An experimental comparison of metaheuristic frameworks for multi-objective optimization

— *Extended study* —

Technical Report

Aurora Ramírez, Rafael Barbudo and José Raúl Romero[1]

Knowledge Discovery and Intelligent Systems Research Group

Department of Computer Science and Numerical Analysis

University of Córdoba

September 27, 2020

[1]Corresponding author: jrromero@uco.es

1 Introduction

Metaheuristic optimization frameworks (MOFs) are software tools or class libraries that provide a collection of extensible building blocks to create algorithms and experiments aimed at solving optimization problems. Several alternatives are currently available to deal with multi-objective optimization (MOO) problems, which are characterized by the definition of two or more objectives.

This technical report is provided as additional material to the research paper of the same title. Therefore, the information presented in this technical report is intended to complement those aspects that have been summarized or omitted due to space limitations. Essentially, it provides further details about the evaluation of ten MOFs in terms of characteristics like the set of algorithms and operators each one offers or the availability of general utilities, among others. This document presents the information extracted from the available documentation and the source code, serving as a reference guide to domain experts and researchers interested in the application of metaheuristic algorithms to solve multi-objective problems. More specifically, the content of this technical report can be summarized as follows:

- The detailed methodology followed to evaluate the frameworks, including technical details of external tools and the experimental environment.
- The exhaustive evaluation of each characteristic is presented in tables.
- The extensive experimental results, which are used to assess execution time and memory at runtime.
- Decisions explaining partial fulfillment of some features.
- The list of original references to the algorithms and operators currently available in the frameworks.

The rest of the document is organized as follows. Section 2 presents the essential concepts related to multi-objective optimization. MOFs and their characteristics are introduced later. Section 3 describes the comparison methodology and the frameworks under evaluation. It also presents an overview of the features that will be evaluated, organized into seven characteristics (C1-C7). Sections 4 to 10 contain the outcomes for each of the seven characteristics. In each section, the characteristic is defined and refined into several features. Then, the results of their evaluation are detailed.

2 Background

2.1 Multi-objective optimization

Multi-objective optimization problems are defined in terms of a set of decision variables, for which at least two objective functions have to be simultaneously optimized [1]. Given that these objectives are often in conflict, a unique optimal solution does not exist. Therefore, the optimization process seeks for candidate solutions representing different trade-offs among the objectives. These solutions are known as non-dominated or Pareto optimal solutions, meaning that the improvement in one objective value implies the detriment of another. The Pareto front is comprised of the projections of non-dominated solutions in the objective space.

Metaheuristics are frequently applied to solve MOO problems, probably because of their efficiency and the possibility of managing a set of solutions in a single run [1]. Evolutionary algorithms (EAs) emerged as the first, and probably most used, bio-inspired method to deal with multi-objective problems [2], though other metaheuristic paradigms like local search (LS) and swarm intelligence (SI) can be found too [3, 4]. Regardless of the metaheuristic model, the goals of any multi-objective algorithm are both promoting the convergence to the Pareto front and preserving diversity by means of specific selection and replacement procedures.

Since the publication of the first multi-objective evolutionary algorithm, several *generations* or *families* of algorithms have appeared. Founded on the Pareto optimality, the *first generation* of MOEAs includes some niching or fitness sharing techniques [5]. The presence of an elitism mechanism, either by means of an external population or specific selection procedures, is the distinctive characteristic of the so-called *second generation* [5].

More recently, researchers have adopted novel techniques aimed at improving the effectiveness of MOEAs as multi-objective problems turn into many-objective ones [6, 2]. Decomposition approaches [2, 6] define multiple weight vectors that represent possible search directions to be explored at the same time, while indicator-based techniques use a performance metric to guide the search towards the front. Relaxing the dominance principle or the use of reference points are other existing approaches. Finally, preference-based methods incorporate information given by decision makers. Taking inspiration from MOEAs, other multi-objective metaheuristics have adopted similar ideas. For instance, external archives have been incorporated to multi-objective particle swarm optimization (MOPSO) [7] and indicator-based local search methods have been proposed too [8].

Multi-objective algorithms are frequently compared taking test suites for both combinatorial and continuous optimization as reference [1]. Most of the combinatorial multi-objective benchmarks, such as the knapsack and the traveling salesman problems, have been adapted from their original single-objective formulations. However, the design of new continuous functions has been extensively explored in order to obtain hard optimization problems with different properties regarding the shape of the front or the number of objectives [9]. Some examples are the DTLZ, WFG and ZDT test functions. The performance of multi-objective algorithms over these problems is often assessed in terms of quality indicators. They quantify diverse properties of the returned front [1] like the hyperarea covered, the distribution of the solutions or the distance to the true Pareto front.

2.2 Metaheuristic optimization frameworks

Metaheuristic optimization frameworks are software tools or libraries that provide a collection of extensible building blocks and generic search models [10]. In addition, a MOF establishes a reference structure to programmers, who can benefit from the existence of reusable and customizable elements, thus reducing development and testing efforts [11]. Ease of use, flexibility and extensibility are the key characteristics underlying the design of a MOF.

The need for generic components in the context of software tools for evolutionary computation was discussed by Gagné and Parizeau [12]. Regarding the solution representation, MOFs should allow both the use of an existing encoding and the definition of new formulations. Similarly, fitness evaluation should support both minimization and maximization problems, as well as the definition of multiple objectives. Focusing on the search process, users should be allowed to choose among a set of available genetic operators, configure their parameters, and apply different metaheuristic models. Regarding algorithm configuration, MOFs usually provide support to load configuration files or include a GUI. Having a configurable output to show results and report statistics is also recommended.

Parejo et al. [13] presented a comprehensive analysis considering some of these characteristics for ten general-purpose MOFs. This study included design issues, available documentation and advanced computing capabilities, thus providing a complete overview of existing alternatives. Concerning MOO, they just covered the list of algorithms available in each framework. Their study showed that not all of them supported multi-objective optimization, while those MOFs with MOO-specific features included only a few MOEAs proposed in the early 2000s. In their review, the authors analyzed the frameworks based on lists of features, but software evaluation in terms of quality metrics or performance behavior was not considered at that time.

Table 1: The set of characteristics and their features.

Characteristic	**Feature**	**Outcomes**
C1: search components and techniques	C1F1: types of metaheuristics	List of available search techniques
	C1F2: families of algorithms	List of algorithms per family
	C1F3: encodings and operators	List of operators per encoding
C2: configuration	C2F1: inputs	List of input types and data formats
	C2F2: batch processing	List of possible ways to run experiments
	C2F3: outputs	List of output types and data formats
C3: execution	C3F1: multi-thread execution	List of possible ways to apply parallelism
	C3F2: distributed execution	List of distributed computing models
	C3F3: stop and restart mode	Support to serialization and checkpointing
	C3F4: fault recovery	Support to parameter tuning and exception handling
	C3F5: execution and control logs	Support to show intermediate results and logs
C4: utilities	C4F1: graphical user interface	List of functionalities associated to the GUI
	C4F2: benchmarks	List of available test problems
	C4F3: quality indicators	List of available quality indicators
C5: documentation and community support	C5F1: software license	Type of license
	C5F2: available documentation	Types of external documentation
	C5F3: software update	Number of releases since January 2015
	C5F4: development facilities	List of auxiliary tools
	C5F5: community	List of communication channels
C6: software implementation	C6F1: implementation and execution	Programming language and execution platform
	C6F2: external libraries	Types of third-party libraries used
	C6F3: software metrics	List of metrics associated to code quality
C7: performance at runtime	C7F1: execution time	Measurement of execution time
	C7F2: memory consumption	Measurement of memory usage

3 Comparison methodology

This section presents a brief introduction to the characteristics under analysis, and explains the methodology followed to perform the analysis and evaluate the selected frameworks.

3.1 Overview of characteristics

The evaluation model consists of a hierarchical categorization of characteristics and features. This kind of model is a common practice when evaluating software tools [13]. The characteristics here defined capture the evaluation goals from different complementary views. They cover not only static properties, but also dynamic properties, which are essential to assess the performance in real contexts. Furthermore, the scope of these characteristics varies from general requirements, such as configuration and execution capabilities, to more specific functionalities and utilities that usually make a difference and offer added value with respect to the other proposals [14].

Table 1 lists the characteristics and breaks them down into their respective features, including their outcomes. Characteristics are defined as follows:

- *C1: search components and techniques.* It refers to the collection of building blocks that can be combined to solve multi-objective problems.
- *C2: configuration.* It evaluates the possibility to create experiments, their parametrization and reporting capabilities.

- *C3: execution.* It covers aspects related to how experiments are run and controlled.
- *C4: utilities.* It encompasses available utilities, divided into GUI, benchmarks and quality indicators.
- *C5: documentation and community support.* It is focused on the available documentation and the technologies for software distribution and interaction with the development team and other users.
- *C6: software implementation.* It analyzes development decisions like the programming language or dependencies to external libraries, as well as source code metrics.
- *C7: performance at runtime.* It evaluates execution time and memory consumption to provide information regarding performance and scalability.

3.2 Evaluation process and supporting tools

The evaluation process started with the definition of a list of features of interest (see Section 3.1), which was iteratively refined during a preliminary analysis. Data were collected from both documentation and source code, using different strategies for their evaluation. For C1 to C6, the conducted process described below:

1. Determine a preliminary set of options—e.g., algorithms, benchmarks or indicators—according to the MOO literature [1, 2].
2. Create a check list for these options by agreeing a common nomenclature, and define the conditions that any MOF should meet for its positive evaluation.
3. Collect evidences of the level of compliance of the available documentation. If the information is not clear or missing, then the source code should be inspected. If the feature should be evaluated from the outcomes of an experiment, e.g., from log files, an example among those available in the MOF is executed. When needed, a new example is specifically implemented.
4. Refine the list of options during the process to add any recent developments. For inclusion, the algorithm, operator, benchmark or indicator should be accompanied by a reference. Otherwise, it should appear in at least two MOFs in order to be considered of general interest and not an in-house development. For this latter case, the option "other specific developments" is considered.

Note that two particular features require special treatment: C3F4 (fault recovery) and C6F3 (software metrics). As part of the fault recovery evaluation, we prepared short experiments with incomplete or erroneous configurations in order to observe how the MOF responds to missing or wrong parameter values, respectively. Such experiments also served to confirm the use of default values to fix an user's mistake. The following situations were considered:

1. Missing parameters. The framework is expected to alert the user to the lack of a mandatory search component or parameter, regardless of the mechanism used to solve such error (if any). The missing elements under evaluation are: a) population

size; b) stopping criterion; c) optimization problem; d) algorithm; and e) crossover operator (extensible to mutation operator).

2. Invalid values. The framework should report that the specific value of a parameter is not valid, regardless of the mechanism used to solve such error, if any. The following situations are considered: a) the population size is a negative number; b) the maximum number of evaluations is a negative number; c) the optimization problem does not exist (a wrong name is used); the algorithm does not exist (a wrong name is used); and d) the crossover probability is greater than 1 (extensible to the mutation operator).

3. Default values. All previous scenarios are evaluated. It is observed whether the MOF gives feedback about the assigned value.

As for C6F3, software metrics related to the maintainability and usability [15] are the most relevant for MOF users. Notice that reliability, portability and efficiency are already covered by other features, whereas other dimensions of the software quality model considered by the ISO Std. 25010, such as security, are less applicable to a MOF, since they are not critical systems. Maintainability involves aspects of modularity, which can be mapped to code size or its organization (number and type of artifacts like classes, functions, etc.), and testability, for which coverage is a well-established indicator. Usability is linked to understandability and learnability, for which complexity and documentation metrics are appropriate. Therefore, we create a list of 14 metrics divided into four groups: code metrics (C6F3a), complexity metrics (C6F3b), testing metrics (C6F3c) and documentation metrics (C6F3d). In terms of support tools, we found that two suites, namely SonarQube[2] and the Eclipse plugin *Metrics*,[3] served our purposes. We use SonarQube 8.3 and the following plugin versions for each programming language: C# (8.6.1.17183), C++ (0.9.5), Java (6.3.0.21585) and Python (2.8.0.6204). It should be noted that the C++ plugin is provided by a third-party[4] and the latest non-commercial version is only compatible with releases below SonarQube 7.x. As a consequence, the complexity and documentation metrics defined in later SonarQube versions cannot be obtained in this case. SonarQube relies on external tools to measure coverage. We configure it to use the following ones: *dotCover*[5] for C#, *eclEmma*[6] for Java, *gcov*[7] for C++ and *coverage*[8] for Python. Notice that Python and C# plugins do not support condition coverage.

Two experiments were planned to evaluate characteristic C7 in order to gather several execution time and memory measurements under different conditions. A first experiment aims to assess the performance of each framework for a variety of algorithms and benchmarks. The second experiment seeks to analyze how well a MOF scales with respect to the population size, the maximum number of generations and the number of objectives. Notice that the choice of algorithms, operators and benchmarks should be founded on the outcomes of C1F2, C1F3 and C4F2, respectively. No algorithm has been implemented

[2] `http://www.sonarqube.org/` (Accessed May 28th, 2020)
[3] `http://metrics.sourceforge.net/` (Accessed May 28th, 2020)
[4] `https://github.com/SonarOpenCommunity/sonar-cxx` (Accessed May 28th, 2020)
[5] `https://www.jetbrains.com/dotcover` (Accessed May 28th, 2020)
[6] `https://www.eclemma.org/` (Accessed May 28th, 2020)
[7] `https://gcovr.com/en/stable/` (Accessed May 28th, 2020)
[8] `https://coverage.readthedocs.io/en/latest/` (Accessed May 28th, 2020)

in order not to introduce bias and to obtain fair results. Only some benchmarks and genetic operators were carefully implemented when no common options were available. Since benchmarks and operators are small pieces of software, they could be implemented following the programming style of the corresponding MOF. The list of algorithms, operators and benchmarks is presented in Section 10.1.1.

Experiments were run on a Debian 8 computer with 8 cores Intel Core i7-2600 CPU at 3.40 GHz and 16 GB RAM. The following compilers or interpreters, with default optimization options, were used: MSBuild 15.0 (C#), g++ 4.92 (C++), JRE 8 (Java) and Python 2.7. Syrupy[9] was used to log CPU time and RAM memory consumption at runtime. This tool can be configured to take values at regular intervals. After preliminary experiments, we observed than setting the same interval for all executions would be inappropriate because of the differences in the total execution time among frameworks, especially under the conditions of the second experiment. We experimentally adjusted the interval depending on the framework and configuration in order to always obtain 30 measurements. Runs were executed several times so as not to introduce bias into the measurement. For the first experiment, we repeated the execution five times, and the process was done using one core only. To keep an affordable experimentation time for the second experiment, six executions were run in parallel (two cores) per each pair MOF/configuration. We established an even number of runs because we interleaved the executions of different pairs of MOFs. This mechanism gave us more guarantees that all MOF had the same changes to execute under more or less overload due to SO processes during the days the experiment took to complete.

All the information has been conveniently structured and revised by different authors, solving disagreements when needed. With the aim of keeping the evaluation process objective, outcomes are reported without considering any qualitative assessment, i.e., no weights were set to reflect the importance of each feature.

3.3 Selected frameworks

In order to obtain a representative sample of software tools, we did not impose any restriction regarding the way they provide support to multi-objective optimization. Therefore, any tool or library including the implementation of at least one multi-objective metaheuristic algorithm was initially selectable. Although we considered diverse programming languages, we assumed that any selected MOF should have been designed under the precepts of the object-oriented paradigm. This gives us certain guarantees as to how the framework was built, ideally based on independent components, e.g., decoupled algorithms and operators, which allow the user to define his/her own experiments. Finally, notice that source code and documentation must be publicly available in order to carry out measurement and experimentation on the frameworks.

Table 2 shows the list of the ten selected frameworks, as well as the analyzed version – the latest stable version available at the time of writing – and website. Two groups can be distinguished with respect to its degree of specialization. DEAP, ECJ, HeuristicLab, EvA and Opt4J are multi-purpose frameworks, whereas jMetal, MOEA Framework and Platypus are mostly focused on multi-objective optimization. Additionally, JCLEC-MO and ParadisEO-MOEO represent intermediate solutions, since they both present an inde-

[9] `https://github.com/jeetsukumaran/Syrupy` (Accessed May 28th, 2020)

Table 2: Basic information of the ten frameworks under analysis.

Name	Ref.	Version	Year	Website
DEAP	[16]	1.3.0	2019	`https://github.com/DEAP/deap`
ECJ	[17]	27	2019	`https://cs.gmu.edu/~eclab/projects/ecj/`
EvA	[18]	2.2.0	2015	`http://www.ra.cs.uni-tuebingen.de/software/eva2/`
HeuristicLab	[19]	3.3.16	2019	`http://dev.heuristiclab.com/`
JCLEC-base + JCLEC-MO	[20] [21]	4.0 1.0	2014 2018	`http://jclec.sourceforge.net/` `http://www.uco.es/kdis/jclec-mo`
jMetal	[22]	5.9	2019	`http://jmetal.github.io/jMetal/`
MOEA Framework	[23]	2.13	2019	`http://moeaframework.org/`
Opt4J	[24]	3.1.4	2015	`http://opt4j.sourceforge.net/`
ParadisEO-MOEO	[25]	2.0.1	2012	`http://paradiseo.gforge.inria.fr/`
Platypus	[26]	1.0.4	2020	`https://github.com/Project-Platypus/Platypus`

pendent core package and complementary modules or wrappers for advanced or specialized techniques.

Table 3: Coverage of C1F1: types of metaheuristics.

Metaheuristic	**DEAP**	**ECJ**	**EvA**	**HeuristicLab**	**JCLEC-MO**	**jMetal**	**MOEA Framework**	**Opt4J**	**ParadisEO-MOEO**	**Platypus**
C1F1a: single-solution-based metaheuristics										
Hill Climbing			✓						●	
Iterated Local Search									●	
Simulated Annealing			✓					●	●	
Tabu Search									●	
Variable Neighborhood Search									●	
Pareto-based Local Search									✓	
C1F1b: population-based metaheuristics										
Cellular Algorithm						✓				
Differential Evolution			✓			✓	✓	✓		
Evolution Strategy			✓		✓	✓	✓			
Evolutionary Programming			✓		✓					
Genetic Algorithm	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Genetic Programming	✓	✓	✓	✓	✓		✓			
Grammatical Evolution		✓					✓			
Particle Swarm optimization			●		✓	✓	✓	✓		✓
Scatter Search			●		✓					

4 Search components and techniques

Three different features have been defined to evaluate the variety of search components and techniques provided by each framework:

C1F1: types of metaheuristics. This feature reports the metaheuristic models that can be applied to solve multi-objective problems. Starting from the set of metaheuristics defined in the literature [27, 28], only those paradigms that are supported by at least one framework have been included in the comparison. Metaheuristics have been classified into two groups. Therefore, C1F1a stands for single-solution-based metaheuristics, while C1F1b considers population-based metaheuristics. Two possibilities were considered to decide whether a multi-objective metaheuristic is really available or not. Firstly, a MOF defines an abstract implementation of the metaheuristic paradigm, i.e., the search iterative procedure, that is instantiated with specific elements of multi-objective problem solvers, e.g., Pareto evaluation. Secondly, the MOF includes an algorithm that is specific to such metaheuristic paradigm, e.g., the Pareto Archive Evolution Strategy confirms the availability of evolution strategies. Other metaheuristics included in the MOF are discarded because their implementation or configuration is tightly coupled to single-objective problems. The outcome for this feature is presented as a check list in Table 3. A partial fulfillment of this feature, represented using the symbol ●, indicates that the metaheuristic can manage multiple objectives but a unique fitness function, either based on an aggregation method or considering only one objective, is considered during the evaluation phase.

C1F2: families of algorithms. Table 4 presents the list of algorithms currently implemented in each framework, classified into families according to the literature [2, 5, 6]. The classification applied is described next:

- *C1F2a: first generation (1G)*: This category was established by Goldberg [5] to group all the algorithms whose selection mechanism is based on the Pareto dominance and also apply niching or fitness sharing techniques.
- *C1F2b: second generation (2G)*: These algorithms define an elitism mechanism based on the use of an external archive or a specific replacement strategy to enhance selective pressure [5]. Some of them are extensions of first-generation algorithms.
- *C1F2c: relaxed dominance (RDOM)*: The modification of the Pareto dominance principle is the main characteristic of these algorithms [6].
- *C1F2d: indicator based (IND)*: Algorithms belonging to this category use a quality indicator to guide the search process [6].
- *C1F2e: decomposition (DEC)*: Also known as scalarized or aggregation methods, these algorithms transform the original problem into multiple scalar subproblems [2].
- *C1F2f: reference set (REFS)*: These algorithms use a set of reference points to evaluate the quality of solutions or discard less interesting solutions [6].
- *C1F2g: preference based (PREF)*: These algorithms manage user's preferences to guide the search towards some regions of the Pareto front [6].

The publication year and the reference to the original publication describing the algorithm are shown in Table 4. In addition, the total number of available algorithms and their novelty have been considered for further evaluation, which is measured in terms of the median of the publication year. Notice that non-native implementations and those algorithms for which an external reference is not provided have not been considered. Similarly, variants of an algorithm, e.g., steady-state NSGA-II, are not counted as different algorithms.

Table 4: Coverage of C1F2: families of algorithms.

Metaheuristic	**DEAP**	**ECJ**	**EvA**	**HeuristicLab**	**JCLEC-MO**	**jMetal**	**MOEA Framework**	**Opt4J**	**ParadisEO-MOEO**	**Platypus**
C1F2a: first generation (1G)										
VEGA (1985) [29]							✓			
MOGA (1995) [30]			✓						✓	
NSGA (1995) [31]			✓						✓	
C1F2b: second generation (2G)										
SPEA (1999) [32]			✓							
PAES (2000) [33]					✓	✓	✓			
PESA (2000) [34]			✓							
PESA2 (2001) [35]			✓			✓	✓			
SPEA2 (2001) [36]	✓	✓	✓		✓	✓	✓	✓	✓	✓
NSGA-II (2002) [37]	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
GDE3 (2005) [38]						✓	✓			✓
CMAES-MO (2007) [39]			✓	✓			✓			
MOCHC (2007) [40]					✓	✓				
AbYSS (2008) [41]						✓				
CellDE (2008) [42]						✓				
MOCell (2009) [43]						✓				
FAME (2019) [44]						✓				
C1F2c: relaxed dominance (RDOM)										
ϵ-MOEA (2002) [45]					✓		✓			✓
OMOPSO (2005) [46]					✓	✓	✓	✓		✓
SMPSO (2009) [47]					✓	✓	✓			✓
GrEA (2013)* [48]					✓					
C1F2d: indicator based (IND)										
IBEA (2004) [49]					✓	✓	✓		✓	✓
SMS-EMOA (2007) [50]					✓	✓	✓	✓		
HypE (2011)* [51]					✓					
MOMBI (2013)* [52]						✓				
MOMBI-II (2015)* [53]						✓				
D-NSGA-II (2018)* [54]						✓				
C1F2e: decomposition (DEC)										
MSOPS (2003) [55]							✓			
MOEA/D (2007) [56]					✓	✓	✓			✓
dMOPSO (2011) [57]						✓				
DBEA (2015)* [58]							✓			
MOEADD (2015)* [59]						✓				
CDG (2018) [60]						✓				
C1F2f: reference set (REFS)										
R-NSGA-II (2006) [61]						✓				
NSGA-III (2014)* [62]	✓	✓			✓	✓	✓			✓
RVEA (2016)* [63]					✓		✓			
C1F2g: preference based (PREF)										
WASF-GA (2014) [64]						✓				
ESPEA (2015) [65]						✓				
GWASF-GA (2015) [66]						✓				
PAR (2016)* [67]					✓					
Number of algorithms	3	3	8	2	15	26	17	4	5	9
Median year of publication	**2002**	**2002**	**2001**	**2005**	**2007**	**2009**	**2005**	**2004**	**2002**	**2005**

* Originally proposed as a many-objective algorithm

Table 5: Coverage of C1F3a: variety of crossover operators (binary, integer and permutation encoding).

Operator	**DEAP**	**ECJ**	**EvA**	**HeuristicLab**	**JCLEC-MO**	**jMetal**	**MOEA Framework**	**Opt4J**	**ParadisEO-MOEO**	**Platypus**
Binary encoding										
Half-Uniform Crossover (HUX) [68]					✓	✓	✓			✓
N-Points Crossover (NPX) [69]	✓	✓	✓	✓	✓	✓	✓	✓	✓	
One Point Crossover (1PX) [70]	✓	✓	✓	✓	✓	✓	✓	✓	✓	
Uniform Crossover (UX) [71]	✓	✓	✓	✓	✓		✓	✓	✓	
Integer encoding										
Arithmetic Crossover (ARX) [72]				✓						
Average Crossover (AVX) [73]				✓						
Blend Alpha Crossover (BLX-α) [74]				✓						
Blend Alpha Beta Crossover (BLX-$\alpha\beta$) [75]				✓						
Discrete Crossover (DX) [76]				✓						
Heuristic Crossover (HX) [77]				✓						
Intermediate Crossover (IX) [76]		✓								
Line Crossover (LIX) [76]		✓								
Local Crossover (LOX) [78]				✓						
N-Points Crossover (NPX) [69]	✓	✓	✓		✓	✓		✓		
One Point Crossover (1PX) [70]	✓	✓	✓	✓	✓		✓	✓	✓	
Simulated Binary Crossover (SBX) [79]						✓				
Uniform Crossover (UX) [71]	✓	✓	✓		✓			✓	✓	
Permutation encoding										
Cosa Crossover (COX) [80]				✓						
Cyclic Crossover (CYX) [81]				✓						
Davis Uniform Crossover (DUX) [82]								✓		
Edge Recombination Crossover (ERX) [83]				✓						
Maximal Preservative Crossover (MPX) [84]				✓						
Order Crossover (OX) [82]	✓			✓	✓				✓	
Order-2 Crossover (2OX) [85]				✓						
Order Based Crossover (OBX) [85]				✓						
Partially Mapped Crossover (PMX) [86]	✓		✓	✓	✓	✓	✓			✓
Position Crossover (POX) [85]				✓				✓		
Tate Uniform Crossover (TUX) [87]				✓						

Table 6: Coverage of C1F3a: variety of crossover operators (double and tree encoding).

Operator	**DEAP**	**ECJ**	**EvA**	**HeuristicLab**	**JCLEC-MO**	**jMetal**	**MOEA Framework**	**Opt4J**	**ParadisEO-MOEO**	**Platypus**
Double encoding										
Adaptive Metropolis Crossover (AMX) [88]							✓			
Arithmetic Crossover (ARX) [72]			✓	✓	✓				✓	
Average Crossover (AVX) [73]				✓						
Blend Alpha Crossover (BLX-α) [74]	✓		✓	✓	✓	✓		✓		
Blend Alpha Beta Crossover (BLX-$\alpha\beta$) [75]				✓						
Discrete Crossover (DX) [76]			✓	✓	✓				✓	
Flat Crossover (FX) [89]			✓		✓					
Heuristic Crossover (HX) [77]				✓	✓					
Intermediate Crossover (IX) [76]		✓	✓						✓	
Line Crossover (LIX) [76]		✓								
Local Crossover (LOX) [78]				✓						
N-Points Crossover (NPX) [69]	✓	✓				✓	✓	✓		
One Point Crossover (1PX) [70]	✓	✓	✓	✓			✓	✓	✓	
Parent-centric Crossover (PCX) [90]			✓				✓			✓
Simulated Binary Crossover (SBX) [79]	✓	✓	✓	✓		✓	✓	✓	✓	✓
Random Convex Crossover (RCX) [78]				✓						
Simplex Crossover (SPX) [91]			✓				✓			✓
Unfair Average Crossover (UAX) [92]								✓		
Uniform Crossover (UX) [71]	✓	✓			✓		✓	✓	✓	
Unimodal Normal Distribution Crossover (UNDX) [93]			✓				✓			✓
Tree encoding										
Koza Subtree Crossover (KSTX) [94]	✓	✓	✓	✓	✓		✓		✓	
One Point Tree Crossover (1PTX) [95]	✓				✓					
Size Fair Tree Crossover (SFTX) [96]		✓								

Table 7: Coverage of C1F3b: variety of mutation operators (binary, integer and permutation encoding).

Operator	**DEAP**	**ECJ**	**EvA**	**HeuristicLab**	**JCLEC-MO**	**jMetal**	**MOEA Framework**	**Opt4J**	**ParadisEO-MOEO**	**Platypus**
Binary encoding										
Bit Flip Mutation (BFM) [70]	✓		✓	✓	✓				✓	✓
N Bit Flip Mutation (NBFM) [70]			✓		✓				✓	
Uniform Bit Flip Mutation (UBFM) [70]		✓	✓	✓	✓	✓	✓	✓	✓	
Uniform Random Resetting Mutation (URRM) [97]		✓								
Integer encoding										
Normal Mutation (NM) [98]				✓						
Random Resetting Mutation [97]					✓					
N Random Resetting Mutation [97]			✓		✓					
Uniform Normal Mutation (UNM) [98]				✓						
Uniform Polynomial Mutation (UPM) [99]						✓				
Uniform Random Resetting Mutation (URRM) [97]	✓	✓			✓		✓	✓		
Permutation encoding										
2-Opt Mutation [97]				✓	✓	✓	✓		✓	
3-Opt Mutation [97]				✓						
N Swap Mutation [97]			✓		✓			✓	✓	✓
Displacement Mutation (DM) [72]				✓						
Insertion Mutation (INSM) [85]				✓			✓	✓		✓
Inversion Mutation (INVM) [85]			✓	✓				✓		
Scramble Mutation (SCM) [85]				✓						
Fogel Mutation (FM) [100]				✓						

Table 8: Coverage of C1F3b: variety of mutation operators (double and tree encoding).

Operator	**DEAP**	**ECJ**	**EvA**	**HeuristicLab**	**JCLEC-MO**	**jMetal**	**MOEA Framework**	**Opt4J**	**ParadisEO-MOEO**	**Platypus**
Double encoding										
Mühlenbein Mutation (MÜM) [76]				✓						
Non Uniform Mutation (NUM) [72]				✓	✓	✓				✓
Normal Mutation (NOM) [98]				✓					✓	
Polynomial Mutation (PM) [99]			✓	✓					●[1]	
Uniform Mutation (UM) [97]				✓		✓			✓	
Uniform Modal Mutation (UMM) [101]					✓					
Uniform Mühlenbein Mutation (UMÜM) [76]					✓					
Uniform Normal Mutation (UNM) [98]	✓	✓						✓		✓
Uniform Polynomial Mutation (UPM) [99]	✓	✓			✓	✓	✓	✓		✓
Uniform Random Resetting Mutation (URRM) [97]		✓			✓	✓	✓			✓
Tree encoding										
All Nodes Mutation (ANM) [102]		✓			✓					
Demote Mutation (DM) [102]		✓			✓					
ERC Mutation (ERCM) [102]		✓								
Grow Mutation (GM) [103]					✓				✓	
Hoist Mutation (HM) [104]									✓	
Koza Subtree Mutation (KSM) [94]		✓	✓		✓		✓		✓	
One Node Mutation (1NM) [102]	✓	✓	✓	✓	✓				✓	
Promote Mutation (PRM) [102]		✓			✓					
Shrink Mutation (SHM) [103]	✓				✓				✓	
Swap Mutation (SWM) [103]		✓								

[1] Available as part of benchmarks (external contribution).

Table 9: Additional information for C1F3: specialized operators.

	DEAP	ECJ	EvA	HeuristicLab	JCLEC-MO	jMetal	MOEA Framework	Opt4J	ParadisEO-MOEO	Platypus
In-house developments for linear encodings	✓	✓	✓					✓	✓	✓
Operators for other types of encodings				✓						✓
Operators for specific metaheuristics (DE, ES, GE or GP)	✓	✓	✓	✓	✓	✓	✓		✓	✓
Operators for mixed encodings		✓				✓	✓	✓	✓	
Support to combine operators				✓			✓			✓
Operators with dynamic probabilities			✓				✓			
Local search as genetic operator						✓				

DE: Differential Evolution, ES: Evolution Strategy, GE: Grammatical Evolution, GP: Genetic Programming

C1F3: encodings and operators. Evolutionary algorithms are the predominant metaheuristic model in the majority of selected frameworks, so the evaluation of this feature is focused on the availability of genetic operators. Preliminary lists of crossover and mutators have been extracted grouping them by type of encoding (binary, integer, permutation, double and tree). Then, a common name has been established by checking both external and internal documentation, as well as references in the literature [13, 70, 72, 78, 94, 97]. Tables 5 and 6 show the list of available crossover operators (C1F3a), while Tables 7 and 8 give the equivalent information concerning mutation operators (C1F3b). Other specialized elements provided by MOFs are listed in Table 9.

5 Configuration

Table 10 summarizes the evaluation of the selected MOFs with respect to their configuration capabilities. The list of features comprising this characteristic is detailed next.

C2F1: inputs. It refers to the mechanism and format used to prepare the execution of an algorithm. The two features used to assess this characteristic are described as follows:

- *C2F1a: type of input.* It reports the alternatives to set-up an experiment. Four options have been identified: implementation of code, use of the command line (CLI), file loading, and by means of the GUI. Given that the source code of all the considered MOFs are publicly available, coding as configuration mechanism refers here to the presence of specific classes aimed at facilitating the algorithm set-up, so that the user does not need to build it from scratch. The symbol ● indicates that the MOF partially supports the corresponding option, e.g., only certain parameters can be configured when such mechanism is used.
- *C2F1b: input data format.* If configuration files are supported, the specific data representation format is analyzed here. Key-value pairs (KVP) and markup languages like XML (eXtensible Markup Language) and YAML (Ain't Another Markup Language) are the formats currently supported.

C2F2: batch processing. Three features have been established with respect to the alternatives to prepare and execute a set of tasks. A single and independent algorithm execution is referred as a task. The gathered information primary comes from the external documentation, though source code has been also inspected to corroborate or complement imprecise information.

- *C2F2a: task replication.* The execution of a task several times should guarantee that the obtained results are exactly the same. In this sense, the possibility to configure the random seed used by stochastic algorithms has been inspected.
- *C2F2b: sequential tasks.* It refers to the capability of running multiple tasks in sequence.
- *C2F2c: parallel tasks.* The execution of several tasks in parallel is supported.

C2F3: outputs. This feature is focused on the type of outcomes generated by the MOF, and to what extent they can be customized by the user. Both external documentation and execution examples have constituted the source of information to identify more concrete features, which are defined as follows:

- *C2F3a: type of output.* Console, file and GUI are the possible elements to control the output flow.

Table 10: Coverage of C2: configuration.

Feature	**Subfeature**	**DEAP**	**ECJ**	**EvA**	**HeuristicLab**	**JCLEC-MO**	**jMetal**	**MOEA Framework**	**Opt4J**	**ParadisEO-MOEO**	**Platypus**
C2F1: inputs	*C2F1a: type of input*										
	Code	✓			✓		✓	✓		✓	✓
	Command line		✓	✓			●[1]				
	File		✓	✓		✓			✓	✓	
	GUI			✓	✓				✓		
	C2F1b: input data format										
	KVP		✓							✓	
	XML					✓			✓		
	YAML			✓							
C2F2: batch processing	*C2F2a: task replication*	✓	✓	✓	✓	✓	✓	✓	✓	✓	
	C2F2b: sequential tasks	✓		✓	✓	✓	✓	✓			✓
	C2F2c: parallel tasks			✓	✓		✓			●[2]	✓
C2F3: outputs	*C2F3a: type of output*										
	Command line	✓	✓	✓	✓	✓		✓		✓	✓
	File		✓	✓	✓	✓	✓	✓	✓	✓	
	GUI		✓	✓	✓		●[3]	✓	✓		
	C2F3b: output data format										
	CSV					✓	✓				
	JSON										✓
	TSV		✓	✓			✓	✓	✓	✓	
	XLSX				✓						
	XML						✓				
	C2F3c: parametrization										
	Report frequency			✓		✓	✓	✓	✓	✓	✓
	Report path		✓			✓	✓		✓	✓	
	C2F3d: degree of flexibility										
	Customizable		✓	✓	✓	✓		✓	✓	●[4]	
	Programmable	✓					✓	✓		✓	✓

[1] Only the problem and the reference Pareto front can be configured without programming.
[2] Not supported on all platforms.
[3] A chart with current results (quality indicator or PF) can be generated dynamically.
[4] Output parameter in configuration files only refers to the storage of the Pareto front.

- *C2F3b: data format.* The specific data format used for output files is detailed here. Tabular structures like those provided by CSV (*Comma Separated Values*) and TSV (*Tabular Separated Values*) are frequently used. MOFs also consider XML and other formats derived from it, such as XLSX (Microsoft Office open XML format for spreadsheet files).

- *C2F3c: parametrization.* It refers to customization option to indicate how often reports should be generated or where they will be stored.
- *C2F3d: degree of flexibility.* It evaluates whether the generation of outputs can be adapted according to the user's preferences, with independence of the type of information. More specifically, output components are defined as customizable, when they can be selected and configured by means of the GUI or external files, or as programmable, i.e., they should be included as part of the main program.

Table 11: Coverage of C3: execution.

Feature	Subfeature	DEAP	ECJ	EvA	HeuristicLab	JCLEC-MO	jMetal	MOEA Framework	Opt4J	ParadisEO-MOEO	Platypus
C3F1: multi thread exec.	*C3F1a: parallel evaluation*	✓	✓		✓	✓	✓	✓	✓	✓	
	C3F1b: parallelism in other phases		✓						✓	✓	
C3F2: distr. execution	*C3F2a: master-slave model*		✓		✓			✓		●[1]	
	C3F2b: island model	✓	✓	✓				✓			
C3F3: stop & restart mode	*C3F3a: object state serialization*	✓	✓	✓	✓	✓	✓	✓	✓	●[2]	
	C3F3b: save and load checkpoints	✓	✓		✓			✓	✓	✓	
C3F4: fault recovery	*C3F4a: control of parameter values*										
	Missing parameters	80%	100%	40%	40%	80%	40%	60%	40%	60%	60%
	Wrong numerical values	40%	100%	60%	40%	40%	100%	60%	60%	60%	40%
	Default values	0%	10%	50%	40%	10%	30%	40%	30%	10%	20%
	C3F4b: exception handling										
	Native exceptions	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	Specific exceptions		✓		✓		✓	✓		✓	✓
C3F5: execution and control logs			✓	✓	✓	✓	✓				✓

[1] Not supported on all platforms.
[2] Each execution state is saved into a text file.

6 Execution

This characteristic evaluates those aspects related to the execution capabilities of a MOF and their fault tolerance mechanisms. Next, detailed descriptions of all the features within this characteristic are provided. The evaluation, whose results are shown in Table 11, is based on the external documentation provided by each MOF, although source code, in form of running examples, has been also considered in case of incomplete or imprecise information.

C3F1: multi-thread execution. According to how multi-threading is currently supported by MOFs, two options can be distinguished: parallel evaluation of solutions (C3F1a) and parallel execution of other phases of the search (C3F1b). The parallel execution of algorithms, as independent tasks was already covered by *C2F2c*.

C3F2: distributed execution. Distributing the execution among several machines does not only provide further alternatives to execute, but also to design metaheuristic algorithms. After consulting the available documentation of each MOF, two distribution mechanisms have been identified: the master-slave model (C3F2a), either to evaluate solutions or to perform the entire search process, and the island model (C3F2b).

C3F3: stop and restart mode. MOFs can support the storage of the state of an execution, probably with the aim of restoring it later. A first feature, *C3F1a: object state serialization*, considers whether the MOF implements a serialization mechanism. The second feature, *C3F2b: save and load checkpoints*, looks for the existence of a working procedure to save and load checkpoints, meaning that serialized objects are effectively used.

C3F4: fault recovery. This feature refers to the capabilities of the MOF to control errors and recover from them. The following subfeatures have been defined:

- *C3F4a: control of parameter values.* It evaluates to what extent a given MOF checks parameter values before launching an execution. Different wrong configurations have been prepared and executed (see Section 3). As a way to guarantee a fair comparison, test cases have been defined in terms of common parameters that can be configured in any framework. When possible, configuration files, console or coding are the preferable configuration options, since it is likely that a GUI does not permit the configuration of an experiment lacking a search component. Table 11 shows the percentage of successful cases for all the proposed scenarios.
- *C3F4b: exception handling.* This feature assesses the completeness of the exception handling mechanism. Firstly, handling exceptions defined by the corresponding programming language reflects that a MOF controls possible runtime errors. In addition, MOFs can define their own specific exceptions to deal with unexpected situations.

C3F5: execution and control logs. This feature establishes whether a framework provides some kind of log system as part of at least one output device.

Table 12: Coverage of C4F1: graphical user interface and C4F2: benchmarks.

Feature	Subfeature	DEAP	ECJ	EvA	HeuristicLab	JCLEC-MO	jMetal	MOEA Framework	Opt4J	ParadisEO-MOEO	Platypus
C4F1: GUI	*C4F1a: design of experiments*			✓	✓						
	C4F1b: parametrization			✓	✓			●[1]	✓		
	C4F1c: execution control		✓	●[2]	✓			●[2]	✓		
	C4F1d: visualization of results			✓	✓		✓	✓	✓		
C4F2: benchmarks	*C4F2a: continuous optimization problems*										
	BBOB16 [105]							✓			
	CEC'09 [106]						✓ (10)	✓ (23)			✓ (23)
	CEC'18 [107]						✓ (15)				
	CDTLZ [108]						✓ (6)	✓ (7)			
	DTLZ [109]	✓ (7)			✓(8)	✓ (7)	✓ (7)	✓ (5)	✓ (7)	● (7)[3]	✓ (5)
	Fonseca & Fleming [110]	✓	✓		✓		✓	✓			
	GLT [111]						✓ (6)				
	IHR [112]				✓						
	Kursawe [113]	✓	✓		✓		✓	✓			
	LGZ [114]						✓				
	LZ [115]						✓ (9)	✓ (9)			
	Osyczka [116]						✓ (1)	✓ (2)			
	Poloni [117]	✓	✓					✓			
	Schaffer [29]	✓	✓ (1)		✓ (2)		✓ (1)	✓ (2)		✓ (1)	
	Srinivas [31]						✓	✓			
	Tanaka [118]						✓	✓			
	Viennet [119]						✓ (3)	✓ (4)			
	WFG [120]						✓ (9)	✓ (9)	✓ (9)	● (9)[3]	✓ (9)
	ZDT [121]	✓(5)	✓ (5)		✓ (5)	✓ (6)	✓ (6)	✓ (6)	✓ (6)	● (4)[3]	✓ (6)
	Other functions	✓	✓	✓			✓	✓			
	C4F2b: combinatorial problems										
	Flowshop [122]									●[3]	
	Knapsack [123]					✓		✓	✓		
	MNK [124]							✓			
	LOTZ [125]							✓	✓		
	Queens [126]								✓		
	QAP [127]									✓	
	TSP [128]					✓	✓			✓	

[1] Only the number of seeds and the maximum number of evaluations can be configured.
[2] Execution can be stopped but not resumed.
[3] Available as external contribution.

7 Utilities

This characteristic compiles additional facilities that can be of interest to different types of user. Table 12 shows the information regarding the graphical user interface and the availability of benchmarks, which represent the first two features within this characteristic. Table 13 details the outcomes for the third feature, which is focused on the implementation of quality indicators. Next, all the features are described in more detail.

C4F1: graphical user interface. Four features are defined to evaluate different aspects of the GUI. They are assessed by running some examples and consulting the available

Table 13: Coverage of C4F3: quality indicators.

Quality indicator	**DEAP**	**ECJ**	**EvA**	**HeuristicLab**	**JCLEC-MO**	**jMetal**	**MOEA Framework**	**Opt4J**	**ParadisEO-MOEO**	**Platypus**
C4F3a: unary indicators										
Generalised Spread (ΔS) [129]					✓	✓				
Hypervolume (HV) [36]	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Overall Nondominated Vector Generation (ONVG) [130]			✓		✓					
Spacing [131]				✓	✓		✓			✓
Spread (S) [37]	✓				✓	✓				
C4F3b: binary indicators										
Additive Epsilon ($I_{\epsilon+}$) [132]					✓	✓	✓		✓	
Average PF Error [37]	✓									
Entropy [133]									✓	
Epsilon (I_{ϵ}) [132]					✓					✓
Error Ratio (ER) [130]			✓		✓	✓				
Generational Distance (GD) [130]			✓	✓	✓	✓	✓			✓
Hyperarea Ratio (HR) [36]					✓					
Inverted Generational Distance (IGD) [134]				✓	✓	✓	✓			✓
Inverted Generational Distance + (IGD+) [135]						✓				
Maximum Pareto Front Error (ME) [130]			✓		✓		✓			
Nondominated Vector Addition (NVA) [1]					✓					
Overall Nondom. Vector Generation Ratio (ONVGR) [130]					✓					
R1 [136]							✓			
R2 [136]					✓	✓	✓			
R3 [136]					✓		✓			
Two Set Coverage (a.k.a. coverage or contribution) [121]					✓	✓	✓		✓	
C4F3c: ternary indicators										
Relative Progress [1]					✓					

documentation. A partial fulfillment is possible, specially as MOFs do not always provide the same flexibility when applying the GUI facilities to the resolution of multi-objective problems, or there are some restrictions. The features are described as follows:

- *C4F1a: design of experiments.* It reports about whether the GUI supports the creation, loading and modification of experiments. The difference between an experiment and a simple execution of an algorithm is that the former can include replicated components, such as algorithms and operators, with different values. In addition, an experiment can include post-processing steps to analyze the results.
- *C4F1b: parametrization.* The configuration of parameters and the choice of search components constitute the main functionalities to be evaluated by this feature.
- *C4F1c: execution control.* It analyzes the possibility to start, pause and stop the execution of an experiment by using the GUI.
- *C4F1d: visualization of results.* It evaluates the availability of specific graphical

elements to show the outcomes of an experiment, e.g., convergence graphics or scatter plots to visualize the returning Pareto front.

C4F2: benchmarks. Researchers need test problems in order to conduct experimental studies. Depending on the nature of the problem, continuous optimization and combinatorial problems can be distinguished, which are represented by C4F2a and C4F2b features, respectively. Source packages and external documentation has been used to extract the list of test problems currently implemented by the selected MOFs. In case of families of test functions, e.g., DTLZ or ZDT, the number of implemented functions is shown in parentheses in Table 12.

C4F3: quality indicators. This type of performance measure serves to quantify the quality of the returning Pareto front in different ways [1]. Given that these measures can be classified according to the number of PFs required to compute them, the following three features have been defined: *C4F3a: unary indicators*, *C4F3b: binary indicators* and *C4F3c: ternary indicators*.

8 Documentation and community support

This characteristic analyzes additional aspects surrounding the development and use of the selected frameworks. Websites and public repositories constitute the main sources of information. It should be noted that most of the features defined within this characteristic refer to the development project, though the evaluation of the documentation (with the exception of research papers) is focused on the last release. Table 14 shows the gathered information organized into the following five features:

C5F1: software license. This characteristic details the type of license each MOF uses. The symbol ● is used to specify that the modules comprising the framework have different licenses.

C5F2: documentation. Four features serve to analyze the available documentation:

- *C5F2a: tutorials.* They are documents containing basic information and examples to users. It can include tutorial lessons, *How to's* and Frequently Asked Questions (FAQs) available in any format.
- *C5F2b: API.* The Application Programming Interface (API) provides the specification of classes, properties and methods that researchers might need to implement their own algorithms.
- *C5F2c: reference manuals.* They refer to advanced topics and design aspects, e.g., architecture.
- *C5F2d: code samples.* Additional code examples might serve to show how to implement new optimization problems or search operators. It should be noted that code snapshots included in tutorials and benchmarks already distributed with the source code are not considered here.
- *C5F2e: research papers.* Some MOFs have been published in journal and conference papers.

C5F3: software update. To evaluate how often each MOF is being updated, the number of releases since January 2015 is counted.

C5F4: development facilities. Two features serve to evaluate to what extent MOFs provide developers with useful mechanisms to access, compile and extend the source code:

- *C5F4a: public repositories.* This feature evaluates whether development teams use public repositories to host and distribute the code.
- *C5F4b: compilation/distribution mechanism.* There exist several ways to facilitate the compilation of the different modules comprising a MOF. Build automation tools like Maven also allow the configuration of MOFs as external dependencies. This feature lists the technologies currently used to address any of these tasks.

Table 14: Coverage of C5: documentation and community support.

Feature	**Subfeature**	**DEAP**	**ECJ**	**EvA**	**HeuristicLab**	**JCLEC-MO**	**jMetal**	**MOEA Framework**	**Opt4J**	**ParadisEO-MOEO**	**Platypus**
C5F1: software license											
Academic Free			✓								
CeCill										●	
GNU GPL					✓	✓					✓
GNU LGPL		✓		✓				✓	✓	●	
MIT							✓				
C5F2: documentation	*C5F2a: tutorials*	✓	✓	✓	✓	✓	✓	✓	✓	✓	
	C5F2b: API	✓	✓	✓		✓	✓	✓	✓	✓	
	C5F2c: reference manuals		✓		✓	✓	✓	●[1]		●[2]	
	C5F2d: code samples	✓		✓	✓	✓		✓		●[3]	✓
	C5F2e: research papers	✓	✓	✓	✓	✓	✓		✓	✓	
C5F3: software update		3	5	1	5	1	9	10	2	0	4
C5F4: development facilities	*C5F4a: public repositories*										
	Git	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
	Mercurial								✓		
	SVN				✓						
	C5F4b: compilation/distribution mechanism										
	Anaconda										✓
	Ant							✓			
	Gradle								✓		
	Makefile		✓							✓	
	Maven		✓	✓			✓	✓	✓		
	MSBuild				✓						
	PIP	✓									
C5F5: community	*C5F5a: contact email*		✓		✓			✓	✓	✓	
	C5F5b: forum or mailing list	✓		✓	✓				✓	✓	
	C5F5c: issue tracker	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

[1] The *Beginner's guide* is not publicly available.
[2] There are slides that include some hierarchies of core classes, but they are conceived as tutorial lessons or seminars.
[3] The available examples are benchmarks or third-party contributions.

C5F5: community. Users are provided with communication mechanisms to report bugs, make suggestions and even request new functionalities. The following features are considered:

- *C5F5a: contact email.* This feature considers whether a MOF provides its own email account to receive feedback from users.
- *C5F5b: forum or mailing list.* Forum and mailing lists are the most common options to allow users to interact each other, as well as with the development team, in order to solve their doubts.

- *C5F5c: issue tracker.* It is a common system to register bugs and requests, as well as follow their evolution.

Table 15: Coverage of C6F1: implementation and execution.

	DEAP	**ECJ**	**EvA**	**HeuristicLab**	**JCLEC-MO**	**jMetal**	**MOEA Framework**	**Opt4J**	**ParadisEO-MOEO**	**Platypus**
C6F1a: programming language										
C#				✓						
C++						●[1]			✓	
Java		✓	✓		✓	✓	✓	✓		
Python	✓					●[1]				✓
C6F1b: execution platform										
Linux	✓	✓	✓	●[2]	✓	✓	✓	✓	✓	✓
MacOS	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Windows	✓	✓	✓	✓	✓	✓	✓	✓	●[3]	✓

[1] Partial versions in C++ and Python are available.
[2] Partial support by using Mono tool.
[3] Module PEO is not compatible with Windows.

9 Software implementation

This characteristic provides different views of the software implementation. The characteristic is divided in three features. Table 15 shows the information regarding the development environment, while Table 16 lists external dependencies. Table 17 collects the metrics extracted from the source code, the symbol $-$ being used to indicate that a metric is not available for that language. Next, each feature is described.

C6F1: implementation and execution. This feature is focused on the technologies and platforms required to implement or execute the frameworks. It is divided into *C6F1a: Programming language* and *C6F1b: Execution platform.*

C6F2: external libraries. This feature is examined in terms of the purpose of the libraries used by the MOF: *C6F2a: configuration*, *C6F2b: graphics*, *C6F2c: mathematical processing*, *C6F2d: programming support*, *C6F2e: reporting*, *C6F2f: statistical analysis* and *C6F2g: testing.*

C6F3: software metrics. Metrics have been classified into four groups according to their scope: code, complexity, testing and documentation.

- *C6F3a: code metrics.* They provide information about the number and type of artifacts comprising the source code:

  - Lines of code, excluding comments and blank lines.
  - Percentage of duplicated lines.
  - Number of directories. Depending on the language, it corresponds to the number of packages in the source code directory.
  - Number of classes.
  - Number of abstract classes.
  - Number of interfaces.
  - Number of functions. Depending on the language, it corresponds to functions, methods or paragraphs.

- *C6F3b: Complexity metrics.* These metrics are related to the complexity of the code and existing dependencies:
  - Cyclomatic complexity. Also known as McCabe metric, it is based on the number of paths through the code.
  - Cognitive complexity. It evaluates the difficulty to understand the code, using on a mathematical model based on human judgment. Details can be found at SonarQube website.[10]

- *C6F3c: testing metrics.* These metrics are related to the available test cases:
  - Lines to cover. Number of code lines that can be covered by unit tests (without considering comments and blank lines).
  - Condition coverage. Percentage of covered conditions.
  - Line coverage. Percentage of code lines executed by unit tests.
  - Coverage. Coverage of conditions, evaluated as both 'true' and 'false', as well as lines. Expressed as percentage.

- *C6F3d: documentation metrics.* It evaluates code documentation as follows:
  - Density of comments. The ratio between comment lines and the sum of comment lines and total code lines. Expressed as percentage.

[10] `https://www.sonarsource.com/resources/white-papers/cognitive-complexity.html` (Accessed May 28th, 2020)

Table 16: Coverage of C6F2: external libraries.

External libraries	**DEAP**	**ECJ**	**EvA**	**HeuristicLab**	**JCLEC-MO**	**jMetal**	**MOEA Framework**	**Opt4J**	**ParadisEO-MOEO**	**Platypus**
C6F2a: configuration										
Commons configuration					✓					
SnakeYAML			✓							
C6F2b: graphics										
GNUplot									✓	
jFreeChart		✓					✓			
matplotlib										✓
NetronDiagraming				✓						
xchart						✓				
C6F2c: mathematical processing										
ALBLIB				✓						
AutoDiff				✓						
Commons Math		✓				✓	✓			
EJML		✓								
JAMA			✓							
LibSVM				✓						
MathJax				✓						
Numpy	✓									
SAT4J								✓		
C6F2d: programming support										
AvalonEdit				✓						
DayView				✓						
Commons Collections					✓					
Commons Lang					✓	✓	✓			
GUIVE								✓		
JZLIK		✓								
MPI4py										✓
ProtocolBuffers				✓						
PSH-ECJ		✓								
SciLab				✓						
Scoop	✓									
C6F2e: reporting										
Commons IO						✓				
Datapro4j					✓					
EPPlus				✓						
iText		✓								
SharpZipLib				✓						
C6F2f: statistical analysis										
rJava					✓					
C6F2g: testing										
Hamcrest					✓	✓		✓		
jaCoCo		✓								
jMock		✓								
jUnit		✓	✓		✓	✓	✓	✓		
Mockito						✓				
Spring-test						✓				

Table 17: Coverage of C6F3: software metrics.

Metric	**DEAP**	**ECJ**	**EvA**	**HeuristicLab**	**JCLEC-MO**	**jMetal**	**MOEA Framework**	**Opt4J**	**ParadisEO-MOEO**	**Platypus**
C6F3a: code metrics										
Lines of code (KLOC)	3.8	54.0	84.0	798.2	30.5	45.1	34.5	26.2	57.2	5.7
Duplicated lines (%)	1.6	8.0	10.4	8.8	12.4	17.7	3.6	1.2	8.5	5.1
Directories or packages	2	109	63	-	63	154	96	50	107	5
Classes	36	637	736	8,125	414	629	508	552	1,289	166
Abstract classes	-	36	33	547	58	28	35[1]	41	343[2]	-
Interfaces	-	30	96	930	41	62	24	66		-
Functions	367	3,496	8,947	76,329	2,827	3,488	2,956	2,220	5,600	659
C6F3b: complexity metrics										
Cyclomatic complexity	971	11,167	18,152	178,893	5,435	7,614	6,658	4,582	-	1,457
Cognitive complexity	1,154	13,162	16,820	15,5964	5,279	6,908	5,789	3,990	-	1,585
C6F3c: testing metrics										
Lines to cover (KLOC)	3.4	33.3	50.3	448.1	19.3	34.6	33.1	13.9	43.5	5.0
Condition coverage (%)	-	9.8	0.1	-	23.2	19.8	63.6	1.1	54.6	-
Line coverage (%)	6.4	10.2	0.3	15.1	23.9	17.3	70.4	1.4	33.3	16.9
Coverage (%)	6.4	10.0	0.3	15.1	23.8	17.9	68.4	1.3	41.2	16.9
C6F3d: documentation metrics										
Density of comments (%)	47.3	30.1	20.9	18.3	35.4	16.6	32.4	33.1	-	15.5

[1] Nine and two abstract classes belong to the *test* and *example* packages, respectively.
[2] Number of code files in which at least one virtual method is defined.

10 Performance at runtime

The aim of this characteristic is to measure execution time and memory consumption at runtime. A series of experiments considering different algorithms and benchmarks is carried out in order to obtain a wide range of measurements. Scalability has been also investigated varying the population size, the maximum number of generations and the number of objectives.

10.1 Experiment #1: comparison of algorithms and benchmarks

Table 18: Experiment #1: algorithms and benchmarks used.

			Group #1						Group #2			
Algorithm	**Benchmark**	**No. Obj.**	**DEAP**	**ECJ**	**EvA**	**HeuristicLab**	**Opt4J**	**ParadisEO-MOEO**	**JCLEC-MO**	**jMetal**	**MOEA Framework**	**Platypus**
NSGA-II	LOTZ	2	∗	∗	∗	∗	✓	∗	∗	∗	✓	∗
	ZDT1	2	✓	✓	∗	✓	✓	✓				
	ZDT4	2	✓	✓	∗	✓	✓	✓				
	ZDT6	2	✓	✓	∗	✓	✓	✓				
	KP	4	∗	∗	∗	∗	✓	∗	✓	∗	✓	∗
	DTLZ1	6	✓	∗	∗	✓	✓	✓	✓	✓	✓	✓
	DTLZ2	6	✓	∗	∗	✓	✓	✓	✓	✓	✓	✓
	DTLZ4	6	✓	∗	∗	✓	✓	✓				
SPEA2	LOTZ	2	∗	∗	∗		✓	∗				
	ZDT1	2	✓	✓	∗		✓	✓				
	ZDT4	2	✓	✓	∗		✓	✓				
	ZDT6	2	✓	✓	∗		✓	✓				
	KP	4	∗	∗	∗		✓	∗				
	DTLZ1	6	✓	∗	∗		✓	✓				
	DTLZ2	6	✓	∗	∗		✓	✓				
	DTLZ4	6	✓	∗	∗		✓	✓				
IBEA	LOTZ	2							∗	∗	✓	∗
	KP	4							✓	∗	✓	∗
	DTLZ1	6							✓	✓	✓	✓
	DTLZ2	6							✓	✓	✓	✓
MOEA/D	ZDT1	2							✓	✓	✓	✓
	ZDT4	2							✓	✓	✓	✓
	DTLZ1	2							✓	✓	✓	✓
	DTLZ2	2							✓	✓	✓	✓
OMOPSO	ZDT1	2							✓	✓	✓	✓
	ZDT4	2							✓	✓	✓	✓
	DTLZ1	6							✓	✓	✓	✓
	DTLZ2	6							✓	✓	✓	✓

10.1.1 Experimental design

The goal of this experiment is to study the behavior of MOFs by simulating how domain experts would usually work with them. In this scenario, algorithms are selected among those available and configured to their default settings. A representative collection of algorithms and benchmarks is required to draw some general conclusions without introducing subjectivity or bias to the analysis. The experiment should preferably compare algorithms from different families, considering both combinatorial and continuous optimization problems. Also, optimization problems should cover different complexities regarding its mathematical formulation and dimension, so those benchmarks with configurable objectives are preferred.

Looking at the outcomes of C1F2 (see Section 4), we found that only two algorithms are available in the majority of MOFs: NSGA-II, implemented by all MOFs; and SPEA2, only missing in HeuristicLab. Using these two algorithms would not allow us to cover a variety of families, because they both belong to the second generation. We also observe that some MOFs —particularly those with a more specialized scope— share some other algorithms in their catalog that could serve our purposes. For these reasons, we take the decision of comparing MOFs in two groups, as show in Table 18. Group #1 is comprised of DEAP, ECJ, EvA, HeuristicLab, Opt4J and ParadisEO-MOEO, for which we will study NSGA-II and, when possible, SPEA2. JCLEC-MO, jMetal, MOEA Framework and Platypus constitute Group #2, for which algorithms from three additional families —indicator-based, decomposition and relaxed dominance with PSO— are available for comparison.

Each selected algorithm is combined with several benchmarks (see Table 19). As a result, we increase the number of samples by changing the type of problem, i.e., continuous and combinatorial, and the number of objectives. We presume that domain experts would be more interested in problems with two to six objectives, leaving highly-dimensional problems for the second experiment. Table 18 details the benchmarks to be used, where the figure in parentheses indicates the number of objectives. The symbol $\ast$ indicates that the corresponding benchmarks has been implemented in the MOD. The number of bechmarks for Group #1 is greater to compensate the lack of algorithms. DTLZ and ZDT are continuous benchmarks and available in the majority of MOFs (see C4F2 in Section 7). The implementation of DTLZ was required for ECJ and EvA. ZDT was also developed for EvA. Regarding the problems, combinatorial problems are less common, and only MOEA Framework include both LOTZ and KP, while JCLEC-MO implements KP. Even so, we consider these problems because of their popularity. Furthermore, implementing them in the rest of MOFs is straightforward. LOTZ is a good introductory example to metaheuristics because it has a simple formulation (it just counts the numbers of zeros and ones in the genotype), and it is frequently used to evaluate search convergence. The knapsack problem is an abstraction of multiple problems that appear in the real-world. Since it is already available as a single-objective benchmark in some MOFs, its adaptation to the multi-objective formulation was just required.

As for the configuration of the algorithms selected for the experiment, Table 20 shows the list of parameters and their values, including genetic operators. Since the purpose of the experiment does not concern how well algorithms solve an optimization problem, standard values for general parameters like the population size and operator probabilities are considered. Crossover and mutation operators are chosen depending on the required

Table 19: Experiment #1: selected benchmarks.

Benchmark	**Description**	**Reference**
DTLZ1	Minimize a variable number of functions. The problem has a linear Pareto front (PF).	[9]
DTLZ2	Minimize a variable number of functions. The PF is the first quadrant of a sphere.	[9]
ZDT1	Minimize two functions. The PF is convex.	[137]
ZDT4	Minimize two functions. The problem is multi-modal, i.e. it has local PFs.	[137]
ZDT6	Minimize two functions. PF with non-uniformly distributed solutions.	[137]
LOTZ	Maximize the number of leading '1' and and trailing '0' in a bit array.	[138]
KP	Find the subset of items that maximizes the profits of k knapsacks.	[139]

encoding among the most commonly available (see C1F3 in Section 4). Only a few developments were needed: SBX for HeuristicLab and JCLEC-MO, one point crossover for Platypus, and UMP for HeuristicLab, EvA and ParadisEO-MOEO. As for the specific parameters, default values provided by the original authors are applied. After the analysis of the available variants, IBEA is executed using hypervolume as indicator, whereas MOEA/D applies the Tchebycheff approach as evaluation mechanism.

10.1.2 Results for execution time

Tables 21 and 22 show the execution time in seconds for Group #1 and #2, respectively. Each cell contains the mean and the standard deviation of the five runs. The figure in parentheses indicates the number of objectives.

10.1.3 Results for memory consumption

Table 23 shows the minimum and maximum RAM consumption for each configuration of Experiment #1 for the MOFs comprising Group#1. Values are expressed in KB and might correspond to different executions of the algorithm and benchmark. Again, the figure in parentheses indicates the number of objectives. Table 24 contains the same information but for Group #2.

Table 20: Experiment #1: parameter set-up.

General parameters	
Population/Swarm size	100
No. of generations	100
Crossover probability	0.9
Non-uniform mutation probability	0.1
Uniform mutation probability	1/genotype-length
Genetic operators	
Crossover operator	1PX (binary), SBX (double)
Mutation operator	UBFM (binary), UPM (double)
SPEA2 parameters	
Population size (P)	50
Archive size (A)	50
Parent selector	Binary tournament
k-neighbour	$\sqrt[2]{P+A}$
IBEA parameters	
Parent selector	Binary tournament
Fitness indicator	Hypervolume
Scaling factor (κ)	0.05
Reference point (ρ)	2
MOEA/D parameters	
Neighbourhood size (τ)	10
Max. No. of replacements (nr)	2
Evaluation function	Tchebycheff
Weights	Uniformly generated
OMOPSO parameters	
Archive size	100
Non uniform mutation probability	0.1
Uniform mutation probability	1/particle-length
ϵ-dominance (ϵ values)	0.0075

Table 21: Experiment #1: execution time for MOFs in Group #1.

Configuration	DEAP	ECJ	EvA	Heuristic Lab	Opt4J	ParadisEO MOEO
NSGA-II – LOTZ(2)	0.778 ± 0.010	0.245 ± 0.014	1.443 ± 0.037	1.546 ± 0.012	0.773 ± 0.050	0.757 ± 0.001
NSGA-II – ZDT1(2)	3.417 ± 0.073	0.299 ± 0.015	1.827 ± 0.019	1.756 ± 0.020	0.785 ± 0.056	0.900 ± 0.008
NSGA-II – ZDT4(2)	4.384 ± 1.112	0.295 ± 0.017	1.537 ± 0.014	1.694 ± 0.026	0.765 ± 0.068	0.937 ± 0.013
NSGA-II – ZDT6(2)	3.493 ± 0.027	0.284 ± 0.034	1.466 ± 0.023	1.733 ± 0.028	0.758 ± 0.058	0.764 ± 0.017
NSGA-II – KP(4)	8.460 ± 0.077	4.879 ± 0.022	1.587 ± 0.041	1.618 ± 0.023	1.142 ± 0.076	1.069 ± 0.017
NSGA-II – DTLZ1(6)	3.954 ± 0.030	0.344 ± 0.021	1.658 ± 0.043	1.762 ± 0.022	0.896 ± 0.067	2.456 ± 0.013
NSGA-II – DTLZ2(6)	3.908 ± 0.018	0.336 ± 0.022	1.803 ± 0.040	1.761 ± 0.031	0.900 ± 0.043	1.873 ± 0.022
NSGA-II – DTLZ4(6)	3.871 ± 0.061	0.351 ± 0.034	1.834 ± 0.039	1.768 ± 0.012	0.922 ± 0.110	1.843 ± 0.003
SPEA2 – LOTZ(2)	8.308 ± 0.406	0.444 ± 0.017	1.445 ± 0.022	–	0.609 ± 0.038	0.948 ± 0.009
SPEA2 – ZDT1(2)	6.522 ± 0.080	0.368 ± 0.028	2.684 ± 0.032	–	0.999 ± 0.087	0.379 ± 0.021
SPEA2 – ZDT4(2)	55.746 ± 0.298	0.291 ± 0.022	1.380 ± 0.029	–	1.020 ± 0.060	0.343 ± 0.014
SPEA2 – ZDT6(2)	6.534 ± 0.045	0.300 ± 0.021	1.385 ± 0.047	–	0.735 ± 0.032	0.360 ± 0.010
SPEA2 – KP(4)	26.518 ± 1.776	3.219 ± 0.033	1.736 ± 0.105	–	1.173 ± 0.065	0.339 ± 0.013
SPEA2 – DTLZ1(6)	10.373 ± 1.180	0.959 ± 0.050	1.646 ± 0.107	–	1.633 ± 0.236	0.778 ± 0.009
SPEA2 – DTLZ2(6)	37.392 ± 1.928	1.442 ± 0.023	2.702 ± 0.038	–	1.583 ± 0.070	0.892 ± 0.004
SPEA2 – DTLZ4(6)	41.509 ± 2.884	1.435 ± 0.088	2.897 ± 0.059	–	1.699 ± 0.103	0.878 ± 0.018

Table 22: Experiment #1: execution time for MOFs in Group #2.

Configuration	**JCLEC-MO**	**jMetal**	**MOEA Framework**	**Platypus**
NSGA-II – LOTZ(2)	0.410 ± 0.009	0.624 ± 0.023	0.328 ± 0.018	7.135 ± 0.024
NSGA-II – KP(4)	0.519 ± 0.027	0.558 ± 0.042	0.374 ± 0.016	103.746 ± 0.433
NSGA-II – DTLZ1(6)	0.537 ± 0.028	0.540 ± 0.066	0.444 ± 0.011	144.508 ± 2.958
NSGA-II – DTLZ2(6)	0.587 ± 0.064	0.595 ± 0.055	0.389 ± 0.008	132.329 ± 1.151
IBEA – LOTZ(2)	4.754 ± 0.037	1.288 ± 0.039	0.844 ± 0.019	39.217 ± 0.199
IBEA – KP(4)	3.012 ± 0.036	1.678 ± 0.142	1.036 ± 0.029	23.502 ± 0.122
IBEA – DTLZ1(6)	1.829 ± 0.030	2.556 ± 0.034	1.254 ± 0.011	8.172 ± 0.054
IBEA – DTLZ2(6)	1.843 ± 0.018	2.538 ± 0.032	1.285 ± 0.007	8.045 ± 0.094
MOEA/D – ZDT1(2)	0.429 ± 0.018	0.337 ± 0.020	0.458 ± 0.008	6.734 ± 0.031
MOEA/D – ZDT4(2)	0.413 ± 0.030	0.277 ± 0.010	0.414 ± 0.010	5.065 ± 0.032
MOEA/D – DTLZ1(2)	0.380 ± 0.022	0.260 ± 0.013	0.434 ± 0.023	4.691 ± 0.017
MOEA/D – DTLZ2(2)	0.492 ± 0.035	0.236 ± 0.004	0.412 ± 0.016	5.191 ± 0.040
OMOPSO – ZDT1(2)	0.556 ± 0.034	0.412 ± 0.050	1.269 ± 0.040	5.440 ± 0.139
OMOPSO – ZDT4(2)	0.530 ± 0.043	0.284 ± 0.010	1.117 ± 0.032	2.919 ± 0.461
OMOPSO – DTLZ1(2)	0.663 ± 0.009	0.484 ± 0.042	1.510 ± 0.016	19.619 ± 1.251
OMOPSO – DTLZ2(2)	0.715 ± 0.035	1.466 ± 0.086	2.139 ± 0.072	61.989 ± 2.557

Table 23: Experiment #1: memory consumption (KB) for MOFs in Group #1.

Configuration	**DEAP**		**ECJ**		**EvA**		**Heuristic Lab**		**Opt4J**		**ParadisEO MOEO**	
	Min.	Max.	Min.	Max.	Min.	Max.	Min.	Max.	Min.	Max.	Min.	Max.
NSGA-II – LOTZ(2)	5,148	20,512	12,976	43,840	13,100	55,980	6,828	75,408	13,120	122,028	1,480	1,544
NSGA-II – ZDT1(2)	4,804	20,484	12,964	55,220	13,152	56,264	6,812	82,872	12,648	116,280	3,112	3,260
NSGA-II – ZDT4(2)	5,172	20,508	12,968	58,700	13,088	58,264	6,800	80,544	12,532	117,308	1,528	3,504
NSGA-II – ZDT6(2)	5,048	20,668	12,908	52,220	13,060	56,976	6,856	78,240	13,116	112,492	3,312	3,552
NSGA-II – KP(4)	5,128	20,640	12,912	864,748	13,040	57,220	6,808	82,224	13,044	138,840	1,456	3,076
NSGA-II – DTLZ1(6)	5,112	20,448	12,992	53,776	12,480	57,076	6,796	79,008	12,544	141,812	1,460	3,440
NSGA-II – DTLZ2(6)	5,196	20,608	13,044	54,100	12,560	54,464	6,168	79,364	12,368	144,196	1,460	3,500
NSGA-II – DTLZ4(6)	5,180	20,600	12,980	57,816	13,136	55,064	6,828	78,376	12,556	139,164	3,272	3,436
SPEA2 – LOTZ(2)	5,204	20,776	12,904	77,236	13,024	55,576	-	-	13,072	95,784	2,984	3,188
SPEA2 – ZDT1(2)	5,092	20,604	12,836	65,568	13,048	60,464	-	-	13,076	123,372	3,392	3,748
SPEA2 – ZDT4(2)	5,092	21,420	12,740	58,024	13,100	54,204	-	-	13,044	125,296	3,336	3,800
SPEA2 – ZDT6(2)	5,168	20,128	12,776	56,516	13,128	54,996	-	-	13,084	96,800	3,732	3,808
SPEA2 – KP(4)	5,180	21,240	12,772	553,832	13,072	54,820	-	-	13,136	122,816	2,872	3,000
SPEA2 – DTLZ1(6)	5,180	21,040	12,876	111,444	13,040	54,872	-	-	13,044	141,320	3,320	3,648
SPEA2 – DTLZ2(6)	5,176	21,308	12,804	109,312	13,060	59,100	-	-	13,096	135,536	3,340	3,740
SPEA2 – DTLZ4(6)	5,176	21,248	12,852	115,880	13,064	58,008	-	-	13,036	144,292	3,324	3,840

Table 24: Experiment #1: memory consumption (KB) for MOFs in Group #2.

Configuration	**JCLEC-MO**		**jMetal**		**MOEA Framework**		**Platypus**	
	Min.	Max.	Min.	Max.	Min.	Max.	Min.	Max.
NSGA-II – LOTZ(2)	13,004	76,320	13,032	98,624	13,000	51,948	5,144	23,612
NSGA-II – KP(4)	13,108	81,224	13,008	75,552	13,124	57,696	5,096	28,732
NSGA-II – DTLZ1(6)	13,136	71,088	13,084	71,904	13,048	68,676	5,192	22,416
NSGA-II – DTLZ2(6)	13,100	69,248	13,048	73,060	13,048	68,780	5,136	22,468
IBEA – LOTZ(2)	13,148	105,372	13,032	178,892	13,036	101,892	5,096	24,932
IBEA – KP(4)	13,136	105,692	13,048	119,132	12,604	105,484	5,092	30,316
IBEA – DTLZ1(6)	13,188	104,764	13,116	105,608	13,052	100,640	5,048	23,804
IBEA – DTLZ2(6)	13,048	105,652	13,012	107,304	13,048	100,812	5,172	23,768
MOEA/D – ZDT1(2)	13,124	63,380	13,048	81,088	12,928	101,576	5,060	21,932
MOEA/D – ZDT4(2)	13,180	57,532	13,008	60,956	12,808	80,808	5,092	22,048
MOEA/D – DTLZ1(2)	13,076	55,076	13,112	57,756	12,516	73,016	5,096	22,064
MOEA/D – DTLZ2(2)	13,128	57,072	13,076	57,348	12,804	74,424	5,136	22,064
OMOPSO – ZDT1(2)	13,140	67,780	13,040	89,204	12,828	109,084	5,172	23,004
OMOPSO – ZDT4(2)	13,060	60,680	12,980	48,816	12,884	107,668	5,048	22,780
OMOPSO – DTLZ1(2)	13,044	70,108	13,032	76,008	12,740	109,792	5,180	23,740
OMOPSO – DTLZ2(2)	13,096	67,080	13,120	111,968	12,760	111,784	5,092	27,720

10.2 Experiment #2: scalability study

10.2.1 Experimental design

This experiment is intended to analyze how MOFs manage runtime resources when more complex configurations are demanded. Therefore, this experiment is more oriented to researchers who are interested in comparing the performance of algorithms under different parameter settings or in solving highly-dimensional problems. Under these scenarios, it is important to find out how well each MOF scales. Scalability is tested in terms of three parameters: population size (100, 500, 1000), number of generations (100, 500, 1000, 5000) and number of objectives (2, 5, 10, 25, 50), resulting in 60 combinations in total. Again, our interest is not to study which algorithm performs best, but how these parameters affect the use of memory and the way in which execution time increases. As a way to reduce the influence of any other aspect, an algorithm without parameters and available in all MOFs, NSGA-II, and a benchmark with configurable objectives, DTLZ1, were selected. Genetic operators and their parameters are the same than those used in the previous experiment (see Table 20).

10.2.2 Results for execution time

Tables 25 to 30 show the execution time in seconds for each value of the population size: 100, 500 and 1000. The first column contains the configuration of the rest of parameters, where G stands for the number of generations and O indicates the number of objectives. The mean and standard deviation of the six executions are shown.

10.2.3 Results for memory consumption

Table 31 summarizes the minimum and maximum memory consumption in KB for those configurations with 100 and 500 individuals. The MOF providing such value is included. Equivalent results for a population size equals to 1000 can be found in Table 32. In both tables, P represents the population size, G is the number of generations and O stands for the number of objectives.

Table 25: Experiment #2: execution time of DEAP, ECJ, HeuristicLab, EvA, and JCLEC-MO (pop. size = 100)

Configuration	**DEAP**	**ECJ**	**EvA**	**Heuristic Lab**	**JCLEC-MO**
G(100) − O(2)	3.55 ± 0.12	0.34 ± 0.04	1.79 ± 0.02	1.84 ± 0.13	0.65 ± 0.15
G(100) − O(5)	3.78 ± 0.04	0.34 ± 0.02	1.93 ± 0.05	1.85 ± 0.12	0.64 ± 0.07
G(100) − O(10)	4.77 ± 0.06	0.46 ± 0.05	2.26 ± 0.08	1.92 ± 0.11	0.73 ± 0.10
G(100) − O(25)	7.92 ± 0.17	0.52 ± 0.05	2.99 ± 0.09	2.20 ± 0.11	0.90 ± 0.09
G(100) − O(50)	13.23 ± 0.20	0.73 ± 0.06	4.27 ± 0.17	2.70 ± 0.12	1.30 ± 0.10
G(500) − O(2)	17.64 ± 0.17	0.71 ± 0.08	6.97 ± 0.06	6.46 ± 0.15	1.32 ± 0.17
G(500) − O(5)	18.67 ± 0.24	0.77 ± 0.06	7.85 ± 0.13	6.75 ± 0.16	1.37 ± 0.14
G(500) − O(10)	23.66 ± 0.16	0.93 ± 0.09	9.50 ± 0.20	6.84 ± 0.15	1.73 ± 0.14
G(500) − O(25)	39.22 ± 0.35	1.32 ± 0.17	14.76 ± 0.22	8.95 ± 1.43	2.71 ± 0.34
G(500) − O(50)	66.94 ± 0.50	1.88 ± 0.14	22.00 ± 0.77	11.15 ± 0.20	4.29 ± 0.17
G(1000) − O(2)	35.17 ± 0.42	1.16 ± 0.15	13.79 ± 0.14	12.20 ± 0.29	1.96 ± 0.21
G(1000) − O(5)	36.55 ± 0.30	1.26 ± 0.16	15.34 ± 0.14	13.24 ± 1.22	2.17 ± 0.23
G(1000) − O(10)	47.00 ± 0.73	1.69 ± 0.11	18.66 ± 0.26	13.92 ± 1.37	2.82 ± 0.18
G(1000) − O(25)	75.74 ± 1.07	2.38 ± 0.25	29.60 ± 0.43	17.68 ± 3.50	4.38 ± 0.22
G(1000) − O(50)	131.53 ± 1.27	3.50 ± 0.33	44.70 ± 1.44	23.44 ± 2.49	8.55 ± 1.08
G(5000) − O(2)	165.18 ± 3.60	3.25 ± 0.24	52.55 ± 0.23	56.28 ± 1.01	6.27 ± 0.19
G(5000) − O(5)	177.96 ± 3.55	3.83 ± 0.20	60.24 ± 0.34	57.96 ± 0.95	7.63 ± 0.39
G(5000) − O(10)	221.57 ± 4.60	5.55 ± 0.49	79.42 ± 0.92	60.80 ± 1.56	10.51 ± 0.29
G(5000) − O(25)	351.09 ± 6.99	8.90 ± 0.41	144.39 ± 1.09	75.46 ± 0.88	17.16 ± 0.56
G(5000) − O(50)	588.13 ± 8.28	14.37 ± 0.92	226.77 ± 2.25	106.45 ± 0.91	30.55 ± 0.58

Table 26: Experiment #2: execution time of jMetal, MOEA Framework, Opt4J, ParadisEO-MOEO and Platypus (pop. size = 100).

Configuration	**jMetal**	**MOEA Framework**	**Opt4J**	**ParadisEO MOEO**	**Platypus**
G(100) − O(2)	0.58 ± 0.06	0.37 ± 0.01	0.93 ± 0.08	0.91 ± 0.01	5.23 ± 0.03
G(100) − O(5)	0.64 ± 0.06	0.41 ± 0.02	0.98 ± 0.06	2.13 ± 0.01	7.43 ± 0.12
G(100) − O(10)	0.73 ± 0.10	0.47 ± 0.04	1.12 ± 0.13	3.68 ± 0.01	11.58 ± 0.20
G(100) − O(25)	0.84 ± 0.05	0.59 ± 0.02	1.34 ± 0.11	10.85 ± 0.07	23.43 ± 0.46
G(100) − O(50)	1.08 ± 0.11	0.76 ± 0.03	1.52 ± 0.12	23.82 ± 0.27	46.30 ± 1.68
G(500) − O(2)	1.35 ± 0.16	0.78 ± 0.04	1.76 ± 0.18	4.86 ± 0.02	24.54 ± 0.13
G(500) − O(5)	1.34 ± 0.12	0.84 ± 0.04	1.94 ± 0.13	9.92 ± 0.04	36.91 ± 0.61
G(500) − O(10)	1.57 ± 0.17	1.10 ± 0.04	2.38 ± 0.20	17.11 ± 0.11	58.53 ± 0.41
G(500) − O(25)	2.21 ± 0.23	1.58 ± 0.03	3.13 ± 0.20	53.67 ± 0.27	113.09 ± 2.74
G(500) − O(50)	3.28 ± 0.19	2.36 ± 0.03	5.10 ± 0.90	77.07 ± 0.14	208.13 ± 4.02
G(1000) − O(2)	1.99 ± 0.31	1.22 ± 0.03	2.66 ± 0.17	9.41 ± 0.16	48.95 ± 0.31
G(1000) − O(5)	2.22 ± 0.29	1.35 ± 0.07	2.97 ± 0.27	17.83 ± 0.21	71.95 ± 0.81
G(1000) − O(10)	2.38 ± 0.17	1.80 ± 0.02	3.74 ± 0.26	29.48 ± 0.12	114.64 ± 2.82
G(1000) − O(25)	3.86 ± 0.24	2.84 ± 0.05	5.31 ± 0.13	97.26 ± 0.22	219.09 ± 3.57
G(1000) − O(50)	6.05 ± 0.24	4.27 ± 0.05	8.68 ± 0.12	151.38 ± 0.27	395.62 ± 7.48
G(5000) − O(2)	5.45 ± 0.34	4.63 ± 0.06	9.28 ± 1.22	35.01 ± 1.09	237.23 ± 2.60
G(5000) − O(5)	6.58 ± 0.28	5.03 ± 0.07	11.11 ± 1.27	73.55 ± 0.30	340.72 ± 3.96
G(5000) − O(10)	9.38 ± 0.73	7.26 ± 0.10	13.18 ± 0.16	128.77 ± 0.54	510.07 ± 17.78
G(5000) − O(25)	17.59 ± 0.42	11.98 ± 0.05	22.12 ± 0.13	208.66 ± 1.22	894.50 ± 18.84
G(5000) − O(50)	30.74 ± 1.21	19.26 ± 0.21	39.18 ± 0.68	184.86 ± 0.30	$1,437.75 \pm 38.90$

Table 27: Experiment #2: execution time of DEAP, ECJ, EvA, HeuristicLab and JCLEC-MO (pop. size = 500).

Configuration	DEAP	ECJ	EvA	Heuristic Lab	JCLEC-MO
G(100) − O(2)	75.36 ± 0.84	0.79 ± 0.05	5.74 ± 0.62	2.65 ± 0.02	3.25 ± 0.10
G(100) − O(5)	78.96 ± 0.63	1.23 ± 0.03	7.68 ± 0.20	2.78 ± 0.09	4.66 ± 0.67
G(100) − O(10)	99.71 ± 0.38	2.08 ± 0.26	12.23 ± 0.32	3.14 ± 0.10	5.46 ± 0.18
G(100) − O(25)	159.05 ± 1.58	2.96 ± 0.06	27.15 ± 1.69	4.39 ± 0.04	8.01 ± 0.10
G(100) − O(50)	264.86 ± 7.66	4.29 ± 0.15	64.06 ± 3.07	6.86 ± 0.18	12.80 ± 0.15
G(500) − O(2)	762.77 ± 18.81	3.00 ± 0.07	27.36 ± 0.65	10.54 ± 0.17	15.05 ± 0.22
G(500) − O(5)	788.49 ± 6.59	5.07 ± 0.04	40.66 ± 0.61	11.34 ± 0.18	17.92 ± 0.57
G(500) − O(10)	983.17 ± 7.78	8.45 ± 0.30	65.24 ± 0.92	12.54 ± 0.11	24.37 ± 0.22
G(500) − O(25)	$1,570.28 \pm 32.17$	12.81 ± 0.17	162.12 ± 3.94	18.29 ± 0.19	36.38 ± 0.43
G(500) − O(50)	$2,539.69 \pm 36.32$	18.74 ± 0.18	357.63 ± 19.87	30.28 ± 3.78	57.18 ± 0.44
G(1000) − O(2)	$3,269.87 \pm 94.70$	6.15 ± 0.92	55.46 ± 0.55	20.96 ± 1.11	29.34 ± 0.07
G(1000) − O(5)	$4,186.92 \pm 78.98$	10.10 ± 0.09	88.85 ± 3.79	22.23 ± 0.24	34.99 ± 0.37
G(1000) − O(10)	$4,809.16 \pm 48.87$	17.48 ± 1.22	134.56 ± 4.23	24.03 ± 0.30	46.04 ± 0.45
G(1000) − O(25)	389.02 ± 2.96	24.74 ± 0.34	339.99 ± 6.73	36.19 ± 0.46	70.63 ± 0.62
G(1000) − O(50)	$7,446.58 \pm 131.93$	36.87 ± 2.74	760.58 ± 16.51	58.12 ± 0.85	109.89 ± 0.98
G(5000) − O(2)	$11,708.99 \pm 100.31$	29.98 ± 0.39	280.61 ± 3.91	94.92 ± 1.25	142.52 ± 0.88
G(5000) − O(5)	391.40 ± 5.34	50.83 ± 0.31	443.64 ± 4.31	107.21 ± 0.93	185.58 ± 1.37
G(5000) − O(10)	501.88 ± 3.27	80.25 ± 0.72	647.77 ± 12.00	112.24 ± 1.85	217.77 ± 1.58
G(5000) − O(25)	780.57 ± 8.90	120.45 ± 1.87	$1,675.01 \pm 61.64$	170.88 ± 2.41	321.95 ± 1.24
G(5000) − O(50)	$1,292.58 \pm 32.87$	170.61 ± 2.44	$3,830.01 \pm 94.69$	311.71 ± 38.41	475.77 ± 4.83

Table 28: Experiment #2: execution time of jMetal, MOEA Framework, Opt4J, ParadisEO-MOEO and Playtpus (pop. size = 500)

Configuration	**jMetal**	**MOEA Framework**	**Opt4J**	**ParadisEO MOEO**	**Platypus**
G(100) − O(2)	2.64 ± 0.07	1.43 ± 0.08	4.36 ± 0.61	20.61 ± 0.27	84.25 ± 1.09
G(100) − O(5)	2.83 ± 0.08	1.93 ± 0.08	4.12 ± 0.09	54.17 ± 0.19	121.22 ± 2.41
G(100) − O(10)	3.69 ± 0.09	3.27 ± 0.06	5.38 ± 0.06	112.00 ± 1.26	218.11 ± 6.39
G(100) − O(25)	6.59 ± 0.07	5.12 ± 0.09	7.67 ± 0.06	446.82 ± 2.18	431.29 ± 9.00
G(100) − O(50)	10.89 ± 0.18	7.69 ± 0.03	11.12 ± 0.17	914.59 ± 8.13	785.53 ± 10.01
G(500) − O(2)	10.95 ± 0.20	8.37 ± 1.86	17.94 ± 0.31	123.24 ± 2.40	771.43 ± 8.38
G(500) − O(5)	11.62 ± 0.17	7.83 ± 0.11	19.57 ± 0.33	232.67 ± 0.70	$1,081.11 \pm 27.04$
G(500) − O(10)	16.55 ± 0.19	14.90 ± 0.20	24.49 ± 1.96	464.23 ± 15.61	$2,011.81 \pm 57.77$
G(500) − O(25)	31.61 ± 0.30	23.55 ± 0.14	37.19 ± 4.69	$1,560.58 \pm 42.20$	$3,711.54 \pm 116.69$
G(500) − O(50)	54.09 ± 0.38	35.45 ± 0.16	52.97 ± 0.34	$2,950.80 \pm 12.87$	$6,204.79 \pm 81.14$
G(1000) − O(2)	20.87 ± 0.47	17.69 ± 4.14	36.01 ± 1.88	206.90 ± 3.09	$3,950.62 \pm 126.50$
G(1000) − O(5)	24.53 ± 3.29	16.03 ± 0.41	39.38 ± 0.49	416.42 ± 3.25	$5,440.04 \pm 76.96$
G(1000) − O(10)	32.60 ± 0.31	28.87 ± 0.49	46.39 ± 0.71	818.98 ± 2.03	$9,606.06 \pm 167.61$
G(1000) − O(25)	62.96 ± 0.73	46.25 ± 0.58	68.13 ± 0.35	$2,827.62 \pm 31.59$	392.19 ± 4.99
G(1000) − O(50)	110.04 ± 2.13	69.39 ± 0.25	107.21 ± 0.63	$4,581.26 \pm 26.46$	$16,188.45 \pm 381.67$
G(5000) − O(2)	101.94 ± 4.16	88.29 ± 4.38	170.20 ± 2.18	898.77 ± 12.75	$24,692.45 \pm 923.18$
G(5000) − O(5)	115.16 ± 1.73	84.89 ± 0.99	191.18 ± 2.86	$1,805.80 \pm 9.78$	576.07 ± 17.31
G(5000) − O(10)	160.63 ± 1.13	135.71 ± 2.87	215.03 ± 3.42	$3,743.38 \pm 63.83$	$1,045.97 \pm 20.44$
G(5000) − O(25)	325.92 ± 3.63	240.90 ± 24.27	330.67 ± 5.99	$13,313.05 \pm 412.57$	$1,937.42 \pm 22.07$
G(5000) − O(50)	576.81 ± 3.86	339.96 ± 1.59	533.96 ± 6.95	$19,352.82 \pm 190.56$	$3,240.30 \pm 55.02$

Table 29: Experiment #2: execution time of DEAP, ECJ, EvA, HeuristicLab and JCLEC-MO (pop. size = 1,000).

Configuration	**DEAP**	**ECJ**	**EvA**	**Heuristic Lab**	**JCLEC-MO**
G(100) – O(2)	298.55 ± 5.00	2.73 ± 0.09	16.38 ± 0.49	4.52 ± 0.05	11.86 ± 0.07
G(100) – O(5)	305.74 ± 5.45	3.40 ± 0.09	22.80 ± 0.80	4.88 ± 0.06	15.24 ± 0.15
G(100) – O(10)	393.34 ± 4.14	6.12 ± 0.04	43.79 ± 2.09	6.22 ± 0.06	19.63 ± 0.09
G(100) – O(25)	631.06 ± 5.73	9.52 ± 0.04	107.96 ± 10.14	10.33 ± 0.25	29.93 ± 0.14
G(100) – O(50)	$1,089.98 \pm 18.38$	14.27 ± 0.39	255.63 ± 6.18	18.29 ± 0.72	47.95 ± 0.20
G(500) – O(2)	$1,529.38 \pm 21.80$	18.57 ± 0.20	88.18 ± 2.86	19.63 ± 0.21	61.03 ± 0.43
G(500) – O(5)	$1,554.52 \pm 24.31$	18.81 ± 0.12	139.51 ± 2.59	21.55 ± 0.14	73.29 ± 0.24
G(500) – O(10)	$1,969.26 \pm 50.70$	30.71 ± 0.30	242.01 ± 3.05	25.45 ± 0.37	87.86 ± 0.59
G(500) – O(25)	$3,065.23 \pm 31.35$	44.49 ± 0.33	656.28 ± 12.50	43.41 ± 0.61	136.27 ± 3.32
G(500) – O(50)	$5,180.26 \pm 49.56$	64.63 ± 0.81	$1,492.61 \pm 94.18$	74.65 ± 1.72	204.29 ± 3.16
G(1000) – O(2)	$2,865.92 \pm 84.36$	38.93 ± 1.03	177.86 ± 1.04	37.38 ± 0.41	121.94 ± 0.47
G(1000) – O(5)	$3,264.83 \pm 25.59$	39.00 ± 0.92	283.31 ± 8.82	43.35 ± 0.42	149.19 ± 0.80
G(1000) – O(10)	$3,861.10 \pm 78.95$	61.73 ± 0.41	494.61 ± 6.88	48.50 ± 0.49	173.10 ± 1.03
G(1000) – O(25)	$6,114.82 \pm 31.48$	87.30 ± 0.38	$1,343.71 \pm 29.88$	82.91 ± 1.39	258.02 ± 2.67
G(1000) – O(50)	$10,275.47 \pm 87.46$	123.70 ± 0.76	$3,064.50 \pm 44.83$	140.11 ± 3.24	370.16 ± 1.62
G(5000) – O(2)	$12,592.69 \pm 229.07$	197.24 ± 4.65	918.54 ± 5.55	179.62 ± 1.62	607.98 ± 2.52
G(5000) – O(5)	$17,221.06 \pm 323.23$	202.35 ± 1.63	$1,507.15 \pm 42.84$	214.99 ± 1.62	765.17 ± 6.18
G(5000) – O(10)	$18,972.78 \pm 158.16$	311.18 ± 0.01	$2,524.16 \pm 43.36$	226.39 ± 1.93	866.93 ± 3.96
G(5000) – O(25)	$29,254.01 \pm 339.15$	435.32 ± 2.59	$6,933.29 \pm 115.55$	391.85 ± 2.53	$1,228.53 \pm 91.90$
G(5000) – O(50)	$48,052.73 \pm 1,023.74$	589.33 ± 3.58	$16,030.96 \pm 512.88$	680.33 ± 2.98	$1,611.35 \pm 9.21$

Table 30: Experiment #2: execution time of jMetal, MOEA Framework, Opt4J, ParadisEO-MOEO and Playtpus (pop. size = 1,000)

Configuration	jMetal	MOEA Framework	Opt4J	ParadisEO MOEO	Platypus
G(100) − O(2)	9.01 ± 0.17	4.27 ± 0.14	13.98 ± 0.31	82.06 ± 1.42	314.05 ± 2.75
G(100) − O(5)	9.30 ± 0.20	5.90 ± 0.08	13.95 ± 0.34	218.51 ± 2.49	434.26 ± 7.44
G(100) − O(10)	12.52 ± 0.18	12.11 ± 0.18	18.59 ± 0.33	480.84 ± 1.71	820.76 ± 11.80
G(100) − O(25)	23.83 ± 0.24	19.05 ± 0.21	25.39 ± 0.24	$1,541.10 \pm 21.53$	$1,782.19 \pm 35.48$
G(100) − O(50)	39.93 ± 0.36	27.98 ± 0.32	36.14 ± 0.51	$3,203.83 \pm 30.23$	$3,085.28 \pm 71.70$
G(500) − O(2)	42.96 ± 0.98	31.63 ± 0.98	70.68 ± 1.62	452.57 ± 27.69	$1,444.20 \pm 19.18$
G(500) − O(5)	46.03 ± 1.37	33.68 ± 5.55	75.72 ± 2.29	875.26 ± 17.32	$1,958.00 \pm 12.28$
G(500) − O(10)	61.42 ± 1.02	60.50 ± 1.04	87.41 ± 1.80	$2,119.48 \pm 8.33$	$3,960.86 \pm 104.85$
G(500) − O(25)	119.93 ± 1.82	91.49 ± 0.94	122.19 ± 1.36	$5,495.18 \pm 31.49$	$7,362.19 \pm 134.17$
G(500) − O(50)	210.16 ± 2.43	131.50 ± 1.07	177.34 ± 1.22	$9,643.98 \pm 51.63$	$11,895.61 \pm 239.56$
G(1000) − O(2)	84.37 ± 1.14	69.70 ± 2.20	143.42 ± 2.31	848.70 ± 14.62	$2,878.85 \pm 33.63$
G(1000) − O(5)	92.41 ± 0.90	69.32 ± 1.42	156.79 ± 4.30	$1,571.01 \pm 9.55$	$3,987.65 \pm 92.90$
G(1000) − O(10)	123.86 ± 2.33	128.27 ± 21.62	172.51 ± 2.58	$3,732.22 \pm 18.43$	$7,615.26 \pm 78.98$
G(1000) − O(25)	240.92 ± 3.70	180.18 ± 1.23	240.89 ± 1.30	$9,844.00 \pm 88.16$	$13,775.35 \pm 327.26$
G(1000) − O(50)	423.16 ± 4.33	260.39 ± 1.79	354.04 ± 2.34	$16,393.57 \pm 128.38$	$22,528.93 \pm 333.49$
G(5000) − O(2)	424.83 ± 12.19	376.79 ± 7.63	695.60 ± 21.77	$3,439.66 \pm 56.18$	$14,546.53 \pm 105.73$
G(5000) − O(5)	478.63 ± 3.37	367.36 ± 7.62	771.86 ± 20.96	$7,031.26 \pm 28.44$	$20.749.33 \pm 311.89$
G(5000) − O(10)	616.48 ± 11.43	573.23 ± 8.10	852.02 ± 14.97	$16,113.68 \pm 153.68$	$36,861.58 \pm 530.84$
G(5000) − O(25)	$1,253.40 \pm 20.79$	893.06 ± 7.24	$1,172.27 \pm 16.75$	$43,566.70 \pm 299.17$	$61,123.81 \pm 1,680.66$
G(5000) − O(50)	$2,237.61 \pm 30.34$	$1,284.00 \pm 7.17$	$1,743.74 \pm 12.82$	$69,172.09 \pm 716.24$	$91,748.88 \pm 1,583.78$

Table 31: Experiment #2: memory consumption (population size = 100, 500).

Configuration	Min. KB	MOF	Max. KB	MOF
P(100)–G(100)–O(2)	1,480	ParadisEO-MOEO	119,980	Opt4J
P(100)–G(100)–O(5)	1,460	ParadisEO-MOEO	129,056	Opt4J
P(100)–G(100)–O(10)	1,456	ParadisEO-MOEO	149,584	Opt4J
P(100)–G(100)–O(25)	1,460	ParadisEO-MOEO	150,956	Opt4J
P(100)–G(100)–O(50)	2,884	ParadisEO-MOEO	157,872	Opt4J
P(100)–G(500)–O(2)	1,480	ParadisEO-MOEO	143,432	Opt4J
P(100)–G(500)–O(5)	1,472	ParadisEO-MOEO	158,068	Opt4J
P(100)–G(500)–O(10)	1,456	ParadisEO-MOEO	160,340	Opt4J
P(100)–G(500)–O(25)	1,492	ParadisEO-MOEO	185,104	Opt4J
P(100)–G(500)–O(50)	2,996	ParadisEO-MOEO	312,256	Opt4J
P(100)–G(1000)–O(2)	1,480	ParadisEO-MOEO	146,132	Opt4J
P(100)–G(1000)–O(5)	1,456	ParadisEO-MOEO	163,236	Opt4J
P(100)–G(1000)–O(10)	1,460	ParadisEO-MOEO	156,908	Opt4J
P(100)–G(1000)–O(25)	1,504	ParadisEO-MOEO	181,644	Opt4J
P(100)–G(1000)–O(50)	2,912	ParadisEO-MOEO	314,792	Opt4J
P(100)–G(5000)–O(2)	1,500	ParadisEO-MOEO	306,840	jMetal
P(100)–G(5000)–O(5)	1,460	ParadisEO-MOEO	165,160	Opt4J
P(100)–G(5000)–O(10)	1,456	ParadisEO-MOEO	141,428	Opt4J
P(100)–G(5000)–O(25)	1,488	ParadisEO-MOEO	171,216	Opt4J
P(100)–G(5000)–O(50)	2,888	ParadisEO-MOEO	305,280	Opt4J
P(500)– G(100)–O(2)	2,976	ParadisEO-MOEO	269,368	jMetal
P(500)–G(100)–O(5)	2,984	ParadisEO-MOEO	128,440	Opt4J
P(500)–G(100)–O(10)	2,916	ParadisEO-MOEO	127,220	Opt4J
P(500)–G(100)–O(25)	3,304	ParadisEO-MOEO	130,940	Opt4J
P(500)–G(100)–O(50)	3,456	ParadisEO-MOEO	189,700	Opt4J
P(500)–G(500)–O(2)	2,996	ParadisEO-MOEO	711,400	jMetal
P(500)–G(500)–O(5)	2,888	ParadisEO-MOEO	171,988	HeuristicLab
P(500)–G(500)–O(10)	2,984	ParadisEO-MOEO	141,868	HeuristicLab
P(500)–G(500)–O(25)	3,324	ParadisEO-MOEO	132,636	Opt4J
P(500)–G(500)–O(50)	3,524	ParadisEO-MOEO	231,632	Opt4J
P(500)–G(1000)–O(2)	3,012	ParadisEO-MOEO	879,304	jMetal
P(500)–G(1000)–O(5)	2,916	ParadisEO-MOEO	192,668	HeuristicLab
P(500)–G(1000)–O(10)	2,984	ParadisEO-MOEO	150,048	HeuristicLab
P(500)–G(1000)–O(25)	3,320	ParadisEO-MOEO	190,800	HeuristicLab
P(500)–G(1000)–O(50)	3,440	ParadisEO-MOEO	317,708	Opt4J
P(500)–G(5000)–O(2)	2,984	ParadisEO-MOEO	1,095,336	jMetal
P(500)–G(5000)–O(5)	2,888	ParadisEO-MOEO	340,544	jMetal
P(500)–G(5000)–O(10)	2,928	ParadisEO-MOEO	157,456	Opt4J
P(500)–G(5000)–O(25)	3,304	ParadisEO-MOEO	329,696	Opt4J
P(500)–G(5000)–O(50)	3,444	ParadisEO-MOEO	361,980	Opt4J

Table 32: Experiment #2: memory consumption (population size = 1,000).

Configuration	Min. KB	MOF	Max. KB	MOF
P(1000)–G(100)–O(2)	2,932	ParadisEO-MOEO	842,128	jMetal
P(1000)–G(100)–O(5)	2,916	ParadisEO-MOEO	130,956	Opt4J
P(1000)–G(100)–O(10)	3,152	ParadisEO-MOEO	127,996	Opt4J
P(1000)–G(100)–O(25)	3,416	ParadisEO-MOEO	151,680	Opt4J
P(1000)–G(100)–O(50)	4,548	ParadisEO-MOEO	359,788	Opt4J
P(1000)–G(500)–O(2)	2,912	ParadisEO-MOEO	1,428,564	jMetal
P(1000)–G(500)–O(5)	2,912	ParadisEO-MOEO	344,696	jMetal
P(1000)–G(500)–O(10)	3,244	ParadisEO-MOEO	175,904	HeuristicLab
P(1000)–G(500)–O(25)	3,440	ParadisEO-MOEO	268,924	Opt4J
P(1000)–G(500)–O(50)	4,560	ParadisEO-MOEO	331,636	Opt4J
P(1000)–G(1000)–O(2)	2,888	ParadisEO-MOEO	1,448,548	jMetal
P(1000)–G(1000)–O(5)	3,016	ParadisEO-MOEO	352,900	jMetal
P(1000)–G(1000)–O(10)	3,196	ParadisEO-MOEO	168,180	HeuristicLab
P(1000)–G(1000)–O(25)	3,528	ParadisEO-MOEO	280,632	Opt4J
P(1000)–G(1000)–O(50)	4,528	ParadisEO-MOEO	349,564	Opt4J
P(1000)–G(5000)–O(2)	2,984	ParadisEO-MOEO	1,428,652	jMetal
P(1000)–G(5000)–O(5)	2,884	ParadisEO-MOEO	490,204	jMetal
P(1000)–G(5000)–O(10)	3,152	ParadisEO-MOEO	261,328	Opt4J
P(1000)–G(5000)–O(25)	3,540	ParadisEO-MOEO	264,836	Opt4J
P(1000)–G(5000)–O(50)	4,544	ParadisEO-MOEO	1,711,236	JCLEC-MO

Table 33: Summary of best frameworks for each feature.

Characteristic	**Feature**	**Best framework**
C1: search components and techniques	C1F1: type of metaheuristics	EvA
	C1F2: families of algorithms	jMetal
	C1F3: encodings and operators	HeuristicLab
C2: configuration	C2F1: inputs	EvA
	C2F2: batch processing	EvA, HeuristicLab, jMetal
	C2F3: outputs	MOEA Framework
C3: execution	C3F1: multi-thread execution	ECJ, Opt4J, ParadisEO-MOEO
	C3F2: distributed execution	ECJ, MOEA Framework
	C3F3: stop and restart mode	ECJ, HeuristicLab, MOEA Framework, Opt4J
	C3F4: fault recovery	ECJ, HeuristicLab, jMetal, MOEA Framework, ParadisEO-MOEO, Platypus
	C3F5: execution and control logs	ECJ, EvA, HeuristicLab, JCLEC-MO, jMetal, Platypus
C4: utilities	C4F1: graphical user interface	HeuristicLab
	C4F2: benchmarks	MOEA Framework
	C4F3: quality indicators	JCLEC-MO
C5: documentation and community support	C5F1: software license	See Table 14
	C5F2: available documentation	JCLEC-MO
	C5F3: software update	MOEA Framework
	C5F4: development facilities	Opt4J
	C5F5: community	HeuristicLab, Opt4J, ParadisEO-MOEO
C6: software implementation	C6F1: implementation and execution	DEAP, ECJ, EvA, JCLEC-MO, jMetal, MOEA Framework, Op4tJ, Platypus
	C6F2: external libraries	ECJ, JCLEC-MO, jMetal, MOEA Framework
	C6F3: software metrics	See Table 17
C7: performance at runtime	C7F1: execution time	ECJ
	C7F2: memory consumption	ParadisEO-MOEO

11 Summary

Table 33 compiles the frameworks that best support each feature under analysis. More specifically, the framework(s) satisfying more items of the corresponding checklists are listed in the last column. For some particular features, the procedure to count the number of options supported has been defined as follows:

- For C2F1a and C2F3a, +1 for each type of input/output format supported by the MOF (code, command line and file).
- For C2F1b and C2F3, +1 for each available format, regardless of its type.
- C3F4a is not considered to compute the total support of C3F4, as it is not defined as a list.
- C5F1 and C6F1a are not quantifiable as all MOFs have a license and a programming language, respectively.
- For C5F4a and C5F4b, +1 for each type of control version system and compilation/distribution mechanism, respectively.
- For C6F1b, +1 for each compatible SO.
- For C6F3, the reader is referred to the values of the different software metrics.

- C7F1 is based on the average ranking in Experiment #2 (see research paper).
- For C7F2, ParadisEO-MOEO is selected because its minimum and maximum memory consumption is always below the rest of frameworks.

The resulting table serves as a quick reference for the user to determine which MOF best suits his/her specific needs, depending on what aspects he/she wants to focus on the development.

References

[1] C. A. Coello Coello, G. B. Lamont, and D. A. Van Veldhuizen, *Evolutionary Algorithms for Solving Multi-Objective Problems*. Springer, 2nd ed., 2007.

[2] A. Zhou, B.-Y. Qu, H. Li, S.-Z. Zhao, P. N. Suganthan, and Q. Zhang, "Multi-objective evolutionary algorithms: A survey of the state of the art," *Swarm Evol. Comput.*, vol. 1, no. 1, pp. 32–49, 2011.

[3] D. Jones, S. Mirrazavi, and M. Tamiz, "Multi-objective meta-heuristics: An overview of the current state-of-the-art," *Eur. J. Oper. Res.*, vol. 137, no. 1, pp. 1–9, 2002.

[4] E.-G. Talbi, M. Basseur, A. J. Nebro, and E. Alba, "Multi-objective optimization using metaheuristics: non-standard algorithms," *Int. Trans. Oper. Res.*, vol. 19, no. 1-2, pp. 283–305, 2012.

[5] C. A. Coello Coello, "Evolutionary Multiobjective Optimization: Current and Future Challenges," in *Advances in Soft Computing*, pp. 243–256, Springer London, 2003.

[6] B. Li, J. Li, K. Tang, and X. Yao, "Many-Objective Evolutionary Algorithms: A Survey," *ACM Comput. Surv.*, vol. 48, pp. 13:1–35, Sept. 2015.

[7] C. A. C. Coello, G. T. Pulido, and M. S. Lechuga, "Handling multiple objectives with particle swarm optimization," *IEEE Trans. Evol. Comput.*, vol. 8, pp. 256–279, June 2004.

[8] M. Basseur, A. Liefooghe, K. Le, and E. K. Burke, "The efficiency of indicator-based local search for multi-objective combinatorial optimisation problems," *J. Heuristics*, vol. 18, no. 2, pp. 263–296, 2012.

[9] K. Deb, L. Thiele, M. Laumanns, and E. Zitzler, *Scalable Test Problems for Evolutionary Multiobjective Optimization*, pp. 105–145. Springer London, 2005.

[10] S. Voß and D. Woodruff, *Optimization Software Class Libraries*, vol. 18 of *Operations Research/Computer Science Interfaces Series*. Springer US, 1st ed., 2002.

[11] A. Fink, S. Voß, and D. Woodruff, "Metaheuristic class libraries," in *Handbook of Metaheuristics* (F. Glover and G. Kochenberger, eds.), vol. 57 of *International Series in Operations Research & Management Science*, pp. 515–535, Springer US, 2003.

[12] C. Gagné and M. Parizeau, "Genericity in evolutionary computation software tools: principles and case-study," *Int. J. Artif. Intell. Tools*, vol. 15, no. 2, pp. 173–194, 2006.

[13] J. A. Parejo, A. Ruiz-Cortés, S. Lozano, and P. Fernandez, "Metaheuristic optimization frameworks: a survey and benchmarking," *Soft Comput.*, vol. 16, no. 3, pp. 527–561, 2012.

[14] Z. Meng, X. Liu, G. Yang, L. Cai, and Z. Liu, "A comprehensive evaluation methodology for domain specific software benchmarking," in *2010 IEEE Int. Conf. on Progress in Informatics and Computing (PIC)*, vol. 2, pp. 1047–1051, 2010.

[15] B. R. Maxim and M. Kessentini, "Chapter 2 - an introduction to modern software quality assurance," in *Software Quality Assurance* (I. Mistrik, R. Soley, N. Ali, J. Grundy, and B. Tekinerdogan, eds.), pp. 19–46, Boston: Morgan Kaufmann, 2016.

[16] F.-A. Fortin, F.-M. De Rainville, M.-A. G. Gardner, M. Parizeau, and C. Gagné, "DEAP: Evolutionary Algorithms Made Easy," *J. Mach. Learn. Res.*, vol. 13, no. 1, pp. 2171–2175, 2012.

[17] E. O. Scott and S. Luke, "ECJ at 20: Toward a General Metaheuristics Toolkit," in *Proc. Genetic and Evolutionary Computation Conference (GECCO Companion)*, pp. 1391—-1398, 2019.

[18] M. Kronfeld, H. Planatscher, and A. Zell, "The EvA2 Optimization Framework," in *4th Int. Learning and Intelligent Optimization Conference (LION)*, pp. 247–250, 2010.

[19] S. Wagner, G. Kronberger, A. Beham, M. Kommenda, A. Scheibenpflug, E. Pitzer, S. Vonolfen, M. Kofler, S. Winkler, V. Dorfer, and M. Affenzeller, "Architecture and Design of the HeuristicLab Optimization Environment," in *Advanced Methods and Applications in Computational Intelligence*, vol. 6 of *Topics in Intelligent Engineering and Informatics*, pp. 197–261, Springer, 2014.

[20] S. Ventura, C. Romero, A. Zafra, J. A. Delgado, and C. Hervás, "JCLEC: a Java framework for evolutionary computation," *Soft Comput.*, vol. 12, no. 4, pp. 381–392, 2008.

[21] A. Ramírez, J. R. Romero, C. García-Martínez, and S. Ventura, "JCLEC-MO: A Java suite for solving many-objective optimization engineering problems," *Eng. Appl. Artif. Intell.*, vol. 81, pp. 14–28, 2019.

[22] A. J. Nebro, J. J. Durillo, and M. Vergne, "Redesigning the jMetal Multi-Objective Optimization Framework," in *Proc. Companion Publication of the 17th Ann. Conf. on Genetic and Evolutionary Computation (GECCO)*, pp. 1093–1100, 2015.

[23] D. Hadka, "MOEA Framework: A Free and Open Source Java Framework for Multiobjective Optimization." Version 2.12, January 2017. http://www.moeaframework.org (Last accessed 14 April 2019).

[24] M. Lukasiewycz, M. Glaß, F. Reimann, and J. Teich, "Opt4J: A Modular Framework for Meta-heuristic Optimization," in *Proc. 13th Ann. Conf. on Genetic and Evolutionary Computation (GECCO)*, pp. 1723–1730, 2011.

[25] A. Liefooghe, L. Jourdan, and E.-G. Talbi, "A software framework based on a conceptual unified model for evolutionary multiobjective optimization: ParadisEO-MOEO," *Eur. J. Oper. Res.*, vol. 209, no. 2, pp. 104–112, 2011.

[26] D. Hadka, "Platypus: A Free and Open Source Python Library for Multiobjective Optimization." Version 1.0.2, April 2018. https://github.com/Project-Platypus/Platypus (Last accessed 14 April 2019).

[27] X.-S. Yang, *Nature-inspired Metaheuristic Algorithms.* Luniver Press, 2nd ed., 2010.

[28] I. Boussaïd, J. Lepagnot, and P. Siarry, "A survey on optimization metaheuristics," *Inf. Sci.*, vol. 237, no. 0, pp. 82–117, 2013.

[29] J. D. Schaffer, "Multiple Objective Optimization with Vector Evaluated Genetic Algorithms," in *Proc. 1st Int. Conf. on Genetic Algorithms*, pp. 93–100, 1985.

[30] C. M. Fonseca and P. J. Fleming, "Genetic Algorithms for Multiobjective Optimization: Formulation, Discussion and Generalization," in *Proc. 5th Int. Conf. on Genetic Algorithms*, pp. 416–423, 1993.

[31] N. Srinivas and K. Deb, "Muiltiobjective Optimization Using Nondominated Sorting in Genetic Algorithms," *Evol. Comput.*, vol. 2, no. 3, pp. 221–248, 1994.

[32] E. Zitzler and L. Thiele, "Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach," *IEEE Trans. Evol. Comput.*, vol. 3, no. 4, pp. 257–271, 1999.

[33] J. D. Knowles and D. W. Corne, "Approximating the Nondominated Front Using the Pareto Archived Evolution Strategy," *Evol. Comput.*, vol. 8, no. 2, pp. 149–172, 2000.

[34] D. W. Corne, J. D. Knowles, and M. J. Oates, "The Pareto Envelope-Based Selection Algorithm for Multiobjective Optimization," in *Parallel Problem Solving from Nature PPSN VI*, vol. 1917 of *Lecture Notes in Computer Science*, pp. 839–848, 2000.

[35] D. W. Corne, N. R. Jerram, J. D. Knowles, M. J. Oates, and M. J, "PESA-II: Region-based Selection in Evolutionary Multiobjective Optimization," in *Proc. 3rd Ann. Conf. on Genetic and Evolutionary Computation (GECCO*, pp. 283–290, 2001.

[36] E. Zitzler, M. Laumanns, and L. Thiele, "SPEA2: Improving the Strength Pareto Evolutionary Algorithm," in *Proc. Conf. on Evolutionary Methods for Design, Optimisation and Control with Applications to Industrial Problems*, pp. 95–100, 2001.

[37] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Trans. Evol. Comput.*, vol. 6, no. 2, pp. 182–197, 2002.

[38] S. Kukkonen and J. Lampinen, "GDE3: the third evolution step of generalized differential evolution," in *Proc. 2005 IEEE Congress on Evolutionary Computation (CEC)*, vol. 1, pp. 443–450, 2005.

[39] C. Igel, N. Hansen, and S. Roth, "Covariance Matrix Adaptation for Multi-objective Optimization," *Evol. Comput.*, vol. 15, no. 1, pp. 1–28, 2007.

[40] A. J. Nebro, E. Alba, G. Molina, F. Chicano, F. Luna, and J. J. Durillo, "Optimal Antenna Placement Using a New Multi-objective CHC Algorithm," in *Proc. of the 9th Ann. Conf. on Genetic and Evolutionary Computation (GECCO)*, pp. 876–883, 2007.

[41] A. J. Nebro, F. Luna, E. Alba, B. Dorronsoro, J. J. Durillo, and A. Beham, "AbYSS: Adapting Scatter Search to Multiobjective Optimization," *IEEE Trans. Evol. Comput.*, vol. 12, no. 4, pp. 439–457, 2008.

[42] J. J. Durillo, A. J. Nebro, F. Luna, and E. Alba, "Solving Three-Objective Optimization Problems Using a New Hybrid Cellular Genetic Algorithm," in *Proc. Parallel Problem Solving from Nature (PPSN X)*, vol. 5199 of *Lecture Notes in Computer Science*, pp. 661–670, 2008.

[43] A. J. Nebro, J. J. Durillo, F. Luna, B. Dorronsoro, and E. Alba, "MOCell: A cellular genetic algorithm for multiobjective optimization," *Int. J. of Intelligent Systems*, vol. 24, no. 7, pp. 726–746, 2009.

[44] A. Santiago, B. Dorronsoro, A. J. Nebro, J. J. Durillo, O. Castillo, and H. J. Fraire, "A novel multi-objective evolutionary algorithm with fuzzy logic based adaptive selection of operators: FAME," *Information Sciences*, vol. 471, pp. 233–251, 2019.

[45] M. Laumanns, L. Thiele, K. Deb, and E. Zitzler, "Combining Convergence and Diversity in Evolutionary Multiobjective Optimization," *Evol. Comput.*, vol. 10, no. 3, pp. 263–282, 2002.

[46] M. R. Sierra and C. A. Coello Coello, "Improving PSO-Based Multi-objective Optimization Using Crowding, Mutation and ϵ-Dominance," in *Proc. Evolutionary Multi-Criterion Optimization*, vol. 3410 of *Lecture Notes in Computer Science*, pp. 505–519, 2005.

[47] A. Nebro, J. Durillo, J. Garcia-Nieto, C. Coello Coello, F. Luna, and E. Alba, "SMPSO: A new PSO-based metaheuristic for multi-objective optimization," in *IEEE Symposium on Computational Intelligence in Multi-Criteria Decision-Making (MCDM)*, pp. 66–73, 2009.

[48] S. Yang, M. Li, X. Liu, and J. Zheng, "A Grid-Based Evolutionary Algorithm for Many-Objective Optimization," *IEEE Trans. Evol. Comput.*, vol. 17, no. 5, pp. 721–736, 2013.

[49] E. Zitzler and S. Künzli, "Indicator-Based Selection in Multiobjective Search," in *Proc. Parallel Problem Solving from Nature (PPSN VIII)*, vol. 3242 of *Lecture Notes in Computer Science*, pp. 832–842, 2004.

[50] N. Beume, B. Naujoks, and M. Emmerich, "SMS-EMOA: Multiobjective selection based on dominated hypervolume," *Eur. J. Oper. Res.*, vol. 181, no. 3, pp. 1653–1669, 2007.

[51] J. Bader and E. Zitzler, "Hype: An Algorithm for Fast Hypervolume-based Many-objective Optimization," *Evol. Comput.*, vol. 19, pp. 45–76, Mar 2011.

[52] R. H. Gómez and C. A. C. Coello, "MOMBI: A new metaheuristic for many-objective optimization based on the R2 indicator," in *Proc. IEEE Congress on Evolutionary Computation (CEC)*, pp. 2488–2495, 2013.

[53] R. Hernández Gómez and C. A. Coello Coello, "Improved Metaheuristic Based on the R2 Indicator for Many-Objective Optimization," in *Proc. 17th Ann. Conf. on Genetic and Evolutionary Computation (GECCO)*, pp. 679–686, ACM, 2015.

[54] X. Cai, H. Sun, and Z. Fan, "A diversity indicator based on reference vectors for many-objective optimization," *Information Sciences*, vol. 430-431, pp. 467–486, 2018.

[55] E. Hughes, "Multiple single objective Pareto sampling," in *Proc. 2003 Congress on Evolutionary Computation (CEC)*, vol. 4, pp. 2678–2684, 2003.

[56] Q. Zhang and H. Li, "MOEA/D: A Multiobjective Evolutionary Algorithm Based on Decomposition," *IEEE Trans. Evol. Comput.*, vol. 11, no. 6, pp. 712–731, 2007.

[57] S. Zapotecas Martínez and C. A. Coello Coello, "A Multi-objective Particle Swarm Optimizer Based on Decomposition," in *Proc. 13th Ann. Conf. on Genetic and Evolutionary Computation (GECCO)*, pp. 69–76, 2011.

[58] M. Asafuddoula, T. Ray, and R. Sarker, "A Decomposition-Based Evolutionary Algorithm for Many Objective Optimization," *IEEE Trans. Evol. Comput.*, vol. 19, no. 3, pp. 445–460, 2015.

[59] K. Li, K. Deb, Q. Zhang, and S. Kwong, "An Evolutionary Many-Objective Optimization Algorithm Based on Dominance and Decomposition," *IEEE Trans. Evol. Comput.*, vol. 19, no. 5, pp. 694–716, 2015.

[60] X. Cai, Z. Mei, Z. Fan, and Q. Zhang, "A Constrained Decomposition Approach With Grids for Evolutionary Multiobjective Optimization," *IEEE Trans. Evol. Comput.*, vol. 22, no. 4, pp. 564–577, 2018.

[61] K. Deb and J. Sundar, "Reference Point Based Multi-objective Optimization Using Evolutionary Algorithms," in *Proc. 8th Ann. Conf. Genetic and Evolutionary Computation (GECCO)*, pp. 635–642, 2006.

[62] K. Deb and H. Jain, "An Evolutionary Many-Objective Optimization Algorithm Using Reference-Point-Based Nondominated Sorting Approach, Part I: Solving Problems With Box Constraints," *IEEE Trans. Evol. Comput.*, vol. 18, no. 4, pp. 577–601, 2014.

[63] R. Cheng, Y. Jin, M. Olhofer, and B. Sendhoff, "A Reference Vector Guided Evolutionary Algorithm for Many-Objective Optimization," *IEEE Trans. Evol. Comput.*, vol. 20, no. 5, pp. 773–791, 2016.

[64] A. B. Ruiz, R. Saborido, and M. Luque, "A Preference-based Evolutionary Algorithm for Multiobjective Optimization: The Weighting Achievement Scalarizing Function Genetic Algorithm," *J. of Global Optimization*, vol. 62, no. 1, pp. 101–129, 2015.

[65] M. A. Braun, P. K. Shukla, and H. Schmeck, "Obtaining Optimal Pareto Front Approximations Using Scalarized Preference Information," in *Proc. 17th Ann. Conf. on Genetic and Evolutionary Computation (GECCO)*, pp. 631–638, ACM, 2015.

[66] M. Luque, A. B. Ruiz, and R. Saborido, "Un Nuevo Algoritmo Evolutivo en Programación Multiobjetivo para Aproximar el Frente Óptimo de Pareto," in *X Congreso Español de Metaheurísticas, Algoritmos Evolutivos y Bioinspirados (MAEB)*, 2015.

[67] F. Goulart and F. Campelo, "Preference-guided evolutionary algorithms for many-objective optimization," *Inf. Sci.*, vol. 329, pp. 236–255, 2016.

[68] L. J. Eshelman, "The CHC Adaptive Search Algorithm : How to Have Safe Search When Engaging in Nontraditional Genetic Recombination," *Foundations of Genetic Algorithms*, pp. 265–283, 1991.

[69] K. A. De Jong, *An Analysis of the Behavior of a Class of Genetic Adaptive Systems.* PhD thesis, University of Michigan, Ann Arbor, MI, USA, 1975.

[70] J. H. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence.* Cambridge, MA, USA: MIT Press, 1992.

[71] D. H. Ackley, *A Connectionist Machine for Genetic Hillclimbing.* Norwell, MA, USA: Kluwer Academic Publishers, 1987.

[72] Z. Michalewicz, *Genetic Algorithms + Data Structures = Evolution Programs.* London, UK, UK: Springer-Verlag, 3rd ed. ed., 1996.

[73] T. Nomura, "An Analysis on Crossovers for Real Number Chromosomes in an Infinite Population Size," in *Proc. 15th Int. Joint Conference on Artificial Intelligence (IJCAI) - Volume 2*, pp. 936–941, Morgan Kaufmann Publishers Inc., 1997.

[74] L. J. Eshelman and J. D. Schaffer, "Real-Coded Genetic Algorithms and Interval-Schemata," *Foundations of Genetic Algorithms*, pp. 187–202, 1993.

[75] M. Takahashi and H. Kita, "A crossover operator using independent component analysis for real-coded genetic algorithms," in *Proc. 2001 Congress on Evolutionary Computation (CEC)*, vol. 1, pp. 643–649, 2001.

[76] H. Mühlenbein and D. Schlierkamp-Voosen, "Predictive Models for the Breeder Genetic Algorithm I. Continuous Parameter Optimization," *Evol. Comput.*, vol. 1, pp. 25–49, March 1993.

[77] A. H. Wright, "Genetic Algorithms for Real Parameter Optimization," in *Foundations of Genetic Algorithms*, pp. 205–218, Morgan Kaufmann, 1991.

[78] D. Dumitrescu, B. Lazzerini, L. Jain, and A. Dumitrescu, *Evolutionary Computation.* International Series on Computational Intelligence, Taylor & Francis, 2000.

[79] K. Deb and R. Agrawal, "Simulated Binary Crossover for Continuous Search Space," Tech. Rep. IITK/ME/SMD-94027, Indian Institute of Technology, Kanpur, India, 1994.

[80] O. Wendt and W. König, "Cooperative Simulated Annealing: How Much Cooperation is Enough?," tech. rep., Institute for Information Systems at Frankfurt University, 1994.

[81] I. M. Oliver, D. J. Smith, and J. R. C. Holland, "A Study of Permutation Crossover Operators on the Traveling Salesman Problem," in *Proc. 2nd Int. Conf. on Genetic Algorithms on Genetic Algorithms and Their Application*, pp. 224–230, L. Erlbaum Associates Inc., 1987.

[82] L. Davis, "Applying Adaptive Algorithms to Epistatic Domains," in *Proc. of the 9th Int. Joint Conference on Artificial Intelligence (IJCAI) - Volume 1*, pp. 162–164, 1985.

[83] D. Whitley, T. Starkweather, and D. Shaner, *Handbook of Genetic Algorithms*, ch. The Travelling Salesman and Sequence Scheduling: Quality Solutions Using Genetic Edge Recombination, pp. 350–372. Van Nostrand Reinhold, 1991.

[84] H. Mühlenbein, "Evolution in time and space - the parallel genetic algorithm," in *Foundations of Genetic Algorithms*, pp. 316–337, Morgan Kaufmann, 1991.

[85] G. Syswerda, *Handbook of Genetic Algorithms*, ch. Schedule Optimization Using Genetic Algorithms, pp. 332–349. Van Nostrand Reinhold, 1991.

[86] D. E. Goldberg and R. Lingle, Jr., "Alleles Loci and the Traveling Salesman Problem," in *Proc. 1st Int. Conf. on Genetic Algorithms*, (Hillsdale, NJ, USA), pp. 154–159, L. Erlbaum Associates Inc., 1985.

[87] D. M. Tate and A. E. Smith, "A genetic approach to the quadratic assignment problem," *Computers & Operations Research*, vol. 22, no. 1, pp. 73–83, 1995.

[88] J. Vrugt, B. Robinson, and J. Hyman, "Self-Adaptive Multimethod Search for Global Optimization in Real-Parameter Spaces," *IEEE Trans. Evol. Comput.*, vol. 13, pp. 243–259, April 2009.

[89] N. J. Radcliffe, "Forma Analysis and Random Respectful Recombination," in *Proc. 4th Int. Conf. on Genetic Algorithms*, 1991.

[90] K. Deb, A. Anand, and D. Joshi, "A Computationally Efficient Evolutionary Algorithm for Real-parameter Optimization," *Evol. Comput.*, vol. 10, no. 4, pp. 371–395, 2002.

[91] S. Tsutsui, M. Yamamura, and T. Higuchi, "Multi-parent Recombination with Simplex Crossover in Real Coded Genetic Algorithms," in *Proc. of the Genetic and Evolutionary Computation Conference*, vol. 1, pp. 657–664, 1999.

[92] T. Nomura and T. Miyoshi, "Numerical Coding and Unfair Average Crossover in GA for Fuzzy Rule Extraction in Dynamic Environments," in *Selected Papers from the EEE/Nagoya-University World Wisepersons Workshop on Fuzzy Logic, Neural Networks, and Evolutionary Computation*, pp. 55–72, Springer-Verlag, 1996.

[93] H. Kita, I. Ono, and S. Kobayashi, "Multi-parental extension of the unimodal normal distribution crossover for real-coded genetic algorithms," in *Proc. 1999 Congress on Evolutionary Computation (CEC)*, vol. 2, pp. 1581–1588, 1999.

[94] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* Cambridge, MA, USA: MIT Press, 1992.

[95] R. Poli and W. B. Langdon, "A new schema theory for genetic programming with one-point crossover and point mutation," in *Proc. 2nd Ann. Conf. on Genetic Programming* (e. a. J. R. Koza, ed.), pp. 278–285, Morgan Kaufmann, July 1997.

[96] W. B. Langdon, "Size Fair and Homologous Tree Crossovers for Tree Genetic Programming," *Genet. Program. Evol. M.*, vol. 1, no. 1-2, pp. 95–119, 2000.

[97] A. E. Eiben and J. E. Smith, *Introduction to Evolutionary Computing.* SpringerVerlag, 2003.

[98] H.-P. Schwefel, *Numerical Optimization of Computer Models.* New York, NY, USA: John Wiley & Sons, Inc., 1981.

[99] K. Deb and M. Goyal, "A combined genetic adaptive search (GeneAS) for engineering design," *Computer Science and Informatics*, vol. 26, pp. 30–45, 1996.

[100] D. B. Fogel, "Applying Evolutionary Programming to Selected Traveling Salesman Problems," *Cybernetics and Systems*, vol. 24, no. 1, pp. 27–36, 1993.

[101] H.-M. Voigt and T. Anheyer, "Modal mutations in evolutionary algorithms," in *Proc. 1st IEEE Conf. on Evolutionary Computation*, pp. 88–92 vol.1, 1994.

[102] K. Chellapilla, "A Preliminary Investigation into Evolving Modular Programs without Subtree Crossover," in *Proc. of the 3rd Annual Conference on Genetic Programming*, pp. 23–31, 1998.

[103] P. J. Angeline, D. B. Fogel, and L. J. Fogel, "A Comparison of Self-Adaptation Methods for Finite State MMachine in Dynamic Environments," in *Proc. 5th Ann. Conf. on Evolutionary Programming*, pp. 441–449, 1996.

[104] J. Kinnear, K.E., "Fitness landscapes and difficulty in genetic programming," in *Proc. of the 1st IEEE Conference on Evolutionary Computation*, pp. 142–147 vol.1, 1994.

[105] "Black-Box Optimization Benchmarking (BBOB 2016)." GECCO Workshop on Real-Parameter Black-Box Optimization Benchmarking, 2016. https://numbbo.github.io/workshops/BBOB-2016/ (Last accessed: 05 November 2018).

[106] Q. Zhang, A. Zhou, S. Zhao, P. N. Suganthan, W. Liu, and S. Tiwari, "Multiobjective optimization Test Instances for the CEC 2009 Special Session and Competition," Working Report CES-887, School of Computer Science and Electrical Engineering, University of Essex, 2008.

[107] R. Cheng, M. Li, Y. Tian, X. Zhang, S. Yang, Y. Jin, and X. Yao, "Benchmark Functions for the CEC'2018 Competition on Many-Objective Optimization," tech. rep., University of Birmingham, 2018.

[108] H. Jain and K. Deb, "An Evolutionary Many-Objective Optimization Algorithm Using Reference-Point Based Nondominated Sorting Approach, Part II: Handling Constraints and Extending to an Adaptive Approach," *IEEE Trans. Evol. Comput.*, vol. 18, no. 4, pp. 602–622, 2014.

[109] K. Deb, L. Thiele, M. Laumanns, and E. Zitzler, *Scalable Test Problems for Evolutionary Multiobjective Optimization*, ch. Evolutionary Multiobjective Optimization, pp. 105–145. London: Springer London, 2005.

[110] C. M. Fonseca and P. J. Fleming, "An Overview of Evolutionary Algorithms in Multiobjective Optimization," *Evol. Comput.*, vol. 3, no. 1, pp. 1–16, 1995.

[111] F. Gu, H.-L. Liu, and K. C. Tan, "Multiobjective evolutionary algorithm using dynamic weight design method," *Int. J. of Innovative Computing, Information and Control*, vol. 8, no. 5B, pp. 3677–3688, 2012.

[112] C. Igel, N. Hansen, and S. Roth, "Covariance Matrix Adaptation for Multi-objective Optimization," *Evol. Comput.*, vol. 15, no. 1, pp. 1–28, 2007.

[113] F. Kursawe, "A variant of evolution strategies for vector optimization," in *Proc. 1st Workshop on Parallel Problem Solving from Nature (PPSN)*, pp. 193–197, 1991.

[114] H. L. Liu, F. Gu, and Q. Zhang, "Decomposition of a Multiobjective Optimization Problem Into a Number of Simple Multiobjective Subproblems," *IEEE Trans. Evol. Comput.*, vol. 18, no. 3, pp. 450–455, 2014.

[115] H. Li and Q. Zhang, "Multiobjective Optimization Problems With Complicated Pareto Sets, MOEA/D and NSGA-II," *IEEE Trans. Evol. Comput.*, vol. 13, pp. 284–302, April 2009.

[116] A. Osyczka and S. Kundu, "A new method to solve generalized multicriteria optimization problems using the simple genetic algorithm," *Structural optimization*, vol. 10, no. 2, pp. 94–99, 1995.

[117] C. Poloni, G. Mosetti, and S. Contessi, "Multiobjective Optimization by GAs: Application to System and Component Design," in *Computational Methods in Applied Sciences*, pp. 258–264, 1996.

[118] M. Tanaka, H. Watanabe, Y. Furukawa, and T. Tanino, "GA-based decision support system for multicriteria optimization," in *Proc. IEEE Int. Conf. on Systems, Man and Cybernetics*, vol. 2, pp. 1556–1561, 1995.

[119] I. M. R. Viennet, C. Fonteix, "Multicriteria optimization using a genetic algorithm for determining a Pareto set," *Int. J. of System Science*, vol. 27, no. 2, pp. 255–260, 1996.

[120] S. Huband, L. Barone, L. While, and P. Hingston, "A Scalable Multi-objective Test Problem Toolkit," in *Proc. 3rd Int. Conf. Evolutionary Multi-Criterion Optimization (EMO)*, pp. 280–295, 2005.

[121] E. Zitzler, K. Deb, and L. Thiele, "Comparison of Multiobjective Evolutionary Algorithms: Empirical Results," *Evol. Comput.*, vol. 8, no. 2, pp. 173–195, 2000.

[122] M. M. Yenisey and B. Yagmahan, "Multi-objective permutation flow shop scheduling problem: Literature review, classification and current trends," *Omega*, vol. 45, pp. 119–135, 2014.

[123] E. Zitzler and L. Thiele, "Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach," *IEEE Trans. on Evol. Comput.*, vol. 3, no. 4, pp. 257–271, 1999.

[124] M. Pelikan, "Analysis of Estimation of Distribution Algorithms and Genetic Algorithms on NK Landscapes," in *Proc. 10th Ann. Conf. on Genetic and Evolutionary Computation (GECCO)*, pp. 1033–1040, 2008.

[125] E. Zitzler, M. Laumanns, and S. Bleuler, *Metaheuristics for Multiobjective Optimisation*, ch. A Tutorial on Evolutionary Multiobjective Optimization, pp. 3–37. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004.

[126] M. Lukasiewycz, M. Glaß, C. Haubelt, and J. Teich, "Solving Multi-objective Pseudo-Boolean Problems," in *Proc. 10th Int. Conft Theory and Applications of Satisfiability Testing (SAT)*, pp. 56–69, 2007.

[127] J. Knowles and D. Corne, "Instance Generators and Test Suites for the Multiobjective Quadratic Assignment Problem," in *Proc. 2nd Int. Conf. Evolutionary Multi-Criterion Optimization (EMO)*, pp. 295–310, 2003.

[128] T. Lust and J. Teghem, *Advances in Multi-Objective Nature Inspired Computing*, ch. The Multiobjective Traveling Salesman Problem: A Survey and a New Approach, pp. 119–141. Springer Berlin Heidelberg, 2010.

[129] A. Zhou, Y. Jin, Q. Zhang, B. Sendhoff, and E. Tsang, "Combining Model-based and Genetics-based Offspring Generation for Multi-objective Optimization Using a Convergence Criterion," in *Proc. IEEE Int. Conf. on Evolutionary Computation*, pp. 892–899, 2006.

[130] D. A. van Veldhuizen and G. B. Lamont, "Multiobjective Evolutionary Algorithm Test Suites," in *Proc. 1999 ACM Symposium on Applied Computing (SAC)*, pp. 351–357, ACM, 1999.

[131] J. R. Schott, "Fault tolerant design using single and multicriteria genetic algorithm optimization," Master's thesis, Massachusetts Institute of Technology. Dept. of Aeronautics and Astronautics, May 1995.

[132] E. Zitzler, L. Thiele, M. Laumanns, C. M. Fonseca, and V. G. da Fonseca, "Performance assessment of multiobjective optimizers: an analysis and review," *IEEE Trans. Evol. Comput.*, vol. 7, no. 2, pp. 117–132, 2003.

[133] M. Basseur, F. Seynhaeve, and E. Talbi, "Design of multi-objective evolutionary algorithms: application to the flow-shop scheduling problem," in *Proc. 2002 Congress on Evolutionary Computation (CEC)*, vol. 2, pp. 1151–1156, 2002.

[134] C. A. C. Coello and N. C. Cortés, "Solving Multiobjective Optimization Problems Using an Artificial Immune System," *Genet. Program. Evol. M.*, vol. 6, no. 2, pp. 163–190, 2005.

[135] H. Ishibuchi, H. Masuda, and Y. Nojima, "A Study on Performance Evaluation Ability of a Modified Inverted Generational Distance Indicator," in *Proc. 17th Ann. Conf. on Genetic and Evolutionary Computation (GECCO)*, pp. 695–702, 2015.

[136] M. Hansen and A. Jaszkiewicz, "Evaluating the quality of approximations to the non-dominated set," Tech. Rep. IMM-REP-1998-7, University of Denmark, 1998.

[137] E. Zitzler, K. Deb, and L. Thiele, "Comparison of Multiobjective Evolutionary Algorithms: Empirical Results," *Evol. Comput.*, vol. 8, no. 2, pp. 173–195, 2000.

[138] E. Zitzler, M. Laumanns, and S. Bleuler, *Metaheuristics for Multiobjective Optimisation*, ch. A Tutorial on Evolutionary Multiobjective Optimization, pp. 3–37. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004.

[139] E. Zitzler and L. Thiele, "Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach," *IEEE Trans. on Evol. Comput.*, vol. 3, no. 4, pp. 257–271, 1999.