The jMetal Framework for Multi-Objective Optimization: Design and Architecture

Juan J. Durillo, Antonio J. Nebro and Enrique Alba

***Abstract*— jMetal is a Java-based framework for multi-objective optimization using metaheuristics. It is an ease-to-use, flexible, and extensible software package that has been used in a wide range of applications. In this paper, we describe the design issues underlying jMetal, focusing mainly in its internal architecture, with the aim of offering a comprehensive view of its main features to interested researchers. Among the covered topics, we detail the basic components facilitating the implementation of multi-objective metaheuristics (solution representations, operators, problems, density estimators, archives), the included quality indicators to assess the performance of the algorithms, and jMetal's support to carry out full experimental studies.**

I. Introduction

The optimization of problems involving two or more conflicting objectives has received an increasing attention in the last years. Usually, these problems are referred as Multi-objective Optimization Problems (MOPs). Typically, there exists no single solution or point which optimizes all the objective functions at the same time; thus, the optimum of a MOP is given by a set of points known as the Pareto optimal set. The elements in this set are said to be non-dominated since none of them is better than the others for all the objective function values. The representation of the Pareto optimal set in the objective space is known as the Pareto front.

As in single-objective optimization problems, MOPs may present features such as non-linear formulations, constraints, and NP-hard complexity that make them difficult to solve. In that context, the application of exact techniques for solving them is not practical, and approximate techniques are required. Among them, Metaheuristics [1] are widely used. They constitute a family of techniques comprising methods such as Evolutionary Algorithms (EAs), Evolution Strategies (ES), Scatter Search (SS), Particle Swarm Optimizers (PSO), and Ant Colony Optimization (ACO), being EAs the most well-known algorithms [2][3].

Obtaining the Pareto front of a MOP is the main goal of multi-objective optimization. In theory, a Pareto front can contain a large (possibly infinite) number of points. In practice, when using metaheuristics the aim is to get an approximation to the Pareto front by computing a finite number of solutions. In this context, this set of solutions must fulfill two features to be useful to the decision maker: the points should be as close as possible to the exact Pareto front (*convergence*) and be uniformly spread (*diversity*). 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.

To cope with those goals, new methods and algorithms are continuously appearing. In consequence, the existence of software libraries which assist the development of these techniques is desirable. An example of this kind of software is jMetal [4], a Java framework aimed at multi-objective optimization with metaheuristics. It is an open-source software that can be obtained from `http://jmetal.sourceforge.net`.

The goals driving the design of jMetal are to:

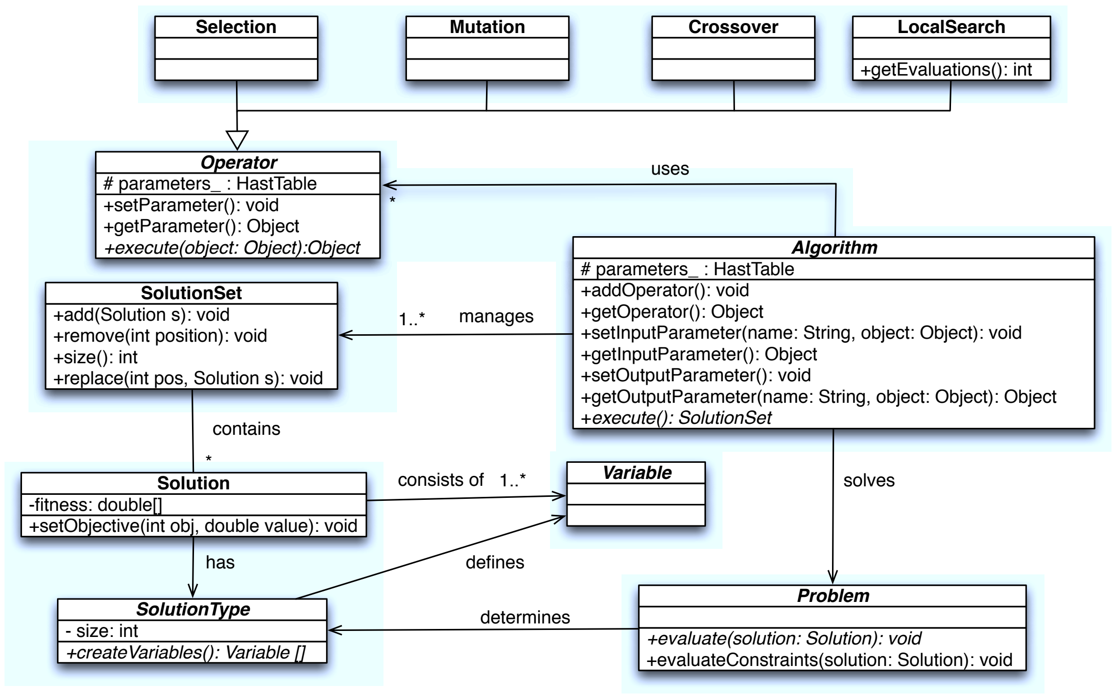
- Provide the users with an easy-to-use, flexible, and extensible tool. This is achieved thanks to its object-oriented design and to an internal architecture focused on multi-objective metaheuristics, where the core components of the algorithms are clearly defined.
- To assist in the overall process of researching with metaheuristics algorithms: implementation of techniques, definition of the problems to solve, performance assessment indicators, and support for carrying out extensive experimental studies.

During the development of jMetal, other related Java-based tools have appeared, e.g., EVA2[1], OPT4j[2], or ECJ[3]. 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 focused mainly on evolutionary algorithms, and many of them are centred in single-objective optimization, offering some extensions to the multi-objective domain. There are other tools, such as PISA [5] or MOEO [6], but they require to use C/C++ instead of Java.

jMetal has been used in a wide range of applications. Some examples are: solving bin packing problems in automobile sheet metal forming processes [7], decision support in several disciplines [8][9], design and comparison of new algorithms [10][11] [12][13][14], the optimization of the flow of experiments to a Robot Scientist [15], solving the record linkage problem [16], replication of overlay networks [17], optimization of planning task [18], optimization of software engineering problems [19] [20], and the optimization of vehicular ad hoc networks [21]. Furthermore, jMetal has been also considered as the main software tool in PhD dissertations such as [22] and [23].

Juan .J Durillo, Antonio .J. Nebro, Enrique Alba are with the Departamento de Lenguajes y Ciencias de la Computación, University of Málaga, Spain (e-mail: {durillo, antonio, eat}@lcc.uma.es).

[1]EVA2: `http://www.ra.cs.uni-tuebingen.de/software/EvA2/`
[2]OPT4j: `http://opt4j.sourceforge.net/`
[3]ECJ: `http://cs.gmu.edu/ eclab/projects/ecj/`



Selection
Mutation
Crossover
LocalSearch
+getEvaluations(): int
Operator
# parameters_ : HastTable
+setParameter(): void
+getParameter(): Object
+execute(object: Object):Object
uses
*
Algorithm
# parameters_ : HastTable
+addOperator(): void
+getOperator(): Object
+setInputParameter(name: String, object: Object): void
+getInputParameter(): Object
+setOutputParameter(): void
+getOutputParameter(name: String, object: Object): Object
+execute(): SolutionSet
SolutionSet
+add(Solution s): void
+remove(int position): void
+size(): int
+replace(int pos, Solution s): void
1..*
manages
contains
*
Solution
-fitness: double[]
+setObjective(int obj, double value): void
consists of
1..*
Variable
solves
has
defines
SolutionType
- size: int
+createVariables(): Variable []
determines
Problem
+evaluate(solution: Solution): void
+evaluateConstraints(solution: Solution): void

Fig. 1. Base classes into jMetal.

The main goal of this paper is to describe the design and architecture of jMetal. The idea is to present a comprehensive view of the framework detailing the elements composing its fundamental components and to detail the main facilities provided. To deal with these issues, we focus first in how to represent solutions (i.e., encodings) and the operations that can be applied to them. After that, we take a look to the implementation of problems and algorithms. We also describe how quality indicators are included in the framework. Finally, we explain how jMetal gives support for making experimental studies involving different metaheuristics and problems.

The rest of this paper is organized as follows. First, the base architecture of jMetal is presented in Section II. Next section is devoted to detailing the quality indicators included in jMetal. In Section V, we describe how our software can be used to carry out experimental studies. Finally, the conclusions and some lines of future work are presented.

II. Architecture of jMetal

jMetal is designed following an object-oriented architecture which facilitates the inclusion of new components and the re-use of existing ones. It provides the user with a set of base classes which can be extended to design new algorithms, implement problems and operators, etc.

Fig. 1 depicts a UML diagram including the core classes within the framework. The working principle of jMetal is that an *Algorithm* solves a *Problem* using one (possibly more) *SolutionSet* and a set of *Operator* objects. We see in the diagram the relationships among these classes and others that will be described next.

A generic terminology to name the classes has been employed in order to make them general enough to be used in a wide variety of metaheuristics. This way, in the context of EAs, populations and individuals correspond to *SolutionSet* and *Solution* jMetal classes, respectively; the same can be applied to PSO algorithms concerning the concepts of swarm and particles.

A. Encoding of Solutions

One of the first decisions that have to be taken when using metaheuristics is to define how to encode or represent the tentative solutions of the problem to solve. Representation strongly depends on the problem and determines the operations (e.g., recombination with other solutions, local search procedures, etc.) that can be applied. Thus, selecting a specific representation has a great impact on the behaviour of metaheuristics and, hence, in the obtained results.

Fig. 2 depicts the basic components that are used for representing solutions into the framework. A *Solution* is composed of set of *Variable* objects, which can be of different types (binary, real, binary-coded real, integer, permutation, etc) plus an array to store the fitness values. With the idea of providing a flexible and extensible scheme, each *Solution* has associated a type (the *SolutionType* class in the figure). The solution type allows to define the variable types of the *Solution* and create them, by using the *createVariables()* method. This is illustrated in Fig. 3, which shows the code of the *RealSolutionType* class (we have omitted irrelevant details), used to characterize solutions composed only by real variables. jMetal provides similar solutions types to represent

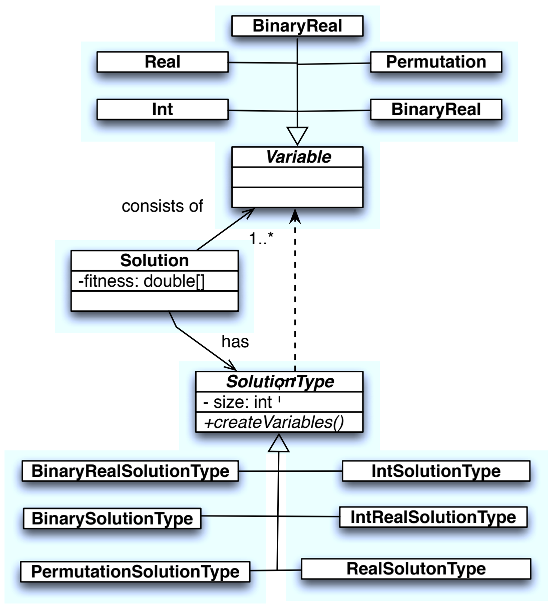


Fig. 2. Elements describing solution representations into jMetal.

```
public class RealSolutionType extends SolutionType {
  // Constructor
  public RealSolutionType(Problem problem)  {
    ...
  } // Constructor

  public Variable[] createVariables() {
    Variable[] variables = new Variable[problem_.getNumberOfVariables()];

    for (int var = 0; var < problem_.getNumberOfVariables(); var++)
       variables[var] = new Real();

    return variables ;
  } // createVariables
} // RealSolutionType
```

Fig. 3. Code of the *RealSolutionType* class, which represents solutions of real variables.

integer, binary, permutation, and other representations, as can be seen in Fig. 2.

The interesting point of using solution types is that it facilitates the definition of more complex representations, mixing different variable types. For example, if we need a new solution representation consisting in a real, an integer, and a permutation of integers, a new class extending *SolutionType* has to be defined for representing the new type, where basically only the *createVariables()* method should be redefined. Fig. 4 shows the code required for this new type of solution.

Once we have the means to define or using existing solution representations, we can create solutions that can be grouped into *SolutionSet* objects (i.e., populations or swarms).

```
  public Variable[] createVariables() {
    Variable[] variables = new Variable[3];

    variables[0] = new Real();
    variables[1] = new Int();
    variables[2] = new Permutation();

    return variables;
  } // createVariable
```

Fig. 4. Code of the *createVariables()* method for creating solutions consisting on a Real, an Integer, and a Permutation

B. Operators

Metaheuristic techniques are based on modifying or generating new solutions from existing ones by means of the application of different operators. For example, EAs make use of crossover, mutation, and selection operatos for modifying solutions. In jMetal, any operation altering or generating solutions (or sets of them) inherits from the *Operator* class, as can be seen in Fig. 1.

The framework already incorporates a number of operators, which can be classified into four different classes:

- *Crossover*. Represents the recombination or crossover operators used in EAs. Some of the included operators are the simulated binary (SBX) crossover [2] and the two-points crossover for real and binary encodings, respectively.
- *Mutation*. Represents the mutation operator used in EAs. Examples of included operators are polynomial mutation [2] (real encoding) and bit-flip mutation (binary encoding).
- *Selection*. This kind of operator is used for performing the selection procedures in many EAs. An example of selection operator is the binary tournament.
- *LocalSearch*. This class is intended for representing local search procedures. It contains an extra method for consulting how many evaluations have been performed after been applied.

Each operator contains the *setParameter()* and *getParameter()* methods, which are used for adding and accessing to an specific parameter of the operator. For example, the SBX crossover requires two parameters, a crossover probability (as most crossover operators) plus a value for the distribution index (specific of the operator), while a single point mutation operator only requires the mutation probability.

It is worth noting that when an operator is applied on a given solution, the solution type of this one is known. Thus, we can define, for example, a unique two-points crossover operator that can be applied to binary and real solutions, using the solution type to select the appropriate code in each case.

C. Problems

In jMetal, all the problems inherits 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 this 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 [25].

A key design feature in jMetal is that each problem defines the allowed solutions types that are suitable to solve it. Fig. 5

```
1. public class Kursawe extends Problem {
2. // Constructor
3.   public Kursawe(String solutionType, Integer numberOfVariables) {
4.      numberOfVariables_   = numberOfVariables ;
5.      numberOfObjectives_  = 2                 ;
6.      numberOfConstraints_ = 0                 ;
7.      problemName_         = "Kursawe"         ;
8.
9.      if (solutionType.compareTo("BinaryReal") == 0)
10.       solutionType_ = new BinaryRealSolutionType(this) ;
11.     else if (solutionType.compareTo("Real") == 0)
12.       solutionType_ = new RealSolutionType(this) ;
16.  } // Kursawe
17.
18.  /**
19.  * Evaluates a solution
20.  */
21.  public void evaluate(Solution solution)  {
22.    double f1 = 0.0, f2 = 0.0;
23.    // computing f1 value
24.    ...
25.    // computing f2 value
26.    ...
27.    solution.setObjective(0, f1);
28.    solution.setObjective(1, f2);
29.  } // evaluate
30. } // Kursawe
```

Fig. 5. Code of the class implementing problem Kursawe.

shows the code used for implementing the known Kursawe's problem (we have omitted again irrelevant code). As we can observe, it extends class *Problem* (line 1). After that, a constructor method is defined for creating instances of this problem (lines 3-16), which has as parameter a string containing a solution type identifier. The basic features of the problem (number of variables, number of objectives, and number of constraints) are defined next (lines 4-6). The sentences between lines 9-12 are used to create the corresponding *SolutionType* object and assign it to a state variable. Specifically, in this case the problem only allows real and binary-coded real as solution representation, i.e., *RealSolutionType* and *BinaryRealSolutionType* solution types.

After the constructor, the *evaluate()* method is redefined (lines 18-29); in this method, after computing the two objective function values, they are stored into the solution by using the *setObjective* method of *Solution* (lines 27 and 28).

Many of commonly used benchmark problems are already included in jMetal. Examples are the ones proposed by Zitzler-Deb-Thiele (ZDT) [36], Deb-Thiele-Laumanns-Zitzler (DTLZ) [37], Walking-Fish-Group (WFG) test problems [38]), and the Li-Zhang benchmark [32].

D. Algorithms

The last core class in the UML diagram in Fig. 1 to comment is *Algorithm*, an abstract class which must be inherited by the metaheuristics included in the framework. In particular, the abstract method *execute()* must be implemented; this method is intended to run the algorithm, and it returns a solution set as a result. As each metaheuristic technique may require some application-specific parameters, *Algorithm* provides two methods for adding and accessing to them: *addParameter()* and *getParameter()*, respectively. Similarly, an algorithm may also make use of some operators, so methods for incorporating operators (*addOperator()*) and to get them (*getOperator()*) are provided.

We do not include in this section the complete description of an algorithm due to space restrictions. Interested readers can find in [4] and in jMetal's user manual (`http://jmetal.sourceforge.net/documentation`) a detailed description of the code for implementing NSGA-II [25].

Besides NSGA-II, jMetal includes the implementation of a number of both classic and modern multi-objective optimizers; some examples are: SPEA2 [27], PAES [26], OMOPSO [29], MOCell [30], AbYSS [31], MOEA/D [32], GDE3 [33], IBEA [34], and SMPSO [35].

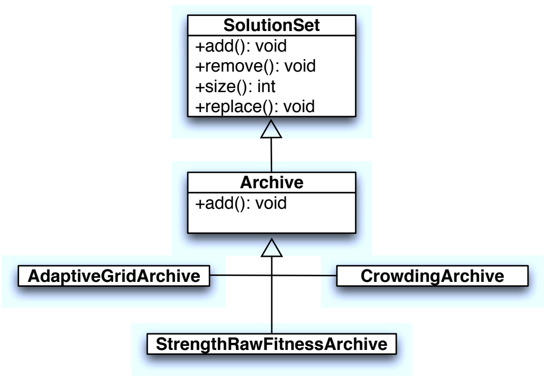


Fig. 6. Archives included in jMetal.

III. Diversity Preservation and Archives

As stated in the introduction, the main goal when solving a MOP is to compute an accurate approximation to its Pareto front in terms of convergence and diversity of solutions. To avoid to converge to small regions of the Pareto front, most of multi-objective metaheuristics incorporate mechanisms to promote diversity. Examples of these mechanisms are the crowding distance of NSGA-II, the hypergrid used by PAES, or the density estimator used by SPEA2.

These diversity mechanisms are frequently applied in combination with external bounded archives. The typical procedure is that, whenever a new solution is evaluated, it is sent to the archive to be inserted. The insertion method compares the new solution against the contents of the archive and the new solution is inserted if it is non-dominated with respect to all the solutions in the archive; otherwise, it is discarded. Furthermore, those solutions in the archive dominated by the new solution are removed. As the archives are bounded, they can become full after a solution insertion, so a given diversity scheme is used to remove the solution which less contributes to the diversity.

All these operations are facilitated by using jMetal archives. An archive is only a kind of *SolutionSet* which redefines the method to add solutions to perform the aforementioned actions. Fig. 6 depicts the archives currently included into jMetal: *CrowdingArchive*, *StrengthRawFitnessArchive*, and *AdaptiveGridArchive* based on the diversity preservation mechanism of NSGAII, SPEA2, and PAES, respectively.

IV. Quality Indicators

This section is devoted to present the quality indicators included in jMetal and to describe how they can be used for different purposes.

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, as an approximation set to the Pareto front of the problem is computed, some kinds of indicators have to be defined in order to assess the quality of the obtained set of solutions. The quality is measured in terms of some criteria which are typically related to the convergence and diversity properties.

A number of quality indicators have been proposed in the literature: Generational Distance (GD) [39], Inverse Generational Distance (IGD), Hypervolume (HV) [40], Epsilon [41], Spread or Δ [2], Generalized Spread indicators, and many others. Some of them are intended to measuring only the convergence or diversity properties, and others take into account both criteria. Fig. 7 depicts a classification of the indicators based in which aspects they measure.

In jMetal, quality indicators are directly implemented into the framework, and they can be used in several ways:

- Inside the algorithms (inline). This allows, for example, to develop indicator based algorithms, such as IBEA, or to establish a termination condition consisting of getting a front with an specific value of a given indicator. We applied this latter idea for studying the scalability and convergence speed of different algorithms using jMetall [42][43][44].
- To assess the performance of the front returned by an algorithm (offline).
- As standalone applications for computing the quality of whatever Pareto front approximation.

V. Support for Experimental Studies

Many research works in the field of multi-objective metaheuristics are related to proposing a new algorithm which is compared at least with some reference metaheuristics (i.e., NSGA-II and, to a lesser extent, SPEA2) in order to put the reader in context about the performance of the results it yields. Other works involve the analysis of variants of the same algorithm in order to study the sensitivity analysis with respect to a given parameter.

In these studies, the algorithm comparisons typically consist in using them to solve a number of benchmark problems

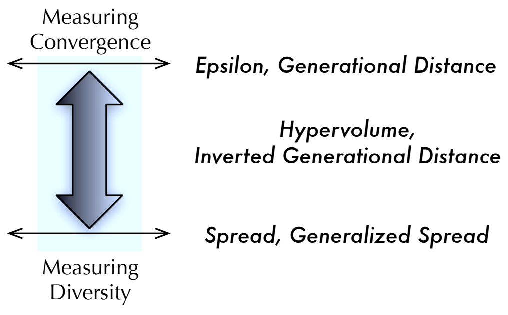

Measuring Convergence
Epsilon, Generational Distance
Hypervolume,
Inverted Generational Distance
Spread, Generalized Spread
Measuring Diversity

Fig. 7. A classification of quality indicators.

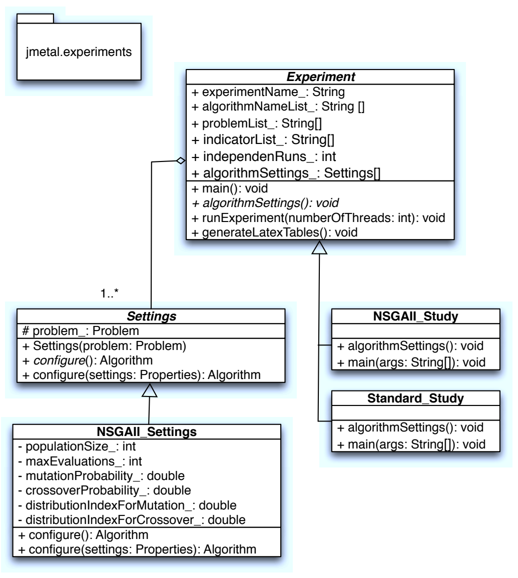

jmetal.experiments
Experiment
+ experimentName_: String
+ algorithmNameList_: String []
+ problemList_: String[]
+ indicatorList_: String[]
+ independenRuns_: int
+ algorithmSettings_: Settings[]
+ main(): void
+ algorithmSettings(): void
+ runExperiment(numberOfThreads: int): void
+ generateLatexTables(): void
1..*
Settings
# problem_: Problem
+ Settings(problem: Problem)
+ configure(): Algorithm
+ configure(settings: Properties): Algorithm
NSGAII_Study
+ algorithmSettings(): void
+ main(args: String[]): void
Standard_Study
+ algorithmSettings(): void
+ main(args: String[]): void
NSGAII_Settings
- populationSize_: int
- maxEvaluations_: int
- mutationProbability_: double
- crossoverProbability_: double
- distributionIndexForMutation_: double
- distributionIndexForCrossover_: double
+ configure(): Algorithm
+ configure(settings: Properties): Algorithm

Fig. 8. Classes and their relationships in the `jmetal.experiments` package.

and to apply some quality indicators for performance assessment. As metaheuristics are non-deterministic techniques, trying to infer some conclusion after one execution of them does not make sense [45], so the usual procedure is to carry out a number N of independent runs per experiment and then to use means and standard deviations (or medians and interquartile ranges) for analyzing the obtained results (a minimum value of $N = 30$ is usually recommended; in our own research works, we use $N = 100$). 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.

To support these kinds of experimental studies, we include in jMetal a package (see Fig. 8) which facilitates the execution of such comparisons.The classes into this package can be used directly modifying their code or extending them, allowing full flexibility and control, or indirectly by using a GUI (Graphical User Interface) tool, which is far more simple but less powerful. We detail the two options in the following.

As it is depicted in the UML diagram in Fig. 8, only two abstract classes, *Experiment* and *Settings* are needed. The first one is intended to fully describe the experiment to carry out, while the second one is used to set the parameters of the algorithms involved in the experiment. An experimental study requires to define the corresponding subclass of *Experiment*, and to use as many *Settings* objects as algorithms (or variants of a same one) to execute.

An example of a subclass of *Settings* is included in Fig. 8, the class named *NSGAII_Settings*, which is used to indicate the parameter settings of NSGA-II. *Setting* subclasses allows to specify the parameters to configure, their default values as well as modifying them. As a consequence, if we plan to run the algorithms with their typical settings, nothing has to be changed. For each metaheuristic in jMetal we have included its corresponding subclass of *Settings* containing all the information related to that particular algorithm. This way, researchers defining their own techniques have plenty of examples to help them in writing the corresponding *Settings* subclasses.

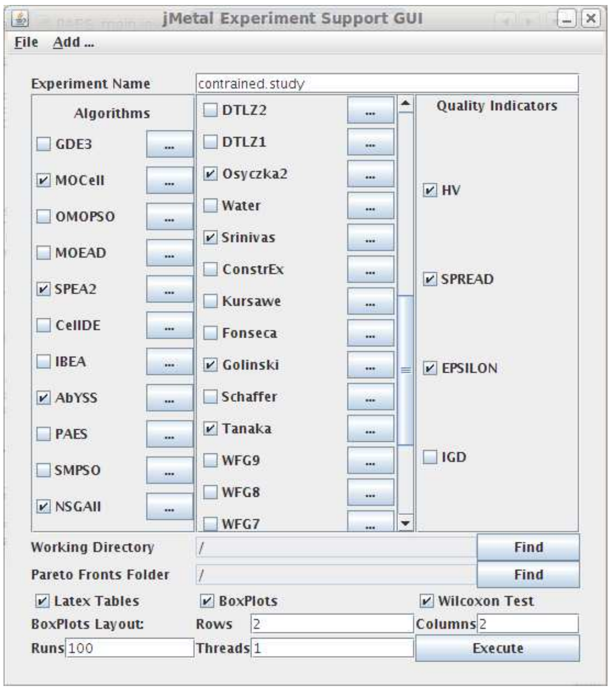
We provide in jMetal two examples of instances of *Experiment*, the so-called *NSGAStudy* and *StandardStudy*. The first one shows how to configure NSGA-II with four crossover probabilities and compare them when solving six problems; the second one contains a complete comparison of five metaheuristics (NSGA-II, SPEA2, MOCell, SMPSO, GDE3) when solving the ZDT, DTLZ, and WFG benchmarks, considering the hypervolume, spread, and epsilon indicators. For each experiment, a number of choices have to be indicated; we comment them by describing the GUI tool next.

The main window of the GUI provided by jMetal is showed in Figure 9. This window is divided into three main regions: top, centre, and bottom. The top part of the GUI is only used to provide the name of the experiment.

The central part is sub-divided into 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. Associated to each algorithm there is a button which allows the user to modify the settings of that algorithm. In this block, users may select those algorithms that they want to include in their study. These rules remains the same for the problems block (on the centre), which allows to select and to configure properly the problems to evaluate. The third block, on the right, contains a list of quality indicators that can be applied to the obtained results.

Finally, the bottom part of the window contains fields related to the location of the Pareto fronts of the problems to solved (which are needed to apply the quality indicators), the folder where to store the output data, the number of independent runs to be executed, and the statistical tests to apply to the final results. As we observe, there is an input field for each of the parameters related to the experiments. Let us note the field named *Threads*, which is used to indicate the number of parallel threads that can be started in order to execute the experiments in parallel. This way, the total execution time can be reduced significantly if we are using a multi-core processor computer.

Once the experiments have been executed, jMetal provides three different ways to show information about the obtained results. First, it is possible to automatically generate LaTeX tables summarizing the information of the different quality indicators applied. An example is included in Table I, which shows the median and interquartile range (tables with means and standard deviations are also generated) of the Epsilon indicator obtained by five different algorithms (Algorithms 1-5) when solving the problems composing the LZ09 benchmark. As we can observe, for each problem the algorithms obtaining the best and second best values of the indicator have been marked with different levels of gray as background



Experiment Name: contrained.study

Algorithms: ☐ GDE3, ✓ MOCell, ☐ OMOPSO, ☐ MOEAD, ✓ SPEA2, ☐ CellDE, ☐ IBEA, ✓ AbYSS, ☐ PAES, ☐ SMPSO, ✓ NSGAII

Problems: ☐ DTLZ2, ☐ DTLZ1, ✓ Osyczka2, ☐ Water, ✓ Srinivas, ☐ ConstrEx, ☐ Kursawe, ☐ Fonseca, ✓ Golinski, ☐ Schaffer, ✓ Tanaka, ☐ WFG9, ☐ WFG8, ☐ WFG7

Quality Indicators: ✓ HV, ✓ SPREAD, ✓ EPSILON, ☐ IGD

Working Directory: /

Pareto Fronts Folder: /

✓ Latex Tables, ✓ BoxPlots, ✓ Wilcoxon Test

BoxPlots Layout: Rows 2, Columns 2

Runs 100, Threads 1

Fig. 9. Configuring an experiment with the GUI tool.

color.

The *Wilcoxon Text* checkbox in the bottom part of the GUI, if selected, generates R scripts to apply the Wilcoxon rank sum test, a non-parametric statistical hypothesis test, which allows to make pairwise comparisons between algorithms to know about the significance of the obtained data [46]. When these scripts are run, they produce LaTeX tables as the one included in Table II, which summarizes the results after applying the test to the hypervolume values (HV) to the approximation sets computed by four metaheuristics (NSGA-II, SPEA2, MOCell, and AbYSS) when solving four problems (Golinski, Srinivas, Tanaka, and Osyczka2). In each cell, each problems is represented with a symbol. In concrete, three different symbols are used:"–" indicates that there is not statistical significance between the performance of the algorithms in the corresponding row and column, "▲" means that the algorithm in the row has obtained 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.

Finally, if the *Boxplots* checkbox is selected, additional R scripts are generated to produce boxplot representation of the results. An example is included Fig. 10, which displays the four graphics resulting when applying the epsilon indicator to the results produced by the same four algorithms used in Table II when solving the same problems. The arrange of the boxplots can be indicated in the layout section, in the bottom part, of the GUI; in the example, the layout is a 2x2 grid.

VI. Conclusions and Future Work

We have detailed the general architecture of the jMetal framework. Concretely, we have presented the main classes

TABLE I

MEDIAN AND INTERQUARTILE RANGE OF THE $I^1_{\epsilon+}$ INDICATOR. CELLS WITH DARK AND LIGHT BACKGROUND INDICATE THE BEST AND SECOND BEST VALUES, RESPECTIVELY.

	Algorithm 1	Algorithm 2	Algorithm 3	Algorithm 4	Algorithm 5
LZ09_F1	$2.31e-03_{2.9e-04}$	$2.28e-03_{3.1e-04}$	$2.32e-03_{2.8e-04}$	$2.27e-03_{3.5e-04}$	$2.29e-03_{2.9e-04}$
LZ09_F2	$7.98e-02_{2.2e-01}$	$2.18e-01_{2.4e-01}$	$2.05e-01_{2.2e-01}$	$1.99e-01_{2.1e-01}$	$6.80e-02_{2.2e-01}$
LZ09_F3	$6.18e-02_{2.2e-01}$	$5.41e-02_{2.1e-01}$	$6.21e-02_{2.2e-01}$	$5.48e-02_{8.7e-02}$	$5.53e-02_{2.2e-01}$
LZ09_F4	$5.55e-02_{2.1e-02}$	$4.98e-02_{2.0e-02}$	$5.05e-02_{1.8e-02}$	$4.99e-02_{2.4e-02}$	$4.92e-02_{1.4e-02}$
LZ09_F5	$7.78e-02_{2.7e-02}$	$7.83e-02_{2.8e-02}$	$7.86e-02_{3.1e-02}$	$7.79e-02_{1.8e-02}$	$7.63e-02_{1.9e-02}$
LZ09_F6	$1.27e-01_{2.8e-02}$	$1.21e-01_{2.7e-02}$	$1.24e-01_{2.4e-02}$	$1.19e-01_{2.1e-02}$	$1.25e-01_{3.2e-02}$
LZ09_F7	$3.25e-02_{3.4e-02}$	$3.04e-02_{2.3e-02}$	$3.03e-02_{3.9e-02}$	$2.72e-02_{2.4e-02}$	$3.18e-02_{2.4e-02}$
LZ09_F8	$1.99e-01_{8.3e-02}$	$1.85e-01_{8.5e-02}$	$2.02e-01_{9.4e-02}$	$2.09e-01_{9.1e-02}$	$2.02e-01_{8.4e-02}$
LZ09_F9	$4.14e-02_{8.8e-02}$	$3.75e-02_{7.1e-02}$	$3.75e-02_{5.1e-02}$	$3.84e-02_{6.4e-02}$	$5.71e-02_{3.6e-01}$

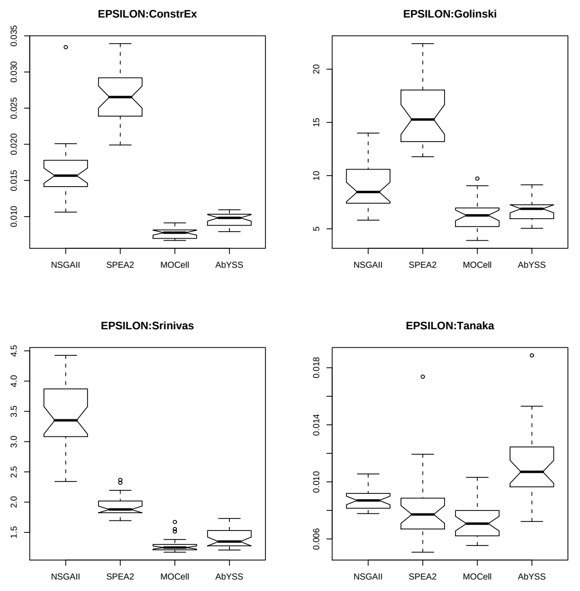


Fig. 10. Boxplots obtained automatically by running the R scripts generated by jMetal.

constituting the core of the multi-objective metaheuristics developed with the tool (solution encodings, operators, density estimators, archives) and the included quality indicators, which can be used offline as standalone applications for performance assessment or inline into the algorithms.

We have also described jMetal's support to carry out experimental studies involving the comparison of a number of techniques to solve a set of problems. By using a graphical tool (easy-to-use) or by encoding directly a set of classes (more flexibility and control), it is possible to configure the metaheuristics to use, the problems to solve, the quality indicators to apply, and how the results can be presented. This includes the automatic generation of LaTeX tables, highlighting the best and second best values, R scripts to display information about the significance of the results after applying the Wilcoxon rank sum test, and R scripts to present the results by using boxplots.

jMetal is a software that is in continuous development. Among the next features to include we are considering improving the graphical tool and including support for decision making.

TABLE II

GOLINSKI SRINIVAS TANAKA OSYCZKA2 .HV.

	SPEA2	MOCell	AbYSS
NSGAII	▲ ▽ ▽ ▲	▽ ▽ ▽ ▲	▽ ▽ ▲ ▲
SPEA2		▽ ▽ ▽ –	▽ ▽ ▲ –
MOCell			▲ ▲ ▲ ▲

ACKNOWLEDGMENTS

This work has been partially funded by the "Consejería de Innovación, Ciencia y Empresa", Junta de Andalucía under contract P07-TIC-03044 DIRICOM project (`http://diricom.lcc.uma.es`), and the Spanish Ministry of Science and Innovation and FEDER under contract TIN2008-06491-C04-01 (the M* project). Juan J. Durillo is supported by grant AP-2006-03349 from the Spanish goverment.

REFERENCES

[1] F. Glover and G. A. Kochenberger. "Handbook of Metaheuristics", Kluwer Academic Publishers, 2003.

[2] K. Deb, "Multi-objective Optimization Using Evolutionary Algorithms", John Wiley & Sons, 2001.

[3] C. A. Coello Coello, G. B. Lamont, D. A. Van Veldhuizen, "Evolutionary Algorithms for Solving Multi-Objective Problems", 2nd Edition, Springer, New York, 2007, ISBN 978-0-387-33254-3.

[4] J.J. Durillo, A.J. Nebro, F. Luna, B. Dorronsoro, and E. Alba, "jMetal: a java Framework for Developing Multi-objective Optimization Metaheuristics", Technical Report ITI-2006-10, Departamento de Lenguajes y Ciencias de la Computación, University of Málaga, E.T.S.I. Informática, Campus de Teatinos, 2006.

[5] S. Bleuler, M. Laumanns, L. Thiele, E. Zitzler, "PISA — a Platform and Programming Language Independent Interface for Search Algorithms", in: C. M. Fonseca, P. J. Fleming, E. Zitzler, K. Deb, L. Thiele (Eds.), Evolutionary Multi-Criterion Optimization (EMO 2003), Lecture Notes in Computer Science, Springer, Berlin, 2003, pp. 494 – 508.

[6] A. Liefooghe, M. Basseur, L. Jourdan, E.-G. Talbi, "ParadisEO-MOEO: A framework for evolutionary multi-objective optimization", in: S. Obayashi, K. Deb, C. Poloni, T. Hiroyasu, T. Murata (Eds.), Fourth International Conference on Evolutionary MultiCriterion Optimization, EMO 2007, Vol. 4403 of Lecture Notes in Computer Science, Springer, Berlin, Germany, 2007, pp. 386–400.

[7] M. Sathe, O. Schenk and H. Burkhart, "Solving Bi-Objective Many-Constraint Bin Packing Problems in Automobile Sheet Metal Forming Processes", Proceedings of the 5th International Conference on Evolutionary Multi-Criterion Optimization.

[8] D. F. Pacheco, F. R. S. Oliveira and F. B. Lima Neto, "Including Multi-Objective Abilities in the Hybrid Intelligent Suite for Decision Support", Proceedings of the IEEE International Joint Conference on Neural Networks, 2008.

[9] P. Jankowski and G. Fraley, "A Geovisual Analytics Approach to Spatial Multiple Objective Optimization', Proceedings of the 24th International Cartographic Conference.

[10] D. Dasgupta, G. Hernandez, D. Garrett, P. K. Vejandla, A. Kaushal, R. Yerneni, "A Comparison Of Multiobjective Evolutionary Algorithms with Informed Initialization and Kuhn-Munkres Algorithm For The Sailor Assignment Problem", Proceedings of the 10th annual conference on Genetic and evolutionary computation, 2008.

[11] A. G. Carvalho and A. F. R. Araujo, "Improving NSGA-II with an Adaptive Mutation Operator", Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers, 2009.

[12] S. Tiwari, P. Koch, G. Fadel and K. Deb, "AMGA: An Archive-based Micro Genetic Algorithm for Multi-objective Optimization", Proceedings of the 10th annual conference on Genetic and evolutionary computation, 2008.

[13] S. S. Fan and J. Chang, "A Parallel Particle Swarm Optimization Algorithm for Multi-objective Optimization Problems", Engineering Optimization. Vol. 41, No. 7, July 2009, 673697.

[14] D. Dasgupta, G. Hernandez, A. Romero, D. Garrett and A. Kaushal, "On The Use of Informed Initialization and Extreme Solutions Sub-population in Multiobjective Evolutionary Algorithms", Proceedings of the IEEE Symposium on Computational Intelligence in Multi-Criteria Decision-Making, 2009.

[15] E. Byrne, "Optimising the flow of experiments to a robot scientist with multi-objective evolutionary algorithms", Proceedings of the 9th annual conference on Genetic and evolutionary computation, 2007.

[16] F. Cusmai, C. Aiello, M. Scannapieco and T. Catarci, "Record Linkage as a Multiobjective Optimization Problem solved by Evolutionary Computation", Proceedings of the 1st International Workshop on Emergent Semantics and cooperaTion in opEn systEMs, 2008.

[17] O. A. Hassan, L. Ramaswamy, J. Miller, K. Rasheed, and E. R. Canfield, "Replication in Overlay Networks: A Multi-objective Optimization Approach", Collaborative Computing: Networking, Applications and Worksharing, Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, 2009.

[18] J. Roca, E. Pugnaghi and G. Libert, "Solving an Extended Resource Leveling Problem with Multiobjective Evolutionary Algorithms", International Journal of Computational Intelligence. Vol 4, N4, Fall 2008.

[19] W. Zhang and M. Hansen, "An Evaluation of the NSGA-II and MOCell Genetic Algorithms for Self-management Planning in a Pervasive Service Middleware", Proceedings of the 14th IEEE International Conference on Engineering of Complex Computer Systems, 2009.

[20] C. L. B. Maia, R. A. Ferreira do Carmo, F. Gomes de Freitas, G. A. Lima de Campos and J. Teixeira de Souza, "A Multi-objective Approach for the Regression Test Case Selection Problem", Proceedings of the XLI Simpsio Brasileiro de Pesquisa Operacional, 2009.

[21] G. Danoy, B. Dorronsoro, P. Bouvry, B. Reljic and F. Zimmer, "Multi-objective Optimization for Information Sharing in Vehicular Ad Hoc Networks", Advances in Information Technology, Communications in Computer and Information Science Serei, 2009.

[22] K. Vergidis, "Business Process Optimisation using an Evolutionary Multi-objective Framework", Ph.D. Dissertation, Cranfield University, 2008.

[23] F. Luna, "Metaheursticas Avanzadas para Problemas Reales en Redes de Telecomunicaciones", Ph. D. Dissertation, Mlaga University, 2008.

[24] F. Kursawe, "A variant of evolution strategies for vector optimization", in: H. Schwefel, R. Männer (Eds.), Parallel Problem Solving for Nature, Springer-Verlag, Berlin, Germany, 1990, pp. 193–197.

[25] K. Deb, A. Pratap, S. Agarwal, T. Meyarivan, "A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II", IEEE Transactions on Evolutionary Computation 6 (2) (2002) 182–197.

[26] J. D. Knowles, D. W. Corne, "Approximating the Nondominated Front Using the Pareto Archived Evolution Strategy", Evolutionary Computation 8 (2) (2000) 149–172.

[27] E. Zitzler, M. Laumanns, L. Thiele, "SPEA2: Improving the Strength Pareto Evolutionary Algorithm", in: K. Giannakoglou, D. Tsahalis, J. Periaux, P. Papailou, T. Fogarty (Eds.), EUROGEN 2001. Evolutionary Methods for Design, Optimization and Control with Applications to Industrial Problems, Athens, Greece, 2002, pp. 95–100.

[28] M. Reyes-Sierra, C. A. Coello Coello, "Multi-Objective Particle Swarm Optimizers: A Survey of the State-of-the-Art", International Journal of Computational Intelligence Research 2 (3) (2006) 287–308.

[29] M. Reyes, C. Coello Coello, "Improving PSO-based Multi-objective Optimization Using Crowding, Mutation and ϵ-dominance", in: C. Coello, A. Hernández, E. Zitler (Eds.), Third International Conference on Evolutionary MultiCriterion Optimization, EMO 2005, Vol. 3410 of LNCS, Springer, 2005, pp. 509–519.

[30] A. Nebro, J. Durillo, F. Luna, B. Dorronsoro, E. Alba, "Design Issues in a Multiobjective Cellular Genetic Algorithm", in: S. Obayashi, K. Deb, C. Poloni, T. Hiroyasu, T. Murata (Eds.), Evolutionary Multi-Criterion Optimization. 4th International Conference, EMO 2007, Vol. 4403 of Lecture Notes in Computer Science, Springer, 2007, pp. 126–140.

[31] A. J. Nebro, F. Luna, E. Alba, B. Dorronsoro, J. J. Durillo, A. Beham, "AbYSS: Adapting Scatter Search to Multiobjective Optimization", IEEE Transactions on Evolutionary Computation 12 (4).

[32] H. Li, Q. Zhang, "Multiobjective Optimization Problems with Complicated Pareto Sets, MOEA/D and NSGA-II", IEEE Transactions on Evolutionary Computation 12 (2) (2009) 284–302.

[33] S. Kukkonen, J. Lampinen, "GDE3: The Third Evolution Step of Generalized Differential Evolution", in: IEEE Congress on Evolutionary Computation (CEC'2005), 2005, pp. 443 – 450.

[34] E. Zitzler, S. Künzli, "Indicator-based Selection in Multiobjective Search", in: X. Yao, et al. (Eds.), Parallel Problem Solving from Nature (PPSN VIII), Springer-Verlag, Berlin, Germany, 2004, pp. 832–842.

[35] A. Nebro, J. Durillo, J. García-Nieto, C. Coello Coello, F. Luna, E. Alba, "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, pp. 66–73.

[36] E. Zitzler, K. Deb, L. Thiele, "Comparison of Multiobjective Evolutionary Algorithms: Empirical Results", Evolutionary Computation 8 (2) (2000) 173–195.

[37] K. Deb, L. Thiele, M. Laumanns, E. Zitzler, "Scalable Test Problems for Evolutionary Multiobjective Optimization", in: A. Abraham, L. Jain, R. Goldberg (Eds.), Evolutionary Multiobjective Optimization. Theoretical Advances and Applications, Springer, USA, 2005, pp. 105–145.

[38] S. Huband, P. Hingston, L. Barone, L. While, "A Review of Multi-objective Test Problems and a Scalable Test Problem Toolkit", IEEE Transactions on Evolutionary Computation 10 (5) (2006) 477–506.

[39] D. A. Van Veldhuizen, G. B. Lamont, "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] E. Zitzler, L. Thiele, "Multiobjective Evolutionary Algorithms: a Comparative Case Study and the Strength Pareto Approach", IEEE Transactions on Evolutionary Computation 3 (4) (1999) 257–271.

[41] J. Knowles, L. Thiele, E. Zitzler, "A Tutorial on the Performance Assessment of Stochastic Multiobjective Optimizers", Tech. Rep. 214, Computer Engineering and Networks Laboratory (TIK), ETH Zurich (2006).

[42] A. J. Nebro, J. J. Durillo, C. Coello Coello, F. Luna, E. Alba, "A Study of Convergence Speed in Multi-objective Metaheuristics", in: G. Rudolph, T. Jensen, S. Lucas, C. Poloni, N. Beume (Eds.), Parallel Problem Solving from Nature - PPSN X, Vol. 5199 of Lecture Notes in Computer Science, Springer, 2008, pp. 763–772.

[43] J. J. Durillo, A. J. Nebro, C. A. Coello Coello, F. Luna, E. Alba, "A Comparative Study of the Effect of Parameter Scalability in Multi-objective Metaheuristics", in: CEC 2008, Hong Kong, 2008, pp. 255–266.

[44] J. Durillo, A. Nebro, C. C. Coello, J. García-Nieto, F. Luna, E. Alba, "A Study of Multi-Objective Metaheuristics when Solving Parameter Scalable Problems", Accepted to be published in IEEE Transactions on Evolutionary Computation.

[45] J. Hooker, "Testing Heuristics: We Have it All Wrong", Journal of Heuristics 1 (1995) 33–42.

[46] J. Demšar, "Statistical Comparisons of Classifiers over Multiple Data Sets", J. Mach. Learn. Res. 7 (2006) 1–30.