Automatic Configuration of NSGA-II with jMetal and irace

Antonio J. Nebro
University of Málaga
Málaga, Spain
antonio@lcc.uma.es

Manuel López-Ibáñez
University of Manchester
Manchester, United Kindom
manuel.lopez-ibanez@manchester.ac.uk

Cristóbal Barba-González
University of Málaga
Málaga, Spain
cbarba@lcc.uma.es

José García-Nieto
University of Málaga
Málaga, Spain
jnieto@lcc.uma.es

ABSTRACT

jMetal is a Java-based framework for multi-objective optimization with metaheuristics providing, among other features, a wide set of algorithms that are representative of the state-of-the-art. Although it has become a widely used tool in the area, it lacks support for automatic tuning of algorithm parameter settings, which can prevent obtaining accurate Pareto front approximations, especially for inexperienced users. In this paper, we present a first approach to combine jMetal and irace, a package for automatic algorithm configuration; the NSGA-II is chosen as the target algorithm to be tuned. The goal is to facilitate the combined use of both tools to jMetal users to avoid wasting time in adjusting manually the parameters of the algorithms. Our proposal involves the definition of a new algorithm template for evolutionary algorithms, which allows the flexible composition of multi-objective evolutionary algorithms from a set of configurable components, as well as the generation of configuration files for adjusting the algorithm parameters with irace. To validate our approach, NSGA-II is tuned with a benchmark problems and compared with the same algorithm using standard settings, resulting in a new variant that shows a competitive behavior.

KEYWORDS

Multi-objective Optimization, Metaheuristics, Software Tools, Automatic Algorithm Configuration

1 INTRODUCTION

The field of multi-objective optimization with evolutionary algorithms and other metaheuristic techniques is an active research area since the last twenty years. Around year 2000, the most widely used algorithm since then, NSGA-II [5], was proposed, and two seminal books about solving multi-objective problems with evolutionary algorithms were published [3, 4].

A factor that has promoted the development and application of multi-objective metaheuristics has been the availability of software tools. Since the emergence of PISA [2] in 2003, many frameworks have been proposed, being jMetal one of them. jMetal started in 2006 [9] as a research project to develop a Java-based software framework for multi-objective optimization with metaheuristic techniques. The source code is freely available since 2008, and it is hosted in GitHub under MIT license[1]. It has become a popular tool, and the papers describing it [7][9][8][13] sum up more than 1300 cites at the time of writing this paper according to Google Scholar.

jMetal was redesigned from scratch in 2015 [13] to be improved in a number of aspects (architecture, code quality, project organization, algorithm templates, parallelism support), but an aspect that is becoming a hot topic, the automatic configuration of metaheuristics, was not considered then. Most of metaheuristics depend on a number of parameters, and their performance is bound largely by finding proper values for them. This is usually a cumbersome task, particularly in the case of users who are expert in the problem domain but are unfamiliar with the algorithms.

In this paper, we present a proposal to extend jMetal with support for automatic parameter tuning. Our approach is based, on the one hand, on designing a very flexible template for evolutionary algorithms in such a way that a multi-objective evolutionary algorithm can be composed of a number of building blocks that can be configured as parameters which can take a range of values. On the other hand, these parameters are described in a way that they can be tuned by irace [11], an R-based package which has not only been used for algorithm configuration, but also for the automatic design of multi-objective evolutionary algorithms [1]. The goal is to facilitate the combined use of both tools to jMetal users interested in solving a given problem, but not in wasting time in adjusting manually algorithms' parameters.

[1]jMetal site in GitHub: https://github.com/jMetal/jMetal

As this is our first attempt to adapt jMetal for automatic parameter configuration, we have focused in multi-objective evolutionary algorithms and, in particular, in configuring the NSGA-II algorithm to solve continuous optimization problems. All the new developed code is included in an experimental package called *jmetal-auto*, as we wanted to avoid confusing jMetal users by modifying the existing code.

To validate our approach, we have carried out an experimental study where NSGA-II has been tuned with a family of benchmark problems and compared with the standard NSGA-II and also with SMPSO [12] as a representative algorithm of the state of the art.

The rest of the paper is organized as follows. Section 2 briefly describes jMetal and irace, the two tools we combine in this work. The proposed approach is detailed in Section 3 and the experimentation is included in Section 4. Finally, the conclusions and lines of future work are presented in Section 5.

2 SOFTWARE TOOLS

In this section, we briefly describe jMetal and irace, the two software tools we combine in this work.

2.1 The jMetal Framework

jMetal is tool for multi-objective optimization with metaheuristics based on an object-oriented architecture comprising four core entities: algorithms, problems, solutions (encodings or representations), and operators, as depicted in Figure 1. They are related among them following the idea that an algorithm solves a problem by using operators that manipulate solutions.

Algorithm 1 Pseudo-code of an evolutionary algorithm

1: $P(0) \leftarrow$ GenerateInitialSolutions()
2: $t \leftarrow 0$
3: Evaluate($P(0)$)
4: **while not** StoppingCriterion() **do**
5: $Q(t) \leftarrow$ Variation($P(t)$)
6: Evaluate($Q(t)$)
7: $P(t+1) \leftarrow$ Update($P(t)$, $Q(t)$)
8: $t \leftarrow t+1$
9: **end while**

A feature appeared in jMetal 5 [13] is the inclusion of algorithmic templates. If we focus on evolutionary algorithms, they can be described with the pseudo-code included in Algorithm 1. The corresponding *AbstractEvolutionaryAlgorithm* abstract class in jMetal is defined as follows (we have omitted some non relevant details):

```
1  public abstract class AbstractEvolutionaryAlgorithm<S,R>
       implements Algorithm<R> {
2    protected List<S> population;
3    ...
4    protected abstract void initProgress();
5    protected abstract void updateProgress();
6    protected abstract boolean isStoppingConditionReached();
7    protected abstract List<S> createInitialPopulation() ;
8    protected abstract List<S> evaluate(List<S> population);
9    protected abstract List<S> selection(List<S> population);
10   protected abstract List<S> reproduction(List<S> population);
11   protected abstract List<S> replacement(List<S> population,
       List<S> offspringPopulation);
12
13   @Override public abstract R getResult();
14
15   @Override public void run() {
16     List<S> offspringPopulation;
17     List<S> matingPopulation;
18
19     population = createInitialPopulation();
20     population = evaluatePopulation(population);
21     initProgress();
22     while (! isStoppingConditionReached()) {
23       matingPopulation = selection(population);
24       offspringPopulation = reproduction(matingPopulation);
25       offspringPopulation = evaluate(offspringPopulation);
26       population = replacement(population, offspringPopulation);
27       updateProgress();
28     }
29   }
```

We can observe that the template consists of a number of methods that are called inside the *run()* method, which defines the flow control of the algorithms. These methods can be defined in any subclass of the template, and this is the way how popular multi-objective metaheuristics including NSGA-II [5], SPEA2 [14], and many others are implemented.

The jMetal source code is sub-divided into four packages:

- *jmetal-core*: Classes of the core architecture plus some utilities, including quality indicators.
- *jmetal-algorithm*: Implementations of the metaheuristics included in the framework (more than 24 multi-objective algorithms, excluding variants).
- *jmetal-problem*: Implementations of problems, including 8 benchmark families, summing up more than 90 instances.
- *jmetal-exec*: Executable programs to configure and run the algorithms.

2.2 The irace Package

irace [11] is a software tool implemented in R aimed at the automatic configuration of optimization algorithms, i.e., to find accurate settings of a given algorithm for a given set of training instances of a problem. In this context, an algorithm configuration is a complete assignment of values to all required parameters of an algorithm.

In its latest versions (irace 2.0 and higher), irace implements an elitist iterated racing algorithm, where algorithm configurations are sampled from a sampling distribution, uniformly at random at the beginning, but biased towards the best configurations found in later iterations. At each iteration, the generated configurations and the "elite" ones from previous iterations are raced by evaluating them on training problem instances. A statistical test is used to decide which configurations should be eliminated from the race. When the race terminates, the surviving configurations become elites for the next iteration. A complete description of elitist iterated racing is provided in the original paper [11].

3 APPROACH FOR EXTENDING JMETAL TO WORK WITH IRACE: REDESIGNING NSGA-II

To configure an algorithm, irace requires a file containing the parameters to be tuned, including their type (categorical, ordinal, real, and integer) and the values they can take. Additionally, the algorithm must be prepared to be configured externally with any valid parameter combination.

As our plan is to use irace to configure NSGA-II, we must determine what kinds of parameters can be adjusted. We can distinguish the following ones:

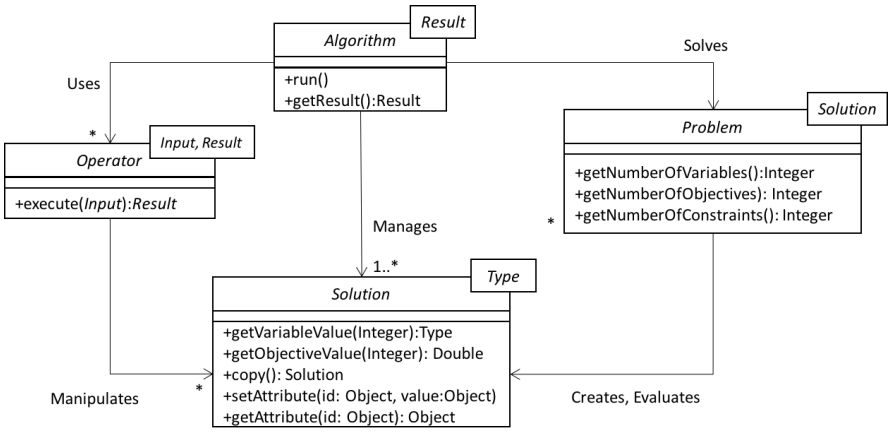

Figure 1: UML class diagram of jMetal 5.0 core classes.

- Selection, crossover, and mutation operators.
- Common operator parameters, such as mutation and crossover probabilities, which are applied to all of them.
- Specific operator parameters, such the distribution index of the simulated binary crossover.
- Algorithm parameters, such as the size of the offspring population size, or the size of the population if an external archive is used.
- Other parameters as, for example, different strategies for initializing the population.

We must note that the standard NSGA-II is a generational evolutionary algorithm, which is featured by using a ranking method based on Pareto ranking and the crowding distance density estimator, both in the selection and replacement steps of the algorithm [5]. When it is used to solve continuous problems, NSGA-II adopts the simulated binary crossover (SBX) and the polynomial mutation. No external archive is included in NSGA-II.

However, as we intend to configure NSGA-II in an automatic way, we need to relax the aforementioned features in order to have enough flexibility to modify the search capabilities of the algorithm. This way, we are going to consider that any multi-objective evolutionary algorithm with the typical parameters (selection, crossover, and mutation) and using ranking and crowding in the replacement step can be considered as a variant of NSGA-II.

Our solution is composed of three parts: the definition of a new template for evolutionary algorithms, the development of a program to configure NSGA-II from any parameter combination to work with irace, and the definition of utilities for creating file with the desired parameters to be configured. These components are located in a new and experimental package named *jmetal-auto*, which does not interfere with the other existing packages mentioned in Section 2.1.

We explore each of the new components next.

3.1 New Template for Evolutionary Algorithms

After analyzing the current implementation of NSGA-II in jMetal we concluded that it did not cope with the defined requirements in an easy way, because configuring NSGA-II and variants of it require to define new subclasses. The approach we adopt here is then based in defining a new template for evolutionary algorithms, which is shown in the following code snippet:

```
1  public class EvolutionaryAlgorithm<S extends Solution<?>>
2      implements Algorithm<List<S>>{
3    protected List<S> population;
4
5    private Evaluation<S> evaluation;
6    private InitialSolutionsCreation<S> createInitialPopulation;
7    private Termination termination;
8    private MatingPoolSelection<S> selection;
9    private Variation<S> variation;
10   private Replacement<S> replacement;
11
12   ...
13
14   public void run() {
15     population = createInitialPopulation.create();
16     population = evaluation.evaluate(population);
17     initProgress();
18     while (!termination.isMet(attributes)) {
19       List<S> matingPop ;
20       List<S> offspringPop ;
21       matingPop= selection.select(population);
22       offspringPop = variation.variate(population, matingPop);
23       offspringPop = evaluation.evaluate(offspringPop);
24
25       population =replacement.replace(population, offspringPop);
26       updateProgress();
27     }
28   }
29   ...
30 }
```

The basic difference with the former template is that we have changed the way it can be extended by replacing inheritance by delegation: instead of methods, the *EvolutionaryAlgorithm* class is composed by a number of objects or components. This way, an evolutionary algorithm can be defined by adding the proper components to the template. Consequently, we have defined a package including the components that can be used to configure a multi-objective evolutionary algorithm.

At the time of writing this paper, the available components are those included in Figure 2. We must note that the components extending *Evaluation* and *Termination* add features that do not take part in an automatic configuration process, as they do not affect the working of the algorithm.

Figure 2: Components included in jMetal to be combined in a multi-objective evolutionary algorithm.

Detailing how NSGA-II can be implemented using this new template and components is out the scope of this paper; notwithstanding, we comment next how the variation component can be configured with crossover and mutation operators. Let us take a look to this code snippet:

```
1  double crossoverProbability = 0.9;
2  double crossoverDistributionIndex = 20.0;
3  RepairDoubleSolution crossoverSolutionRepair = new
       RepairDoubleSolutionWithRandomValue();
4  CrossoverOperator<DoubleSolution> crossover = new SBXCrossover(
       crossoverProbability, crossoverDistributionIndex,
       crossoverSolutionRepair);
5
6  RepairDoubleSolution mutationSolutionRepair = new
       RepairDoubleSolutionWithRandomValue();
7  double mutationProbability = 1.0/problem.getNumberOfVariables();
8  double mutationDistributionIndex = 20.0;
9  MutationOperator<DoubleSolution> mutation =
10     new PolynomialMutation(
11     mutationProbability,
12     mutationDistributionIndex,
13     mutationSolutionRepair);
14
15 Variation<DoubleSolution> variation =
16     new CrossoverAndMutationVariation<>(
17     offspringPopulationSize, crossover, mutation);
```

The variation component we have to configure is an instance of the *CrossoverAndMutationVariation* class, which requires three parameters (lines 15-17): the size of the offspring population and the crossover and mutation operators. The first parameter indicates the number of solutions that the variation component has to produce.

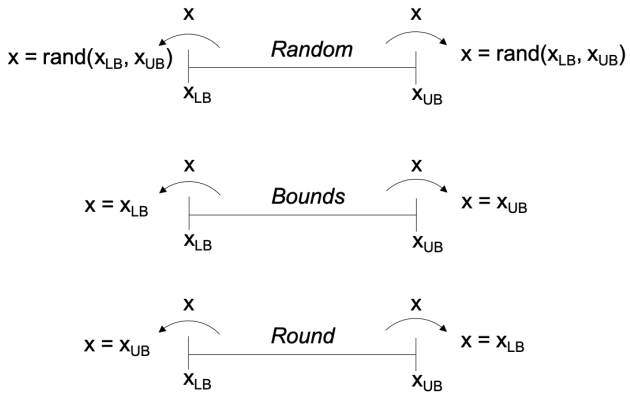

Figure 3: Repairing strategies when a variator operator produces a value out of the valid bounds.

The crossover operator is SBX, which requires values for the crossover probability and the distribution index (lines 1-4). We can see also that the operator has a parameter of class *RepairDoubleSolution* (line 3). This defines which strategy to follow when the operator produces a solution having some of its decision variables out of range (i.e., as we are dealing with continuous problems, the value of each decision variable v_i must be in a range $[lowerBound_i, upperBound_i]$). In jMetal we provide three repairing strategies (they are illustrated in Figure 3):

- Random: the variable takes a random value between the lower and upper bounds. This is the default strategy.
- Bounds: if the value is lower/higher than the lower/upper bound, the variable is assigned the lower/upper bound.
- Round: If the value is lower/higher than the lower/upper bound, the variable is assigned upper/lower bound.

The polynomial mutation operator is configured in a similar way (lines 7-13).

3.2 Autoconfiguring NSGA-II

The template for evolutionary algorithms presented in the last section can be used directly so, to instantiate a particular algorithm, it must be properly configured with any valid parameter configuration. In fact, the way of working of irace implies that it call a program with different parameter settings, run the resulting algorithm to solve a problem, and get as a result a value representing somehow the quality of the Pareto front approximation found. Consequently, with the idea of making easy to any user to auto configure NSGA-II, we have developed a program called *AutoNSGAIIConfigurator* for these purposes.

This program receives as a parameter a string containing a sequence of pairs < –parameter, value> that is read and analyzed to assign the proper parameter values. To process the string we have used Picocli[2], a tool for creating Java command line applications.

[2]Picocli: `https://picocli.info/`

To show an example of how it works, let us suppose that we want to configure the SBX crossover operator. The corresponding configuration substring would be the following:

```
"--crossover SBX
 --crossoverProbability 1.0
 --crossoverRepairStrategy bounds
 --sbxCrossoverDistributionIndex 30.0"
```

and the code to process in the *AutoNSGAIIConfigurator* program is included in this code snippet:

```
1     @Option(
2         names = {"--crossover"},
3         required = true,
4         description = "Crossover: ${COMPLETION-CANDIDATES}")
5     private CrossoverType crossoverType;
6
7     @Option(
8         names = {"--crossoverProbability"},
9         description = "Crossover probability (default: ${DEFAULT-
          VALUE})")
10    private double crossoverProbability = 0.9;
11
12    @Option(
13        names = {"--sbxCrossoverDistributionIndex"},
14        description =
15            "SBX crossover distribution index (default: ${DEFAULT-
              VALUE})")
16    private double sbxCrossoverDistributionIndex = 0.20;
17
18    @Option(
19        names = {"--crossoverRepairStrategy"},
20        description = "Crossover repair strategy (default: ${
          DEFAULT-VALUE})")
21    private RepairStrategyType crossoverRepairStrategy =
          RepairStrategyType.random;
```

We can observe that a crossover is always required, and its parameters have a default value in case they are not indicated.

Once all the parameters have been read, they are used to configure an instance of the *EvolutionaryAlgorithm* class, whose after calling its *run()* method, executes the algorithm and produces a front of solutions. As irace needs a value about the quality of these front, the relative hypervolume [15] is computed and returned. The relative hypervolume I_H^r of an approximation front P is defined as, given a reference Pareto front R, as $I_H^r = (I_H(R) - I_H(P))/I_H(R)$.

3.3 Creating the irace Parameter File

irace needs, as input, a parameter file containing all the parameters and the valid values they can take. This file is a text file that can be written with any text processing tool, but we have developed a package including an utility that allows to indicate all the desired parameters to be taken into account and to generate automatically the file.

By using this package, we have created the parameter file for the automatic configuration of NSGA-II included in Figure 4. As commented in a previous section, we feel free to configure the evolutionary algorithm template, but keeping a replacement strategy based on ranking and crowding distance in order to keep the most characteristic feature of NSGA-II. We assume that the resulting algorithm always has to return a population of 100 solutions.

We comment now the main parameters described in the file:

- *algorithmResult*: the algorithm can use the typical population of fixed size, but it can also use an external archive which is updated whenever a new solution is evaluated. This archive is bounded to 100 solutions, and the crowding distance is applied when it becomes full. When the archive is used, the population size can take a value between 10 and 400) and the result of the algorithm is the archive's content.
- *offspringPopulation*: this is the population containing the new solutions produced after the selection and variation steps, and its size can range between 1 and 400.
- *createInitialsolutions*: by default, the solutions of the initial population are randomly created, so we have add two additional procedures, one based on the scheme used in the scatter search algorithm and another one based on latin hypercube sampling.
- *crossover* and *mutation*: the available crossover operators are SBX and BLX-alpha, while the mutation operators can be polynomial or uniform.
- *selection*: the selection scheme can be random or n-ary tournament (with n ranging between 2 and 10).

4 EXPERIMENTATION

This section is devoted to validate our proposal of combining jMetal and irace by applying it to find a configuration of NSGA-II suitable for a set of benchmark problems. It is worth noting that in no way our goal here is to search for the best configuration of NSGA-II, but to carry out a proof of concept.

The experiment we have designed consists in using the WFG [10] problems for tuning NSGA-II and then use the resulting best configuration yielded by irace to test it against a default configuration of NSGA-II, when solving both the WFG and DTLZ problems [6] (the problems of both families have been configured with two objectives). We intend not only to determine the degree of improvement over the base NSGA-II that can be obtained, but also to know if the auto-NSGAII can compete with algorithms of the state-of-the-art. With that idea in mind, we have added SMPSO [12] to the comparative.

All the algorithms have been set to return 100 solutions after computing 25 000 function evaluations. No attempt has been done to adjust the parameters of SMPSO, with is configured with default settings.

The first step is to run irace with the parameter file described in Figure 4 by using the nine problems of the WFG benchmark. After a few hours of execution in a virtual Linux machine with 24 cores, the best configuration found by irace is the one included in Table 1 (right). We include in the left column of the same table the default settings of the NSGA-II algorithm in jMetal.

We can observe that there some noticeable differences. The auto tuned NSGA-II uses the external archive and the sizes of the population and offspring populations are 20 and 200, respectively; the crossover is BLX-alpha instead of SBX; the distribution index of the polynomial mutation is 158.05 instead of 20.0; and the size of the tournament is 9 instead of 2.

The second step is to compare the default and auto tuned NSGA-II algorithms along with SMPSO. For that purpose, we have made 25 independent runs of all of them to solve the WFG and DTLZ problems. After that, we have computed four quality indicators for measuring convergence (additive epsilon), diversity (spread), and both properties (hypervolume and IGD+). We report in the result tables the median and interquartile ranges, highlighting the best and second best indicator values in dark and light grey background, respectively.

```
algorithmResult                  "--algorithmResult "                  c   (externalArchive,population)
populationSize                   "--populationSize "                   c   (100)                          | algorithmResult %in% c("population")
populationSizeWithArchive        "--populationSizeWithArchive "        o   (10,20,50,100,200,400)         | algorithmResult %in% c("externalArchive")
#
offspringPopulationSize          "--offspringPopulationSize "          o   (1,5,10,20,50,100,200,400)
#
createInitialSolutions           "--createInitialSolutions "           c   (random,latinHypercubeSampling,scatterSearch)
#
variation                        "--variation "                        c   (crossoverAndMutationVariation)
#
crossover                        "--crossover "                        c   (SBX,BLX_ALPHA)
crossoverProbability             "--crossoverProbability "             r   (0.0, 1.0)                     | crossover %in% c("SBX","BLX_ALPHA")
crossoverRepairStrategy          "--crossoverRepairStrategy "          c   (random,round,bounds)          | crossover %in% c("SBX","BLX_ALPHA")
sbxCrossoverDistributionIndex    "--sbxCrossoverDistributionIndex "    r   (5.0, 400.0)                   | crossover %in% c("SBX")
blxAlphaCrossoverAlphaValue      "--blxAlphaCrossoverAlphaValue "      r   (0.0, 1.0)                     | crossover %in% c("BLX_ALPHA")
#
mutation                         "--mutation "                         c   (uniform,polynomial)
mutationProbability              "--mutationProbability "              r   (0.0, 1.0)                     | mutation %in% c("polynomial","uniform")
mutationRepairStrategy           "--mutationRepairStrategy "           c   (random,round,bounds)          | mutation %in% c("polynomial","uniform")
polynomialMutationDistributionIndex "--polynomialMutationDistributionIndex " r (5.0, 400.0)              | mutation %in% c("polynomial")
uniformMutationPerturbation      "--uniformMutationPerturbation "      r   (0.0, 1.0)                     | mutation %in% c("uniform")
#
selection                        "--selection "                        c   (random,tournament)
selectionTournamentSize          "--selectionTournamentSize "          i   (2, 10)                        | selection %in% c("tournament")
```

Figure 4: irace parameter file to auto configure NSGA-II

Default settings for NSGA-II	Settings of auto-NSGAII
–algorithmResult population	–algorithmResult externalArchive
–populationSize 100	–populationSizeWithArchive 20
–offspringPopulationSize 100	–offspringPopulationSize 200
–variation crossoverAndMutationVariation	–variation crossoverAndMutationVariation
–crossover SBX	–crossover BLX_ALPHA
–crossoverProbability 0.9	–crossoverProbability 0.9874
–crossoverRepairStrategy random	–crossoverRepairStrategy bounds
–sbxDistributionIndexValue 20.0	–blxAlphaCrossoverAlphaValue 0.5906
–mutation polynomial	–mutation polynomial
–mutationProbability $1/L$ (L: number of variables)	–mutationProbability 0.0015
–mutationRepairStrategy random	–mutationRepairStrategy random
–polynomialMutationDistributionIndex 20.0	–polynomialMutationDistributionIndex 158.05
–selection tournament	–selection tournament
–selectionTournamentSize 2	–selectionTournamentSize 9

Table 1: Settings for NSGA-II: default (left) and auto tuned (right).

Table 2: Additive epsilon indicator. Median and Interquartile Range

	NSGAII	SMPSO	AutoNSGAII
WFG1	$4.52e-01_{2.4e-01}$	$4.55e-01_{9.8e-03}$	$5.99e-03_{7.3e-04}$
WFG2	$5.41e-03_{2.4e-03}$	$6.04e-03_{1.1e-03}$	$3.88e-03_{5.5e-04}$
WFG3	$3.34e-01_{5.3e-04}$	$3.34e-01_{1.9e-04}$	$3.33e-01_{2.7e-07}$
WFG4	$1.29e-02_{2.2e-03}$	$2.16e-02_{3.4e-03}$	$6.72e-03_{1.2e-03}$
WFG5	$3.31e-02_{3.7e-03}$	$2.77e-02_{1.9e-04}$	$2.76e-02_{1.6e-04}$
WFG6	$1.50e-02_{5.9e-03}$	$6.37e-03_{4.9e-04}$	$6.13e-03_{7.2e-03}$
WFG7	$1.28e-02_{4.0e-03}$	$6.52e-03_{4.9e-04}$	$5.20e-03_{1.9e-04}$
WFG8	$1.68e-01_{1.0e-01}$	$1.69e-01_{1.7e-02}$	$2.45e-01_{1.3e-03}$
WFG9	$1.42e-02_{2.0e-03}$	$1.10e-02_{1.9e-03}$	$7.39e-03_{1.2e-03}$
DTLZ1	$3.53e-02_{1.5e-01}$	$6.30e-03_{5.5e-04}$	$3.08e+01_{1.7e+01}$
DTLZ2	$1.12e-02_{4.2e-03}$	$5.53e-03_{3.2e-04}$	$5.28e-03_{2.5e-04}$
DTLZ3	$1.00e+01_{6.4e+00}$	$5.97e-03_{3.5e-01}$	$1.05e+02_{3.6e+01}$
DTLZ4	$1.18e-02_{5.4e-03}$	$5.61e-03_{3.1e-04}$	$5.32e-03_{1.9e-04}$
DTLZ5	$1.01e-02_{2.2e-03}$	$5.12e-03_{3.5e-04}$	$5.03e-03_{2.7e-04}$
DTLZ6	$3.72e-01_{5.3e-02}$	$5.15e-03_{4.5e-04}$	$5.06e-03_{2.9e-04}$
DTLZ7	$7.78e-03_{2.6e-03}$	$4.30e-03_{2.3e-04}$	$4.06e-03_{2.8e-04}$

Table 3: Spread quality indicator. Median and Interquartile Range

	NSGAII	SMPSO	AutoNSGAII
WFG1	$7.54e-01_{5.6e-02}$	$1.02e+00_{3.6e-02}$	$1.23e-01_{1.1e-02}$
WFG2	$7.84e-01_{1.5e-02}$	$8.04e-01_{2.9e-02}$	$7.58e-01_{1.5e-03}$
WFG3	$5.67e-01_{2.2e-02}$	$3.79e-01_{7.9e-03}$	$3.62e-01_{9.7e-03}$
WFG4	$3.64e-01_{5.0e-02}$	$4.55e-01_{5.1e-02}$	$1.30e-01_{1.6e-02}$
WFG5	$3.91e-01_{5.3e-02}$	$1.35e-01_{1.4e-02}$	$1.33e-01_{2.8e-02}$
WFG6	$3.55e-01_{4.8e-02}$	$1.49e-01_{2.5e-02}$	$1.05e-01_{2.8e-02}$
WFG7	$3.56e-01_{4.1e-02}$	$1.51e-01_{2.5e-02}$	$1.10e-01_{2.5e-02}$
WFG8	$6.17e-01_{6.5e-02}$	$7.43e-01_{3.5e-02}$	$5.46e-01_{3.2e-02}$
WFG9	$3.75e-01_{3.4e-02}$	$2.14e-01_{3.1e-02}$	$1.29e-01_{1.6e-02}$
DTLZ1	$1.04e+00_{2.6e-01}$	$6.83e-02_{1.4e-02}$	$6.00e-01_{6.1e-03}$
DTLZ2	$3.38e-01_{4.6e-02}$	$1.23e-01_{2.2e-02}$	$9.89e-02_{3.1e-02}$
DTLZ3	$9.25e-01_{1.3e-01}$	$1.26e-01_{4.8e-01}$	$6.04e-01_{1.0e-02}$
DTLZ4	$3.43e-01_{5.8e-02}$	$1.19e-01_{2.4e-02}$	$9.41e-02_{2.2e-02}$
DTLZ5	$3.26e-01_{3.0e-02}$	$1.32e-01_{2.1e-02}$	$1.01e-01_{2.5e-02}$
DTLZ6	$7.66e-01_{9.2e-02}$	$1.11e-01_{2.6e-02}$	$9.61e-02_{2.7e-02}$
DTLZ7	$6.20e-01_{2.4e-02}$	$5.25e-01_{1.9e-03}$	$5.25e-01_{1.8e-03}$

Tables 2,3,4, and 5 contain the obtained values of the four quality indicators. Some conclusions can be drawn from a first examination of these results. The auto tuned NSGA-II (called *AutoNSGAII* in the tables) does not only outperform globally NSGA-II, but also SMPSO. Although the hypervolume has been used as quality measure in the tuning step, AutoNSGAII is the best performing algorithm when considering the other three indicators, what suggests that the found configuration is robust; only in three problem instances (WFG8, DTLZ1, and DTLZ3) out of the sixteen problems AutoNSGAII have performed poorly.

This analysis is supported by the computing of the statistical Wilcoxon rank sum test (at a 5% level of significance), whose results for all the quality indicators are included in Table 6. Each cell contains a symbol representing each of the 17 problems (WFG1-9 and DTLZ1-7). There are three different symbols in this table: "–" indicates that there not statistical significance between the algorithms in the row and in the column, "▲" means that the algorithm

Table 4: Hypervolume quality indicator. Median and Interquartile Range

	NSGAII	SMPSO	AutoNSGAII
WFG1	$4.49e-01_{7.6e-02}$	$1.16e-01_{7.7e-03}$	$6.34e-01_{2.6e-05}$
WFG2	$5.64e-01_{9.5e-04}$	$5.62e-01_{1.2e-03}$	$5.65e-01_{5.1e-05}$
WFG3	$4.41e-01_{3.8e-04}$	$4.41e-01_{2.2e-04}$	$4.42e-01_{1.1e-05}$
WFG4	$2.17e-01_{7.6e-04}$	$2.03e-01_{2.4e-03}$	$2.17e-01_{3.0e-03}$
WFG5	$1.95e-01_{2.9e-04}$	$1.96e-01_{7.5e-05}$	$1.96e-01_{1.0e-04}$
WFG6	$2.03e-01_{8.9e-03}$	$2.09e-01_{4.3e-04}$	$2.08e-01_{1.3e-02}$
WFG7	$2.09e-01_{3.5e-04}$	$2.09e-01_{3.2e-04}$	$2.11e-01_{3.1e-05}$
WFG8	$1.48e-01_{2.3e-02}$	$1.48e-01_{1.0e-03}$	$1.39e-01_{2.3e-03}$
WFG9	$2.37e-01_{2.9e-03}$	$2.35e-01_{8.2e-04}$	$2.39e-01_{1.9e-03}$
DTLZ1	$4.66e-01_{1.6e-01}$	$4.94e-01_{1.9e-04}$	$0.00e+00_{0.0e+00}$
DTLZ2	$2.09e-01_{2.7e-04}$	$2.10e-01_{1.5e-04}$	$2.11e-01_{4.1e-05}$
DTLZ3	$0.00e+00_{0.0e+00}$	$2.10e-01_{6.3e-02}$	$0.00e+00_{0.0e+00}$
DTLZ4	$2.10e-01_{7.1e-04}$	$2.10e-01_{1.5e-04}$	$2.11e-01_{4.2e-05}$
DTLZ5	$2.11e-01_{3.5e-04}$	$2.12e-01_{1.3e-04}$	$2.12e-01_{4.1e-05}$
DTLZ6	$1.89e-05_{1.4e-03}$	$2.12e-01_{6.9e-05}$	$2.12e-01_{5.6e-05}$
DTLZ7	$3.29e-01_{2.8e-04}$	$3.30e-01_{9.8e-05}$	$3.30e-01_{7.3e-05}$

Table 5: IGD+ quality indicator. Median and Interquartile Range

	NSGAII	SMPSO	AutoNSGAII
WFG1	$2.39e-01_{1.4e-01}$	$4.76e-01_{1.0e-02}$	$1.94e-03_{9.1e-05}$
WFG2	$1.61e-03_{6.1e-04}$	$2.68e-03_{6.7e-04}$	$9.95e-04_{9.9e-05}$
WFG3	$2.15e-02_{2.3e-04}$	$2.13e-02_{1.4e-04}$	$2.06e-02_{1.7e-05}$
WFG4	$3.34e-03_{4.9e-04}$	$1.14e-02_{1.4e-03}$	$3.42e-03_{1.4e-03}$
WFG5	$2.75e-02_{9.8e-05}$	$2.71e-02_{5.7e-05}$	$2.70e-02_{3.9e-05}$
WFG6	$6.84e-03_{5.9e-03}$	$3.05e-03_{3.2e-04}$	$3.54e-03_{8.5e-03}$
WFG7	$3.47e-03_{4.5e-04}$	$3.15e-03_{1.8e-04}$	$2.30e-03_{6.1e-05}$
WFG8	$4.24e-02_{2.3e-02}$	$4.26e-02_{2.3e-03}$	$5.80e-02_{3.0e-03}$
WFG9	$4.97e-03_{1.6e-03}$	$6.05e-03_{4.1e-04}$	$4.07e-03_{1.1e-03}$
DTLZ1	$1.93e-02_{9.0e-02}$	$2.81e-03_{1.0e-04}$	$4.31e+01_{2.1e+01}$
DTLZ2	$3.15e-03_{2.7e-04}$	$2.48e-03_{8.0e-05}$	$2.30e-03_{5.2e-05}$
DTLZ3	$1.01e+01_{8.3e+00}$	$2.09e-03_{1.6e-01}$	$1.35e+02_{4.7e+01}$
DTLZ4	$2.59e-03_{4.1e-04}$	$2.13e-03_{6.9e-05}$	$1.95e-03_{3.0e-05}$
DTLZ5	$2.50e-03_{1.7e-04}$	$2.05e-03_{9.0e-05}$	$1.81e-03_{5.8e-05}$
DTLZ6	$3.88e-01_{5.4e-02}$	$1.91e-03_{9.1e-05}$	$1.83e-03_{4.3e-05}$
DTLZ7	$2.42e-03_{2.2e-04}$	$1.88e-03_{1.2e-04}$	$1.82e-03_{9.3e-05}$

Table 6: Wilcoxon rank sum test results. The symbols in each cell correspond to problems WFG1-9 and DTLZ1-7.

Additive epsilon

	SMPSO	AutoNSGAII
NSGAII	– – ▽ ▲ ▽ ▽ ▽ – ▽ ▽ ▽ ▽ ▽ ▽ ▽ ▽	▽ ▽ ▽ ▽ ▽ ▽ ▽ ▲ ▽ ▲ ▽ ▲ ▽ ▽ ▽ ▽
SMPSO		▽ ▽ ▽ ▽ ▽ – ▽ ▲ ▽ ▲ ▽ ▲ ▽ ▽ – ▽

Spread

	SMPSO	AutoNSGAII
NSGAII	▲ ▲ ▽ ▲ ▽ ▽ ▽ ▲ ▽ ▽ ▽ ▽ ▽ ▽ ▽ ▽	▽ ▽ ▽ ▽ ▽ ▽ ▽ ▽ ▽ ▽ ▽ ▽ ▽ ▽ ▽ ▽
SMPSO		▽ ▽ ▽ ▽ – ▽ ▽ ▽ ▽ ▲ ▽ ▲ ▽ ▽ ▽ –

Hypervolume

	SMPSO	AutoNSGAII
NSGAII	▲ ▲ ▽ ▲ ▽ ▽ – – ▲ ▽ ▽ ▽ ▽ ▽ ▽ ▽	▽ ▽ ▽ – ▽ – ▽ ▲ ▽ ▲ ▽ – ▽ ▽ ▽ ▽
SMPSO		▽ ▽ ▽ ▽ – – ▽ ▲ ▽ ▲ ▽ ▲ ▽ ▽ ▽ ▽

IGD+

	SMPSO	AutoNSGAII
NSGAII	▲ ▲ ▽ ▲ ▽ ▽ ▽ – ▲ ▽ ▽ ▽ ▽ ▽ ▽ ▽	▽ ▽ ▽ – ▽ – ▽ ▲ ▽ ▲ ▽ ▲ ▽ ▽ ▽ ▽
SMPSO		▽ ▽ ▽ ▽ – – ▽ ▲ ▽ ▲ ▽ ▲ ▽ ▽ ▽ –

in the row has produced a best indicator value 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. We can observe that statistical confidence has been found in most of the results.

An issue that arises after the conducted experiment is, given that the auto configured NSGA-II has yielded unsatisfactory indicator values on three problems, to use irace to find a particular configuration for only those three instances and to see how it differs from the previously found configuration. We have proceeded then to repeat the experiment using only the DTLZ1, DTLZ3, and WFG8 instances.

The resulting settings found by irace are reported in Table 7 (right). When comparing the obtained configuration with the former one, included in the left column of the table, we can observe a number of differences: the offspring population size is 5 instead of 100, the crossover operator is SBX instead of BLX-Alpha, the mutation operator is uniform instead of polynomial, and the size of the tournament is 4 instead of 9. The similarities of both configurations are the use of an external archive, a population size of 20 individuals, and a crossover probability around 0.98.

The indicator values obtained for this second configuration are included in the last column of Table 8 (we have named *AutoNSGAIIb* the new version of autoNSGAII in the table), which contains also the values of the former experiment for the three considered problems. We can observe that all the results have been improved, particularly in the case of the DTLZ problems, where the second auto configured NSGA-II outperforms NSGA-II and the first AutoNSGAII, although it cannot beat the values obtained by SMPSO.

5 CONCLUSIONS AND FUTURE WORK

We have presented a first approximation to adapt the jMetal framework to work jointly with the irace package to allow the automatic parameter configuration of multi-objective evolutionary algorithms. The proposal involves the creation of a flexible and configurable template for this family of techniques, and NSGA-II has been used as target algorithm. We have developed also a program able of interact with irace in such a way that it can receive any configuration from irace, set the corresponding NSGA-II variant, run the resulting algorithm, and return as a result a quality measure of the produced front. All this new functionality has been included in a new package called *jmetal-auto*.

We have validated our proposal in two steps. First, we have auto configured NSGA-II from a wide set of parameters that can be incorporated into the algorithm; the nine benchmark problems of the WFG family have been used for this auto-tuning. Second, the resulting NSGA-II with the found configuration has been compared against the same algorithm with default settings and with SMPSO on a benchmark composed of 16 problems. The reported results after applying four quality indicators show that the auto configured NSGA-II clearly outperforms the other two algorithms in all the indicators.

There are several lines for further research. We have conducted a case study, but others can be developed by considering different scenarios involving the automatic configuration of NSGA-II, such as fixing the stopping condition in a low number of evaluations or using problems with more than two objectives. We plan also to extend the features of the *jmetal-auto* package to incorporate more algorithmic components, in particular from indicator-based and decomposition-based multi-objective algorithms.

6 ACKNOWLEDGEMENTS

This work has been partially funded by Spanish Grants TIN2017-86049-R (Spanish Ministry of Education and Science) and P12-TIC-1519 (Plan Andaluz de Investigación, Desarrollo e Innovación). Cristóbal Barba-González is supported by Grant BES-2015-072209

Settings for auto-NSGA-II	Settings of auto-NSGAII (DTLZ1, DTLZ3, WFG8)
–algorithmResult externalArchive	–algorithmResult externalArchive
–populationSizeWithArchive 20	–populationSizeWithArchive 20
–offspringPopulationSize 100	–offspringPopulationSize 5
–variation crossoverAndMutationVariation	–variation crossoverAndMutationVariation
–crossover BLX_ALPHA	–crossover SBX
–crossoverProbability 0.9874	–crossoverProbability 0.9791
–crossoverRepairStrategy bounds	–crossoverRepairStrategy random
–blxAlphaCrossoverAlphaValue 0.59.6	–sbxDistributionIndexValue 5.0587
–mutation polynomial	–mutation uniform
–mutationProbability 0.0015	–mutationProbability 0.0463
–mutationRepairStrategy random	–mutationRepairStrategy random
–polynomialMutationDistributionIndex 158.05	–uniformMutationPerturbation 0.2307
–selection tournament	–selection tournament
–selectionTournamentSize 9	–selectionTournamentSize 4

Table 7: Settings for auto-NSGAII: first experiment (left) and second experiment (right).

Additive epsilon	NSGAII	SMPSO	AutoNSGAII	AutoNSGAIIb
DTLZ1	$3.53e-02_{1.5e-01}$	$6.30e-03_{5.5e-04}$	$3.08e+01_{1.7e+01}$	$7.06e-03_{2.4e-03}$
DTLZ3	$1.00e+01_{6.4e+00}$	$5.97e-03_{3.5e-01}$	$1.05e+02_{3.6e+01}$	$2.87e-02_{2.6e-02}$
WFG8	$1.68e-01_{1.0e-01}$	$1.69e-01_{1.7e-02}$	$2.45e-01_{1.3e-03}$	$2.44e-01_{1.0e-01}$
Spread	NSGAII	SMPSO	AutoNSGAII	AutoNSGAIIb
DTLZ1	$1.04e+00_{2.6e-01}$	$6.83e-02_{1.4e-02}$	$6.00e-01_{6.1e-03}$	$6.28e-02_{1.8e-02}$
DTLZ3	$9.25e-01_{1.3e-01}$	$1.26e-01_{4.8e-01}$	$6.04e-01_{1.0e-02}$	$2.14e-01_{2.1e-01}$
WFG8	$6.17e-01_{6.5e-02}$	$7.43e-01_{3.5e-02}$	$5.46e-01_{3.2e-02}$	$5.33e-01_{5.5e-02}$
Hypervolume	NSGAII	SMPSO	AutoNSGAII	AutoNSGAIIb
DTLZ1	$4.66e-01_{1.6e-01}$	$4.94e-01_{1.9e-04}$	$0.00e+00_{0.0e+00}$	$4.93e-01_{3.9e-03}$
DTLZ3	$0.00e+00_{0.0e+00}$	$2.10e-01_{6.3e-02}$	$0.00e+00_{0.0e+00}$	$1.76e-01_{3.1e-02}$
WFG8	$1.48e-01_{2.3e-02}$	$1.48e-01_{1.0e-03}$	$1.39e-01_{2.3e-03}$	$1.46e-01_{3.4e-03}$
IGD+	NSGAII	SMPSO	AutoNSGAII	AutoNSGAIIb
DTLZ1	$1.93e-02_{9.0e-02}$	$2.81e-03_{1.0e-04}$	$4.31e+01_{2.1e+01}$	$3.61e-03_{2.1e-03}$
DTLZ3	$1.01e+01_{8.3e+00}$	$2.09e-03_{1.6e-01}$	$1.35e+02_{4.7e+01}$	$2.78e-02_{2.5e-02}$
WFG8	$4.24e-02_{2.3e-02}$	$4.26e-02_{2.3e-03}$	$5.80e-02_{3.0e-03}$	$5.06e-02_{9.1e-03}$

Table 8: Median and Interquartile Range of the indicator values for the DTLZ1, DTL3, and WFG8 instances experiment (AutoNSGAIIb is NSGA-II auto tuned with the second configuration).

(Spanish Ministry of Economy and Competitiveness). José García-Nieto is the recipient of a Post-Doctoral fellowship of "Captación de Talento para la Investigación" Plan Propio at Universidad de Málaga.

REFERENCES

[1] L. C. T. Bezerra, M. López-Ibáñez, and T. Stützle. 2016. Automatic Component-Wise Design of Multiobjective Evolutionary Algorithms. *IEEE Transactions on Evolutionary Computation* 20, 3 (June 2016), 403–417.

[2] Stefan Bleuler, Marco Laumanns, Lothar Thiele, and Eckart Zitzler. 2003. PISA — A Platform and Programming Language Independent Interface for Search Algorithms. In *Evolutionary Multi-Criterion Optimization (EMO 2003) (Lecture Notes in Computer Science)*, Carlos M. Fonseca, Peter J. Fleming, Eckart Zitzler, Kalyanmoy Deb, and Lothar Thiele (Eds.). Springer, Berlin, 494 – 508.

[3] C.A. Coello Coello, G.B. Lamont, and D.A. van Veldhuizen. 2007. *Multi-Objective Optimization Using Evolutionary Algorithms*. John Wiley & Sons, Inc. 2nd Ed., NY, USA.

[4] K. Deb. 2001. *Multi-Objective Optimization Using Evolutionary Algorithms*. John Wiley & Sons, New York, NY, USA.

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

[6] K. Deb, L. Thiele, M. Laumanns, and E. Zitzler. 2005. Scalable test problems for evolutionary multiobjective optimization. In *Evolutionary Multiobjective Optimization. Theoretical Advances and Applications*, Ajith Abraham, Lakhmi Jain, and Robert Goldberg (Eds.). Springer, USA, 105–145.

[7] J.J. Durillo and A.J. Nebro. 2011. jMetal: A Java framework for multi-objective optimization. *Advances in Engineering Software* 42, 10 (2011), 760 – 771. https://doi.org/10.1016/j.advengsoft.2011.05.014

[8] J.J. Durillo, A.J. Nebro, and E. Alba. 2010. The jMetal Framework for Multi-Objective Optimization: Design and Architecture. In *CEC 2010*. IEEE, Barcelona, Spain, 4138–4325.

[9] J.J. Durillo, A.J. Nebro, F. Luna, B. Dorronsoro, and E. Alba. 2006. *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.

[10] S. Huband, L. Barone, R.L. While, and P. Hingston. 2005. A Scalable Multi-objective Test Problem Toolkit. In *Third International Conference on Evolutionary MultiCriterion Optimization, EMO 2005 (Lecture Notes in Computer Science)*, C.A. Coello, A. Hernández, and E. Zitler (Eds.), Vol. 3410. Springer, Berlin, Germany, 280–295.

[11] Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Thomas Stützle, and Mauro Birattari. 2016. The irace package: Iterated Racing for Automatic Algorithm Configuration. *Operations Research Perspectives* 3 (2016), 43–58. https://doi.org/10.1016/j.orp.2016.09.002

[12] A.J. Nebro, J.J. Durillo, J. García-Nieto, C.A. Coello Coello, F. Luna, and E. Alba. 2009. 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, 66–73.

[13] A.J. Nebro, Juan J. Durillo, and M. Vergne. 2015. Redesigning the jMetal Multi-Objective Optimization Framework. In *Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation (GECCO Companion '15)*. ACM, New York, NY, USA, 1093–1100. https://doi.org/10.1145/2739482.2768462

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

[15] E. Zitzler and L. Thiele. 1999. Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach. *IEEE Transactions on Evolutionary Computation* 3, 4 (Nov 1999), 257–271. https://doi.org/10.1109/4235.797969