Soft Comput
DOI 10.1007/s00500-011-0754-8

ORIGINAL PAPER

Metaheuristic optimization frameworks: a survey and benchmarking

José Antonio Parejo · Antonio Ruiz-Cortés · Sebastián Lozano · Pablo Fernandez

Abstract This paper performs an unprecedented comparative study of Metaheuristic optimization frameworks. As criteria for comparison a set of 271 features grouped in 30 characteristics and 6 areas has been selected. These features include the different metaheuristic techniques covered, mechanisms for solution encoding, constraint handling, neighborhood specification, hybridization, parallel and distributed computation, software engineering best practices, documentation and user interface, etc. A metric has been defined for each feature so that the scores obtained by a framework are averaged within each group of features, leading to a final average score for each framework. Out of 33 frameworks ten have been selected from the literature using well-defined filtering criteria, and the results of the comparison are analyzed with the aim of identifying improvement areas and gaps in specific frameworks and the whole set. Generally speaking, a significant lack of support has been found for hyper-heuristics, and parallel and distributed computing capabilities. It is also desirable to have a wider implementation of some Software Engineering best practices. Finally, a wider support for some metaheuristics and hybridization capabilities is needed.

1 Introduction and motivation

Heuristic methods have proven to be a comprehensive tool to solve hard optimization problems; they bring a balance of "good" solutions (relatively close to global optimum) and affordable time and cost. However, heuristics are usually based on specific characteristics of the problem at hand, which makes their design and development a complex task. In order to solve this drawback, metaheuristics appear as a significant advance (Glover 1977); they are problem-agnostic algorithms that can be adapted to incorporate the problem-specific knowledge. Metaheuristics have been remarkably developed in recent decades (Voß 2001), becoming popular and being applied to many problems in diverse areas (Glover and Kochenberger 2002; Back et al. 1997). However, when new are considered, metaheuristics should be implemented and tested, implying costs and risks.

As a solution, object-oriented paradigm has become a successful mechanism used to ease the burden of application development and particularly, on adapting a given metaheuristic to the specific problem to solve. Based on this paradigm, there are a number of proposals which jointly offer support for the most widespread techniques, platforms and languages. In this article, we coin these kind of approaches as metaheuristic optimization frameworks (MOFs).

In addition to the advantages of having pre-implemented metaheuristics in terms of testing and reuse, using a MOF can provide a valuable benefit. They support the evaluation and comparison of different metaheuristics to select the best performing one for the problem at hand.

However, as the number of alternatives is extensive (we have identified 33 different MOFs in literature) this becomes a double-edged sword and the choice of the right MOF results in a major issue. Due to the wide number of metaheuristics (and variants), each of the MOFs is focused on a particular subset; in this context, not choosing the right MOF leads to a no-win situation; this would imply further costs due to the change from one MOF to another,

J. A. Parejo (✉) · A. Ruiz-Cortés · S. Lozano · P. Fernandez
University of Sevilla, Seville, Spain
e-mail: japarejo@us.es

Published online: 28 August 2011

or the risk of obtaining a sub-optimal solution due to the use of inappropriate metaheuristics.

A comparative framework is a useful tool to guide a selection of the MOF that best suits a particular scenario. However comparisons of frameworks in literature are either informal evaluations using author criteria or focused on performance (Wilson et al. 2004). Gagnè and Parizeau (2006) present a comparison (over 6 features) of MOFs supporting evolutionary algorithms. Voß (2002) presents a constructive discussion of various software libraries, but there is a lack of a comparative analysis. Alternatively, some articles (such as Cahon et al. 2004; Di Gaspero and Schaerf 2003) presenting a concrete MOF, include a related work section with a comparison of specific features with other MOFs; however, those works present a narrow perspective with a comparison of a reduced set of MOFs.

To the best of our knowledge, no general reviews nor detailed comparative studies of MOFs have been conducted in the literature. Moreover, a conceptual discussion about the desirable set of features of a MOF has not been carried out.

The key point of this article is to provide a general comparative framework to guide the selection of a particular MOF and to evaluate the current MOFs found in the literature from a research perspective. In doing so, this article extends the comparative framework of Gagnè and Parizeau (2006) including frameworks that incorporate several types of metaheuristic techniques (cf. Sect. 4) and presents a comparative analysis of a large set of features.

Specifically, this paper advances the state of the art in the following:

1. A general comparative framework for MOFs that can be used to classify, evaluate and compare them.
2. An analysis of the current relevant MOFs in the literature based in the comparative framework proposed.
3. An evaluation of the current state of the art of MOFs from the research context that can be used: (i) to guide newcomers in the area and (ii) to identify relevant gaps to MOF developers.

It is important to highlight that the main value of this study lies neither in comparing the rankings of two concrete MOFs in a feature or characteristic, nor in stating which MOF better fulfills the benchmark criteria. The main contribution of the paper is the establishment of a general comparison framework which clearly defines the set of desirable features of MOFs; depicting a real "state of the art" MOF with improvement directions and gaps in features support. This comparison framework has shown its value an generality, allowing the evaluation of the new versions of assessed MOFs released during the realization of this study without modifications (four MOFs released new versions). Moreover, the possibility of downloading the benchmark as a spreadsheet and tailoring it to user needs by modifying its weights is also crucial for making it more relevant and applicable.

The remainder of this article is organized as follows: Section 2 defines what a metaheuristic optimization framework is and outlines the advantages and disadvantages of using such tools. Next, Sect. 3 describes the methodology used to create our comparative framework divided into six areas. In further sections, each area is developed in detail (Sects. 4 to 9), defining a set of characteristics, its importance, metrics and data sources used for its evaluation. In each section, charts and interesting results on the current support by the selected MOFs are provided. In Sect. 10 we discuss the results obtained from a global perspective, showing significant gaps and general tendencies. Finally, in Sect. 11 we summarize and present the main conclusions and future work.

Details about MOF assessment are provided as tables in "Appendix" and at http://www.isa.us.es/MOFComparison.

2 Metaheuristic optimization frameworks

Problem types that model real-life situations (e.g. traveling salesman problem, knapsack problem, MAX-SAT problem, etc.) have concrete instances that have a solution space that contains specific solutions. When those solutions are evaluated using an objective function (or a set of functions for multi-objective problems) we can define an optimization problem as searching for the solution that provides the maximum (or minimum) value.

According to Glover and Kochenberger (2002) we define metaheuristics as: "An iterative process that guides the operation of one or more subordinate heuristics (which may be from a local search process, to a constructive process of random solutions) to efficiently produce quality solutions for a problem". An interesting concept in this definition is the establishment of two distinct levels for metaheuristic problem solving: the heuristic level that is by definition highly dependent on the problem, and the metaheuristic level based on the aforementioned level but expressed as a problem-independent process. For instance, when we apply simulated annealing (SA) (Kirkpatrick et al. 1983), we use three subordinate heuristics: the creation of an initial solution to the problem, the generation of similar (neighboring) solutions to another solution by some criterion; and the evaluation of solutions (objective function). These heuristics are highly dependent on the specific problem addressed and how we encode solutions, but based on them, we can establish a general iterative algorithm that has been successfully applied to a huge variety of problems.

Thus, for each metaheuristic technique and type of problem, we have a set of subordinate heuristics that define how the metaheuristic is adapted to the problem-type at hand. Note that a given problem-type may have multiple valid sets of subordinate heuristics. For instance, when using a genetic algorithm we can have different solution encodings (e.g. using bit strings or integer vectors) and consequently, different ways of generating the initial population, crossover and mutation operators, etc. For a specific instance of a problem, the application of a metaheuristic will provide a solution depending on the specific subordinate heuristics used.

A MOF can be defined as "a set of software tools that provide a correct and reusable implementation of a set of metaheuristics, and the basic mechanisms to accelerate the implementation of its partner subordinate heuristics (possibly including solution encodings and technique-specific operators), which are necessary to solve a particular problem instance using techniques provided". Figure 1 depicts a conceptual map showing these elements and their relationships. In this figure, MOFs and their components are shaded.

Specifically, MOFs not only provide a set of implemented techniques, but also facilities to simplify the adaptation of those implementations to the specific problem to address and additional tools to help the whole optimization problem solving activities. Moreover, MOFs usually provide mechanisms to monitor the optimization processes, supporting tools to determine appropriate values of parameters of techniques, and to identify the reasons that prevent techniques from finding optimal solutions.

2.1 Why are MOFs valuable?

The No Free Lunch (NFL) theorem of Wolpert and Macready (1997) can be summarized as follows: "There is no strategy or algorithm that generally behaves better than another for the entire set of possible problems". Ho and Pepyne (2002) expressed it as follows: "Universal optimizers are impossible".

The NFL theorem has been used as an argument against the use of MOFs, since there can be no universal optimal solver nor a software implementation of it (Voß 2002, Chapter 4, pp 82–83). Frameworks are not intended to be a universal optimal implemented solution. Frameworks are tailorable tools that allow us to perform this implementation in an better way in terms of the implementation cost and effort.

The NFL theorem implies the need to "match" a problem and the optimization technique used to solve it in order to obtain optimal or near optimal solutions. Metaheuristics allow to perform such a matching by adapting its underlying heuristics. The purpose of MOFs is the optimization of such adaption mechanisms in a more reusable and effortless way.

Furthermore, if no algorithm behaves better than another (as stated by NFL), when trying to solve a new problem without specific knowledge (with regards to well-known similar problems and their best-matching techniques), it is even more advantageous to use various metaheuristics to ensure a proper matching to the problem. The benefits of using MOFs that implement several metaheuristics are even more obvious.

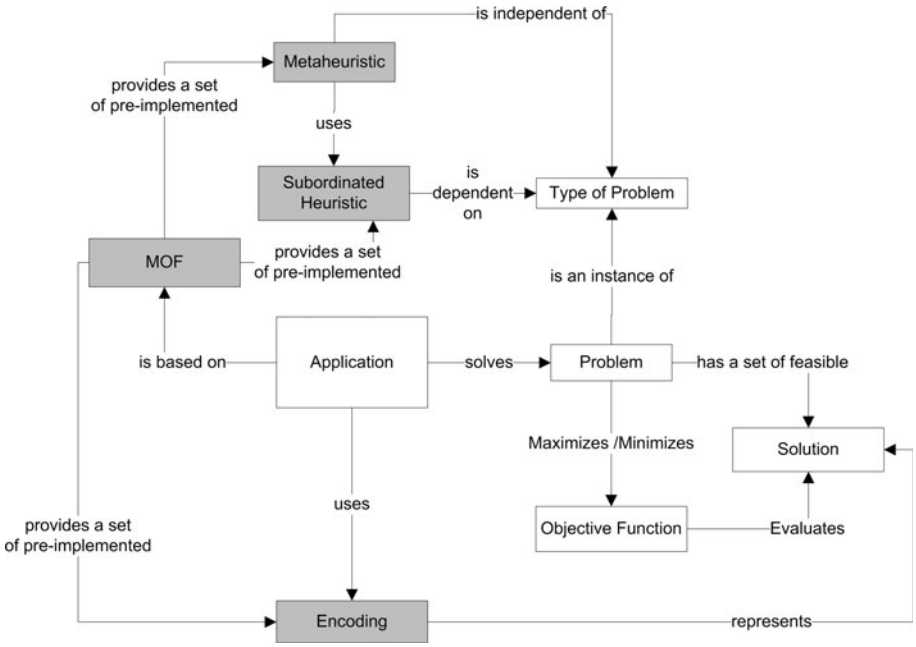


Fig. 1 MOFs' conceptual map

Table 1 Advantages and drawbacks of the use of MOFs

Advantages	Drawbacks
Reduced implementation effort and ability to apply various techniques and variants with little additional effort	Steep learning curve
Additional tools to help problem solving (monitoring, reporting, parallel and distributed computing)	Advanced knowledge needed for adaptation and inflexibility to adapt to use some metaheuristics variants
Optimized and error-free implementation (except the extensions and adaption created by users or the undetected errors that could be present in the MOF)	Induced complexity (when debugging and testing) and additional dependences
New users with little knowledge can use the framework not only as a tool for software application development environment but as a methodological aid	The choice of the right MOF may be an issue, since switching from one MOF to another has a high cost, they provide diverse features support and there are no comparative benchmarks in literature

The main advantage of using MOFs is that they provide correct, fully functional and optimized versions of a set of metaheuristic techniques and their variants. Moreover, they provide mechanisms to facilitate the proper implementation of the underlying heuristics, depending on the problem, the representation of solutions, etc. As a consequence, we only have to implement those elements which are directly related to the problem, freeing us, as far as possible, of worrying abouts the aspects that do not depend on it. In addition, the use of MOFs decreases in general the risk of bugs in the implementation and therefore the time (and associated cost) invested in debugging. Complementary, some MOFs provide additional features to aid solving the optimization problem, such as optimization process monitoring and results analysis tools, capabilities for parallel and distributed optimization tasks execution, supporting mechanisms for techniques parameters value determination, graphical reports and user-friendly interfaces (see Sects. 6 and 7 for a detailed review of this kind of features).

Some of the advantages shown above as well as features described in the following sections are more valuable than others depending on the application context. Specifically, we identify three main MOF usage contexts:

- *Industrial application of optimization problem solving*: In this context, implementation burden reduction and its optimization are the most valuable features.
- *Research on metaheuristics and optimization problem solving*: In this context, optimization process monitoring and results analysis tools are likely to be the most valuable features.
- *Teaching of metaheuristics*: In this context, ease of use of the graphical interface, reports and graphical representation of solutions and methodological guidance through wizards and GUI are likely to be the most valuable features.

2.2 Drawbacks: all that glitters are not gold

MOFs also have some drawbacks. One is their steep learning curve. The user needs to know the set of variation and extension points to use in order to adapt the framework to the problem and understand how they are related to the behavior of the software. This means that when we exactly know which technique to apply and we are confident in our implementation skills, using a MOF may be discouraged unless you have expertise in using it. Another drawback to consider when using MOFs is that the flexibility to adapt the MOF is limited by its design. Consequently, a proper framework design is essential to achieve the most favorable balance between the capabilities provided and its flexibility. This drawback implies that it could be impossible to implement certain variants or modify certain behavior when using a MOF, this drawback is specially serious in the context of research, where experimentation with different variants and capability of customization is a key feature (cf. Sect. 2.1 with the definition of usage contexts identified). An increased testing and debugging complexity is a disadvantage resulting from the inversion of control (i.e. loss of explicit control over the execution flow of our application) that involves the use of a framework. The use of MOFs implies increasing the size of the software, creating dependencies on third-party libraries and an increase on the complexity of the application.

The advantages and drawbacks of using MOFs discussed in this section are shown in Table 1.

3 Review method

The present comparative is based on the software technology evaluation methodology proposed by Brown and Wallnau (1996), which seeks to identify the value added by technology through the establishment of a descriptive model in terms of its features of interest and their

relationship and importance to its usage contexts. In this work, only the first phase (descriptive modeling) of the proposed methodology is performed, providing a solid basis for the evaluation of technologies and a context for describing the features of interest. The second phase includes conducting experiments with each of the MOFs associated with specific use scenarios and is beyond the scope of this article.

In order to establish our descriptive model of characteristics to be supported by MOFs, and select the set of MOFs to assess, we followed a systematic and structured method inspired by the guidelines of Kitchenham (2004). First, we stated a set of research questions (see next subsection). Second, in order to find the list of candidate MOFs, we established information sources used for the search (cf. Sect. 3.2). Then, we applied filtering criteria to obtain the final set of MOFs to be analyzed (cf. Sect. 3.3). Finally, we composed and grouped the full set of comparison criteria and used them to assess MOFs.

3.1 Research questions

The aim of this study is to answer the following research questions:

- RQ1: What metaheuristics are currently supported by MOFs? This question motivates the following sub-questions:
  - Is there a MOF that supports the whole set of techniques?
  - What is the most popular technique? i. e., Which is the technique implemented by most MOFs?
  - Is there a "core set of techniques" supported by more than a half of the assessed MOFs?
- RQ2: What tailoring mechanisms do current MOFs provide in order to adapt to solve a problem, and to what extent are those mechanisms supported? This question motivates the following sub-questions:
  - Is there a "core set of adaption mechanisms" (such as solution encoding mechanisms, operators, etc.) supported by more than half of the assessed MOFs?
  - What MOF is better suited to adapt to specific problem solving?
- RQ3: What combination of techniques (hybrid approaches) are supported when using a MOF? This question motivates the following sub-question:
  - Is hybridization a widely supported feature (supported by more than half of the assessed frameworks)?
  - What is the most common hybridization mechanism supported by MOFs?
- RQ4: Can current MOFs help to find out the best parameter values for their supported metaheuristics (perform hyper-heuristic search)?
- RQ5: To what extent do current MOFs take advantage of parallelization capabilities of metaheuristics and distributed computing?
- RQ6: What additional tools are provided by current MOFs in order to support the whole of the optimization problem-solving process?
- RQ7: Which costs and licensing model do current MOFs go by?
- RQ8: What platforms (operating system, programming languages, etc.) are supported by current MOFs? This question motivates the following sub-question:
  - Given the current support of techniques by MOFs, are all techniques available on each platform?
- RQ9: Are current MOFs using software engineering best practices in order to improve code quality, maintainability, stability and performance?

After reviewing all this information we also want to answer some more general questions:

- RQ10: What degree of maturity and popularity do current MOFs have? This question motivates the following sub-questions:
  - What problems have been solved with each MOF?
  - What documentation and help on its use does each MOF provide to its users?
  - Are current MOFs supported by scientific publications?
  - What is the user community of each current MOF?
  - Which is currently the most popular MOF?
- RQ11: What are the challenges to be faced in the evolution and development of MOFs?

3.2 Source material

The information sources used for the search of MOFs have primarily been electronic databases through their online search engines. Specifically, we have searched on: IEEE Xplore, ACM Digital Library, SpringerLink and Scopus. The following search strings have been used: "Metaheuristic Optimization Framework", "Heuristic Optimization Framework", "Metaheuristic Software library", "Metaheuristic Optimization Library" and "Metaheuristic Optimization Tool".

Based on the results obtained, a list of candidate MOFs was generated that later was enlarged using direct web searches (using Google and the search strings described above) and references present on papers and frameworks'

Table 2 Selected MOFs

Name	Ver.	Web
EasyLocal (Di Gaspero and Schaerf 2003)	2.0	http://satt.diegm.uniud.it/EasyLocal++/
ECJ (Luke et al. 2009)	20	http://cs.gmu.edu/∼eclab/projects/ecj/
EO/ ParadisEO/ MOEO/ PEO (Cahon et al. 2004)	1.2	http://paradiseo.gforge.inria.fr http://eodev.sourceforge.net/
EvA2 (Kronfeld et al. 2010)	2	http://www.ra.cs.uni-tuebingen.de/software/EvA2/
FOM (Parejo et al. 2003)	0.8	http://www.isa.us.es/fom
HeuristicLab (Wagner 2009)	3.3	http://dev.heuristiclab.com
JCLEC (and KEEL) (Ventura et al. 2008)	4.0	http://JCLEC.sourceforge.net http://sci2s.ugr.es/keel/
MALLBA (Alba et al. 2007)	2.0	http://neo.lcc.uma.es/mallba/easy-mallba/index.html
Optimization Algorithm Toolkit (Brownlee 2007)	1.4	http://optalgtoolkit.sourceforge.net
Opt4j (Martin Lukasiewycz and Helwig 2009)	2.1	http://opt4j.sourceforge.net

web sites. Key references obtained during this phase were Voß (2002) and Gagnè and Parizeau (2006). However, framework web sites were a key data source, given that their links, articles and related work sections allowed us establish the full reference set to study. After a detailed analysis of these references, an initial set of main supported features and MOFs were established, and basic information gathering of those tools was performed. The list of candidate optimization tools contains 33 entries: Comet, EvA2, evolvica, Evolutionary::Algorithm, GAPlayground, jaga, JCLEC, JGAP, jMetal, n-genes, Open Beagle, Opt4j, ParadisEO/EO, Pisa, Watchmaker, FOM, Hypercube, HotFrame, Templar, EasyLocal, iOpt, OptQuest, JDEAL, Optimization Algorithm Toolkit, HeuristicLab, MAFRA, Localizer, GALIB, DREAM, Discropt, MALLBA, MAGMA and UOF.

3.3 Inclusion and exclusion criteria

Some MOFs were discarded to keep the size and complexity of the review at a manageable level, establishing the following filtering criteria:

- The development of MOFs must be alive, and error fixing supported by their developers. A MOF where users must debug all errors found by themselves and that will not provide future improvements or features is not a valid option. Consequently this is our first filtering criterion. We consider as abandoned those frameworks without new versions (even minor bug fixes) or papers published in the past 5 years. This criterion eliminated eight frameworks, specifically: jaga, hotframe, templar, MAFRA, DREAM, Discropt and UOF.
- Optimization tools to be evaluated must be frameworks implemented in general purpose Object Oriented languages (such as Java or C++). They must provide a general design where user-defined classes are integrated in order to produce an optimization application for solving the problem at hand. There are useful optimization tools that do not meet those requirements and are consequently out of the scope of this article, but might be studied in a similar comparative research work. This criterion eliminated three optimization tools: Evolutionary::Algorithm, PISA, Comet and OptQuest.
- MOFs must support at least two different optimization techniques (we consider multi-objective variants of techniques as different techniques). Otherwise, they are considered specific applications, even if they can adapt to various problems using the mechanisms that characterize OO frameworks. This criterion eliminated nine MOFs, namely: evolvica, n-genes, GALib, GAPlayground, Hypercube, JGAP, Open Beagle, jmetal, watchmaker.
- Those frameworks for which an executable version or source code with its documentation could not be obtained were also eliminated (after contacting authors and requesting from them a valid version). This criterion eliminated four frameworks, namely: iOpt, JDEAL, OptQuest and MAGMA.

Table 2 shows the final set of frameworks compared along with their specific versions and web sites.

As a consequence, only a subset of possible optimization tools has been evaluated. In spite of the considerable effort during the development of this work, and that the MOFs have been chosen based on well-defined and consistent filtering criteria, some metaheuristic optimization libraries of great practical interest did not qualify and therefore have not been included in this study (e.g. JGAP, Hypercube, Watch-maker or Comet).

Table 3 Areas of interest and comparison characteristics

Area	Characteristic	Rel. RQ
C1 Metaheuristic techniques	C1.1 Steepest descent/hill climbing	RQ1
	C1.2 Simulated annealing	
	C1.3 Tabu search	
	C1.4 GRASP	
	C1.5 Variable neighborhood search (VNS)	
	C1.6 Evolutionary algorithms	
	C1.7 Particle swarm optimization	
	C1.8 Artificial immune systems	
	C1.9 ACO	
	C1.10 Scatter search	
	C1.11 Multi-objective metaheuristics	
C2 Adaption to the problem and its structure	C2.1 Solution enconding	RQ2
	C2.2 Neighborhood structure definition	
	C2.3 Auxiliary mechanisms supporting population based heuristics (genetic operators)	
	C2.4 Solution selection mechanisms	
	C2.5 Fitness function specification	
	C2.6 Constraint handling	
C3 Advanced characteristics	C3.1 Hybridization	RQ3
	C3.2 Hyper-heuristics	RQ4
	C3.3 Parallel and distributed computing	RQ5
C4 Global optimization process support	C4.1 Termination conditions	
	C4.2 Batch execution	
	C4.3 Experiments design	
	C4.4 Statistical analysis	RQ6
	C4.5 User interface and graphical reports	
	C4.6 Interoperability	
C5 Design, implementation and licensing	C5.1 Implementation language	RQ8
	C5.2 Licensing model	RQ7
	C5.3 Platforms availability	RQ8
	C5.4 Usage of soft. eng. best practices (test, design patterns, UML)	RQ9
	C5.5 Size (classes and packages/modules)	
C6 Documentation and support	C6.1 Sample problems types	RQ10
	C6.2 Articles and papers	
	C6.3 Documentation	
	C6.4 Users and popularity	

3.4 Comparision criteria

Evaluating a software tool usually implies understanding and balancing competing concerns regarding the new technology. In this sense, the proposed comparative criteria cover six areas of interest that in turn are subdivided into 30 specific characteristics (and an additional subdivision level that comprises 271 features). Table 3 shows the areas and corresponding set of characteristics in this study, along with the associated research question that we intend to answer through the evaluation of each characteristic.

Table 3 covers a wide range of concerns, from MOF-specific characteristics such as supported metaheuristic techniques or solution encoding (covered in areas C1, C2 and C3), to general concerns such as usability, documentation and licensing model (covered in areas C4, C5 and C6). Consequently, the use of this six areas allows us to easily discern the interesting features on the three usage

contexts described in Sect. 2.1. This benchmark, by focusing on key areas and features for each context, provides a general view of the state of the art MOFs and provides a global assessment. Specifically, these areas are directly related to our research questions:

- Area C1 is related to RQ1, establishing a set of metaheuristic techniques and variants to be supported by MOFs. The assessment of this area for each framework allows us to answer both RQ1 and its sub-questions.
- Area C2 is related to RQ2; the characteristics of this area describe the possible ways of tailoring to the problem through metaheuristic. Thus its assessment provides a basic way of answering RQ2, showing the support provided by each framework and also which tasks are the responsibility of the user.
- Area C3 is related to RQ3, RQ4 and RQ5, grouped as advanced capabilities support.
- Area C4 is related to RQ6, by defining different kinds of additional tools that are (or could be) supported by MOFs.
- Area C5 is related to RQ7, RQ8 and RQ9 showing the platforms and programming languages supported by each framework, along with the use of software engineering best practices.
- Area C6 is related to RQ10, by defining characteristics that assess the issues concerning the sub-questions of RQ10.

As there are different kinds of characteristics, a proper quantification of the facilities provided by MOFs is a complex issue. Sometimes it is meaningless to use quantitative values for assessing certain characteristics (e.g. it makes no sense to associate a quantitative value to the language in which the MOF is implemented). Therefore, for some characteristics we avoid defining metrics, treating them simply as attributes of MOFs which might be relevant to users. In other cases (such as MOF size), the characteristics have been left out of the comparative analysis because they do not affect the research questions. However, the information harvested can be useful for further analysis.

In our comparative approach, we have attempted to obtain a knowledge base about real capabilities provided by MOFs which are as objective as possible. In so doing, each characteristic has been defined, and a set of features is identified to evaluate its support (with minor exceptations). Features are defined taking into account themaximum possible support that could provide an ideal MOF, not the current state of the art MOFs in order to identify gaps, and answer RQ11. Consequently, there are characteristics that are not fully supported by any MOF and even some for which current support is nearly non-existent. In case we need a subjective criteria, we have adopted the perspective of the research-use context (cf. Sect. 2) and the research questions stated. We are working on three levels: areas, characteristics and features; where characteristics are aggregated into areas and various features are used to evaluate individual characteristics. For each feature and MOF a value is measured with two methods: First, features corresponding characteristics of areas C1–C4 are evaluated using a binary true/false value avoiding subjectivity on the value assignment. This information is defined as feature coverage and is the base of a more general evaluation that provides a global quantitative value for each characteristic and area. Second, areas C5 and C6 respresent non-functional characteristics corresponding to transversal aspects that cannot be measured in an objective way; as a consequence, each feature is defined with a score marked by the research use context.

A specific value has been given to each characteristic based on these features. In so doing, a weighting that defines the contribution of each feature to the general support of the characteristic has been set ("Weight" column of Table 4). In the same way; each area is measured based on a weighted sum of the evaluation of its corresponding characteristics. The proposed weights range from 0.0 to 1.0, meaning none and full contribution to characteristics support, respectively.

Three different types of metrics have been devised:

- Uniform: weighting is associated evenly to each feature of the characteristic. This metric type is usually associated with variants or features with no clear predominance in terms of popularity or performance.
- Proportional: a basis feature is given a significant weight (usually 0.5) and the remaining weight is evenly associated with the other features of the characteristic. This metric type is associated with a characteristic with a more useful feature with some rare variants or additional features.
- Ad Hoc: weighting is associated with features based on specific author criteria.

It is important to note that we have set weights from a research use context on optimization problem solving; however in other specific scenarios such as teaching, or industrial problem solving, weights could vary in order to reflect the exact importance of features, characteristics and areas on those contexts. This mechanism allows customized versions of the comparative study and tailored conclusions. This information is published as a public google documents spreadsheet at: http://www.isa.us.es/MOFComparison (moreover, this document contains comments about cover of features and why some features are assessed as partially supported by MOFs). In this way data can be verified and reused, and weights can be redefined.

Table 4 Coverage of features in area C1

Area	Characteristic	Feature	Weight	ECJ	ParadisEO	EvA2	FOM	JCLEC	OAT	Opt4j	EasyLocal	HeuristicLab	MALLBA	Sum
C1 Supported Metaheuristics	**SD/HC**	Basic Implementation	0.5	✔	✔	✔	✔		✔		✔	✔	✔	8
		Multi-Start	0.5	✔	✔	✔	✔		✔		∼	✔	✔	7.5
	SA	Basic Impl.	0.5		✔	✔	✔			✔	✔	✔	✔	7
		Lineal Annealing	0.1		✔		✔					✔		3
		Exponential/Geometric Annealing	0.1		✔		✔				✔	✔	✔	5
		Logaritmic Annealing	0.1				✔							1
		Metropolic Acceptance	0.1		✔	✔	✔			✔	✔	✔	✔	7
		Logistic Acceptance	0.1											0
	TS	Basic Impl.	0.3		✔		✔				✔	✔		4
		Recent Features/Moves Based Tabu Memory	0.2		✔		∼				✔	✔		3.5
		Frecuency Based Tabu Memory	0.3				✔				✔			2
		Basic Aspiration Criteria	0.2		✔		✔				✔	✔		4
	GRASP		1				✔							1
	VNS	Basic VNS (VNS)	0.2		✔		✔							2
		Variable Neighborhood Descent (VND)	0.2								✔			1
		Reduced VNS (RVNS)	0.2											0
		VNS with Decomposition	0.2											0
		Skewed VNS (SVNS)	0.2											0
	EA	Basic EA Implementation of GA	0.2	✔	✔	✔	✔	✔	✔	✔		✔	✔	9
		Basic EA Implementation of ES	0.2	✔	✔	✔		✔	✔	✔		✔	✔	8
		Basic EA Implementation of GP	0.2	✔	✔	✔		✔	✔	✔		✔		7
		GAVaPS	0.05											0
		Diploid Individuals support	0.05				✔							1
		Coevolution support	0.1	✔										1
		Differential evolution	0.1	✔		✔			✔	✔				4
		Niching Methods	0.1	∼		✔		✔				✔		3.5
	PSO	Basic Implementation	0.3	✔	✔	✔				✔		✔	✔	6
		Discrete Variable Support	0.2		✔									1
		Customizable Dynamic Equations	0.2		✔	✔				✔				3
		Topologies	0.2		✔	✔						✔		3
		Lifetime support	0.1											0
	AIS	CLONAG	0.25						✔					1
		optIA	0.25											0
		Immune Networks	0.25											0
		Detritic Cell Algorithms	0.25											0
	ACS	AS	0.1				✔		✔				✔	3
		ACS	0.2				✔		✔				✔	3
		MMAS	0.4				✔		✔					2
		ASrank	0.2						✔					1
		API	0.1											0
	Scatter Search	Basic. Impl.	1			✔								1
	Multi-Objective Metaheuristics	PGA	0.0625											0
		MOGA	0.0625		✔	✔		✔						3
		NSGA	0.0625		✔	✔								2
		NSGA-II	0.0625	✔	✔	✔		✔		✔		✔		6
		NPGA	0.0625											0
		SPEA	0.0625			✔								1
		SPEA-II	0.0625	✔		✔		✔		✔				4
		PAES	0.0625											0
		PESA	0.0625			✔								1
		PESA-II	0.0625			✔								1
		MOMGA	0.0625											0
		ARMOGA	0.0625											0
		Multiobjective AS/ACO	0.0625											0
		Multiobjective PSO	0.0625											0
		POSA	0.0625											0
		MOSA	0.0625											0
		Feature Support Count		10.5	20	19	17.5	7	11	10	9.5	16	10	130.5
		Weighted Sum		0.207	0.381	0.394	**0.450**	0.081	0.259	0.175	0.264	0.324	0.245	

Table 5 Coverage of features in area C2

Area	Char.	Feature	Weight	ECJ	ParadisEO	EvA2	FOM	JCLEC	OAT	Opt4j	EasyLocal	Heur-Lab	MALLBA	Sum
C2 Problem Adaption/Encoding	Solution encoding	Bit Vector	0,04	✔	✔	✔	✔	✔	✔	✔		✔	✔	9
		Bit Matrix	0,04											0
		Bit Map	0,04	✔	✔	✔				✔				4
		Bit GrayCode Vector	0,04											0
		Integer Vector	0,04	✔	✔	✔	✔	✔	✔	✔		✔	✔	9
		Integer Matrix	0,04											0
		Integer Map	0,04	✔	✔	✔				✔				4
		Real Vector	0,04		✔	✔		✔	✔	✔		✔	✔	7
		Real Matrix	0,04											0
		Real Map	0,04		✔	✔								2
		String Vector	0,04	✔		✔				✔				3
		String Matrix	0,04											0
		String Map	0,04	✔	✔	✔				✔				4
		Permutation	0,04			✔				✔		✔	✔	4
		Expression Tree	0,04	✔	✔	✔		✔		✔		✔		6
		State-Machine / Graph	0,04	✔				✔						2
		Combined / Arbitrary representations	0,4	✔	✔	✔		✔		✔		✔		6
	N. D.	Neighborhood structures for solution encodings	0,6		✔						✔	✔		3
		Neighborhood structures for composite solution encodings	0,3		✔						✔	✔		3
		Complex Neighborhood structures	0,1											0
	E/A Auxiliary Methods	(1PX) Integer/Bit Vector One Point Crossover	0.0102	✔	✔	✔		✔	✔	✔		✔	✔	8
		(NPX) Integer/Bit Vector n-Points Crossover (including 2)	0.012	✔	✔	✔		✔	✔	✔		✔	✔	8
		(UX) Integer/Bit Vector Uniform crossover	0.0102	✔	✔	✔		✔		✔		✔	✔	7
		(HCX) Half Uniform Crossover	0.0102			✔		✔					✔	3
		(PNCTX) Integer/Bit Vector Punctuated Crossover	0.0102											0
		(SX) Integer/Bit Vector Shuffled Crossover	0.0102			✔								1
		(RRX) Integer/Bit Vector Random Respectfull Recombination	0.0102											0
		(R1PX) Real Vector One Point Crossover	0.0071	✔	✔	✔		✔				✔	✔	6
		(RNPX) Real Vector n-Points Crossover (including 2)	0.0071	✔	✔	✔		✔				✔	✔	6
		(RUX) Real Vector Uniform Crossover	0.0071	✔	✔	✔		✔				✔	✔	6
		(RAX) Real Vector Arithmetic Crossover	0.0071	✔	✔	✔		✔		✔		✔		6
		(RHX) Real Vector Heuristic Crossover	0.0071									✔		1
		(RSPLX) Real Vector Simplex Crossover	0.0071											0
		(GEOMX) Geometric crossover	0.0071											1
		(BLX-alpha) Blended crossover	0.0071	✔	✔	✔		✔		✔		✔		6
		(F-BSX) Real Vector Fitness Scaning Based Crossover	0.0071					✔				✔		2
		(DMPX) Real Vector Diagonal Multi-Parental Crossover	0.0071					✔						1
		(POX) Permutation Davis order Crossover	0.01									✔	✔	2
		(PPMX) Permutation Partially mapped Crossover	0.01			✔						✔	✔	3
		(P2OX) Permutation Order 2 Crossover	0.01									✔	✔	2
		(PPX) Permutation Position Crossover	0.01							✔		✔	✔	3
		(PDUX) Permutation Davis Unifform Crossover	0.01							✔				1
		(PMPX) Permuation Maximal Preservative Crossover	0.01									✔		1
		(PCX) Permutatio Cycle Crossover	0.01									✔	✔	2
		(TCX) Tree Cramer Crossover	0.023		✔	✔		✔				✔		4
		(TKX) Tree Koza Crossover	0.023	✔	✔	✔		✔						4
		(TMX) Tree Montana Crossover	0.023					✔						1
		(SMFc) State Machine Fogel Crossover	0.0178											0
		(SM1Pc) State Machine Zou & Grefenstette Crossover	0.0178											0
		(SMUc) State Machine Uniform Crossover	0.0178											0
		(SMJo) State Machine Join Operator	0.0178											0
		(CSc) Composite/Combined Solution Encoding Crossover	0.07		✔	✔		✔		✔		✔		5
		(CPXc) Complex Crossover operator	0.07											0
		(Bm) Binary/Integer Vector Basic Mutation	0.06	✔	✔	✔		✔		✔		✔	✔	7
		(RUm) Real Vector Unifform Mutation	0.01		✔	✔		✔		✔		✔	✔	6
		(RNm) Real Vector Normal Mutation	0.01	✔	✔	✔		✔		✔		✔		6
		(RCM) Real Vector Cauchy Mutation	0.01											0
		(RLm) Real Vector Laplace Mutation	0.01											0
		(RSDm) Real Vector Schwefel Dynamic Mutation	0.01											0
		(RFDm) Real Vector Fogel Dynamic Mutation	0.01											0
		(P2Optm) Permutation 2-Opt mutation	0.01									✔		1
		(P3Optm) Permutation 3-Opt mutation	0.01									✔		1
		(PKOptm) Permutation K-Opt mutation	0.01											0
		(PSWm) Permutation Swap Mutation	0.01							✔		✔	✔	3
		(PIm) Permutation Insertion Mutation	0.01							✔		✔		2
		(PSCm) Permutation Scramble Mutation	0.01							✔		✔		2
		(TGm) Tree Grow Mutation	0.015	✔		✔		✔				✔		4
		(TSHm) Tree Shrink Mutation	0.015	✔				✔				✔		3
		(TSWm) Tree Swapping Mutation	0.015	✔	✔	✔		✔				✔		5
		(TCm) Tree Cycle Mutation	0.015	✔										1
		(SMBm) State Machine Basic Mutation Operator	0.06											0
		(CSm) Composite/Combined Solution Encoding Mutation	0.06		✔	✔		✔		✔		✔		5
		(CPXm) Complex Mutation operator	0.06									✔		1
		(DEm) Dynamic probaility mutation	0.06		✔									1
	Solution Selection	Elitist Selector (Es)	0.07	✔	✔	✔	✔	✔	✔	✔		✔	✔	9
		Expected Value Selector (EVs)	0.066											0
		Elitist Expected Value Selector (EEVs)	0.066											0
		Proportional Selector (Ps)	0.07	✔	✔	✔	✔	✔	✔			✔	✔	8
		Determinist Sampling Selector (DSs)	0.066											0
		Remaining Stocastic Sampling Selector (RSSs)	0.066			✔								1
		Stocastic Tournament Selector (STs)	0.066	✔	✔	✔	✔	✔	✔	✔		✔	✔	9
		Stocastic Universal Sampling Selector (SUSs)	0.066	✔		✔								2
		Linear Ranking Selector (LRs)	0.066			✔		✔				✔		3
		Mu.Lambda Selector (MLs)	0.066	✔	✔	✔	✔	✔	✔			✔		7
		Mu+Lambda Selector (M+Ls)	0.066	✔	✔	✔	✔	✔	✔			✔		7
		Threshold Selector (Ths)	0.066											0
		Boltzman Selector (Bs)	0.066	✔		✔		✔						3
		Random Selector (RNDs)	0.066	✔	✔	✔	✔	✔		✔		✔	✔	8
		Combined Selector (CMBs)	0.066	✔	✔		✔							3
	O.F.S.	DSL for Objective Function Definition (DSLof)	0.33									∼		0.5
		GUI & Graph. tools for Objective Function Definition (GUIof)	0.33									∼		0.5
		Interactive Objective Function Definition (Iof)	0.33											0
	Const. H.	Explicit Constraint Modeling	0.1				✔							1
		Penalization on Objective Function	0.3		✔	✔	✔							3
		Individual Constraint Solution Repairing Mechanism	0.3				✔							1
		Global Solution Repair Mechanism	0.3			✔	✔							2
			Feature Support Count	33	36	43	12	36	10	29	2	48	23	272
			Weighted Sum	0.254	0.413	0.306	0.090	0.258	0.078	0.210	0.150	**0.484**	0.102	

Table 6 Coverage of features in area C3

Area	Characteristic	Feature	Weight	ECJ	ParadisEO	EvA2	FOM	JCLEC	OAT	Opt4j	EasyLocal	HeuristicLab	MALLBA	SUM
C3 Advanced Metaheuristic Characteristics	**Hybridization**	(BEMIh) Batch Execution Multiple Instances Hybridization	0.1	✔	✔		~	✔	✔	✔	✔	✔		7.5
		(BEMMh) Batch Execution Multiple Metaheuristics Hybridization	0.2				~				✔	✔		2.5
		(IMMh) Interleaved Multiple Metaheuristics Hybridization	0,6	~	✔		~	~				✔	~	4
		(Ch) Combined Hibridizacion	0.1				~							0.5
	Hyper-Heuristics	Pre-implemented Parameter Setting meta-problem	0.25				✔							1
		Pre-Implemented Technique selection meta-problem	0,25				✔							1
		Pre-implemented Operators/Low level heuristics selection meta-problem	0.25											0
		Pre-implemented solution encoding selection meta-problem	0.25											0
	Paral. & Dist,	(IPDM) Independent Parallel & Distributed Metaueristics execution	0.2	✔	✔	✔							✔	4
		(SSPDM) Shared Solutions (or Populations) Parallel & Distributed Metaheuristics	0.2	✔	✔	✔							✔	4
		(LSPDNM) Local Search using Parallel & Dist Neighborhood exporation . Metah.	0,2											0
		(PDPEDM) Parallel & Distributed Population Evaluation Metah.	0.2	✔	✔	✔							✔	4
		(PDESSM) Paralell & Dist. Evaluation of Single Solution Metah.	0.2	✔	✔								✔	3
			Feature Support Count	5.5	6	3	4	1.5	1	1	2	3	4.5	31.5
			Weighted Sum	0.400	**0.500**	0.200	0.333	0.133	0.033	0.033	0.100	0.300	0.367	

Moreover, for areas C1, C2, C3 and C4, tables showing feature cover per framework (and weights associated as an additional column) are provided in this article, corresponding to Tables 4, 5, 6 and 7, respectively. In the following sections we describe each area, its characteristics, corresponding features and weights and global scores obtained by each MOF. Tables 9, 10 and 11 in the appendix show these scores in detail.

4 Metaheuristic techniques (C1)

The main feature of any MOF is the set of supported metaheuristics. A characteristic is defined for each metaheuristic, which indicates the support the MOFs provide for it.

4.1 Characteristics description

A set of 11 characteristics has been defined, with 52 features, comprising most major metaheuristics proposed in the literature, either based on intelligent search (characteristics C1.1, C1.2, C1.3 and C1.5), on solution building (C1.4, C1.9 and C1.10) or populations (C1.6, C1.7, C1.8, C1.9 and C1.10). Furthermore, we have evaluated the incorporation of techniques for multi-objective problem solving (C1.11). Metaheuristics and variants described in this section have been chosen following Glover and Kochenberger (2002) and some technique-specific references such as Aarts and Lenstra (1997), Back et al. (1997) and Clerc (2006). We next describe in detail each of these characteristics; the cover of features by frameworks and their weights are shown in Table 4.

C1.1 Steepest descent/hill climbing This technique searches successively for the best neighbor solution until reaching a local optimum. This technique is commonly used for hybridization (c.f. characteristic C3.1). *Metric*: We have defined two different features: (1) basic implementation until local optimum is found, and (2) multi-start implementation using a random initial solution when local optimum is found. A uniform metric is used (with each feature weighing 0.5).

C1.2 Simulated annealing This technique is inspired by the natural process of slow cooling used in metallurgy. It was proposed by Kirkpatrick et al. (1983). We have defined a feature associated with the basic implementation of this technique and features for some of its variants.

Table 7 Coverage of features in area C4

Area	Characteristic	Feature	Weight	ECJ	ParadisEO	EvA2	FOM	JCLEC	OAT	Opt4j	EasyLocal	HeuristicLab	MALLBA	SUM
C4 Optimization Process Support	**Finalization Conditions**	Max iterations terminator	0.1	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	10
		Fitness value terminator	0.1	✔	✔	✔	✔		✔	✔	✔		✔	8
		Max execution time terminator	0.1	✔	✔	✔	✔		✔				✔	6
		Max objective function evaluations terminator	0.1	✔	✔	✔	✔		✔	✔			✔	7
		Max iter./exec. time/obj. func. ev. without improvement terminator	0.1		✔	✔	✔	✔	~					4.5
		Composite Logical combinations terminator	0.1		✔	✔	✔							3
		Technique specific	0.1			~					✔	✔		2.5
	Batch processing	Automated repetition of a single optimization task	0.2	✔		✔	✔	✔	✔	✔	✔	✔		8
		Automated repetition of a task varying parameters of the technique	0.2					✔	~		✔			2.5
		Automated repetition of different tasks (possibly varying the parameter of each one)	0.2	✔							✔	✔		3
		Automated repetition of different tasks (possibly varying the parameter of each one) over different instances of the problem	0.2											0
		Random execution of planified tasks	0.2											0
	Experiment Design	Hypothesis Definition Support	0.2				~	✔	✔					2.5
		Experiment Modelling Support (Definition of Dependent and Independent Variables)	0.2				~	✔	✔					2.5
		Experiment Design Suppoort (Factorial. fractional. latin squares. nested. etc.)	0.2				~							0.5
		Experiment Execution Support	0.2				✔	✔	✔					3
		Experiment Execution Plan Generation Support	0.2				~	~	~		~	~		2.5
		Experimental Design Systems import/export	0.2											
	Statistical Analysis	T-Student	0.1				✔	✔	✔		✔			4
		One way ANOVA	0.1				✔	✔	✔					3
		Two way ANOVA	0.1				✔	✔	✔					3
		Multi-way ANOVA	0.1					✔	✔					2
		Wilcoxon	0.1				✔	✔	✔					3
		Mann-Withney	0.1				✔	✔	✔		✔			4
		Kolmogorov-Smirnov	0.1						✔					1
		Statistical Analysis Systems export/import	0.3			~								0.5
	GUI	Design & Usability	$0.1\widehat{6}$	0.4		0.8	0.2	0.5	1	1		1		4.9
		Metaheuristic techniques configuration & spec.	$0.1\widehat{6}$	0.5		0.5	1		0.5	0.8		1		4.3
		Problem definition. modelling and data import	$0.1\widehat{6}$			0.5			0.5			1		2
		Support for planning of optimization tasks	$0.1\widehat{6}$	1					1			1		3
		Graphical Support for methodological guidance	$0.1\widehat{6}$			0.4			0.5	0.4		1		2.3
		Charting	$0.1\widehat{6}$	1		1	1	1	1	0.5		1	0.5	7
	Interoperability	Data Export	0.25	✔	~	~			~			✔		3.5
		Data Import	0.25	✔	~	~			~			✔		3.5
		Web Services Facade	0.25											0
		XML usage for projects & config	0.25	~				✔				✔		2.5
Feature Support Count				12.4	8	12.7	17.2	18.1	23.2	7.7	8.2	14.5	5.5	127.5
Weighted Sum				0.368	0.192	0.347	0.411	0.470	**0.582**	0.208	0.165	0.458	0.131	

Specifically, we have evaluated variants on the cooling scheme: linear and exponential scheme proposed by Kirkpatrick et al. (1983), logarithmic scheme defined by Geman and Geman (1987) and schemes based on thermodynamics (defined by Nulton and Salamon (1988) and Andresen and Gordon (1994). Addtionally, we have evaluated the variants on the acceptance criterion of worsening solutions: metropolis acceptance proposed by Kirkpatrick et al. (1983) and logistic acceptance (Goldberg 1990). *Metric*: A proportional metric is used, where the basic implementation has a weight of 0.5, each cooling scheme variant weighs 0.1 and each acceptance criterion variant weighs 0.1.

C1.3 Tabu search Basic ideas of tabu search were proposed by Glover (1989). This technique uses procedures designed to cross boundaries of local optima by establishing an *adaptive memory to guide the search process*, avoiding searching in circles through the solution space. This memory scheme is implemented using data structures that store either visited solutions (tabu list) or components of those solutions and even the frequency of appearance of each solution component. In order to avoid discarding promising solutions, *aspiration criteria* is implemented, for instance it allows the selection of a tabu solution if it improves the current solution by a percentage. *Metric*: An ad hoc. metric is used to assess this characteristic. A feature representing the basic implementation of this technique using a tabu list weighs 0.3, components recency memory feature weighs 0.2, components frequency-based memory weighs 0.3 and aspiration criteria feature weighs 0.2.

C1.4 GRASP This technique was proposed by Feo and Resende (1989, 1995), and specifies two stages for each interaction: (1) solution building adding components in a stochastic greedy way (one among the best choices for each component is selected sequentially until the solution is built) and (2) a local search is performed based on the built solution. Candidate components of the first stage are sorted and evaluated using a greedy value function, generating a *restricted candidate list* (RCL). *Metric*: A unique feature indicating support for this technique is used, evaluated as a binary value indicating if the framework provides some kind of support for it.

C1.5 Variable neighborhood search (VNS) This technique proposes a systematic exchange on neighborhood structure in a local search context. It was propose by Mladenović (1995). Many variants of this technique have been proposed in literature, and based on them we propose the following features: (1) Original proposal implementation (VNS); (2) Variable neighborhood descent (VND); (3) Reduced VNS (RVNS); (4) Variable neighborhood decomposition search (VNDS) by Hansen et al. (2001) and (5) Skewed VNS by de Souza and Martins (2008). *Metric*: A uniform metric is used (having a weight of 0.2 for each one of the five features).

C1.6 Evolutionary algorithms (EA) There are many techniques based on principles of biological evolution that can be called evolutionary algorithms. These techniques can be divided into three independently developed approaches: *evolutionary strategies (ES)* proposed by Rechenberg (1965), *evolutionary programming* according to Fogel et al. (1966) and *genetic algorithms* as developed by Holland (1992). These techniques present different variants based on the elements used for adapting to the problem (some of them present in other techniques) and some additional variation points. In order to create a global and coherent comparative criteria, we have identified various characteristics for those variations. Remarkably, the selection of individuals for crossover and survival is independent of the solution encoding; thus, frameworks can provide implementations using different selection criteria and can reuse them, since mechanisms for selecting solutions are used in various metaheuristics. We have created a characteristic for evaluating the support for solution selection (C2.4). Crossover and mutation mechanisms are dependent on the representation scheme used, and the efficiency of a specific mechanism will strongly depend on the problem to be solved. Consequently, we have created an associated characteristic in the area of adaptation to the problem (C2.3).

Thus, this feature (C1.6) only measures the support provided by frameworks for general evolutionary algorithms, without taking into account solution encoding capabilities, the genetic operators nor the selection mechanisms available. Of the many variants that have been proposed in literature for the basic evolutionary algorithm, we take into account (1) the use of variable population sizes (e.g. GAVaPS Arabas et al. (1994), (2) niching methods (commonly used to solve multi-modal optimization problems), (3) individuals that encode more than one solution to the problem (usually diploid). Goldberg and Smith (1987), (4) coevolution of multiple populations in competitive and cooperative environments as described in (Back et al. 1997, Chapter on Coevolutionary Algorithms) and (5) differential evolution as developed by Price et al. (2005). Variants (1), (3) and (4) as well as some versions of (2) can be implemented regardless of the problem, the solution encoding or the operators used.

Metric: An ad hoc. metric is defined to assess this characteristic. Three features have been identified to evaluate the support of the different evolutionary approaches, with each feature weighing 0.2. With regard to the variants, (1) weighs 0.05, (2) weighs 0.1 and (3) weighs 0.05, (4) weighs 0.1 and (5) weighs 0.1. We evaluate variants as binary variables, in terms of the support afforded by frameworks.

C1.7 Particle swarm optimization (PSO) This technique is a stochastic algorithm is inspired by the behavior of birds flocking and fish schooling. The algorithm iteratively modifies a population of solutions (named the swarm), whose interactions are expressed as equations. Solutions in the swarm are represented as particles in an n-dimensional space with a position and speed. The original proposal by Kennedy and Eberhart (1995) has been applied successfully to a variety of problems (Clerc 2006; Parsopoulos and Vrahatis 2002a). Moreover, this technique has been adapted to support discrete variables, and different equations to rule swarm interaction have been proposed (Chatterjee and Siarry 2006; Wilke et al. 2007; Vesterstrm and Riget 2002; Rahman et al. 2009). The topology of the neighborhood of particles, i.e. the particles that influence the position of a given particle according to the equations, generate a full set of possible variants. In the original PSO, two different kinds of topologies were defined: (1) global, specifying that all particles are neighbors of each other; and (2) local, specifying that only a specific number of particles can affect a given particle. In Kennedy and Mendes (2002) a systematic review of neighborhood topologies is described, and in Suganthan (1999) the concept of "dynamic" neighborhood topology is proposed. Another interesting variant is the use of a "life time" for solutions in the swarm; after this time solutions are randomized. *Metrics*: We have created a feature to represent the original proposal for real variables and classic equations. It weighs 0.3. Discrete variable support weighs 0.2. Equation customization weighs 0.2. The explicit modeling and support of different neighborhood topologies weighs 0.2. Finally, lifetime support weighs 0.1.

C1.8 Artificial immune systems (AIS) This technique intends to use the structure and operation of biological immune systems of mammals and apply it to solving optimization problems. This technique comprises various proposals: *Clonal Selection algorithms* originally proposed by Nossal and Lederberg (1958) and its variants such as CLONALG, developed by de Castro and Von Zuben (2002) and optIA; *Immune Network algorithms* and *Dentritic Cell algorithms Metrics*: A uniform metric is used to assess this characteristic (with each feature weighing 0.25).

C1.9 Ant Colony System (ACS) This technique, also known as Ant Systems (AS) is a probabilistic optimization algorithm inspired by the food foraging behavior of ants. Ant Systems use a data structure called "pheromone trace" to support communication between ants. In this article the following variants are taken into account: The original proposal of Ant System (AS) and Ant Colony System (ACS) as propsed by Dorigo and Gambardella (1997), Ant System using Rankings (ASrank), Min–Max Ant System (MMAS) according to Stutzle and Hoos (1997) and API as developed by Monmarchè et al. 2000). *Metrics*: An ad hoc metric is defined for this characteristic; corresponding weights are shown in Table 4.

C1.10 Scatter search (SS) This technique was proposed by Glover (1977). It operates on a set of solutions, the *reference set*, by combining existing solutions to create new ones. In contrast to other evolutionary methods like genetic algorithms, scatter search is based on systematic designs and methods, where new solutions are created from the linear combination of two solutions of the reference set, using strategies for search diversification and intensification. *Metrics*: This technique has a unique feature, evaluated as a binary value, which indicates if the framework provides it with some kind of support.

C1.11 Multi-objective metaheuristics The technique most commonly used to solve multi-objective optimization problems is EA (Dreo et al. 2005). However, some variants of other techniques have also been taken into account: SA (MOSA as proposed by Ulungu et al. (1999) and PASA as developed by Suresh and Mohanasundaram (2004), PSO (Parsopoulos and Vrahatis (2002b) and ACO (Iredi et al. (2001). Those variants have been adapted to solve multi-objective optimization problems. Regarding the EA variants to evaluate, we have taken into account the original proposal by Goldberg (1989) (PGA), MOGA as proposed by Fonseca and Fleming (1993), Non Dominated Sorting Genetic Algorithm (NSGA and NSGA-II) as developed by Deb et al. (2002), Niched Pareto Genetic Algorithm (NPGA) according to Horn et al. (1994), Strength Pareto Evolutionary Algorithm (SPEA and SPEA-II (Zitzler and Thiele 1999; Zitzler et al. 2001), Pareto Envelope based Selection Algorithms (PESA and PESA-II) (Corne et al. 2000), Pareto-archived ES (PAES) (Knowles and Corne 2000), multi-objective messy GA (MOMGA) (Van Veldhuizen and Lamont 2000) and ARMOGA (Sasaki 2005). *Metrics*: A uniform metric is used to assess this characteristic.

4.2 Assessment and feature coverage analysis

In order to assess this area, we have crawled the source code, user and technical documentation and user interface of each selected MOF. Table 4 shows the feature coverage of Area C1, along with the weight corresponding to each feature in its associated characteristic. The last column of this table shows the number of MOFs supporting each feature. The last two rows show the number of features supported by each MOF and a score computed as the weighted sum of features supported divided by the number of characteristics in the area. It is remarkable that only four features of this area are supported by a minimum of six out of the ten MOFs under study. Only the core techniques (namely SD/HC, SA and EA) have features in this range. This shows a dispersion in techniques supported by MOFs,

and consequently, implies that users have little choice if they want to use techniques out of this set. Thus MOF is determined by the technique the user wants to apply.

An interesting fact shown in Table 4 is that 39% of features in this area are not supported by any MOF. Consequently, current MOFs have ample scope for improvement in this area. Moreover the distribution of those unsupported features imply that MOF technique support is aimed at the basic variants. This does not apply to the techniques in the core set, TS and some multi-objective variants, since those techniques only have features that represent variants with more than 30% of MOFs supporting them. ParadisEO, Eva2 and FOM have the highest number of features supported in this area, followed by HeuristicLab and OAT.

4.3 Comparative analysis

FOM is the framework that provides a broader support of optimization techniques, closely followed by Paradiseo, Eva2 and HeuristicLab. It is important to note that more features supported do not imply more techniques supported, since some techniques have a number of variants and specific heuristics implementations modeled as features. The weights contribute to express this fact by making that each technique sums a total score of 1 unit once the features are weighted. Figure 2 shows a stacked columns diagram for the C1 area characteristics. Each color or texture represents a metaheuristic and each column the support provided by a MOF. The number of techniques supported by each MOF can be easily identified by the number of different colors/textures in its column. The degree of support for each technique is expressed through each color's height (computed based on the weight associated to their features and the feature support information shown). The total height of each column provides a measure of the global support of metaheuristics by its corresponding MOF.

The almost universal support for EA and the lack of support for AIS are remarkable. SS is only supported by Eva2, and GRASP is only supported by FOM. Other metaheuristics with very little support are ACO, TS and VNS. This could be due to the complexity of modeling in abstract, the elements involved in their operation and reusing or customizing them (ACO and TS are based on features of solutions, and VNS needs to apply different neighborhood structures). When applying EAs using java; ECJ, JCLEC and EvA2 appear as highly competitive options; while Paradiseo and MALLBA are the MOFs available if the user plans to use C++. In .NET environments, the only option available for applying EAs is HeuristicLab.

We can provide an answer to RQ1 and its sub-questions based on information shown in Table 4 and Fig. 2. characteristics of area 1 summarize the whole set of metaheuristics currently supported by assessed frameworks. Most variants of those techniques are unsupported. The most widely supported techniques are EA, SD/HC and SA, which are supported by more than 60% of asssessed frameworks. Finally, there is no universal MOF, which provides support for all the techniques.

5 Adapting to a problem and its structure (C2)

As stated in the previous section, MOFs provide implementation of metaheuristic techniques for problem solving. They also provide mechanisms to express problems properly in order to apply these techniques. MOFs allow for the adaptation of their supported metaheuristics for better problem solving.

For instance, frameworks can provide appropriate data structures that the techniques can handle. This two-way adaptation (techniques to problem for efficient problem solving, and problems to techniques for proper solution handling and underlying heuristics implementation) is

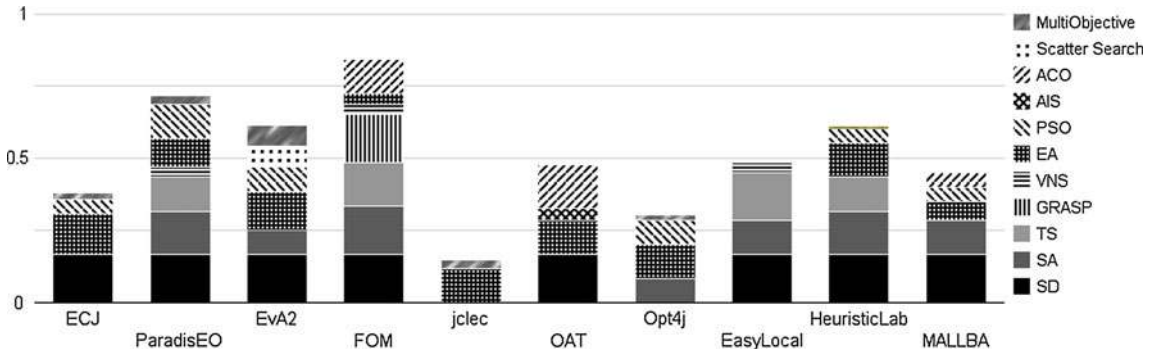


Fig. 2 Stacked bar chart showing MOFs techniques support

basically done in three ways: selecting an appropriate solution representation/encoding, specifying the objective function to optimize and implementing the set of underlying heuristics required by the metaheuristic used to solve the problem.

5.1 Characteristics description

This area evaluates the capabilities provided by MOFs to support this adaption. Characteristic C2.1 aims at assessing capabilities to represent solutions to optimization problems based on the set of data structures provided by frameworks. Characteristics C2.2, C2.3 and C2.4 aim to assess the supported set of underlying heuristics. Characteristic C2.5 aims to assess the capabilities of declarative objective function specification based on the representations assessed in C2.1. Finally, C2.6 aims to assess capabilities of constraint handling. Features and characteristics described in this section have been structured following Back et al. (1997) and Rothlauf (2006) for solution encodings (C2.1), Back et al. (1997) for selection and genetic operators (C2.3 and C2.4), Aarts and Lenstra (1997) for neighborhood definition capabilities (C2.2) and Michalewicz and Fogel (2004) for constraint handling techniques (C2.6). Next we describe in detail each of these characteristics:

- *C2.1 Solution encoding*: Solution encodings are data structures that allow the modeling of solutions for metaheuristic techniques to handle. In this sense, the increased flexibility and the more data structures provided, the lower the effort invested by the users to address problems. *Metric*: In order to evaluate this characteristic, we have taken into account three criteria: provided data structures (vectors, matrices, trees, graphs and maps), data types and information encoding and the ability to use combined representations as described by Rothlauf (2006). A proportional metric is used, where this last feature weighs 0.4. Data types taken into account are bits (with usual or Gray encoding), integers, floating point numbers and strings. The remaining weight is evenly divided among these combination of data type and data structure.
- *C2.2 Neighborhood structure definition*: A proper neighborhood structure definition is a key factor for the success of intelligent search-based heuristics. Neighborhood structure strongly depends on solution representation, and its suitability depends on the problem to be solved and the technique used to solve it (as stated by Aarts and Lenstra (1997). *Metric*: The assessment is divided into three features: pre-defined neighborhood structures provided by MOFs weigh 0.6; neighborhood structures of composite representations weigh 0.3, and a weight of 0.1 is given to complex neighborhood structures that apply different neighborhood structures randomly or based on some rule.
- *C2.3 Auxiliary Mechanisms supporting population-based heuristics (genetic operators)*: Genetic operators are the main underlying heuristics on EA. Their implementation (except for selection operators, evaluated in C2.4) is usually dependent on solution representation; therefore, MOFs must provide the corresponding implementations for their supported representations. Various alternatives for implementing each genetic operator have been proposed in literature as described below. We have relied primarily on (Back et al. 1997, chapter C3.3) to develop the definition and features of this characteristic.

The most common genetic operators are crossover and mutation. Weights have been evenly distributed among all variants provided for each operator. Next, we enumerate the crossover operators proposed in literature for solution encodings of Table 3.

- *Binary and integer vectors:* The original crossover operator was proposed by Holland (1992) and named "one point crossover" (1PX), the generalization of this operator for n crossover points (NPX) was proposed by Jong (1975), uniform crossover (UX) (Ackley 1987), punctuated crossover (PNCTX) (Schaffer and Morishima 1987), shuffled crossover (SX) (Eshelman et al. 1989), half uniform crossover (HCX) (Eshelman 1991) and random respectful crossover (RRX) as proposed by Radcliffe (1991).
- *Floating Point vectors:* Operators 1PX, NPX and UX are in principle applicable to floating point vectors, but they support a set of specific crossover operators for being implemented by MOFs: arithmetic crossover (AX/BLX) (Michalewicz 1994, p 112), heuristic crossover (HX) (Wright 1994), simplex crossover (SPLX) (Renders and Bersini 1994), geometric crossover (GEOMX) (Michalewicz 1994), blend crossover (BLX-alpha) (Eshelman and Schaffer 1993), crossover operators based on objective function scanning (F-BSX) and diagonal multi-parental crossover (DMPX) as proposed by Eiben et al. (1994).
- *Permutations:* Basic crossover operators, such as 1PX, NPX, UX, etc., generate infeasible individuals when using permutation-based representations; it is therefore necessary to design specific operators for such representations, such as order crossover operator (OX) (Davis 1985), partially mapped crossover (PMX) (Goldberg and Lingle 1985), order-2 and position crossover operators (Syswerda 1991), uniform crossover for permutations (UPX) (Davis 1985, p 80), maximal preservative crossover (MPX) (Muhlenbein 1991, p 331), cycle crossover (CX) (Oliver et al. 1987)

and merge crossover (MX) ad defined by Blanton and Wainwright (1993).

- *State machines:* Crossover operators for state machines (SMFx) were initially proposed by Fogel (1964), Fogel et al. (1966) (pp 21–23). In this comparative study, we evaluate those operators and 1PX using a vectorial representation of the state machine (SM1PX) as defined by Zhou and Grefenstette (1986), state one to one state interchange as proposed by Fogel and Fogel (1986), uniform crossover for state machines (SMUX) and the merge operator (SMJO) as defined by Birgmeier (1996).
- *Trees*: There is real difficulty in defining proper crossover operators for trees, and specifically trees representing programs, since generally constraints have to be imposed on their structure, semantics and associate data types. The most common crossover operator for trees were proposed by Cramer (1985). In this comparative, we also considered those defined by Koza (1992) and the adaptations proposed by Montana (1995).
- *Crossover operators for composite representations (CSX):* Crossover operators for individuals using composite representations can be used by applying the corresponding operators to each component of the representation.
- *Composite crossover operators (CMPX):* By assigning a probability (or decision rule) to the application of an operator from a set of valid crossover operators for the representation used, composite crossover operators are possible.

Next we enumerate the mutation operators proposed in literature for the solution encodings of Table 3.

- *Binary and integer vectors:* We have taken into account the original mutation operator proposed by (Holland 1975, pp 109–111).
- *Floating point vectors:* The mutation operator based on an uniform distribution $U(b, -b)$ (RUm) proposed by Davis (1989), the normal mutation operator (RNm) developed by Schwefel (1981), the mutation operators based on Cauchy (RCm) and Laplace (RLm) distribution as proposed by Montana and Davis (1989, Yao and Liu (1996), and the proposals of adaption of mutation ratio according to Schwefel (1981) and Fogel et al. (1991), are the mutation operator for floating vectors that have been considered.
- *Permutations:* The mutation operators for permutations covered by this comparison are 2-opt (P2Optm), 3-opt (P3Optm) and k-opt (PKOptm), simple interchange mutation operator (PSWm) o insertion operator (deleting the item from its original position) of 2 element (PIm) and "scramble mutation operator" (PSCm) (Syswerda 1991).
- *State machines:* The basic mutation operator for state machines is based on the set of its states and transitions, slightly modifying any state or transition as porposed by (Back et al. 1997, C3.2.4).
- *Trees:* The mutation operators for trees covered by this comparison are those proposed by Angeline et al. (1996): (1) grow mutation operator (TGm); (2) reduction mutation operator (TSHRm); (3) swapping mutation operator (TSWm); (4) cycle mutation operator (TCm); and (5) the gaussian mutation operator for numeric nodes (TGNm). The adaption proposed by Montana (1995) is also taken into account.
- *Mutation operators for composite representations (CSm):* Mutation operators for individuals using composite representations can be created by applying the corresponding operators to each component of the representation.
- *Composite Mutation operators (CPXm):* Composite mutation operators are possible through the assignment of a probability (or decision rule) to the application of an operator from a set of valid operators for the representation used.
- *Mutation operators using dynamic probability (DEm):* There exists empirical evidence (Fogarty 1989) that the use of a dynamic mutation probability that decreases exponentially along the evolution process, improves the performance of EAs. In this comparison, we have taken this feature it into account.

Metric: A uniform metric is defined, where the weight evenly distributed among mutation (0.5) and crossover (0.5) operators. For each variant of those operators, weights are uniformly associated.

- *C2.4 Selection mechanisms*: This characteristic assesses the support for the different criteria for solution selection. The problem of selecting a subset amongst a larger set of solutions appears as a specific heuristic on a number of metaheuristic techniques (SA, TS, EA, ACO, etc.). By applying OO analysis and design methodologies and specifically the strategy design pattern[1], objects encapsulating the solution selection logic are called *selectors*. The use of different selectors allows for controlling the trade-off between exploration and exploitation of the search space. As a consequence, performance of metaheuristic techniques in finding good solutions to problems is drastically affected by those selection criteria. Usually, selection criteria are

[1] The strategy pattern is a particular software design pattern, whereby algorithms can be selected at runtime. This pattern is useful for situations where it is necessary to dynamically swap the algorithms used in an application. The strategy pattern is intended to provide a means to define a family of algorithms, encapsulate each one as an object and make them interchangeable Gamma et al. (1994).

based on the adequacy of solutions, but there is a wide set of possibilities, from random to elitism (stochastic and deterministic).

In this comparison the following criteria are taken into account: (1) elitist selector (Es), that picks the best solutions, and its variants; expected value selector (EVs) and elitist expected value selector (EEVs) as proposed by Jong (1975); (2) proportional selector (Ps) as proposed by Holland (1975), where probability of select s, $P(s)$ is proportional to their fitness, and its variants, random sampling selector (RSSs) and stochastic tournament selector (STs) Brindle (1981); stochastic universal sampling selector (SUSs) as proposed by Baker (1987); (3) ranking based selectors: linear (LRs) and non-linear (NLRs), developed by Whitley (1989); (4) selection schemas $(\mu, \lambda), (\mu + \lambda)$ and (5) threshold based selectors (Ths); (6) Boltzman selector (Bs), (7) a fully random selector (RNDs) (8) and a selector that combines a pair of different selectors (COMBs) by dividing the set of elements to select amongst its components. *Metric*: A uniform metric is used to assess this characteristic.

- *C2.5 Fitness function specification Support*: The most problem dependent element of metaheuristic techniques is the objective function to be optimized. Therefore, even when using MOFs, its evaluation is usually implemented explicitly by users and integrated into the framework through its extension points. However, based on the solution encodings supplied by MOFs, it is possible to provide tools for declarative objective function specification, freeing the user from the low-level task of implementing it.

In this case, a Domain-Specific Language (DSL) is a tool of great interest for objective function specification. The advantages of using a DSL, compared with classical implementation, are that the DSL can be a much simpler language than the implementation language, and integration of the objective function can be automatic if the MOF supports it. If the MOF provides suitable DSL tools for the specification of the objective function (such as syntax highlighting and in-line debugging and error information), it could lead to a more declarative paradigm for metaheuristic problem solving, improving the usability of metaheuristics and contributing to a wider application of such techniques. There are aslo drawbacks when using DSLs for objective function specification, such as the need to learn a new language, performance loss and the inability to model some objective functions using the language constructs.

Finally, there are problem types for which the automatization of objective function evaluation is impossible, since it relies on a human operator's interaction to evaluate solutions. In order to support this kind of problems, MOFs can provide a form in which users can directly provide the evaluation of solutions. Moreover, a partial implementation would be provided, where MOF users would customize the data entry form and solution representation (graphical or textual), designing a user friendly interface integrated within the framework. *Metric*: A uniform metric is defined to assess this characteristic, using features enumerated above: DSL support, DSL tools and forms for solution evaluation by human operators.

- *C2.6 Constraint Handling:* A feature of great importance for proper problem modeling is constraint definition support. There are usually two different ways to handle constraints when solving optimization problems[2]: (1) include constraint meeting in objective function definition as penalties; (2) and create repairing mechanisms that are applied to infeasible solutions. There are three alternatives of implementation for those mechanisms on MOFS: (a) provide global repairing mechanisms that users can implement for the problem at hand, (b) explicit modeling of each constraint and (c) specific repairing mechanisms for each constraint. In the same way as in characteristic C2.5, (3) the use of a DSL can make it easier to specify constraints for users, and some mechanisms, such as penalization [cf. (1)], can be applied without the need of implementation by users. *Metric*: An ad hoc metric is defined to assess this characteristic, where the weights have been associated with each feature as follows: (1) penalization 0.3, (2.a) global repairing mechanism 0.2, (2.b) individual constraint modeling 0.2 (2.c) individual constraints repairing mechanisms 0.2 and (3) DSL support 0.1.

5.2 Assessment and feature coverage analysis

Table 5 shows the feature coverage of area C2, along with the weight corresponding to each feature in its associated characteristics. The last row and last column of this table, respectively, show the sum of features supported by each MOF and the number of MOFs supporting each feature. It is remarkable that only 9.57% of features of this area are supported by a minimum of six out of the ten MOFs under study. Moreover, those features are associated with only three characteristics (namely C2.1, C2.3 and C2.4) and are mainly related to EA. An interesting fact shown in Table 5 is that more than 25% of features in this area are not supported by any framework.

[2] Various techniques to adapt metaheuristics to constrained problems have been proposed in literature (c.f. Michalewicz and Fogel (2004) for instance). However, most of these approaches require ad hoc implementation of the techniques depending on the problem and type of constraints to handle; consequently, it is difficult to integrate those proposals into a MOF. Those ad hoc techniques have been omitted in our comparison.

5.3 Comparative analysis

Area C2 along with C3 have the smallest average score of our benchmark, evidencing that framework developers have put more emphasis on coding algorithms for problem solving than in the support for an easy and efficient adaptation of these algorithms to the problem. Remarkably, there is a lack of support for: (1) the definition of neighborhood structures (except EasyLocal, ParadisEO and HeuristicLab), (2) the specification of the objective function and (3) constraint handling (exceptions are FOM, Eva2, ParadisEO and HeuristicLab).

In Fig. 3 a stacked columns diagram is shown for the characteristics of this area. Just like in Fig. 2 colors represent characteristics of this area and columns their support by the assessed MOFs.

Based on information shown in Table 5 and Fig. 3, we can provide an answer for RQ2. The means of problem adaption are summarized by the characteristics of area C2; however, current support of these mechanisms is limited and strongly depends on the MOF and metaheuristic to use for problem solving.

It is important to note that characteristic C2.4 is intimately related to EA support, and consequently those MOFs that do not support this technique are not able to support the features of this characteristic. However, those MOFs, such as EasyLocal, are still able to provide support for the rest of the area and constitute very useful alternatives when applying other techniques. Thus, users must have this into account when comparing different MOFs.

6 Advanced characteristics (C3)

In this area we evaluate general and advanced characteristics, not related to specific metaheuristics techniques. Specifically, the characteristics assessed in this area are the use of hybrid techniques, the implementation of hyper-heuristics and distributed and parallel execution. These characteristics are of great interest since they can either drastically improve the results obtained or simplify the application of techniques. They are especially interesting because their implementation involves high cost and complexity, preventing their application in many contexts. As MOFs can provide these characteristics pre-implemented, their applicability is significantly broadened.

6.1 Characteristics description

The following describes these characteristics:

C3.1 hybridization Hybrid metaheuristic techniques are those that combine several techniques. There is ample empirical evidence of the success of hybrid techniques for optimization problem solving (as stated by Talbi 2002). Several authors have described taxonomies of hybrid metaheuristics, to discern the ways techniques can be combined such as Talbi (2002) and Roli and Blum (2008). In this work we restrict the concept of hybrid metaheuristic to a combination of techniques integrated at a high level (as defined by Raidl 2006), where each technique keeps its overall structure except at the point of invocation of the other. Specifically, we have considered four different types of hybridization: (1) batch execution of the same technique (BEMIh), in which the technique is executed several times; (2) batch execution of different techniques (BEMMh), where various techniques are executed sequentially and where the results of one can be used as an initial solution of others; (3) interleaved execution of a technique as a step in each iteration of another, possibly affecting the internal variables (IMMh); and (4) combinations of various types of the above (Ch). *Metric*: An ad hoc metric is defined to assess this characteristic, with the weights of the features being (1) BEMIh 0.1, (2) BEMMh 0.2, (3) IMMh 0.6 and (4) Ch 0.1.

C3.2 hyper.heuristics A hyper-heuristic is readily defined as *a heuristic that selects heuristics*. Hyper-heuristics are

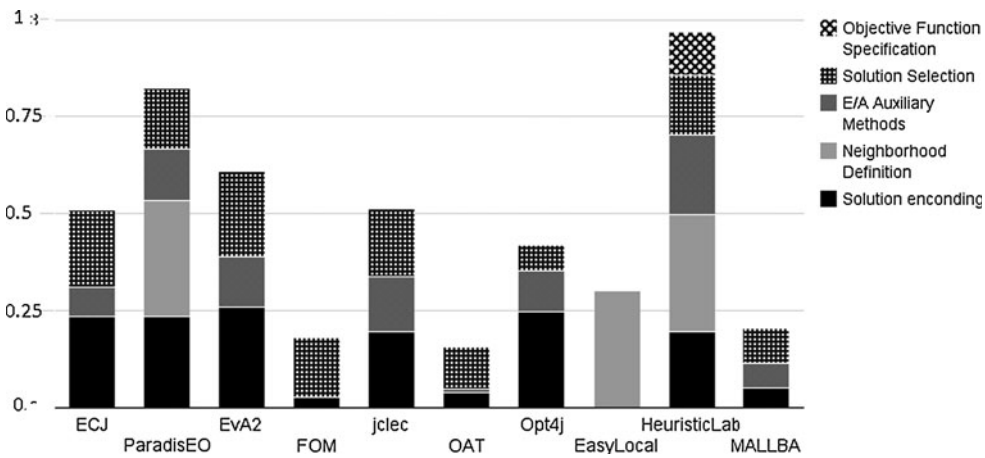


Fig. 3 Adaption to the problem and its structure support

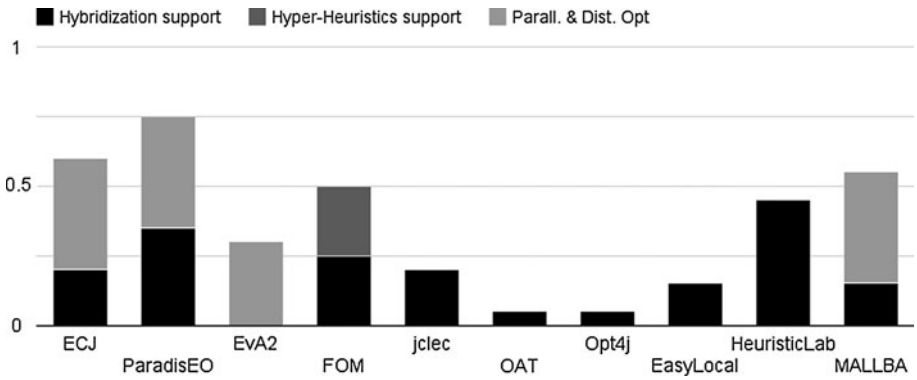


Fig. 4 Advanced characteristics support

intended to provide robust and general techniques of broad applicability without needing extensive knowledge of both the technique and the problem to solve. Hyper-heuristics have received much attention in recent years (Chakhlevitch and Cowling 2008; Cowling et al. 2002). Hyper-heuristics search from the heuristic space the heuristic that best solves a particular problem. The search space for hyper-heuristics could consist of four different subspaces: (1) optimization techniques space, with fixed parameters for each technique; (2) parameter values space for a technique; (3) underlying heuristics space for a technique (e.g. searching on a space of applicable selection, mutation or crossover operators when using an evolutionary algorithm); and (4) search space of possible solution encodings. *Metric*: A uniform metric is defined to assess these characteristics (with each search space weighing 0.25).

C3.3 paralell and distributed computation Many adaptations of metaheuristics have been proposed in the literature to exploit the paralell processing capabilities available in current distributed environments. Incorporating these strategies in a MOF is a significant improvement in their applicability and relevance to the resolution of a great number of real problems, given the complexity and cost of its implementation. Parallel and distributed execution of metaheuristics techniques without intercommunication (IPDM) can be implemented independently of the technique to apply. The only requirement is the installation of the MOF in each of the computers of the distributed environment and enabling a mechanism for communication and control in order to design, plan, launch execution and control optimization tasks in that distributed environment. Another similar variant is one in which techniques can exchange solutions (SSPDM). A parallel EA-based on islands with migration (as proposed by Whitley et al. (1999) would qualify as a SSPDM technique. Finally, techniques that need a change on the implementation of metaheuristics are sub-classified by Cahon et al. (2004) into *Parallel Local Search Metaheuristics* a unique executing instance of the metaheuristic controls the distributed and parallel exploration of its current solution's neighborhood (LSPDNM).

Parallel population-based metaheuristics There are two different approaches to create paralell population-based metaheuristics: (1) parallel and distributed objective function evaluation for the individuals of the population (PDPEDM), where in each network node a different subset of individuals conform the current population to be evaluated. The main difference with SSPDM is that a unique instance of the metaheuristic algorithm is executed in the distributed environment. (2) Parallel evaluation of the objective function, where computing objective function of a solution implies parallel processing in various nodes (PDESSM). *Metric*: A uniform metric is defined to assess this characteristic, where variants taken into account are IPDM, SSPDM, LSPDNM, PDPEDM and PDESSM.

6.2 Assessment and feature cover analysis

Table 6 shows feature coverage of Area C3, along with the weight corresponding to each feature in its associated characteristic. The last row and last column of this table, respectively, show the sum of features supported by each MOF and the number of MOFs supporting each feature. It is remarkable that only 6.25% of features of this area are supported by a minimum of six out of the ten MOFs under study. Furthermore, 40% of MOFs provide a nearly nil support (fewer than 10% of features) in this area.

6.2.1 Comparative analysis

With respect to the features of this criterion, the highest scores correspond to ParadisEO and FOM. Although both frameworks support the first characteristic, FOM does not support Parallel and Distributed Optimization whilst ParadiseEO does not support Hyper-heuristics. Currently, FOM is the only framework that supports Hyper-heuristics. In Fig. 4 a stacked columns diagram is shown for the characteristics of this area.

Table 6 and Fig. 4, answer RQ3, RQ4 and RQ5. Basic hybridization, such as (BEMIh) and (BEMMh) is currently supported by many MOFs, but more advanced hybridization techniques, such as (IMMh) and (Ch) are not. Parallel and distributed computing is currently supported by ParadisEO, ECJ, MALLBA and to a limited extent by other mainly EA-oriented frameworks such as JCLEC and EvA2.

7 Global optimization process support (C4)

One of the strengths of MOFs is their capacity to support the optimization process in its broadest sense, from problem modeling to experimentation, execution and results analysis. This support allows users without a deep knowledge in the area to apply metaheuristic techniques and obtain useful real results. This area evaluates these capacities.

7.1 Characteristics description

Seven characteristics have been established, covering the various stages of execution of the global optimization problem-solving process (4.1, 4.2, 4.3, 4.4 and 4.7) and the ability to interact with the user (4.5) and with other systems (4.6). The following describes those characteristics:

C4.1 termination conditions Metaheuristics do not provide explicit temination criteria, since, in general it is not possible to evaluate whether it has reached the global optimum solution. Therefore, users have to set criteria based on the specific needs and context of the problem to decide when to stop the execution of the metaheuristic. MOFs can provide implementations of the usual criteria for reuse, among which we find the following: (1) maximum number of iterations, (2) maximum execution time, (3) maximum number of objective function evaluations, (4) maximum number of iterations or execution time without improvement in the optimal solution found (5) reaching a concrete objective function value (6) and logical combinations (using operators AND/OR) of the above (e.g. *ExecTime* $\leq$ 36,000 *OR ExecTimeWithOutImprovement* $\geq$ 3,600. (7) Furthermore, termination conditions can be established independently of the problem to solve but dependent on the technique used, such as a termination criterion based on the diversity of the population when using an EA. Finally, (8) we evaluate the facilities provided to enable the definition of specific criterion by its implementation. In this sense, we have assessed the use of abstract classes or interfaces to evaluate the termination condition and its use in the implementation of the metaheuristic techniques provided. *Metric*: A proportional metric is defined, where (8) weighs 0.3, and the remaining weight is evenly distributed among the other described criteria.

C4.2 Batch mode execution The ability to automatically run a set of optimization tasks, where the user only has to specify the sequence and number of times to execute each task is important when performing experiments. The support of this feature promotes cost reduction, by automating one of the most tedious tasks of research and studies with empirical validation. We have defined four features related to this automation: (1) repeated execution of a task (using the same technique, parameters values and instance of the problem); (2) repeated execution of a task with different parameters (defined a range or set of values for the parameters of the technique); (3) execution of various tasks on the same instance of the problem; and (4) execution of various tasks on multiple instances of the problem. *Metric*: A weight of 0.2 has been given for the four features described above. In addition the ability to randomize the optimization task execution sequence and the generation and loading of a document or file where tasks are defined (the task execution plan, where description of tasks to execute can be user-supplied or generated by MOFs) weighs 0.2.

C4.3 Experimental design The appropriate design of experiments is essential to obtain valid conclusions in any study. This characteristic assesses the support provided by MOFs to establish hypothesis, identify dependent and independent variables and select and define experiments properly using standard designs (factorial, latin squares, fractional, etc.). This characteristic is assessed independently of the previous characteristic (C4.2) and the capacity for statistical analysis of results (C4.4). There are two different ways to support this characteristic: (1) provide integration mechanisms with design of experiments systems (such as GOSSET Sloane and Hardin (1991–2003)); and (2) implement the utilities for experimental design in the MOF itself. The alternative (1) implies that capabilities for experiment design are those of the system to integrate with and are difficult to assess in the context of this comparative. We have created a set of features in order to assess the capabilities of frameworks that use this approach (2): (a) hypothesis definition support, specifically common hypothesis, such as equality of performance of two techniques or irrelevance of the value of a parameter in a range; (b) experiments modeling, supporting the definition of dependent and independent variables and their nature (nominal, ordinal or scalar); (c) experiments design based on the previous model using common schemes; and finally (d) the capability of executing the experiments automatically, this feature assess the capability of generating a proper task execution plan for the experiments designed (C4.2 evaluates capabilities of automation of those plans execution). *Metric*: A proportional metric is defined, where approach (1) weighs 0.2, and the remaining weight is evenly distributed among features of approach (2).

C4.4 Statistical analysis One of the most important elements to ensure the validity of any study is the ability to perform statistical tests on its results. Therefore, one of the most common tasks in solving optimization problems (and in any study with an empirical component) is the statistical analysis of experimental data and results. There are two different ways to support this characteristic: (1) to provide integration mechanisms with statistical analysis systems (such as R or SPSS); and (2) to implement the utilities for statistical analysis in the MOF itself. One of the disadvantages of approach (1) is that the user must import data into the statistical analysis system and perform statistical tests on it, interpret results and return to the framework to change parameters or implementations if necessary. This approach frees the MOF from the implementation of the statistical tests. Moreover, statistical analysis systems are usually more complete and powerful than implementations of tests integrated on frameworks. On the other hand, the use of strategy (2) allows the framework to automate the tests and associated data exchange, showing the results integrated in its user interface and even react autonomously to the results of tests. A set of features have been created in order to assess capabilities of frameworks that use approach (2), concerning the support of various tests both parametric and non-parametric: (a) *t* student; (2) one-way ANOVA; (3) two-way ANOVA; (4) *n*-way ANOVA; (5) Mann–Withney *U* test; (6) Wilcoxon test; and (7) Kolmogorov–Smirnov test (or any test to assess the distribution of normal data). The use of approach (2) does not necessarily imply that approach (1) cannot be applied. In this sense the integration with the statistical software can be performed at the test execution level (to free the implementation burden), while providing programmatic support or graphical interfaces integrated in the MOF. *Metric*: A proportional metric is defined, where approach (1) weighs 0.3, and the remaining weight is distributed uniformly among features of approach (2).

C4.5 User interface, graphical reports and charts The usability of applications strongly depends on the proper design of its Graphical User Interface (GUI). Specifically, an appropriate GUI for MOFs requires taking into account the rest of the characteristics of this comparison criteria: the ability to select and configure the parameters of the different techniques, reporting of the results and monitoring of the status of optimization tasks and of the global execution plan, the control of nodes in distributed and parallel computing environments, the on-line technical support and the assistance or communication with the user forums and developers of the MOF. Moreover, although GUI design and usability could be assessed, the evaluation would include a subjective bias. In order to avoid it, we have defined the following set of features to be evaluated: (1) Integrated help and basic usability (menus, shortcut buttons, etc.); (2) technique specification and parameter configuration support, (3) problem modeling and data import, (4) Graphical support of advanced features (subdivided into batch mode execution configuration, design of experiments and statistical analysis of results) (5) the use of *optimization project* where all the information about problem instances, techniques and results are stored and (6) the graphical representation of results through diagrams and figures. *Metric*: A uniform metric is defined to assess this characteristic (each feature weighs 0.2). If the MOF only shows the evolution of the objective function of the best solution, but no additional metrics are provided (such as population diversity when using EA, or current solution when using TS or SA), then feature (6) has been evaluated with half of the weight.

C4.6 Interoperability This characteristic assesses the set of capabilities that frameworks provide to exchange information and interact with other systems. Specifically the following features are taken into account: (1) results and data export capabilities (considering formats such as CSV or excel/odf files); (2) data import capabilities (using formats such as CSV, excel/odf files or specific formats of standard libraries of each problem type, such as SATLIB or TSPLIB); (3) the capability of deployment and invocation as a web service (as in García-Nieto et al. (2007); and (4) the use of XML to store information associated with optimization projects (selected solution encoding, objective function and problem model, techniques and their parameters, experiment design and results and statistical analysis, etc.), so that other systems can process these data and parameters in a simple way. *Metric*: A uniform metric is defined to assess this characteristic (each feature weighs 0.25).

7.2 Assessment and feature cover analysis

The feature coverage of C4 area is shown on Table 7, along with the weight corresponding to each feature in its associated characteristic. As an exception, the features of the GUI characteristic have been assessed using a real value between 0.0 and 1.0. The last row and last column of this table, respectively, show the sum of features supported by each MOF and the number of MOFs supporting each feature.

7.2.1 Comparative analysis

The low score obtained by ParadisEO in this area is surprising, highlighting this as a potential area of improvement for that framework. OAT is among the highest scored frameworks (which has a well-designed GUI as well as powerful experiments execution and statistical analysis

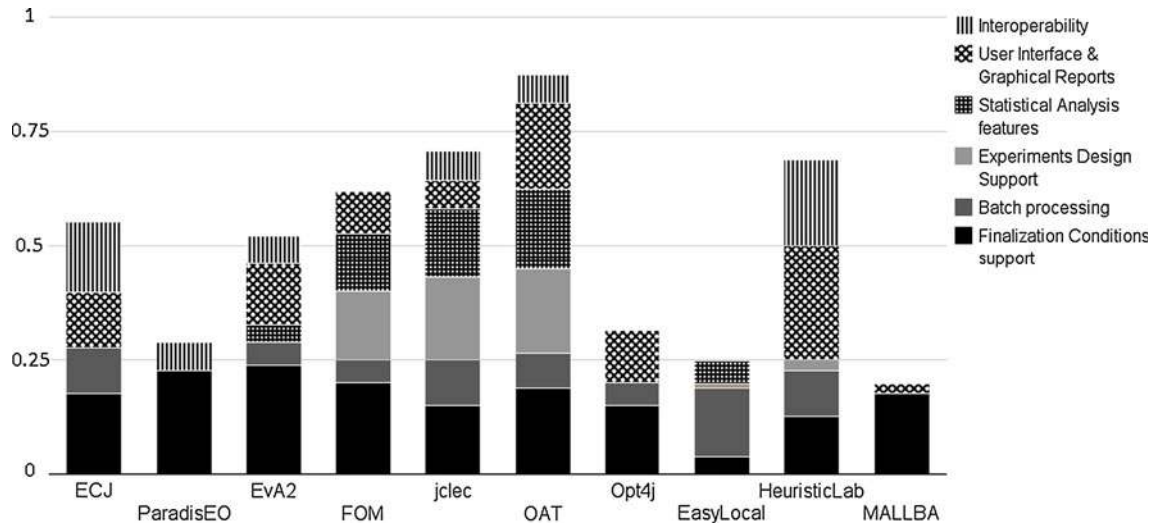


Fig. 5 General optimization process support

support) followed by JCLEC, whose characteristics in this area have been evaluated together with those of its associated project KEEL (focused on Data Mining and classification applications). Note that this area has, together with areas C2 and C3, the lowest support levels, thus representing significant areas of improvement in the present framework ecosystem. In Fig. 5 a stacked columns diagram is shown for the characteristics of this area.

Table 7 and Fig. 5 answer the requirements for RQ6. Area C4 characteristics summarize the capabilities provided by current MOFs for helping conducting research studies and the general problem-solving process. Those characteristics vary from statistical analysis and experiment execution engines, to GUIs with wizards and chart generation. These tools, however, are not inter-operable, and the quality and support of each MOF is not homogeneous; it is dispersed on the set of frameworks. Consequently, those tools are not available for all techniques or for programming languages and platforms.

8 Design, implementation and licensing (C5)

Both a suitable licensing model and the availability to run in multiple platforms are essential to the success of any software product. In the case of software frameworks, incorporating proper design and effective implementation is also very important, since applications created using it incorporate their design therein (with the errors and problems that they may contain). Moreover, the efficiency of those applications is limited by the efficiency of the framework. As a consequence, a comparison area has been defined to group this set of characteristics as described below.

8.1 Characteristics description

C5.1 Language Implementation language can be a key factor for users of MOFs, since the use of a well-known programming language reduces development costs and likelihood of errors. Frameworks under consideration in this share are implemented in C++, C# and Java.

C5.2 Licensing Cost is not a characteristic of interest since all the frameworks assessed are free; however, licensing of MOFs can limit the context and purposes of their use, or they can be forced to provide the client with the source code of the generated application. From this perspective the types of license we take into account are (1) commercial; (2) free without providing MOF source code nor commercial use; (3) free with MOF source code available only for certain organization and usages (usually universities and non profit activities); (4) MOF source code available under GPL (GNU General Public License) or similar, that forces the distribution of the source code of derived products under GPL license; and (5) MOF source code available under LGPL (GNU Lesser General Public License) or similar, that allows the use for commercial application without restrictions on source code availability. *Metric*: This feature is not evaluated using a set of features but we establish a direct score, based on the freedom that each license provides: (1) Commercial Licensing = 0; (2) Free binaries (no commercial use) = 0.25; (3) Restricted availability of source code = 0.5; (4) GPL = 0.75 and (5) LGPL =1.

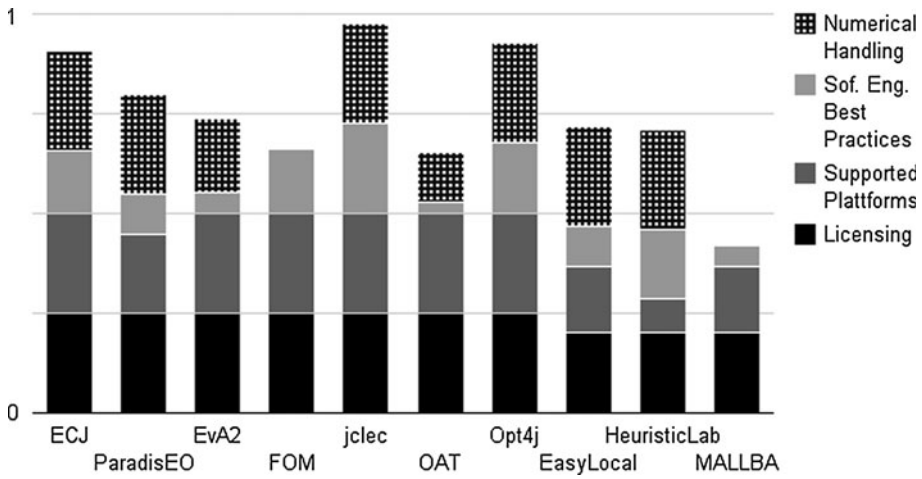


Fig. 6 Design, implementation and licensing assessment

C5.3 Supported platforms The set of platforms taken into account are: Windows, Unix (Linux, Solaris, HPUX, etc.) and Mac. *Metric*: A uniform metric is defined, with each platform weighing $0.\widehat{3}$; in the case of partial support (only a limited set of features are available on a certain platform) we penalize it with 50%.

C5.4 Software engineering best practices A proper design and following of software engineering best practices is especially important for MOFs. However, assessing the design of a framework in a quantitative and objective way is a difficult task. As a result, features only evaluate basic use of certain tools and processes recognized as best practices such as (1) the use of design patterns to promote flexibility in variation points; (2) the use of automated tests (unit tests): this characteristic is evaluated based on the source code of MOFs (for those that do not provide the source code, evaluation is based on the documentation, if tests exists); (3) explicit documentation of the MOF variation and extension points; and (4) the use of reflective capabilities and dependence injection to promote flexibility as described by Fowler (2004). The latter feature corresponds to the capabilities of the framework to dynamically load types of problems, objective functions and other elements associated with customization or extension without having to recompile the framework. With regard to feature (4), MOFs that perform runtime loading of modules have been associated with half of the weight, while those that use a dependence injection system for the management of modules have full weight. *Metric*: A uniform metric is defined to assess this characteristic.

C5.5 Size A basic measure of the complexity of a framework is its size. The size of a framework can be measured by various metrics, number of lines of code, number of classes and packages/modules, number of variation points and possible combinations of components, etc. It would be inappropriate to use the size of frameworks as a quantitative evaluative criteria, since the functionalities supported are not directly related to it, and an increase in its size does not necessarily imply greater complexity in its use. Therefore, we consider it as a qualitative criterion. As a consequence, we consider some of these measures for each framework, but they will not be included in the quantitative assessments.

C5.6 Numerical handling Most metaheuristic techniques are stochastic, requiring the use of a random number generator. This fact has two consequences: (1) choosing a good random number generator is a key point for the proper behavior of the techniques implemented by MOFs; and (2) in order to support experiments replicability, a unique seed must be used on all random number generators used by along the framework and its customizations/extensions developed by users. Features evaluating this two important points are defined for this characteristic, where (1) evaluates if a proper random number generator is provided (either a Mersene Twister implementation or support for customization of the random number generation scheme); and (2) evaluates the replicability of experiments based on the support of a global seed and provision of a random number generator using this seed to user implemented modules. *Metric*: A uniform metric is defined to assess this characteristic.

8.2 Assessment and feature cover analysis

This area seems to be the most homogeneous and supported in the sense that most frameworks support almost all the features and to a high degree. The platforms supported is practically universal, except for HeuristicLab, EsayLocal and some modules of ParadisEO. It is remarkable also the general adoption of the UML notation, as well as the open source licensing models. In Fig. 6 a stacked columns diagram is shown for some characteristics of this area. With

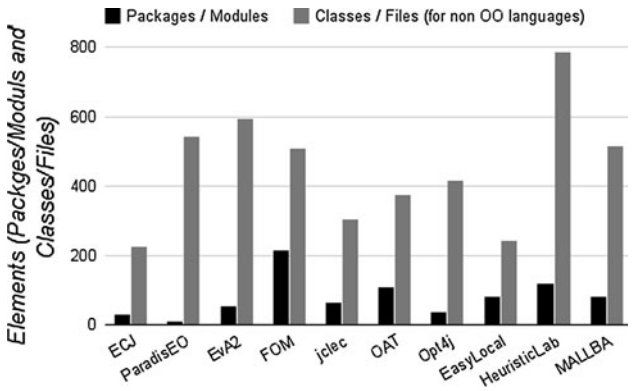


Fig. 7 Framerowks size

regard to the size of MOFs, Fig. 7 shows the framework sizes in terms of number of packages (or modules) and classes (or files, when there is not a direct relation from files to classes). These attributes may be of interest because the size of a framework may be an indirect measure of its complexity and therefore of its possible difficulty of use. However, the restrictions imposed by the language should be taken into account, as for example in java each public class must be in a separate file.

Table 8 and Fig. 6 provide the answers to RQ7, RQ8 and RQ9. There is wide availability of MOFs per platform, where each technique is available on nearly all platforms. This fact is due to the use of platform independent programming languages such as Java and C++ (using standard libraries). However, as there is no MOF supporting all techniques, users must be careful since although there could be available alternative MOFs providing implementations for missing techniques, the effort needed for changing from one MOF to other one is considerable and implies giving up other features or variants. All the frameworks evaluated provide GPL or free licenses for academic/research purposes. Finally, basic software engineering best practices, such as UML diagrams of MOFs architecture and dynamic module loading are widely supported, but more advanced ones, such as automated tests, use of dependence injection libraries and explicit variation point documentation are not supported. Notably, some frameworks do not support the use of a proper random number generator nor its customization.

9 Documentation and support (C6)

When selecting a framework for developing any kind of application, documentation, technical support and user community responsiveness are important. These are the factors that can smooth out the learning curve when users have no experience and need to solve problems or errors that arise during use. Consequently we have considered those factors, including additional features, in order to measure the maturity of the frameworks such as types of problems that MOFs bring as samples and the number of scientific articles published using the framework.

C6.1 Sample problem types As a measure of maturity and supportiveness of frameworks, we have this characteristic that assesses the implemented problem types that MOFs provide. This characteristic can also measure to what extent MOFs have been applied and tested with different kinds of problems. Moreover, solved problem types can be excellent starting points if users try to solve problems to some extent similar to those provided. The set of problem types considered comprises problem families such as TSP, SAT, QAP, Job Shop Scheduling, Flow Shop Scheduling and knapsack, iterated prisoners dilemma, symbolic regression problems and others. The exact problem types can be consulted in the evaluation data sheet

Table 8 MOFs Programming languages, platforms and licenses

MOF	Prog. Lang.	Platforms	License
EasyLocal	C++	Unix	GPL
ECJ	Java	All	Open Source (Academic free license)
ParadisEO	C++	All (Except for windows if using PEO)	CECILL (ParadisEO) and LGPL (EO)
EvA2	Java	All	LGPL
FOM	Java	All	GPL
HeuristicLab	C#	Windows	GPL
JCLEC (and KEEL)	Java	All	LGPL
MALLBA	C++	Unix	Open source
Optimization Algorithm Toolkit	Java	All	LGPL
Opt4j Martin Lukasiewycz and Helwig (2009)	Java	All	LGPL

mentioned previously. *Metric*: A uniform metric is defined where the weight is distributed evenly amongst the evaluated problem types. The set is comprised of 59 different problem types.

C6.2 Articles and papers Another way to assess the maturity and quality of MOFs is through scientific publications that describe MOFs or report their use. The assessment of this characteristic relies on publications found during our literature review and on publications enumerated on MOFs websites. A total number of 285 publications were found for the selected MOFs, searching for papers from 2000 to 2010. *Metric*: An ad hoc metric is defined: the maximum score (1.0) was assigned to the framework with the most publications, namely ECJ with 113, and the score of the other frameworks were computed based on this formula: *score = (publications of MOF N)/(maximun number of publications per MOF)*. The whole set of publications found per framework is available at http://www.isa.us.es/uploads/34MOFs/bib/N.bib, where N is the name of each MOF; for instance, ECJ bibliography is available at http://www.isa.us.es/uploads/MOFs/bib/ECJ.bib.

C6.3 Documentation Documentation is the main source of information for users in a framework, a capital element to enable its use. This characteristic is assessed based on the presence (or absence) of the following features: (1) User manual; (2) Technical/development documentation; (3) "How to" document, where short recipes are provided to perform usual actions; (4) "Frequently asked question" section on the web site of framework documentation; and (5) MOF web site. *Metric*: A uniform metric is defined, where each feature weighs 0.2.

C6.4 Users and popularity This characteristic intends to assess the number of users of each framework. The evaluation of this characteristic is based on the number of researchers using each framework outside the MOF creators research group and development team; we name them "external users". In order to evaluate this characteristic we have filtered publications found during our literature review using each MOF and on publications enumerated on MOF websites, removing those where one of its authors is member of the development team or research group of MOF creators. *Metric*: An ad hoc metric is defined: the maximum score (1.0) was assigned to the framework with more external publications, namely ECJ with 84, and the scores of the other frameworks were computed based on this formula: *score = (external publications of MOF N)/(maximun number of external publications per MOF)*. The whole set of publications found per framework is available at http://www.isa.us.es/uploads/MOFs/bib/N-external.bib, where N is the name of each MOF; for instance, ECJ bibliography of external publications is available at http://www.isa.us.es/uploads/MOFs/bib/ECJ-external.bib.

9.1 Comparative analysis

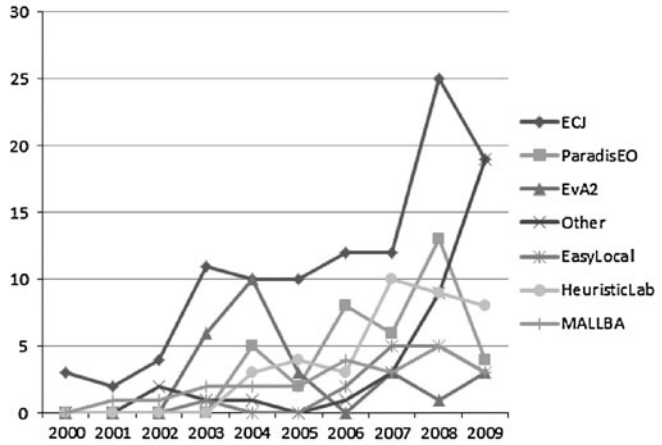
In general, the feature that is less supported in this area is the implemented problem types. With regard to papers that describe or apply MOFs and popularity between external authors, ECJ is the most salient framework, that dwarfs the other MOFs in this comparative. Figure 8 four charts that illustrate this effect: sub-figure (a) shows the number of publications per MOF and year. ECJ appears as the senior framework, obtaining a dominant position early in the procedures which it still holds (we ignore 2010 since early publications could not be updated). Sub-figure (b) shows the total number of publications per MOF. Subfigure (c) shows the number of external and internal publications per MOF as an stacked columns chart. Subfigure (d) shows the number of external authors per MOF. ECJ is followed by ParadisEO and HeuristicLab in number of publications. ECJ has nearly 75% of external publications and 65% of external authors. The less popular frameworks are FOM and OAT with nearly null external usage and a small number of publications.

Note that there are two frameworks that score low on Documentation, namely OAT and EasyLocal. All frameworks have active and supportive communities of users/developers. Figure 9 uses a stacked columns diagram to summarize the support of this area's characteristics.

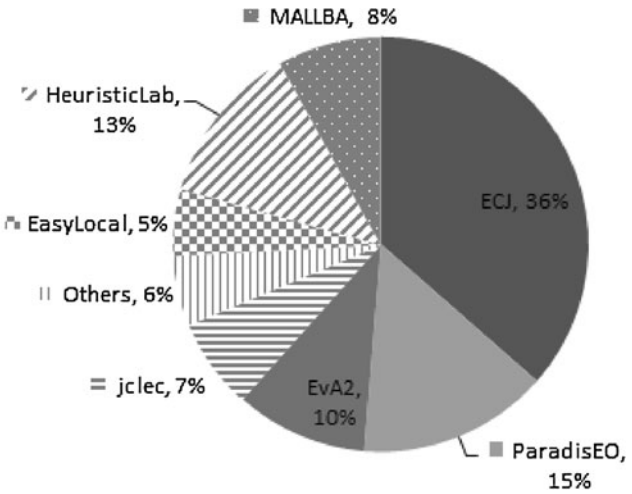
Figure 9 and the information gathered along this study provide an answer to RQ10 question. Currently, a high number of MOFs are available which support a wide set of features. So when addressing new problems or performing research studies on well-known ones, the use of MOFs becomes a valid approach. MOF use outside of developers research groups could be boosted by an improvement of framework documentation and support. Currently, the most popular framework is ECJ, which has a large community of external users and a wealth of publications year on year. Moreover, there seems to be a correlation between the score in area C3 and MOFs' popularity, since frameworks with higher scores in that area are those with higher popularity. This fact is not surprising, since that area contains some of the features that add more value for user. These features, such as distributed and parallel optimization, make MOFs tools capable of solving extremely complex problems and are difficult to implement from scratch, thus making those frameworks more attractive for users that need those features and contributing to make those MOFs popular.

10 Discussion and challenges

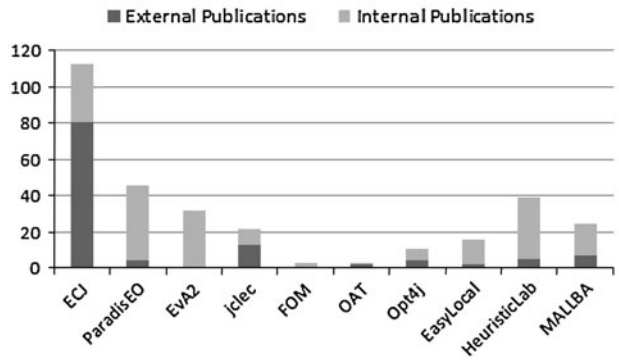
In this section, we discuss the results obtained in this study. Based on these results, we identify a number of challenges (RQ11) to be addressed in the future. Challenges are part of



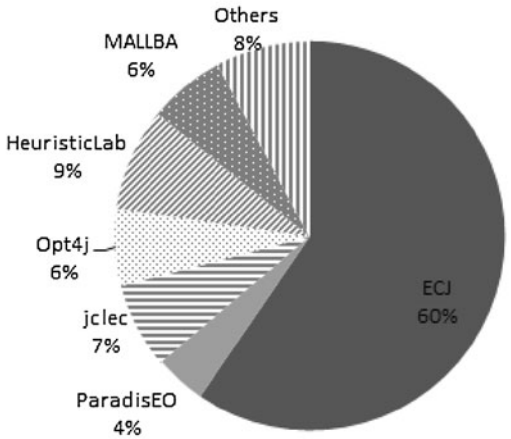
(a) Publications per frameworMOF and year



(b) Total publcations per MOF



(c) Total number of internal and external publications per MOF



(d) Total number of external authors per MOF

Fig. 8 Figures showing publications and popularity of each framework

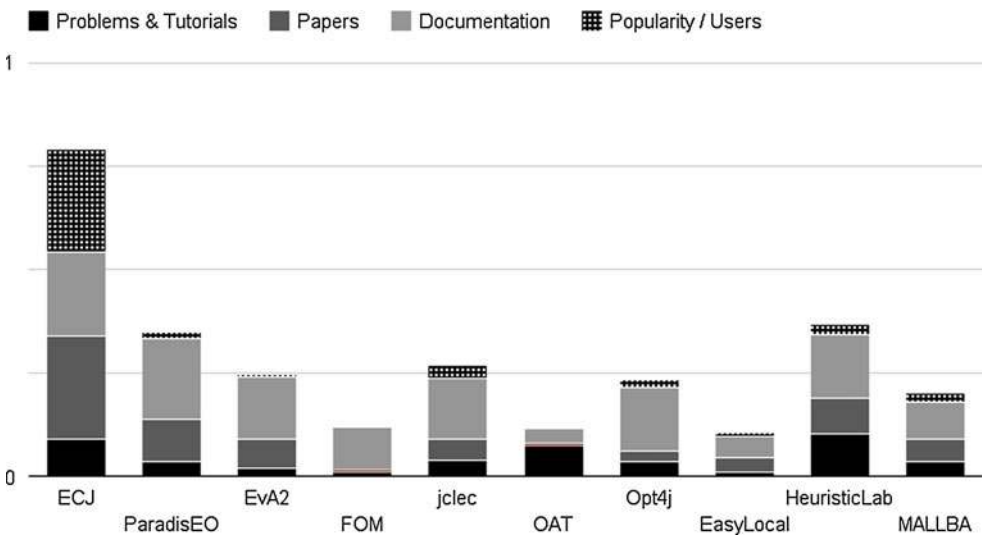


Fig. 9 Documentation and technical support

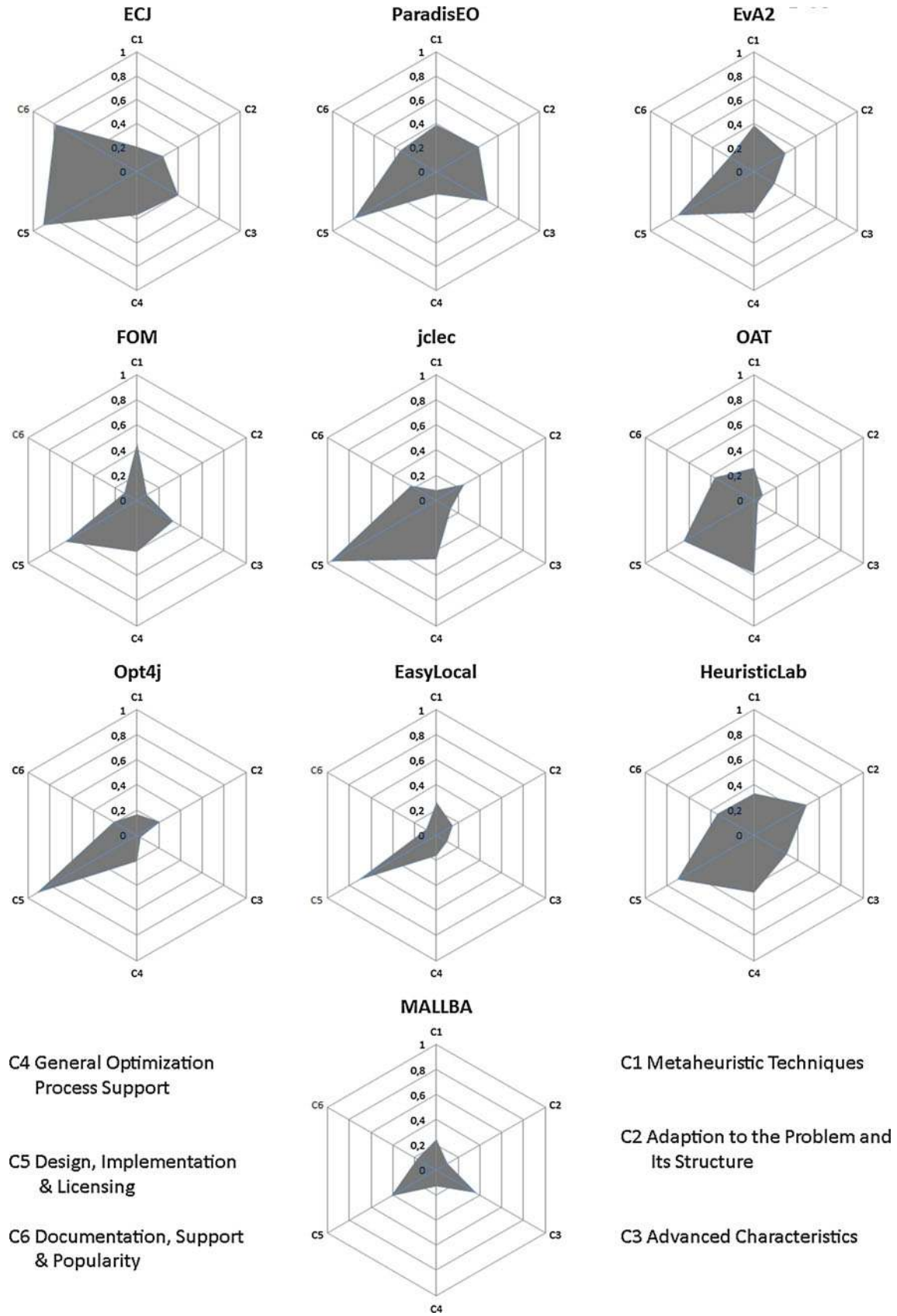


Fig. 10 General scores of MOFS as Kiviat diagrams

the authors' own personal view of open questions, based on the analysis presented in this paper. Figure 10 shows global score results for MOFs as Kiviat diagrams, summarizing the results of this study; evaluating MOFs from a research user perspective. In the appendix, Table 7 shows the global score obtained for each MOF and characteristic as well as the average for each area.

To achieve the maximum score in areas C1, C2 and C3, each MOF would have to implement an ample subset of the current state of the art on metaheuristics, so it is not surprising that the scores do not generally reach the maximum possible value. On the contrary, the small average values on areas C4 and C6 are significant and therefore show a general improvement direction for current MOFs.

10.1 Capabilities discussion

On average, the MOF with the best score is ECJ (maximum area in Fig. 10), making it a preferred choice if users can use EA on java. However, this MOF scores below average in areas C1 and C5, which are clear improvement areas for it, and could lead users to evaluate different options (C1 measures techniques available). The next best scored MOF is ParadisEO, salient in areas C1 and C3, which uses C++ as its implementation language. This MOF, however, scores below average in C4 area, making this a clear improvement area. The MOFs that provide the amplest support in terms of the variety of metaheuristics (criterion C1) are FOM and ParadisEO. The score obtained by OAT in C4 area is remarkable, much above average, and it is due to its GUI, execution of experiments and statistical analysis tooling. In this same area the support of JCLEC (and its twin project KEEL) is also above average. However, the best score of the GUI characteristic is obtained by HeuristicLab that in its last version (3.3) provided a complete, highly configurable and intuitive user interface. C5 area is where all of MOFs provide better average results. This is not surprising given that these characteristics are key for frameworks use and success and are clear signs of technical competence and maturity. In this sense, MOFs without good design or implementation simply do not survive. Finally, the average value of area C6 indicates the need to improve documentation, user guidance and support. Thus we define *Challenge 1*: Improve documentation, user guidance and support and GUI tooling.

10.2 Evolution of the ecosystem of MOFs

The creation of this benchmark has been a time-consuming and demanding task. However, the length of this task has allowed the evaluation of an additional feature of the ecosystem of MOFS: its liveliness and evolution speed. During the creation of this benchmark, various frameworks released new major versions with important improvements, namely ECJ, PARADISEO, JCLEC and HeuristicLab; moreover, other frameworks such as EvA2 and Opt4j released minor versions with bug fixes and minor features. This evolution allowed us to test the evaluation framework presented in this study. No modifications were needed in order to assess those new versions of the MOFs and their features; thus it validates the flexibility and completeness of our approach. Moreover, both the previous and new versions of those frameworks were evaluated, providing a dynamic view of the ecosystem, in contrast with the static one shown in the previous sections. In this sense, we can evaluate the "hot areas", i.e. those areas where more evolution has been performed and the speed in the evolution of the assessed MOFs. In this sense the area with bigger improvements are C4 and C5, primarily due to the improvements in the GUI and licensing model of HeuristicLab and the new GUI of ECJ. Additionally, C1 and C6 have also improved significantly but in a smaller scale, since new techniques and better documentation are provided by the assessed MOFs. The MOF with a bigger improvement in this time was HeuristicLab, changing directly form version 1.1 to version 3.3. In this new version significant improvement is demonstrated in the licensing model (it becomes an open source project under GPL license) and the GUI and documentation have been improved significantly. The next framework in terms of improvement during the creation of this benchmark was ECJ, where a multi-objective technique and a GUI were added, i.e. complementary, significant improvements in documentation have been developed. Finally, the evolution measured shows that the current MOFs Ecosystem is a vibrant and living one, where new versions and important features are added continuously.

Both the final evaluation of current versions and the previous one are available as Google Docs spreadsheets at http://www.isa.us.es/MOFComparison and http://www.isa.us.es/MOFComparison-OLD, respectively. They can be downloaded and exported to different formats such as MS Office or open office for customization and tailoring.

10.3 Potential areas of improvement of current frameworks

In addition to the points stated above about area C6, based on the finished comparative study carried out, and on results described above, we enumerate below some gaps and unsupported features that have been identified. The areas where we see the most room for improvement are C2 (adaption to the problem and its structure), C3 (advance characteristics) and C4 (general optimization process support). Specifically, some features that have room for improvement are

- Hyper-heuristics support.
- Support for designing and automated running of experiments and for analyzing results.

- User guides together with wizards, project templates and GUI to aid the optimization process.
- Parallel and distributed computing support.
- Domain-Specific Languages for objective function and constraints formulation.

Thus we define *Challenge 2*: Provide added-value features for optimization, such as hyper-heuristics and parallel and distributed computing capabilities.

In particular, in the context of area C5 (design, implementation and licensing), we have identified the following issues regarding software engineering best practices:

- *Absence of unit tests*. Note that one of the discarded EA-oriented optimization library (JGAP) is recognized reference for this practice Meffert (2006); however, assessed MOFs do not provide unit tests in general (except for JCLEC and HeuricLab).
- *Heterogeneity of project building and description mechanisms*. It would be interesting that, as in ParadisEO, projects provide files for framework compilation using standard mechanisms such as *makefiles* in C++, or *ant* or *maven* builds files in java.
- *Absence of explicit documentation of variation points*. Although all the frameworks that have been evaluated provide extensive technical documentation of the different classes and modules, none of them provide a scheme (such as *feature models*) to describe the variation points of the framework, nor are these even described explicitly in natural language in the documentation. Moreover, none of the frameworks use the UML profiles for framework documentation Fontoura et al. (2001).
- *Limited dynamic and reflexive capabilities* for loading problems, heuristics and techniques variants. Thus, only Opt4j uses a dependency injection mechanism (such as Google Juice or Spring).

Finally, regarding area C1 (Metaheuristic techniques) there is always the possibility of enlarging the portfolio of techniques implemented. The current support is uneven, with some techniques (such as EA) practically universally supported and others (such as *GRASP, SS, ACO or AIS*) being rarely implemented.

Thus we define *Challenge 3*: Improve techniques and variants support and *Challenge 4*: Develop standard benchmarks for MOFs.

11 Conclusions

In this paper an assessment based on the state of the art of the main MOFs has been made. The motivation of the study is based on the implications of the NFL theorem in terms of the desirability and advantages of using such tools, on the complexity and difficulty of learning and mastering the use of any of these frameworks and on the availability of a good number of MOFs.

From the MOFs assessment carried out, we can draw the following conclusions:

- Frameworks are useful tools that can speed up the development of optimization-based problem solving projects, reducing their development time and costs. They might also be applied by non-expert users as well as extend the user base and the applications scope for metaheuristics techniques.
- There are many MOFs available, which overlap and provide similar capabilities which means that a certain duplication of efforts has been made. It would be great if a certain coordination and standardization of these MOFs were carried out in order to improve the support given to the user community.
- There are visible gaps in the support of specific key characteristics, as shown in Sect. 10.3.
- There is impending work we have to face in the near feature, namely
- Perform the second phase of the technology evaluation methodology followed in this study as defined by Brown and Wallnau (1996), establishing a set of specific use scenarios and conducting experiments of application using the evaluated MOFs.
- As the authors of one of the frameworks studied (FOM), we plan to enhance it according to the potential improvement areas identified in this paper.

Acknowledgments We would like to thank Stefan Wagner, Andreas Schaerf, Sebastián Ventura, Sean Luke, Marcel Kronfeld and David L. Woodruff for their helpful comments in earlier versions of this article. We are thankful to David Benavides and Sergio Segura for providing us their inspirational work Benavides et al. (2009), and Ana Galan for her linguistic support. This work has been partially funded by the European Commission (FEDER) and Spanish Government under CICYT project SETI (TIN2009-07366) and the Andalusian Government projects ISABEL (TIC-2533) and THEOS (TIC-5906).

Appendix: Data tables

In this section, we provide detailed information about the scores obtained in each characteristic by each framework. Interested readers can obtain more detailed information about assessment on characteristics and features (including comments on problems found on the assessment, penalizations on some features and its underlying reasons and informations sources used to assess it) in http://www.isa.us.es/MOFComparison. Moreover, this spreadsheet can be downloaded and exported to various formats, and it is provided in such a way that user can customize weights of each characteristic, feature and area, allowing the creation of tailored benchmarks more adapted to its specific needs (see Tables 9, 10, 11).

Table 9 Scores for C1–C4 and C6

Characteristic	ECJ	ParadisEO	EvA2	FOM	JCLEC	OAT	Opt4j	EasyLocal	HeuristicLab	MALLBA	Avg
C1-Metaheuristic techniques											
Steepest descent/fill climbing (SD)	**1**	**1**	**1**	**1**	0	**1**	0	**1**	**1**	**1**	0.800
Simulated annealing (SA)	0	0.8	0.6	**0.9**	0	0	0.6	0.7	0.8	0.7	0.510
Tabu search (TS)	0	0.7	0	0.9	0	0	0	**1**	0.7	0	0.330
GRASP	0	0	0	**1**	0	0	0	0	0	0	0.100
Variabl neighborhood search (VNS)	0	**0.2**	0	**0.2**	0	0	0	**0.2**	0	0	0.060
Evolutionary algorithms (EA)	**0.85**	0.6	0.8	0.25	0.7	0.7	0.7	0	0.7	0.4	0.570
Particle swarn optimization) (PSO)	0.3	**0.7**	0.5	0	0	0	0.5	0	0.3	0.3	0.260
Artificial immune systems (AIS)	0	0	0	0	0	**0.25**	0	0	0	0	0.025
ACO	0	0	0	0.7	0	**0.9**	0	0	0	0.3	0.190
Scatter search	0	0	**0.438**	0	0	0	0	0	0	0	0.044
Multiobjective metaheuristics	0.125	0.188	**0.438**	0	0.188	0	0.125	0	0.0625	0	0.113
C2-Adaption to the problem and its structure											
Solution enconding	0.7	0.7	**0.775**	0.075	0.588	0.113	0.738	0	0.588	0.15	0.443
Neighborhood definition	0	**0.9**	0	0	0	0	0	**0.9**	**0.9**	0	0.270
E/A auxiliary methods	0.226	0.409	0.393	0	0.426	0.02	0.321	0	**0.616**	0.197	0.261
Solution selection	0.6	0.467	**0.667**	0.467	0.533	0.333	0.2	0	0.467	0.267	0.400
Objective function specification	0	0	0	0	0	0	0	0	**0.333**	0	0.033
Contraint handling	0	0.3	0.6	**0.7**	0	0	0	0	0	0	0.160
C3-Advanced characteristics											
Hybridization support	0.4	0.7	0	0.5	0.4	0.1	0.1	0.3	**0.9**	0.3	0.370
Hyper-heuristics support	0	0	0	**0.5**	0	0	0	0	0	0	0.050
Parall. and dist. opt	**0.8**	**0.8**	0.6	0	0	0	0	0	0	**0.8**	0.300
C4-General optimization process support											
Finalization conditions support	0.7	0.9	**0.95**	0.8	0.6	0.75	0.6	0.15	0.5	0.7	0.665
Batch processing	0.4	0	0.2	0.2	0.4	0.3	0.2	**0.6**	0.4	**0**	0.270
Experiments design support	0	0	0	0.6	0.72	**0.74**	0	0.04	0.1	**0**	0.220
Statistical Analysis features	0	0	0.15	0.5	0.6	**0.7**	0	0.2	0	0	0.215
User interface and graphical reports	0.483	0	0.533	0.367	0.25	0.75	0.45	0	**1**	0.083	0.392
Interoperability	0.625	0.25	0.125	0	0.25	0.25	0	0	**0.75**	0	0.225
C6-Documentation and support											
Problems and tutorials	0.36	0.136	0.068	0.034	0.153	0.288	0.136	0.033898305	**0.407**	0.136	0.175
Papers	**1**	0.407	0.283	0.027	0.195	0.027	0.097	0.142	0.345	0.221	0.274
Documentation	**0.8**	0.79	0.6	0.41	0.59	0.14	0.62	0.2	0.61	0.35	0.511
Popularity/ users	**1**	0.0595238	0	0	0	0.027	0	0	0	0	0.118

Boldface values denote the best (higher) values of each row

Table 10 Scores for C5 design. Implementation and licensing

Characteristic	ECJ	ParadisEO	EvA2	FOM	JCLEC	OAT	Opt4j	EasyLocal	HeuristicLab	MALLBA
Licensing	Open source (academic free license)	CECILL (ParadisEO)y LGPL (EO)	LGPL	LGPL	LGPL	LGPL	LGPL	GPL	GPL	Open Source
Supported platforms	All	All (except for windows if using PEO)	All	All	All	All	All	Windows and Unix	Windows	Unix
Sof. eng. best practices	0.62	0.4	0.2	0.64	**0.9**	0.1	0.7	0.4	0.7	0.2
Packages/ modules	28	10	54	215	63	109	35	80	119	80
Classes/files (for non OO languages)	226	542	594	510	304	373	417	244	785	514
Numerical handling	1	1	0.75	0	1	0.5	1	1	1	0

Boldface value denotes the best (higher) value of the row

Table 11 Global scores

Area	ECJ	ParadisEO	EvA2	FOM	JCLEC	OAT	Opt4j	EasyLocal	HeuristicLab	MALLBA	Avg
1 Supported metaheuristics	0.207	0.381	0.394	**0.450**	0.081	0.259	0.175	0.264	0.324	0.245	0.282
2 Problem adaption/encoding	0.254	0.413	0.306	0.090	0.258	0.078	0.210	0.150	**0.484**	0.102	0.249
3 Advanced metaheuristic characteristics	0.400	**0.500**	0.200	0.333	0.133	0.033	0.033	0.100	0.300	0.367	0.226
4 Optimization process support	0.368	0.192	0.347	0.411	0.470	**0.582**	0.208	0.165	0.458	0.131	0.356
5 Design, implementation and licensing	0.905	0.797	0.738	0.660	**0.975**	0.650	0.925	0.717	0.708	0.417	0.786
6 Documentation, samples and popularity	**0.789**	0.348	0.238	0.118	0.234	0.364	0.213	0.094	0.340	0.177	0.304
Average per framework	**0.487**	0.439	0.371	0.344	0.359	0.328	0.294	0.248	0.436	0.240	0.367

Boldface values denote the best (higher) values of each row

References

Aarts E, Lenstra J (1997) Local search in combinatorial optimization. Wiley

Ackley DH (1987) A connectionist machine for genetic hillclimbing. Kluwer Academic Publishers, Norwell, MA, USA

Alba E, Luque G, García-Nieto J, Ordonez G, Leguizamon G (2007) Mallba: a software library to design efficient optimisation algorithms. Int J Innov Comput Appl 1:74–85. doi:10.1504/IJICA.2007.013403, http://portal.acm.org/citation.cfm?id=1359342.1359349

Andresen B, Gordon JM (1994) Constant thermodynamic speed for minimizing entropy production in thermodynamic processes and simulated annealing. Phys Rev E 50(6):4346–4351. doi:10.1103/PhysRevE.50.4346

Angeline PJ, Fogel DB, Fogel LJ (1996) A comparison of self-adaptation methods for finite state machines in a dynamic environment. In: Proc. 5th Ann. Conf. on Evolutionary Programming, pp 441–450

Arabas J, Michalewicz Z, Mulawka J (1994) Gavaps-a genetic algorithm with varying population size. Evolutionary Computation. In: Proceedings of the First IEEE Conference on Computational Intelligence, vol 1, pp 73–78. doi:10.1109/ICEC.1994.350039

Back T, Fogel DB, Michalewicz Z (eds) (1997) Handbook of evolutionary computation. IOP Publishing Ltd., Bristol. http://portal.acm.org/citation.cfm?id=548530

Baker J (1987) Reducing bias and inefficiency in the selection algorithm. In: Proceedings of the Second International Conference on Genetic Algorithms, pp 14–21

Benavides D, Segura S, Ruiz-Cortès A (2009) Automated analysis of feature models after 20 years: a literature review. (Information Systems Accepted for publication)

Birgmeier M (1996) Evolutionary programming for the optimization of trellis-coded modulation schemes. In: Proc. 5th Ann. Conf. on Evolutionary Programming

Blanton JL Jr., Wainwright RL (1993) Multiple vehicle routing with time and capacity constraints using genetic algorithms. In: Proceedings

of the 5th International Conference on Genetic Algorithms, Morgan Kaufmann Publishers Inc., San Francisco, pp 452–459
Brindle A (1981) Genetic algorithms for function optimization. Ph.D. thesis, University of Alberta, Edmonton
Brown AW, Wallnau KC (1996) A framework for evaluating software technology. IEEE Softw 13(5):39–49. doi:10.1109/52.536457
Brownlee J (2007) Oat: the optimization algorithm toolkit. Tech. rep., Complex Intelligent Systems Laboratory, Swinburne University of Technology
Cahon S, Melab N, Talbi EG (2004) Paradiseo: a framework for the reusable design of parallel and distributed metaheuristics. J Heuristics 10(3):357–380. doi:10.1023/B:HEUR.0000026900.92269.ec
Chakhlevitch K, Cowling P (2008) Hyperheuristics: recent developments. In: Adaptive and multilevel metaheuristics, pp 3–29
Chatterjee A, Siarry P (2006) Nonlinear inertia weight variation for dynamic adaptation in particle swarm optimization. Comput Oper Res 33(3):859–871. doi:10.1016/j.cor.2004.08.012, http://www.sciencedirect.com/science/article/B6VC5-4DBJG28-2/2/57210fee165fec156db017ff5e59aa5f
Clerc M (2006) Particle swarm optimization. ISTE Publishing Company
Corne D, Knowles JD, Oates MJ (2000) The pareto envelope-based selection algorithm for multi-objective optimisation. In: Proceedings of the 6th International Conference on Parallel Problem Solving from Nature (PPSN VI), Springer, London, pp 839–848
Cowling PI, Kendall G, Soubeiga E (2002) Hyperheuristics: a tool for rapid prototyping in scheduling and optimisation. In: Proceedings of the Applications of Evolutionary Computing on EvoWorkshops 2002, Springer, London, pp 1–10
Cramer NL (1985) A representation for the adaptive generation of simple sequential programs. In: Proceedings of the 1st International Conference on Genetic Algorithms, L. Erlbaum Associates Inc., Hillsdale, pp 183–187
Davis L (1985) Applying adaptive algorithms to epistatic domains. In: Proceedings of the 9th international joint conference on Artificial intelligence (IJCAI'85). Morgan Kaufmann Publishers Inc., San Francisco, pp 162–164
Davis L (1989) Adapting operator probabilities in genetic algorithms. In: Proceedings of the third international conference on Genetic algorithms. Morgan Kaufmann Publishers Inc., San Francisco pp 61–69
Deb K, Pratap A, Agarwal S, Meyarivan T (2002) A fast and elitist multiobjective genetic algorithm: Nsga-ii. IEEE Trans Evol Comput 6:182–197
de Castro L, Von Zuben F (2002) Learning and optimization using the clonal selection principle. IEEE Trans Evol Comput 6(3):239–251
Di Gaspero L, Schaerf A (2003) Easylocal++: An object-oriented framework for flexible design of local search algorithms. Softw Pract Exp 33(8):733–765. doi:10.1002/spe.524
Dorigo M, Gambardella L (1997) Ant colony system: a cooperative learning approach to the traveling salesman problem. IEEE Trans Evol Comput 1(1):53–66. doi:10.1109/4235.585892
Dreo, Pétrowski A, Siarry P, Taillard E (2005) Metaheuristics for hard optimization: methods and case studies. Springer
Eiben AE, Raué PE, Ruttkay Z (1994) Genetic algorithms with multi-parent recombination. In: Proceedings of the International Conference on Evolutionary Computation. The Third Conference on Parallel Problem Solving from Nature (PPSN III), Springer, London, pp 78–87
Eshelman LJ (1991) The chc adaptive search algorithm: how to have safe search when engaging in nontraditional genetic recombination. Foundations of Genetic Algorithms pp 265–283. http://ci.nii.ac.jp/naid/10000024547/en/
Eshelman LJ, Schaffer JD (1993) Real-coded genetic algorithms and interval-schemata. In: Whitley DL (ed) Foundation of Genetic Algorithms 2, Morgan Kaufmann., San Mateo, pp 187–202
Eshelman LJ, Caruana RA, Schaffer JD (1989) Biases in the crossover landscape. In: Proceedings of the third international conference on Genetic algorithms, Morgan Kaufmann Publishers Inc., San Francisco, pp 10–19
Feo T, Resende M (1989) A probabilistic heuristic for a computationally difficult set covering problem. Oper Res Lett 8:67–71
Feo TA, Resende MG (1995) Greedy randomized adaptive search procedures. J Glob Optim 6:109–133
Fogarty TC (1989) Varying the probability of mutation in the genetic algorithm. In: Proceedings of the 3rd International Conference on Genetic Algorithms, Morgan Kaufmann Publishers Inc., San Francisco, pp 104–109
Fogel D, Fogel L, Atmar J (1991) Meta-evolutionary programming, vol 1. In: Conference Record of the Twenty-Fifth Asilomar Conference on Signals, Systems and Computers, 1991. pp 540–545. doi:10.1109/ACSSC.1991.186507
Fogel LJ (1964) On the organization of intellect. Ph.D. thesis, UCLA
Fogel LJ, Fogel DB (1986) Artificial intelligence through evolutionary programming. Tech. rep., Final Report for US Army Research Institute, contract no PO-9-X56-1102C-1
Fogel LJ, Owens AJ, Walsh MJ (1966) Artificial intelligence through simulated evolution. Wiley
Fonseca CM, Fleming PJ (1993) Genetic algorithms for multiobjective optimization: Formulationdiscussion and generalization. In: Proceedings of the 5th International Conference on Genetic Algorithms, Morgan Kaufmann Publishers Inc., San Francisco, pp 416–423
Fontoura M, Lucena C, Andreatta A, Carvalho S, Ribeiro C (2001) Using uml-f to enhance framework development: a case study in the local search heuristics domain. J Syst Softw 57(3):201–206
Fowler M (2004) Inversion of control containers and the dependency injection pattern. http://www.martinfowler.com/articles/injection.html
Gagnè C, Parizeau M (2006) Genericity in evolutionary computation software tools: principles and case-study. Int J Artif Intell Tools 15(2):173–194
Gamma E, Helm R, Johnson R, Vlissides J (1994) Design patterns: elements of reusable object-oriented software, illustrated edition. Addison-Wesley Professional
García-Nieto J, Alba E, Chicano F (2007) Using metaheuristic algorithms remotely via ros. In: Proceedings of the 9th annual conference on Genetic and evolutionary computation (GECCO '07), ACM, New York, pp 1510–1510. doi: 10.1145/1276958.1277239
Geman S, Geman D (1987) Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. Readings in computer vision: issues, problems, principles, and paradigms, pp 564–584
Glover F (1977) Heuristics for integer programming using surrogate constraints. Decis Sci 8(1):156–166. doi:10.1111/j.1540-5915.1977.tb01074.x
Glover F (1989) Tabu search—part i. ORSA J Comput 1:190–206
Glover F, Kochenberger GA (2002) Handbook of metaheuristic. Kluwer Academic Publishers
Goldberg D, Lingle R (1985) Alleles loci and the traveling salesman problem. In: Proc. 1st Int. Conf. on Genetic Algorithms and their Applications, pp 154–159
Goldberg DE (1989) Genetic algorithms in search, optimization, and machine learning. Addison-Wesley.
Goldberg DE (1990) A note on boltzmann tournament selection for genetic algorithms and population-oriented simulated annealing. Complex Syst 4(4):445–460

Goldberg DE, Smith RE (1987) Nonstationary function optimization using genetic algorithm with dominance and diploidy. In: Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application, L. Erlbaum Associates Inc., Hillsdale, pp 59–68

Hansen P, Mladenović N, Perez-Britos D (2001) Variable neighborhood decomposition search. J Heuristics 7(4):335–350. doi:10.1023/A:1011336210885

Ho YC, Pepyne DL (2002) Simple explanation of the no-free-lunch theorem and its implications. J Optim Theory Appl 115(3):549–570. http://www.ingentaconnect.com/content/klu/jota/2002/00000115/00000003/00450394

Holland JH (1975) Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence. University of Michigan Press

Holland JH (1992) Adaptation in natural and artificial systems. MIT Press, Cambridge

Horn J, Nafpliotis N, Goldberg D (1994) A niched pareto genetic algorithm for multiobjective optimization. In: Proceedings of the First IEEE Conference on Evolutionary Computation, 1994, vol 1, pp 82–87. doi:10.1109/ICEC.1994.350037

Iredi S, Merkle D, Middendorf M (2001) Bi-criterion optimization with multi colony ant algorithms. In: Proceedings of the First International Conference on Evolutionary Multi-Criterion Optimization (EMO '01), Springer, London, pp 359–372

Jong KAD (1975) An analysis of the behavior of a class of genetic adaptive systems. Ph.D. thesis, University of Michigan

Kennedy J, Eberhart R (1995) Particle swarm optimization, vol 4. In: Proceedings., IEEE International Conference on Neural Networks, 1995. pp 1942–1948. doi:10.1109/ICNN.1995.488968.

Kennedy J, Mendes R (2002) Population structure and particle swarm performance. In: Proceedings of the IEEE Congress on Evolutionary Computation (CEC)

Kirkpatrick S, Gelatt CD, Vecchi MP (1983) Optimization by simulated annealing. Science 220(4598):671–680. http://www.jstor.org/stable/1690046

Kitchenham BA (2004) Procedures for undertaking systematic reviews. Tech. rep., Computer Science Department, Keele University

Knowles JD, Corne DW (2000) Approximating the nondominated front using the pareto archived evolution strategy. Evol Comput 8(2):149–172. doi:10.1162/106365600568167

Koza JR (1992) Genetic programming: on the programming of computers by natural selection. MIT Press

Kronfeld M, Planatscher H, Zell A (2010) The EvA2 optimization framework. In: Blum C, Battiti R (eds) Learning and Intelligent Optimization Conference, Special Session on Software for Optimization (LION-SWOP), Springer, Venice, no. 6073 in Lecture Notes in Computer Science, LNCS, pp 247–250. http://www.ra.cs.uni-tuebingen.de/publikationen/2010/Kron10EvA2Short.pdf

Luke S, Panait L, Balan G, Paus S, Skolicki Z, Popovici E, Sullivan K, Harrison J, Bassett J, Hubley R, Chircop A, Compton J, Haddon W, Donnelly S, Jamil B, O'Beirne J (2009) Ecj: A java-based evolutionary computation research system. http://cs.gmu.edu/eclab/projects/ecj/

Martin Lukasiewycz FR Michael Glaβ, Helwig S (2009) Opt4j—the optimization framework for java. http://www.opt4j.org

Meffert K (2006) JUnit Profi-Tips. Entwickler Press

Michalewicz Z (1994) Genetic algorithms plus data structures equals evolution programs. Springer, New York

Michalewicz Z, Fogel DB (2004) How to solve it: modern heuristics. Springer

Mladenović N (1995) A variable neighborhood search algorithm - a new metaheuristic for combinatorial optimization. Abstracts of papers published at Optimization Week, p 112

Monmarchè N, Venturini G, Slimane M (2000) On how pachycondyla apicalis ants suggest a new search algorithm. Futur Gener Comput Syst 16(9):937–946

Montana DJ (1995) Strongly typed genetic programming. Evol Comput 3(2):199–230. doi:10.1162/evco.1995.3.2.199

Montana DJ, Davis L (1989) Training feedforward neural networks using genetic algorithms. In: Proceedings of the 11th international joint conference on Artificial intelligence (IJCAI' 89). Morgan Kaufmann Publishers Inc., San Francisco, pp 762–767

Muhlenbein H (1991) Evolution on time and space-the parallel genetic glgorithm. Foundations of Genetic Algorithms. http://ci.nii.ac.jp/naid/10016718767/en/

Nossal GJV, Lederberg J (1958) Antibody production by single cells. Nature 181(4620):1419–1420. doi:10.1038/1811419a0

Nulton JD, Salamon P (1988) Statistical mechanics of combinatorial optimization. Phys Rev A 37(4):1351–1356. doi:10.1103/PhysRevA.37.1351

Oliver IM, Smith DJ, Holland JRC (1987) A study of permutation crossover operators on the traveling salesman problem. In: Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application, L. Erlbaum Associates Inc., Hillsdale, pp 224–230

Parejo JA, Racero J, Guerrero F, Kwok T, Smith K (2003) Fom: A framework for metaheuristic optimization. Computational Science ICCS 2003 Lecture Notes in Computer Science 2660:886–895, no-indexada

Parsopoulos K, Vrahatis M (2002a) Recent approaches to global optimization problems through particle swarm optimization. Nat Comput 1(2):235–306

Parsopoulos KE, Vrahatis MN (2002b) Particle swarm optimization method in multiobjective problems. In: Proceedings of the 2002 ACM symposium on Applied computing (SAC '02), ACM, New York, pp 603–607. doi:10.1145/508791.508907

Price KV, Storn RM, Lampinen JA (2005) Differential evolution: a practical approach to global optimization. Natural Computing Series, Springer, Berlin. http://www.springer.com/west/home/computer/foundations?SGWID=4-156-22-32104365-0&teaserId=68063&CENTER_ID=69103

Radcliffe NJ (1991) Forma analysis and random respectful recombination. In: In Foundations of Genetic Algorithms, pp 222–229

Rahman I, Das AK, Mankar RB, Kulkarni BD (2009) Evaluation of repulsive particle swarm method for phase equilibrium and phase stability problems. Fluid Phase Equilibria. doi:10.1016/j.fluid.2009.04.014

Raidl GR (2006) Hybrid metaheuristics. Springer, chap a unified view on hybrid metaheuristics, pp 1–12

Rechenberg I (1965) Cybernetic solution path of an experimental problem. Royal Aircraft Establishment Library Translation 1122, Farnborough

Renders JM, Bersini H (1994) Hybridizing genetic algorithms with hill-climbing methods for global optimization: two possible ways. In: Proceedings of the First IEEE Conference on Computational Intelligence, vol 1, pp 312–317. doi:10.1109/ICEC.1994.349948

Roli A, Blum C (2008) Hybrid metaheuristics: an introduction. In: Hybrid metaheuristics, Springer

Rothlauf F (2006) Representations for genetic and evolutionary algorithms, 2nd edn. Springer

Sasaki D (2005) Armoga: an efficient multi-objective genetic algorithm. Tech. rep.

Schaffer JD, Morishima A (1987) An adaptive crossover distribution mechanism for genetic algorithms. In: Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application, L. Erlbaum Associates Inc., Hillsdale, pp 36–40

Schwefel HP (1981) Numerical optimization of computer models. Wiley, New York

Sloane NJA, Hardin RH (1991–2003) Gosset: a general-purpose program for designing experiments. http://www.research.att.com/njas/gosset/index.html

de Souza MC, Martins P (2008) Skewed vns enclosing second order algorithm for the degree constrained minimum spanning tree problem. Eur J Oper Res 191(3):677–690. doi:10.1016/j.ejor.2006.12.061, http://www.sciencedirect.com/science/article/B6VCT-4N2KTC4-7/2/7799160d76fbba32ad42f719ee72bbf9

Stutzle T, Hoos H (1997) Max–min ant system and local search for the traveling salesman problem. In: EEE International Conference on Evolutionary Computation, 1997, pp 309–314. doi:10.1109/ICEC.1997.592327.

Suganthan PN (1999) Particle swarm optimiser with neighbourhood operator. In: Proceedings of the IEEE Congress on Evolutionary Computation (CEC), pp 1958–1962

Suresh R, Mohanasundaram K (2004) Pareto archived simulated annealing for permutation flow shop scheduling with multiple objectives, vol 2. In: IEEE Conference on Cybernetics and Intelligent Systems, 2004, pp 712–717. doi:10.1109/ICCIS.2004.1460675

Syswerda G (1991) Foundations of genetic algorithms. Morgan Kaufmann, chap a study of reproduction in generational and steady-state genetic algorithms

Talbi EG (2002) A taxonomy of hybrid metaheuristics. J Heuristics 8(5):541–564

Ulungu E, Teghem J, Fortemps P, Tuyttens D (1999) Mosa method: a tool for solving multiobjective combinatorial optimization problems. J Multi Criter Decis Anal 8(4):221–236

Van Veldhuizen DA, Lamont GB (2000) Multiobjective optimization with messy genetic algorithms. In: Proceedings of the 2000 ACM symposium on Applied computing (SAC '00), ACM, New York, pp 470–476. doi:10.1145/335603.335914

Ventura S, Romero C, Zafra A, Delgado J, Hervás C (2008) Jclec: a java framework for evolutionary computation. Soft computing—a fusion of foundations, methodologies and applications 12(4):381–392. 10.1007/s00500-007-0172-0

Vesterstrm JS, Riget J (2002) Particle swarms: extensions for improved local, multi-modal, and dynamic search in numerical optimization. Ph.D. thesis, Dept. of Computer Science, University of Aarhus

Voß S (2001) Meta-heuristics: the state of the art. pp 1–23

Voß S (2002) Optimization software class libraries. Kluwer Academic Publishers

Wagner S (2009) Heuristic optimization software systemsm odeling of heuristic optimization algorithms in the heuristic lab software environment. Ph.D. thesis, Johannes Kepler University, Linz

Whitley D (1989) The genitor algorithim and selection pressure: Why rank-based allocation of reproductive trials is best. In: Proceedings of the Third International Conference on Genetic Algorithms, pp 116–121

Whitley D, Rana S, Heckendorn RB (1999) The island model genetic algorithm: on separability, population size and convergence. CIT J Comput Inf Technol 7(1):33–47

Wilke DN, Kok S, Groenwold AA (2007) Comparison of linear and classical velocity update rules in particle swarm optimization: notes on diversity. International. Int J Numer Methods Eng 70(8):962–984

Wilson GC, Mc Intyre A, Heywood MI (2004) Resource review: Three open source systems for evolving programs–lilgp, ecj and grammatical evolution. Genet Program Evolv Mach 5(1):103–105. doi:10.1023/B:GENP.0000017053.10351.dc

Wolpert DH, Macready WG (1997) No free lunch theorems for optimization. IEEE Trans Evol Comput 1(1):67–82. doi:10.1109/4235.585893.

Wright AH (1994) Genetic algorithms for real parameter optimization. In: Foundations of genetic algorithms, Morgan Kaufmann, pp 205–218

Yao X, Liu Y (1996) Fast evolutionary programming. In: Proc. 5th Ann. Conf. on Evolutionary Programming

Zhou H, Grefenstette JJ (1986) Induction of finite automata by genetic algorithms. In: Proceedings of the 1986 IEEE International Conference on Systems, Man, and Cybernetics, pp 170–174

Zitzler E, Thiele L (1999) Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach. IEEE Trans Evol Computation 3(4):257–271

Zitzler E, Laumanns M, Thiele L (2001) Spea2: Improving the strength pareto evolutionary algorithm. Tech. rep., Computer Engineering and Networks Laboratory (TIK). Department of Electrical Engineering. Swiss Federal Institute of Technology (ETH)