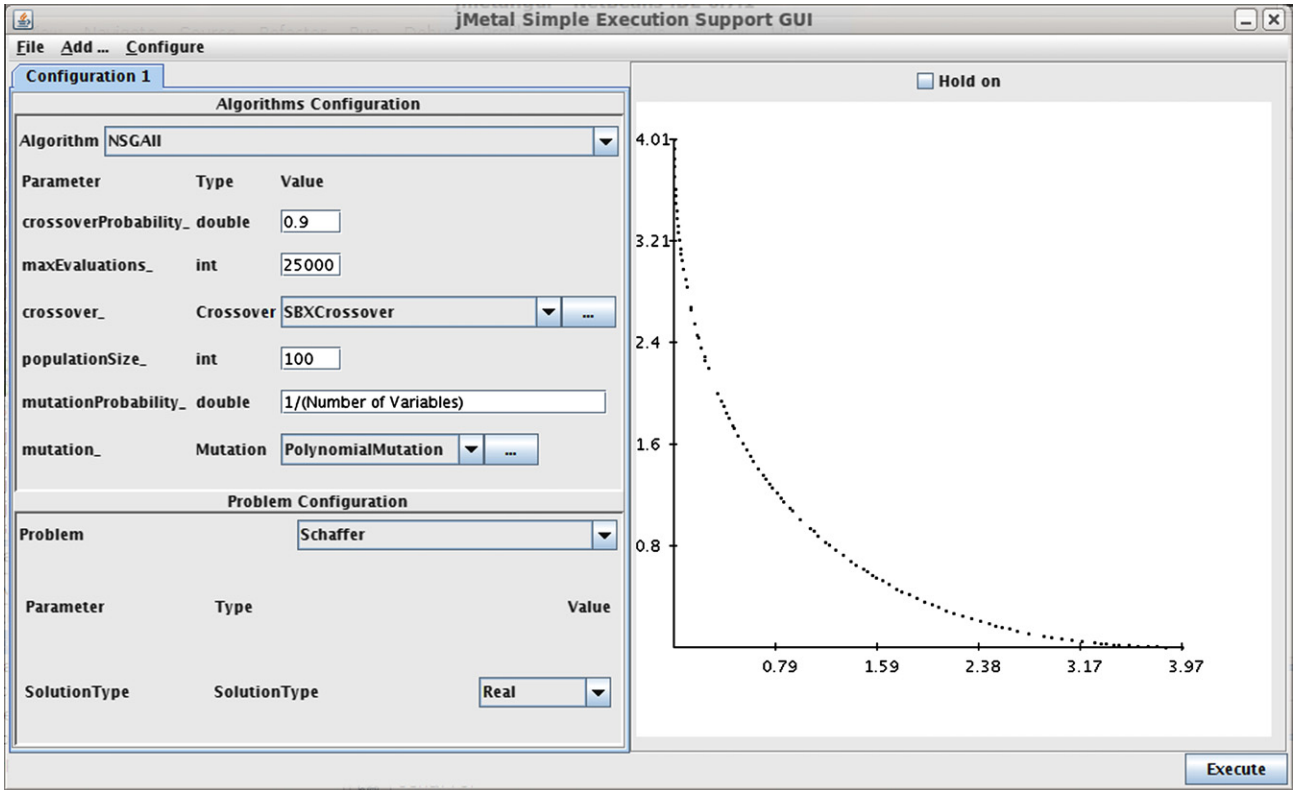


**Fig. 5.** Solving a problem and showing the resulting Pareto front. In this figure, the Schaffer problem has been solved by using the NSGA-II algorithm.

we show how to include that problem into jMetal; after that, we describe how to solve it by using the NSGA-II algorithm. For this last issue, there are two choices: to use the jMetal GUI tool, or to use a Java development environment for configuring the algorithm, and then to display the obtained approximation to the Pareto front using an external application, like R or Gnuplot. The former option is the easiest one, while the second one allows the user a higher degree of control and flexibility over the tool. Here, we describe the first option. Users interested in how to configure the algorithm by modifying the source code can find out more information in the jMetal user manual.

### 5.1. Adding MOPs to jMetal

Schaffer's study is a bi-objective problem having only one decision variable; its definition is included in Table 1. Fig. 6 shows the Java class implementing this problem using jMetal.

As we mentioned in the last section, all the problems in jMetal inherit from class *Problem*. First of all, a constructor for creating instances of the problem is required (lines 14–37). This constructor defines the features of the problem: number of variables (line 15), number of functions (line 16), number of constraints (line 17), and the range of the values of the decision variables (line 20–25). Implementing a new problem only requires to redefine two methods: *evaluate ()* and *evaluateConstraints ()*, which evaluate the objectives values and the overall constraint violation of a given solution, respectively. Whenever a problem has no constraints, like the Schaffer one, the second method should not be redefined: superclass *Problem* defines it as an empty method (i.e., it does nothing). Lines 45–57 show the code of the evaluate method: first, the objective values are computed; then, they are stored into the solution.

### 5.2. Simple execution support GUI: a graphical interface for solving bi-objective problems

jMetal provides a GUI for launching the execution of an algorithm for solving a problem and plotting the computed front. This GUI is called Simple Execution Support GUI.

This tool loads all the metaheuristics, and all the bi-objective problems included in jMetal, so that the user only has to select the desired choices, as it is depicted in Fig. 5. Specifically, this figure shows the result of computing the Pareto front of the Schaffer problem by using the NSGA-II algorithm. Text files containing the values of the objective functions and decision variables are also generated and stored in the working folder. The left part of the figure allows to select and to configure the algorithm and the problem. The right part shows the computed Pareto front approximation with the selected algorithm.

## 6. Including new algorithms within the framework

This section is aimed at describing how new algorithms can be developed by using our framework. To deal with this issue, we will briefly describe the basic components that many of existing multi-objective algorithms include, and then we will detail how to implement them by considering the NSGA-II algorithm as case study.

Practically every multi-objective metaheuristics follow a common pattern. They work over a set of solutions that undergoes stochastic operators in order to create new (better) solutions to a target problem. Typically, that set is initialized in a random fashion (although other more advanced initializations are possible) and new solutions are created until a termination criterion, usually consisting in the number of performed function evaluation, is meet. Finally, each metaheuristic returns a *SolutionSet* object containing the solutions found as a final result. Thus, having that typical behavior in mind, there are some tasks that are shared by most of multi-objective algorithm: to obtain the parameters set for configuring the algorithm, to get the operators to be applied, to initialize the solution set the algorithm works with, and to create new solutions by applying the stochastic operators.

```
 1: /**
 2: * Class representing problem Schaffer
 3: */
 4: public class Schaffer extends Problem {
 5:
 6: /**
 8: * Constructor.
 9: * Creates a default instance of the Schaffer problem
10: * @param solutionType The solution type must "Real"
12: * or "BinaryReal"
13: */
14: public Schaffer(String solutionType) {
15:  numberOfVariables_   = 1 ;
16:  numberOfObjectives_  = 2 ;
17:  numberOfConstraints_ = 0 ;
18:  problemName_   = "Schaffer";
19:
20:  lowerLimit_ = new double[numberOfVariables_];
21:  upperLimit_ = new double[numberOfVariables_];
22:  for (int var = 0; var < numberOfVariables_; var++){
23:    lowerLimit_[var] = -10000;
24:    upperLimit_[var] = 10000;
25:  } // for
26:
27:  // code omitted to avoid hinder the reader
...:  // with non-relevant details
37: } //Schaffer
38:
39:
40: /**
41: * Evaluates a solution
42: * @param solution The solution to evaluate
43: * @throws JMException
44: */
45: public void evaluate(Solution solution) throws JMException {
46:  DecisionVariables dv = solution.getDecisionVariables();
47:
48:  double [] f = new double[numberOfObjectives_];
49:  f[0] = dv.variables_[0].getValue() *
50:   dv.variables_[0].getValue();
51:
52:  f[1] = (dv.variables_[0].getValue() - 2.0) *
53:   (dv.variables_[0].getValue() - 2.0);
54:
55:  solution.setObjective(0,f[0]);
56:  solution.setObjective(1,f[1]);
57: } //evaluate
58: } //Schaffer
```

**Fig. 6.** Class implementing the Schaffer problem.

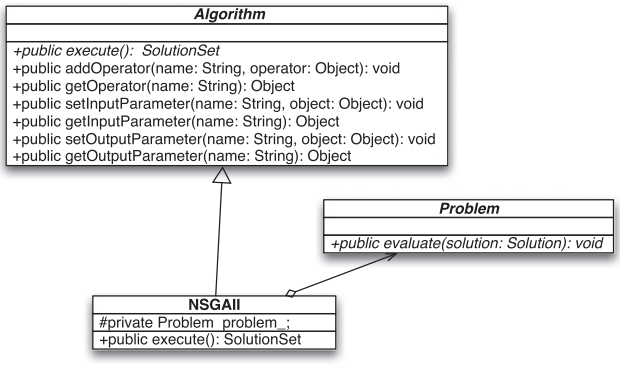


**Fig. 7.** UML diagram of class `NSGAII`.

```
2 * NsgaII.java
6 package jmetal.metaheuristics.nsgaII;
7
8 import jmetal.base.*;
10
13 // This class implements the NSGA-II algorithm.
15 public class NSGAII extends Algorithm {
16
18 // stores the problem to solve
20 private Problem problem_;
21
22 /**
23 * Constructor
24 * @param problem Problem to solve
25 */
26 public NSGAII(Problem problem){
29
30 // Runs of the NSGA-II algorithm.
36 public SolutionSet execute() throws JMException {
169 } // NSGA-II
```

**Fig. 8.** Scheme of the implementation of class `NSGAII`.

In the following, taking as case study the algorithm NSGA-II, we will detail how these tasks can be implemented by using jMetal. The UML diagram of the *NSGAII* class is depicted in Fig. 7. As we stated before, as every metaheuristic developed in jMetal, *NSGAII* inherits from *Algorithm* and it only contains a method, called *execute ()*, which implements the behavior of the algorithm. This method has no parameters and returns as a result a *SolutionSet* object, which should contain the front computed by the technique. Fig. 8 includes the code structure for implementing NSGA-II. We observe that the problem to solve should be included as a parameter for creating the algorithm (line 26).

To implement the *execute ()* method, the first step is to obtain the parameters that should be used by our technique. To this end, jMetal provides two methods: *getInputParameter ()* and *setInputParameter ()*. The former one allows the algorithm to retrieve a parameter, while the latter allows adding a parameter to the algorithm. The same scenario repeats with the operators: the algorithm needs to get the operators, and there should exist a method for adding operators externally to the algorithm. In jMetal those methods are *getOperator ()* and *addOperator ()*, respectively. In the case of NSGA-II, two parameters are required: the population size, and the maximum number of evaluations to perform. Additionally, it requires three operators: crossover, mutation, and selection. Fig. 9 depicts how these parameters and operators can be retrieved by the NSGA-II algorithm. More specifically, the parameters are obtained in lines 56–57, and operators are obtained in lines 60–62.

Once we have set all the needed parameters and operators, we proceed by creating and initializing the solution set used by the algorithm. In NSGA-II, only one solution set is required and it is initialized with random solutions. Fig. 10 shows the code required to

```
36  public SolutionSet execute() throws JMException {
   ...
55   //Read the parameters
56   populationSize  =
    ((Integer)getInputParameter("populationSize")).intValue();
57   maxEvaluations  =
    ((Integer)getInputParameter("maxEvaluations")).intValue();
58
59   //Read the operators
60   mutationOperator = operators_.get("mutation");
61   crossoverOperator = operators_.get("crossover");
62   selectionOperator = operators_.get("selection");
   ...
168  } // execute
```

**Fig. 9.** `execute ()` method: initializing objects.

```
   ...
69   SolutionSet population = new SolutionSet(populationSize);
70
71   // Create the initial solutionSet
72   Solution newSolution;
73   for (int i = 0; i < populationSize; i++) {
74    newSolution = new Solution(problem_);
75    problem_.evaluate(newSolution);
76    problem_.evaluateConstraints(newSolution);
77    evaluations++;
78    population.add(newSolution);
79   } //for
   ...
```

**Fig. 10.** `execute ()` method: initializing the population.

do it. In that code, the solution set is created in line 69; the maximum size of this set is established when it is created, setting properly the *populationSize* parameter. After that, the population is initialized in the loop included in lines 73–79. Concretely, a random solution is created in line 72; then, that solution is evaluated and the overall constraints violation is computed (lines 75 and 76, respectively). Finally, after creating and evaluating it, the new solution is added to the population (line 78).

The main loop of the algorithm is included in the piece of code contained in Fig. 11. We can observe the inner loop performing the generations (lines 87–103), where the genetic operators are applied. The number of iterations of this loop is *populationSize/2* because it is assumed that the crossover returns two solutions; in case of using a crossover operator returning only one solution, the sentence in line 87 should be modified accordingly. Then, the code between lines 103 and 167, which has been omitted in Fig. 11, perform algorithm-specific tasks. In NSGA-II, those tasks are: to joint the parent and offspring population, to sort them by means of the rank of the individuals, and selecting those individuals with the lowest rank and the highest crowding distance to be part of the population for the next generation. Finally, after performing these algorithm-specific tasks, the result is returned in line 167.

After detailing how to implement the NSGA-II algorithm, we comment how to execute it. There are two choices: by creating

```
36  public SolutionSet execute() throws JMException {
   ...
81   // Generations ...
82   while (evaluations < maxEvaluations) {
83
84    // Create the offSpring solutionSet
85    offspringPopulation = new SolutionSet(populationSize);
86    Solution [] parents = new Solution[2];
87    for (int i = 0; i < (populationSize/2); i++){
88     //obtain parents
89     if (evaluations < maxEvaluations) {
90     parents[0] =
      (Solution)selectionOperator.execute(population);
91     parents[1] =
      (Solution)selectionOperator.execute(population);
92     Solution [] offSpring =
      (Solution []) crossoverOperator.execute(parents);
93     mutationOperator.execute(offSpring[0]);
94     mutationOperator.execute(offSpring[1]);
95     problem_.evaluate(offSpring[0]);
96     problem_.evaluateConstraints(offSpring[0]);
97     problem_.evaluate(offSpring[1]);
98     problem_.evaluateConstraints(offSpring[1]);
99     offspringPopulation.add(offSpring[0]);
100    offspringPopulation.add(offSpring[1]);
101    evaluations += 2;
102   } if
103  } // for
   ...
167  return population;
168 } // execute
```

**Fig. 11.** `execute ()` method: main loop.

an additional class, where the parameters and operators of the algorithm are set and then the execute method is invoked (we provide those classes in jMetal, e.g., the *NSGAII_main* class in the case of NSGA-II), and by using the Single Execution Support GUI, as it was shown in Section 5.2.

## 7. Experimentation studies with jMetal: a case study

In Section 5, we computed a front of solutions to Schaffer's problem after running NSGA-II and we displayed it by using the Single Execution Support GUI. While this approach is useful to test whether or not a multi-objective metaheuristics works, and even it is valuable when trying to adjust the algorithm parameters, it is clearly insufficient if our purpose is to carry out a rigorous study. In this section, we describe how jMetal can help in carrying out these kinds of studies.

### 7.1. Description of the experiments

To cope with this issue let us consider the following case study. We intend to assess the performance of a cellular genetic algorithm (cGA), MOCell [15], a scatter search multi-objective algorithm, AbYSS [16], both of them designed by us, and the reference algorithms in the area, i.e., NSGA-II and SPEA2 over a set of four constrained MOPs included in jMetal. These problems are: Osyczka2, Srinivas, Golinski, and Tanaka (see Table 2 for a detailed description).

To start with, it is worth highlighting that metaheuristics are non-deterministic techniques, so trying to draw some conclusion after running once one of them does not make sense [48]. The usual procedure is to carry out a number $N$ of independent runs and then to use means and standard deviations (or medians and interquartile ranges) for summarizing the obtained results. A minimum value of $N = 30$ is usually recommended; in our research works, we use $N = 100$ whenever it is possible. Additionally, this must be complemented with the application of statistical tests to ensure the significance of the results; otherwise, the drawn conclusions may be wrong because differences between the algorithms could have occurred by chance. For comparing the algorithms, we have selected one indicator measuring convergence (Epsilon), one measuring diversity (Spread), and one measuring both criteria (HV).

### 7.2. Configuring the study: experiment support GUI

As for executing a single algorithm for solving a problem, there are also two choices for carrying out these kinds of experiments in jMetal: to use the Experimental Support GUI, or to made use of a Java development environment and setting properly the algorithms modifying the code. As before, we detail here how jMetal support to this kind of experiment by using the Experimentation Support GUI. Again, those readers interested in the latter option are referred to the jMetal manual.

The main window of this GUI is showed in Fig. 12. This window is divided in three main blocks: top, center, and bottom. The top part of the GUI is only used to provide the name of the experiment to carry out. This name is only used to refer to the experiment.

The central part is divided in three blocks: algorithms, problems, and quality indicators. The first one (on the left side of the figure) contains a list with all the algorithms included in jMetal.

**Table 2**
Constrained test functions.

| Problem | Objective functions | Constraints | Variable bounds | n |
|---|---|---|---|---|
| Osyczka2 | $f_1(\vec{x}) = -(25(x_1-2)^2+(x_2-2)^2+(x_3-1)^2(x_4-4)^2+(x_5-1)^2)$<br>$f_2(\vec{x}) = x_1^2+x_2^2+x_3^2+x_4^2+x_5^2+x_6^2$ | $g_1(\vec{x}) = 0 \leqslant x_1+x_2-2$<br>$g_2(\vec{x}) = 0 \leqslant 6-x_1-x_2$<br>$g_3(\vec{x}) = 0 \leqslant 2-x_2+x_1$<br>$g_4(\vec{x}) = 0 \leqslant 2-x_1+3x_2$<br>$g_5(\vec{x}) = 0 \leqslant 4-(x_3-3)^2-x_4$<br>$g_6(\vec{x}) = 0 \leqslant (x_5-3)^3+x_6-4$ | $0 \leqslant x_1, x_2 \leqslant 10$<br>$1 \leqslant x_3, x_5 \leqslant 5$<br>$0 \leqslant x_4 \leqslant 6$<br>$0 \leqslant x_6 \leqslant 10$ | 6 |
| Tanaka | $f_1(\vec{x}) = x_1$<br>$f_2(\vec{x}) = x_2$ | $g_1(\vec{x}) = -x_1^2-x_2^2+1+0.1\cos(16\arctan(x_1/x_2)) \leqslant 0$<br>$g_2(\vec{x}) = (x_1-0.5)^2+(x_2-0.5)^2 \leqslant 0.5$ | $-\pi \leqslant x_i \leqslant \pi$ | 2 |
| Srinivas | $f_1(\vec{x}) = (x_1-2)^2+(x_2-1)^2+2$<br>$f_2(\vec{x}) = 9x_1-(x_2-1)^2$ | $g_1(\vec{x}) = x_1^2+x_2^2 \leqslant 225$<br>$g_2(\vec{x}) = x_1-3x_2 \leqslant -10$ | $-20 \leqslant x_i \leqslant 20$ | 2 |
| Golinski | $f_1(\vec{x}) = 0.7854x_1x_2^2(10x_3^2/3+14.933x_3-43.0934)-1.508x_1(x_6^2+x_7^2)+7.477(x_6^3+x_7^3)+0.7854(x_4x_6^2+x_5x_7^2)$<br>$f_2(\vec{x}) = \frac{\sqrt{\left(\frac{745.0x_4}{x_2x_3}\right)^2+1.69*10^7}}{0.1x_6^3}$ | $g_1(\vec{x}) = \frac{1.0}{x_1x_2^2x_3}-\frac{1.0}{27.0} \leqslant 0$<br>$g_2(\vec{x}) = \frac{1.0}{x_1x_2^2x_3}-\frac{1.0}{27.0} \leqslant 0$<br>$g_3(\vec{x}) = \frac{x_4^3}{x_2x_3^2x_6^4}-\frac{1.0}{1.93} \leqslant 0$<br>$g_4(\vec{x}) = \frac{x_5^3}{x_2x_3x_7^4}-\frac{1.0}{1.93} \leqslant 0$<br>$g_5(\vec{x}) = x_2x_3-40 \leqslant 0$<br>$g_6(\vec{x}) = x_1/x_2-12 \leqslant 0$<br>$g_7(\vec{x}) = 5-x_1/x_2 \leqslant 0$<br>$g_8(\vec{x}) = 1.9-x_4+1.5x_6 \leqslant 0$<br>$g_9(\vec{x}) = 1.9-x_5+1.1x_7 \leqslant 0$<br>$g_{10}(\vec{x}) = f_2(x) \leqslant 1300$<br>$a = 745.0x_5/x_2x_3$<br>$b = 1.575*10^8$<br>$g_{11}(\vec{x}) = \frac{\sqrt{(a)^2+b)}}{0.1x_7^3} \leqslant 1100$ | $2.6 \leqslant x_1 \leqslant 3.6$<br>$0.7 \leqslant x_2 \leqslant 0.8$<br>$17.0 \leqslant x_3 \leqslant 28.0$<br>$7.3 \leqslant x_4 \leqslant 8.3$<br>$7.3 \leqslant x_5 \leqslant 8.3$<br>$2.9 \leqslant x_6 \leqslant 3.9$<br>$5.0 \leqslant x_7 \leqslant 5.5$ | 7 |

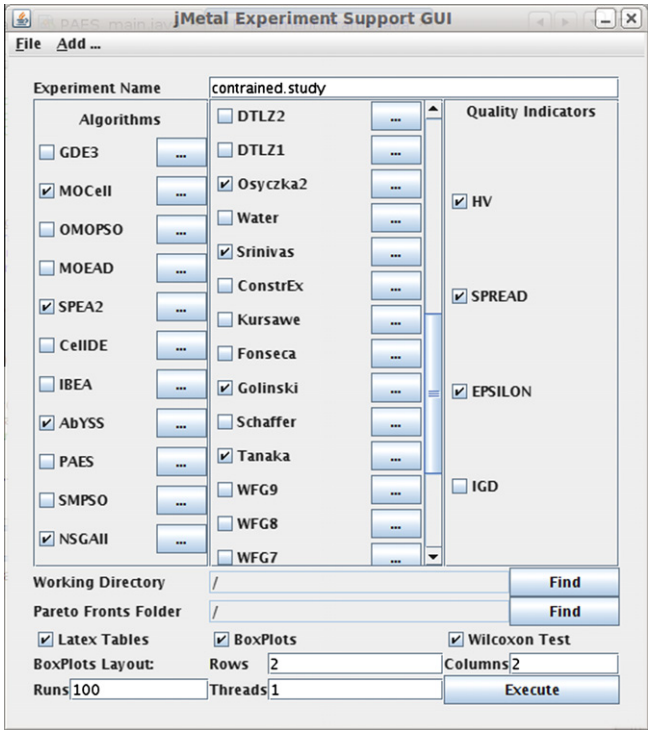


**Fig. 12.** Experiment Support GUI. Configuring an experiment.

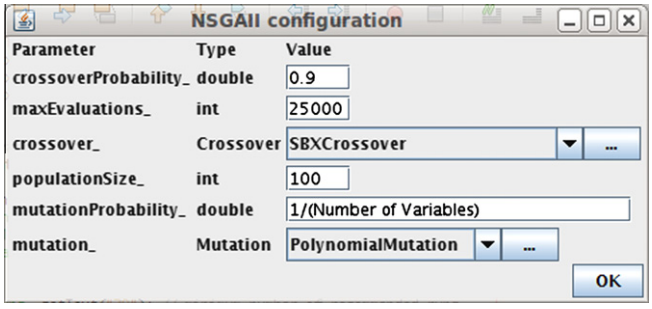


**Fig. 13.** Experiment Support GUI. Configuring an experiment.

Associated to each algorithm there is a button that allows the user to modify the configuration of that algorithm. Fig. 13 shows an example of the window which allows to configure the NSGA-II algorithm. In this block, users must select those algorithms that they want to include in their study. For example, in our case, we select NSGA-II, SPEA2, MOCell, and AbYSS. These comments remain the same for the problems block (on the center), which allows users to select and to configure properly the problems to evaluate (in our case, the constrained ones). The third block, on the right, contains a list of quality indicators that can be applied to the obtained results. For our purpose, we have to select Epsilon Indicator, Spread, and Hypervolume.

Finally, the bottom part of the window contains fields related to the location of the Pareto fronts, where to store the output data, the number of independent runs to be executed, and the statistical test to apply to the final results. In the following, we describe the most relevant fields.

- *Pareto Fronts Folder*. This field must be filled with the folder where files containing the optimal Pareto fronts of the problems involved in the comparison are stored. It is important to indicate the correct path to the location of those fronts; otherwise, the quality indicators will not be computed.
- *LaTeX tables*. This field indicates if LaTeX tables summarizing the results of the experiments (means, medians, standard deviations, interquartile ranges) should be generated.
- *Wilcoxon Test*. When this field is selected, a R script is automatically generated by jMetal. This script is aimed at applying a Wilcoxon test between each pair of algorithms in each problem. After executing it, a Latex table summarizing the test is generated.
- *Boxplots*. This field allows the user to obtain a boxplot representation of the data. Actually, it generates a R script that after being executed (just by typing *Rscript < name of the generated script>* in a command prompt in a system where R is installed), produces a boxplot summarizing the results of all the algorithms involved in the study for each problem and quality indicator used. The fields *Rows* and *Columns* are related to the boxplot layout. They indicate how many boxplots should be included by row and column, respectively.
- *Runs*. This field is used to indicate how many independent runs should be carried out.
- *Threads*. It allows the user to indicate how many threads should be used to execute the experimental study. It is useful in multicore CPUs. Nowadays, these kinds of processors are becoming very common, bringing parallel power to PCs and laptops. As a consequence, users can speed-up the execution of experiments. It is important to note that this value has not influence in the final results.

Thus, to proceed with the proposed study, we followed the next steps. First, we provided the name of the experiment. Then, we selected the algorithms and problems to include, and we configured them properly (algorithms and problems blocks, respectively).

**Table 3**
Epsilon. Median and IQR.

| | NSGAII | SPEA2 | MOCell | AbYSS |
|---|---|---|---|---|
| Golinski | $8.79e+00_{3.0e+00}$ | $1.50e+01_{5.8e+00}$ | $5.67e+00_{1.7e+00}$ | $6.71e+00_{2.4e+00}$ |
| Srinivas | $3.13e+00_{7.6e-01}$ | $1.76e+00_{1.9e-01}$ | $1.36e+00_{1.5e-01}$ | $1.56e+00_{2.5e-01}$ |
| Tanaka | $8.41e-03_{1.2e-03}$ | $1.70e-02_{1.4e-02}$ | $9.17e-03_{5.3e-03}$ | $1.57e-02_{6.2e-03}$ |
| Osyczka2 | $1.84e+00_{1.4e+01}$ | $2.51e+01_{1.6e+01}$ | $2.09e+01_{1.5e+01}$ | $1.64e+01_{1.1e+02}$ |

**Table 4**
SPREAD. Median and IQR.

| | NSGAII | SPEA2 | MOCell | AbYSS |
|---|---|---|---|---|
| Golinski | $4.49e-01_{4.9e-02}$ | $7.10e-01_{4.8e-02}$ | $1.39e-01_{4.3e-02}$ | $1.65e-01_{3.8e-02}$ |
| Srinivas | $4.03e-01_{5.1e-02}$ | $1.74e-01_{2.2e-02}$ | $6.72e-02_{1.6e-02}$ | $9.58e-02_{1.6e-02}$ |
| Tanaka | $8.05e-01_{3.7e-02}$ | $8.80e-01_{9.8e-02}$ | $7.65e-01_{8.1e-02}$ | $8.49e-01_{9.6e-02}$ |
| Osyczka2 | $5.69e-01_{1.1e-01}$ | $7.73e-01_{1.3e-01}$ | $4.91e-01_{2.1e-01}$ | $5.02e-01_{2.1e-01}$ |

**Table 5**
HV. Median and IQR.

| | NSGAII | SPEA2 | MOCell | AbYSS |
|---|---|---|---|---|
| Golinski | $9.69e-01_{2.0e-04}$ | $9.67e-01_{1.1e-03}$ | $9.69e-01_{8.8e-05}$ | $9.69e-01_{2.4e-04}$ |
| Srinivas | $5.38e-01_{4.3e-04}$ | $5.40e-01_{1.9e-04}$ | $5.41e-01_{1.0e-04}$ | $5.40e-01_{2.7e-04}$ |
| Tanaka | $3.08e-01_{4.2e-04}$ | $3.08e-01_{1.6e-03}$ | $3.09e-01_{5.2e-04}$ | $3.07e-01_{9.9e-04}$ |
| Osyczka2 | $7.51e-01_{7.8e-03}$ | $7.22e-01_{3.4e-02}$ | $7.31e-01_{2.5e-01}$ | $7.22e-01_{3.5e-01}$ |

**Table 6**
Epsilon indicator. Wilcoxon test in problems Golinski, Srinivas,Tanaka, and Osyczka2 (each problem is represented by a symbol).

| | SPEA2 | | | | MOCell | | | | AbYSS | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| NSGAII | ▲ | ▽ | ▲ | ▲ | ▽ | ▽ | ▲ | ▲ | ▽ | ▽ | ▲ | ▲ |
| SPEA2 | | | | | ▽ | ▽ | ▽ | – | ▽ | ▽ | – | – |
| MOCell | | | | | | | | | ▲ | ▲ | ▲ | – |

**Table 7**
Spread indicator. Wilcoxon test in problems Golinski, Srinivas, Tanaka, and Osyczka2 (each problem is represented by a symbol).

| | SPEA2 | | | | MOCell | | | | AbYSS | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| NSGAII | ▲ | ▽ | ▲ | ▲ | ▽ | ▽ | ▽ | ▽ | ▽ | ▽ | ▲ | ▽ |
| SPEA2 | | | | | ▽ | ▽ | ▽ | ▽ | ▽ | ▽ | ▽ | ▽ |
| MOCell | | | | | | | | | ▲ | ▲ | ▲ | – |

Once the quality indicators to apply in the study (Epsilon, Spread, and HV, in our case) were selected on the quality indicators list, we specified where to store all generated data, and where to find the Pareto front of the problems used as benchmark. After that, we indicated that LaTeX tables, Wilcoxon tests, and boxplots should be generated. As we had four problems, we marked two boxplots per row and column. The number of independent runs was set to 100, and the number of threads to one.

**Table 8**
HV indicator. Wilcoxon test in problems Golinski, Srinivas, Tanaka, and Osyczka2 (each problem is represented by a symbol).

| | SPEA2 | | | | MOCell | | | | AbYSS | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| NSGAII | ▲ | ▽ | ▽ | ▲ | ▽ | ▽ | ▽ | ▲ | ▽ | ▽ | ▲ | ▲ |
| SPEA2 | | | | | ▽ | ▽ | ▽ | – | ▽ | ▽ | ▲ | – |
| MOCell | | | | | | | | | ▲ | ▲ | ▲ | ▲ |

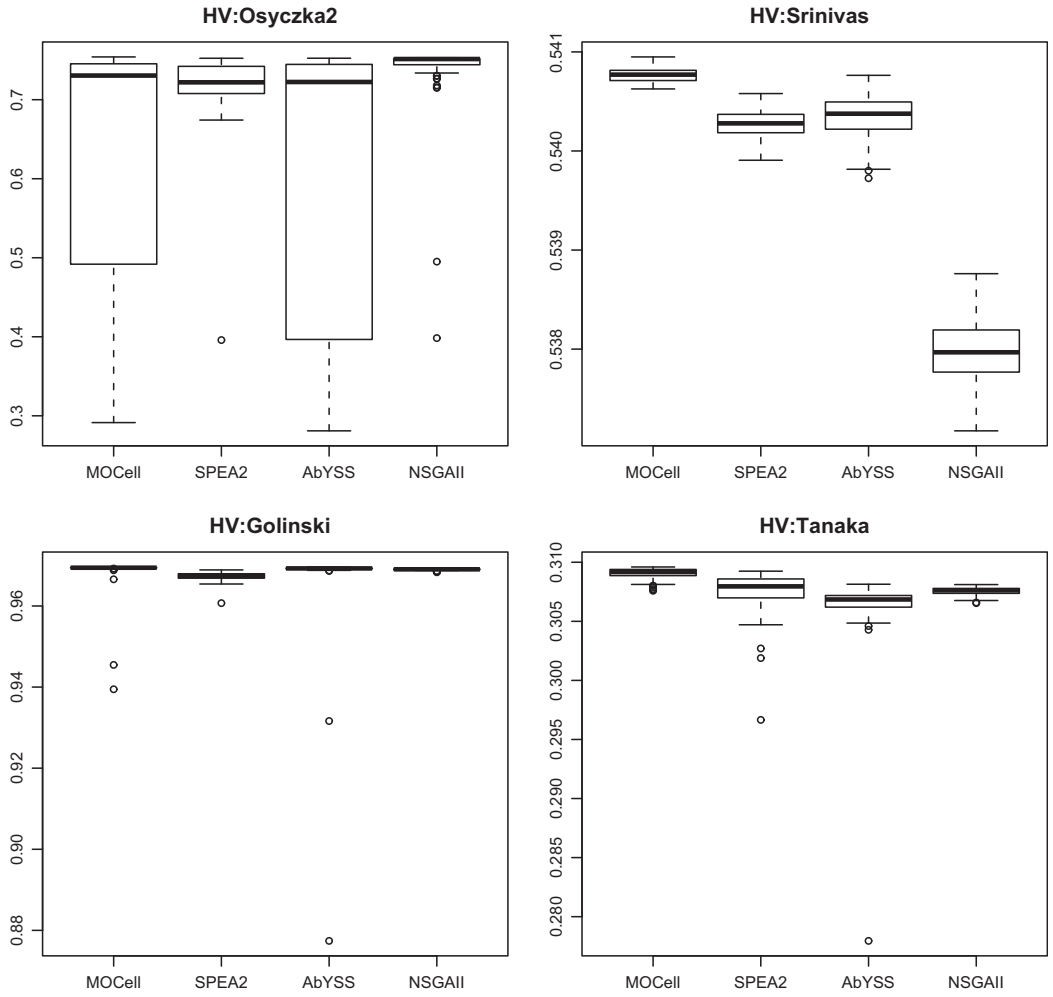


**Fig. 14.** Boxplots of the HV obtained by the different algorithms in the evaluated problems.

### 7.3. Analysis of the results

In this section, we analyze the results obtained by using jMetal in the experiment described before.

Tables 3–5 have been generated by jMetal. They included the median and interquartile range (IQR) of all the independent runs carried out for the Epsilon, $\Delta$, and HV indicators, respectively. To ease the analysis of these tables, some cells have a gray colored background in each row; particularly, there are two different gray levels: a darker one, pointing out the algorithm obtaining the best value of the indicator, and a lighter one, highlighting the algorithm obtaining the second best value of the indicator.

We start by describing the values obtained in the Epsilon indicator. For this and the $\Delta$ indicator, the lower the value the better the computed fronts. Thus, attending to Table 3, we see that the best or second best indicator values are distributed between two algorithms: MOCell and NSGA-II. MOCell has obtained the best or second best values in this indicator for all the problems but one; NSGA-II has computed the best fronts regarding to this indicator in two out of the four evaluated problems.

We turn now to analyze the computed fronts in terms of the diversity through the use of the $\Delta$ indicator. In this case, MOCell has obtained the best values for this indicator in all the cases. Another algorithm obtaining good figures in this indicator has been AbYSS: it has computed the fronts with the second best values of the indicator in three out of the four evaluated problems.

Finally, we pay attention to the HV values. Regarding to this indicator, the higher the value the better the computed fronts. The obtained results in this indicator should verify somehow the ones obtained in the previous analyzed indicators. In fact, they do: MOCell has been also the best algorithm; after it NSGA-II has obtained the best value in one out of the four problems and the second best values of the indicator in three cases.

As mentioned before, jMetal also generates R scripts for performing statistical test comparisons between each pair of algorithms. After executing these scripts, LaTeX tables summarizing the comparison between algorithms are generated. Tables 6–8 are examples of these tables and summarize the results of that test in the Epsilon, $\Delta$, and HV indicators, respectively. In each cell, the four considered MOPs are represented with a symbol. Three different symbols are used: "–" indicates that there not statistical significance between the algorithms, "▲" means that the algorithm in the row has yielded better results than the algorithm in the column with confidence, and "▽" is used when the algorithm in the column is statistically better than the algorithm in the row. Thus, deepening in the data contained in those tables, we observe that statistical confidence has been found in most of the results.

More detailed information can be obtained if we display the results by using boxplots, which constitutes a useful way of depicting groups of numerical data. Boxplots can be also built after executing a R script generated by jMetal after performing a experimental study. In this case, the boxplot representing the distribution of values for the HV indicator in the comparison carried out are showed in Fig. 14. Some interesting facts can be worked out by observing the figure. For example, in problems Srinivas and Tanaka, we observe that MOCell has clearly outperformed to the other algorithms.

Summarizing, according the problems, parameter settings, and quality indicators used, MOCell has been the most salient algorithm in our study.

## 8. Summary, conclusions and future work

In this paper we have presented jMetal, a Java framework for multi-objective optimization. Particularly, we have described the core classes composing jMetal and we have illustrated how to implement new problems and algorithms using it. In the same way, we have described how to use jMetal for computing and displaying the Pareto front of a problem by using the Single Execution Support GUI provided by this tool.

Furthermore, we have described how using the Experimental Support GUI jMetal can be used to assist in carrying out experimental studies. To deal with this issue, a case study consisting in evaluating the performance of four *state-of-the-art* multi-objective algorithms (NSGA-II, SPEA2, MOCell, and AbYSS) using a benchmark composed of four constrained problems has been proposed. The obtained results have been analyzed on the basis of three quality indicators (Epsilon, $\Delta$, and HV) computed with jMetal. Furthermore, the differences between the results has been also assessed through the use of statistical tests applied by jMetal.

As further work, we plan to extend jMetal including some other features. Particularly, we want to include some mechanisms for allowing users to interactively include preferences to guide the search towards regions of interest and for supporting decision-making.

## Acknowledgements

This work has been partially funded by the "Consejería de Innovación, Ciencia y Empresa", Junta de Andalucía, under contract P07-TIC-03044 (DIRICOM project), and the Spanish Ministry of Science and Innovation and FEDER under contract TIN2008-06491-C04-01 (M* project).

## References

[1] Weise T, Zapf M, Chiong R, Nebro AJ. Why is optimization difficult? In: Chiong R, editor. Nature-inspired algorithms for optimisation. Studies in computational intelligence, vol. 193/2009. Springer; 2009. p. 1–50.
[2] Glover FW, Kochenberger GA. Handbook of metaheuristics. Kluwer; 2003.
[3] Blum C, Roli A. Metaheuristics in combinatorial optimization: overview and conceptual comparison. ACM Comput Surv 2003;35(3):268–308.
[4] Deb K. Multi-objective optimization using evolutionary algorithms. John Wiley & Sons; 2001.
[5] Coello Coello CA, Lamont GB, Van Veldhuizen DA. Evolutionary algorithms for solving multi-objective problems. 2nd ed. New York: Springer; 2007. iSBN 978-0-387-33254-3.
[6] Deb K, Pratap A, Agarwal S, Meyarivan T. A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans Evolut Comput 2002;6(2):182–97.
[7] Knowles JD, Corne DW. Approximating the nondominated front using the pareto archived evolution strategy. Evol Comput 2000;8(2):149–72.
[8] Zitzler E, Laumanns M, Thiele L. SPEA2: Improving the strength pareto evolutionary algorithm. In: Giannakoglou K, Tsahalis D, Periaux J, Papailou P, Fogarty T, editors. EUROGEN 2001. Evolutionary methods for design, optimization and control with applications to industrial problems, Athens, Greece; 2002. p. 95–100.
[9] Reyes-Sierra M, Coello Coello CA. Multi-objective particle swarm optimizers: a survey of the state-of-the-art. Int J Comput Intell Res 2006;2(3):287–308.
[10] Bleuler S, Laumanns M, Thiele L, Zitzler E. PISA – a platform and programming language independent interface for search algorithms. In: Fonseca CM, Fleming PJ, Zitzler E, Deb K, Thiele L, editors. Evolutionary multi-criterion optimization (EMO 2003). Lecture notes in computer science. Berlin: Springer; 2003. p. 494–508.
[11] Durillo J, Nebro A, Luna F, Dorronsoro B, Alba E. jMetal: a Java framework for developing multi-objective optimization metaheuristics, Tech. Rep. ITI-2006-10, Departamento de Lenguajes y Ciencias de la Computación, University of Málaga, E.T.S.I. Informática, Campus de Teatinos; 2006.
[12] Sağ T, Çunkaş M. A tool for multiobjective evolutionary algorithms. Adv Eng Softw 2009;40(9):902–12.
[13] Corne D, Jerram N, Knowles J, Oates M. PESA-II: Region-based selection in evolutionary multiobjective optimization. In: Genetic and evolutionary computation conference (GECCO-2001), Morgan Kaufmann; 2001. p. 283–90.
[14] Reyes M, Coello Coello C. Improving PSO-based multi-objective optimization using crowding, mutation and $\epsilon$-dominance. In: Coello C, Hernández A, Zitler E, editors. Third international conference on evolutionary multicriterion optimization, EMO 2005. LNCS, vol. 3410. Springer; 2005. p. 509–19.
[15] Nebro A, Durillo J, Luna F, Dorronsoro B, Alba E. Design issues in a multiobjective cellular genetic algorithm. In: Obayashi S, Deb K, Poloni C, Hiroyasu T, Murata T, editors. Evolutionary multi-criterion optimization. 4th International conference, EMO 2007. Lecture notes in computer science, vol. 4403. Springer; 2007. p. 126–40.

[16] Nebro AJ, Luna F, Alba E, Dorronsoro B, Durillo JJ, Beham A. AbYSS: Adapting Scatter Search to Multiobjective Optimization. IEEE Trans Evol Comput 12(4):439–457.
[17] Li H, Zhang Q. Multiobjective optimization problems with complicated pareto sets, moea/d and nsga-ii. IEEE Trans Evol Comput 2009;12(2):284–302.
[18] Greiner D, Emperador J, Winter G, Galván B. Improving computational mechanics optimum design using helper objectives: an application in frame bar structures. In: Obayashi S, Deb K, Poloni C, Hiroyasu T, Murata T, editors. Fourth international conference on evolutionary multicriterion optimization, EMO 2007. Lecture notes in computer science, vol. 4403. Berlin, Germany: Springer; 2006. p. 575–89.
[19] Durillo JJ, Nebro AJ, Luna F, Alba E. Solving three-objective optimization problems using a new hybrid cellular genetic algorithm. In: Rudolph G, Jensen T, Lucas S, Poloni C, Beume N, editors. Parallel problem solving from nature-PPSN X. Lecture notes in computer science, vol. 5199. Springer; 2008. p. 661–70.
[20] Kukkonen S, Lampinen J. GDE3: The third evolution step of generalized differential evolution. In: IEEE congress on evolutionary computation (CEC'2005); 2005. p. 443–50.
[21] Eskandari H, Geiger CD, Lamont GB. FastPGA: a dynamic population sizing approach for solving expensive multiobjective optimization problems. In: Obayashi S, Deb K, Poloni C, Hiroyasu T, Murata T, editors. Evolutionary multi-criterion optimization. 4th International conference, EMO 2007. Lecture notes in computer science, vol. 4403. Springer; 2007. p. 141–55.
[22] Zitzler E, Künzli S. Indicator-based selection in multiobjective search. In: Yao X et al., editors. Parallel problem solving from nature (PPSN VIII). Berlin, Germany: Springer Verlag; 2004. p. 832–42.
[23] Nebro A, Durillo J, García-Nieto J, Coello Coello C, Luna F, Alba E. Smpso: a new pso-based metaheuristic for multi-objective optimization. In: 2009 IEEE symposium on computational intelligence in multicriteria decision-making (MCDM 2009). IEEE Press; 2009. p. 66–73.
[24] Nebro A, Alba E, Molina G, Chicano F, Luna F, Durillo J. Optimal antenna placement using a new multi-objective chc algorithm. In: GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation. New York (NY, USA): ACM Press; 2007. p. 876–83.
[25] Beume N, Naujoks B, Emmerich M. SMS-E MOA: Multiobjective selection based on dominated hypervolume. Eur J Oper Res 2007;181(3):1653–69. URL http://ideas.repec.org/a/eee/ejores/v181y2007i3p1653-1669.html.
[26] Zitzler E, Deb K, Thiele L. Comparison of multiobjective evolutionary algorithms: empirical results. Evol Comput 2000;8(2):173–95.
[27] Deb K, Thiele L, Laumanns M, Zitzler E. Scalable test problems for evolutionary multiobjective optimization. In: Abraham A, Jain L, Goldberg R, editors. Evolutionary multiobjective optimization. theoretical advances and applications. USA: Springer; 2005. p. 105–45.
[28] Huband S, Hingston P, Barone L, While L. A review of multiobjective test problems and a scalable test problem toolkit. IEEE Trans Evol Comput 2006;10(5):477–506.
[29] Zhang Q, Zhou A, Zhao SZ, Suganthan PN, Liu W, Tiwari S. Multiobjective optimization test instances for the cec 2009 special session and competition, Tech. Rep. CES-487, University of Essex and Nanyang Technological University, Essex, UK and Singapore, September 2008.
[30] Kursawe F. A variant of evolution strategies for vector optimization. In: Schwefel H, Männer R, editors. Parallel problem solving for nature. Berlin, Germany: Springer-Verlag; 1990. p. 193–7.
[31] Fonseca C, Flemming P. Multiobjective optimization and multiple constraint handling with evolutionary algorithms-part ii: application example. IEEE Trans System, Man, Cybern 1998;28:38–47.
[32] Schaffer J. Multiple objective optimization with vector evaluated genetic algorithms. In: Grefensttete J, editor. First international conference on genetic algorithms, Hillsdale, NJ; 1987. p. 93–100.
[33] Srinivas N, Deb K. Multiobjective function optimization using nondominated sorting genetic algorithms. Evol Comput 1995;2(3):221–48.
[34] Tanaka M, Watanabe H, Furukawa Y, Tanino T. Ga-based decision support system for multicriteria optimization. In: Proceedings of the IEEE international conference on systems, man, and cybernetics, vol. 2; 1995. p. 1556–1561.
[35] Osyczka A, Kundo S. A new method to solve generalized multicriteria optimization problems using a simple genetic algorithm. Struct Optimiz 1995;10:94–9.
[36] Kurpati A, Azarm S, Wu J. Constraint handling improvements for multi-objective genetic algorithms. Struct Multidiscipl Opt 2002;23(3):204–13.
[37] Ray T, Tai K, Seow K. An evolutionary algorithm for multiobjective optimization. Eng Opt 2001;33(3):399–424.
[38] Zitzler E, Thiele L. Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach. IEEE Trans Evol Comput 1999;3(4):257–71.
[39] Van Veldhuizen DA, Lamont GB. Multiobjective evolutionary algorithm research: A history and analysis, Tech. Rep. TR-98-03, Dept. Elec. Comput. Eng., Graduate School of Eng., Air Force Inst.Technol., Wright-Patterson, AFB, OH; 1998.
[40] Knowles J, Thiele L, Zitzler E. A tutorial on the performance assessment of stochastic multiobjective optimizers, Tech. Rep. 214, Computer Engineering and Networks Laboratory (TIK), ETH Zurich; 2006.
[41] Nebro AJ, Durillo JJ, Coello Coello C, Luna F, Alba E. A study of convergence speed in multi-objective metaheuristics. In: Rudolph G, Jensen T, Lucas S, Poloni C, Beume N, editors. Parallel problem solving from nature-PPSN X. Lecture notes in computer science, vol. 5199. Springer; 2008. p. 763–72.
[42] Durillo JJ, Nebro AJ, Coello Coello CA, Luna F, Alba E. A comparative study of the effect of parameter scalability in multi-objective metaheuristics. In: CEC 2008, Hong Kong; 2008. p. 255–66.
[43] Durillo J, Nebro A, Coello CC, García-Nieto J, Luna F, Alba E. A study of multi-objective metaheuristics when solving parameter scalable problems. IEEE Trans Evol Comput 2010;14(4):618–35.
[44] Durillo JJ, Nebro AJ, Luna F, Alba E. On the effect of the steady-state selection scheme in multi-objective genetic algorithms. In: Ehrgott M, Fonseca C, Gandibleux X, Hao J, Sevaux M, editors. 5th International conference, EMO 2009. Lecture notes in computer science, vol. 5467. Springer; 2009. p. 183–97.
[45] Durillo J, Nebro A, Alba E. The jmetal framework for multi-objective optimization: Design and architecture. In: CEC 2010, Barcelona, Spain; 2010, p. 4138–325.
[46] Nebro A, Durillo J. jMetal 3.1 User Manual; 2010.
[47] Zhou A, Jin Y, Zhang Q, Sendhoff B, Tsang E. Combining model-based and genetics-based offspring generation for multi-objective optimization using a convergence criterion. In: 2006 IEEE Congress on evolutionary computation; 2006. p. 3234–41.
[48] Hooker J. Testing heuristics: we have it all wrong. J Heuristics 1995;1:33–42.