ParadisEO-MOEO: A Software Framework for Evolutionary Multi-Objective Optimization

Arnaud Liefooghe, Laetitia Jourdan, Thomas Legrand, Jérémie Humeau, El-Ghazali Talbi

5

ParadisEO-MOEO: A Software Framework for Evolutionary Multi-Objective Optimization

Arnaud Liefooghe[1,2], Laetitia Jourdan[1,2], Thomas Legrand[2], Jérémie Humeau[2], and El-Ghazali Talbi[1,2]

[1] Laboratoire d'Informatique Fondamentale de Lille (LIFL), UMR CNRS 8022, Université Lille 1, Bâtiment M3, 59655 Villeneuve d'Ascq cedex, France
[2] INRIA Lille-Nord Europe, Parc Scientifique de la Haute Borne, 40 avenue Halley, 59650 Villeneuve d'Ascq, France
Arnaud.Liefooghe@lifl.fr, Laetitia.Jourdan@lifl.fr, Thomas.Legrand@inria.fr, Jeremie.Humeau@inria.fr, talbi@lifl.fr

Summary. This chapter presents ParadisEO-MOEO, a white-box object-oriented software framework dedicated to the flexible design of metaheuristics for multi-objective optimization. This paradigm-free software proposes a unified view for major evolutionary multi-objective metaheuristics. It embeds some features and techniques for multi-objective resolution and aims to provide a set of classes allowing to ease and speed up the development of computationally efficient programs. It is based on a clear conceptual distinction between the solution methods and the problems they are intended to solve. This separation confers a maximum design and code reuse. This general-purpose framework provides a broad range of fitness assignment strategies, the most common diversity preservation mechanisms, some elitist-related features as well as statistical tools. Furthermore, a number of state-of-the-art search methods, including NSGA-II, SPEA2 and IBEA, have been implemented in a user-friendly way, based on the fine-grained ParadisEO-MOEO components.

5.1 Introduction

A large number of existing real-world optimization problems are characterized by multiple conflicting objective functions. Evolutionary algorithms are commonly used to solve these multi-objective problems since they are particularly well-suited to find a spread set of good-quality solutions. Over the past few years, major contributions have been made in the field of evolutionary multi-objective optimization. In this work, we propose a new software framework for evolutionary multi-objective optimization called ParadisEO-MOEO. Its modular implementation follows a general purpose model based on a fine-grained decomposition. This model is founded on a unified view of evolutionary algorithms for multi-objective optimization where the fundamental issues of fitness assignment, diversity preservation and elitism are

C.A. Coello Coello et al. (Eds.): Adv. in Multi-Obj. Nature Inspired Computing, SCI 272, pp. 87–117.
springerlink.com

involved. ParadisEO[1] is a free open-source C++ white-box object-oriented framework dedicated to the reusable design of metaheuristics. It attempts to simplify and accelerate the development process of efficient solver programs while having a minimal programming effort. It is based on a clear conceptual separation between the problem-specific and the invariant part of the solution method. This separation is expressed at the implementation level, which confers a maximum design and code reuse. ParadisEO is composed of four connected modules: ParadisEO-EO for population-based metaheuristics, ParadisEO-MO for single solution-based metaheuristics, ParadisEO-MOEO for multi-objective metaheuristics and ParadisEO-PEO for parallel and distributed models for metaheuristics and their hybridization. Each module has been validated and successfully applied to solve a wide range of academic and real-world optimization problems of both continuous and combinatorial nature. Historically, ParadisEO was especially dedicated to parallel and distributed metaheuristics and was the result of the PhD work of Sébastien Cahon, supervised by Nouredine Melab and El-Ghazali Talbi [10]. The initial version already contained a few number of features related to evolutionary multi-objective optimization, mainly with regard to elitism. This work has been partially extended and presented in [33]. But since then, the ParadisEO-MOEO module has been completely redesigned in order to confer an even more fine-grained decomposition, and major additional features have been integrated into the framework.

In this chapter, we provide a general presentation of ParadisEO, and a detailed description of the ParadisEO-MOEO module. First, a unified view of evolutionary multi-objective optimization techniques is presented in Sect. 5.2. Then, software frameworks are discussed in Sect. 5.3. Sect. 5.4 is devoted to the design and the implementation of evolutionary multi-objective metaheuristics with ParadisEO-MOEO. A case study on a bi-objective scheduling problem is given in Sect. 5.5. Finally, the last section concludes the chapter.

5.2 Evolutionary Multi-Objective Optimization, a Unified View

This section presents some basic concepts about Evolutionary Multi-objective Optimization (EMO). Next, a couple of related design issues are briefly discussed and a unified model for EMO algorithms is proposed.

5.2.1 Evolutionary Multi-Objective Optimization

A Multi-objective Optimization Problem (MOP) can be defined by a set f of $n \geq 2$ objective functions $f_1, f_2, \ldots, f_n$, a set X of feasible solutions in the *decision space*, and a set Z of feasible points in the *objective space*. Without loss of generality, we here assume that $Z \subseteq \mathbb{R}^n$ and that all n objective functions are to be minimized. To each decision vector $x \in X$ is assigned an objective vector $z \in Z$ on the basis of the vector function $f : X \rightarrow Z$ with $z = f(x) = f_1(x), f_2(x), \ldots, f_n(x)$.

[1] `http://paradiseo.gforge.inria.fr`

Definition 1 (Pareto dominance). *An objective vector $z \in Z$ is said to* dominate *another objective vector $z' \in Z$ if $\forall i \in \{1,2,\ldots,n\}$, $z_i \leq z'_i$ and $\exists j \in \{1,2,\ldots,n\}$ such as $z_j < z'_j$.*

Definition 2 (Non-dominated point). *An objective vector $z \in Z$ is said to be* non-dominated *if there does not exist any other objective vector $z' \in Z$ such that z' dominates z.*

By extension, we say that a decision vector $x \in X$ *dominates* a decision vector $x' \in X$ if $f(x)$ dominates $f(x')$, and that a decision vector $x \in X$ is *non-dominated* (or *efficient*, *Pareto optimal*) if $f(x)$ is non-dominated. Note that other dominance relations exist in the frame of multi-objective optimization and will be discussed later in the chapter. A possible MOP resolution method is to find the minimal set of efficient solutions, *i.e.* one feasible solution per non-dominated point. However, generating the entire efficient set is usually infeasible, due to the complexity of the underlying problem or the large number of optima. Therefore, in many approaches, the overall goal is to identify a good approximation of it. Evolutionary algorithms are commonly used to this end, as they are particularly well-suited to find multiple efficient solutions in a single simulation run. The reader is referred to [11, 13] for more details about EMO.

5.2.2 Design Issues

As pointed out by various authors (see *e.g.* [11, 49]), approximating the efficient set is itself a bi-objective problem. Indeed, the approximation to be found must have both good convergence and distribution properties, as its mapping in the objective space has to be (*i*) close to, and (*ii*) well-spread over the (unknown) Pareto optimal front. As a consequence, the main differences between the design of a single- and a multi-objective metaheuristic in general, and EA in particular, deal with these two goals. As noticed by Zitzler et al. [49], in the EMO literature, initial approaches were mainly focused on moving toward the Pareto front [19, 39]. Afterward, diversity preservation mechanisms quickly emerged [18, 26, 41]. Then, at the end of the twentieth century, the concept of elitism, related to the preservation of non-dominated solutions, became very popular and is now employed in most recent EMO methods [50, 51]. The importance of the issues of *fitness assignment*, *diversity preservation* and *elitism* are commonly approved and are also presented under different names in, for instance, [11, 49]. They are discussed in details below.

Fitness Assignment

In the single-objective case, the fitness value assigned to a given solution is most often its unidimensional objective value. While dealing with MOPs, fitness assignment aims to guide the search toward Pareto optimal solutions for a better convergence. We propose to classify existing fitness assignment schemes into four different classes:

- *Scalar approaches*, where the MOP is reduced to a single-objective optimization problem. A popular example consists in combining the n objective functions into a single one by means of a weighted-sum aggregation. Other examples are ε-constraint or achievement function-based methods, see [36].
- *Criterion-based approaches*, where each objective function is treated separately. For instance, in VEGA (Vector Evaluated GA) [39], a parallel selection is performed where solutions are discerned according to their values on a single objective function, independently to the others. In lexicographic methods [19], a hierarchical order is defined between objective functions.
- *Dominance-based approaches*, where a dominance relation is used to classify solutions. Existing strategies are *dominance-rank* [18], *dominance-count* [51] and *dominance-depth* [21] techniques. And different schemes can also be combined, as for instance in [51]. In the frame of dominance-based approaches, the most commonly used dominance relation is the Pareto-dominance relation given in Definition 1. But some recent techniques are based on other dominance operators such as ε-dominance in [15] or g-dominance in [37].
- *Indicator-based approaches*, where the fitness values are computed by comparing individuals on the basis of a quality indicator I. The chosen indicator represents the overall goal of the search process. Generally speaking, no particular diversity preservation mechanism usually necessary, with regards to the indicator being used. Examples of indicator-based EAs are IBEA (Indicator-Based EA) [48] and SMS-EMOA (S-Metric Selection EMO Algorithm) [6].

Diversity Preservation

As noticed earlier, aiming at approximating the efficient set is not only a question of convergence. The final approximation also has to be well spread over the objective space. However, classical fitness assignment schemes often tend to produce premature convergence by privileging non-dominated solutions, which does not guarantee a uniformly sampled output set. In order to prevent that issue, a diversity preservation mechanism, based on a given distance measure, is usually integrated into the metaheuristic to uniformly distribute the population over the trade-off surface. In the frame of EMO, a common distance measure is based on the euclidean distance between objective vectors. But, this measure can also be defined in the decision space or can even combine both spaces. Popular examples of diversity preservation techniques are *fitness sharing* [22] and *crowding* [25], respectively used in *e.g.* Fonseca and Fleming's MOGA (Multi-Objective GA) [18] and Deb et al.'s NSGA-II (Non-dominated Sorting GA II) [14].

Elitism

Another essential issue about MOP solving is the notion of *elitism*. It mainly consists in maintaining an external set, the so-called *archive*, that allows to store either all or a subset of non-dominated solutions found during the search process. This secondary population mainly aims at preventing the loss of these solutions during

the stochastic optimization process. The update of the archive contents with new potential non-dominated solutions is mostly based on the Pareto-dominance criteria. But, in the literature, other dominance criteria are found and can be used instead of the Pareto-dominance relation. Examples are weak-dominance, strict-dominance, ε-dominance [24], etc. When dealing with archiving, one may distinguished four different techniques depending on the problem properties and on the designed algorithm: (*i*) *no archive*, (*ii*) an *unbounded archive*, (*iii*) a *bounded archive* or (*iv*) a *fixed-size archive*. First, if the approximation set is maintained by, or contained into the population itself, there can be no archive at all. On the other hand, if an archive is maintained, it usually comprises the current non-dominated set approximation, as dominated solutions are removed. Then, an unbounded archive can be used in order to save the whole set of non-dominated solutions found since the beginning of the search process. However, as some continuous optimization problems may contain an infinite number of non-dominated solutions, it is simply not possible to save them all. Therefore, additional operations must be used to reduce the number of stored solutions. Then, a common strategy is to bound the size of the archive according to some fitness and/or diversity assignment scheme(s). Finally, another archiving technique consists of a fixed size storage capacity, where a bounding mechanism is used when there are too many non-dominated solutions, and some dominated solutions are integrated into the archive if the non-dominated set is too small. This is done, for instance, in the frame of SPEA2 [50]. Usually, an archive is used as an external storage only. However, archive members can also be integrated during the selection phase of an EMO algorithm [51].

5.2.3 A Unified Model

An Evolutionary Algorithm (EA) [21] is a search method where a population of solutions is iteratively improved by means of some stochastic operators. EAs belong to the class of population-based metaheuristics. Starting from an initial population, each individual is evaluated in the objective space and a selection scheme is performed to build a so-called parent population. An offspring population is then created by applying variation operators. Next, a replacement strategy determines which individuals will survive in the next EA generation. The search process is iterated until a given stopping criterion is satisfied. As noticed earlier in the chapter, in the frame of EMO, the main expansions deal with the issues of *fitness assignment*, *diversity preservation* and *elitism*. Fitness and diversity informations are necessary to discriminate individuals at to the selection and the replacement steps of the EA. Moreover, the update of the archive contents possibly appears at each EA iteration. As a consequence, whatever the MOP to be solved, the common concepts for the design of an EMO algorithm are the following ones.

1. Design a representation.
2. Design a population initialization strategy.
3. Design a way of evaluating a solution.

4. Design suitable variation operators.
5. Decide a fitness assignment strategy.
6. Decide a diversity preservation strategy.
7. Decide a selection strategy.
8. Decide a replacement strategy.
9. Decide an archive management strategy.
10. Decide a continuation strategy.

When dealing with any kind of metaheuristics, one may distinguish problem-specific and generic components. Indeed, the first four common concepts presented above strongly depend of the MOP at hand, while the six last ones can be considered as problem-independent, even if some problem-dependent strategies can also be envisaged in some particular cases. Note that concepts of representation and evaluation are shared by any metaheuristic, concepts of population initialization and stopping criterion are shared by any population-based metaheuristic, concepts of variation operators, selection and replacement are shared by any EA, whereas concepts of fitness, diversity and archiving are specific to EMO.

5.3 Software Frameworks for Evolutionary Multi-Objective Optimization

In this section, the motivations in using a software framework for metaheuristics are outlined. Next, ParadisEO, a platform dedicated to the reusable design of metaheuristics, and ParadisEO-MOEO, its module devoted to EMO are presented. The main characteristics of ParadisEO are then detailed, and a comparative study of existing software frameworks for EMO is given.

5.3.1 Motivations

In practice, there exists a large diversity of optimization problems to be solved, giving rise to a wide number of possible models to be handled, in the context of a metaheuristic solution method. Moreover, a growing number of general-purpose search methods are proposed in the literature, with evolving complex mechanisms. ¿From a practitioner's point of view, there is a popular demand to provide a set of ready-to-use metaheuristic implementations, allowing a minimum programming effort. On the other hand, an expert generally wants to be able to design new algorithms, to integrate new components into an existing method, or even to combine different search mechanisms. As a consequence, an approved approach for the development of metaheuristics is the use of frameworks. A framework may be defined by a set of components based on a strong conceptual separation of the invariant part and the problem-specific part of metaheuristics. Then, each time a new optimization problem is tackled, both code and design can directly be reused in order to redo as little code as possible.

5.3.2 ParadisEO and ParadisEO-MOEO

ParadisEO[2] is a white-box object-oriented software framework dedicated to the flexible design of metaheuristics for optimization problems of both discrete and combinatorial nature. Based on EO (Evolving Objects)[3] [30], this template-based, ANSI-C++ compliant computation library is portable across both Unix-like and Windows systems. Moreover, it tends to be used both by non-specialists and optimization experts. ParadisEO is composed of four connected modules that constitute a global framework. Each module is based on a clear conceptual separation of the solution methods from the problems they are intended to solve. This separation confers a maximum code and design reuse to the user. The first module, ParadisEO-EO [30], provides a broad range of components for the development of population-based metaheuristics, including evolutionary algorithms and particle swarm optimization techniques. Second, ParadisEO-MO [8] contains a set of tools for single-solution based metaheuristics, *i.e.* hill climbing, simulated annealing, tabu search, iterated local search and variable neighborhood search. Next, ParadisEO-MOEO is specifically dedicated to the reusable design of metaheuristics for multi-objective optimization. Finally, ParadisEO-PEO [10] provides a powerful set of classes for the design of parallel and distributed metaheuristics: parallel evaluation of solutions, parallel evaluation function, island model and cellular model. In the frame of this chapter, we will exclusively focus on the module devoted to multi-objective optimization, namely ParadisEO-MOEO.

ParadisEO-MOEO provides a flexible and modular framework for the design of metaheuristics for multi-objective optimization. Its implementation is based on the unified model proposed in the previous section and is conceptually divided into fine-grained components. On each level of its architecture, a set of abstract classes is proposed and a wide range of sub-classes, corresponding to different state-of-the-art strategies, are also provided. Moreover, as the framework aims to be extensible, flexible and easily adaptable, all its components are generic in order to provide a modular architecture allowing to quickly and conveniently develop any new scheme with a minimum of code writing. The underlying goal here is to follow new strategies coming from the literature and, if necessary, to provide any additional components required for their implementation. Moreover, ParadisEO-MOEO constantly evolves and new features might be added to the framework regularly in order to provide a wide range of efficient and modern concepts and to reflect the most recent advances of the EMO field.

5.3.3 Main Characteristics

A framework is usually intended to be exploited by a large number of users. Its exploitation could only be successful if a range of user criteria are satisfied. Therefore, the main goals of the ParadisEO software framework are the following ones:

[2] `http://paradiseo.gforge.inria.fr`
[3] `http://eodev.sourceforge.net`

- *Maximum design and code reuse.* The framework must provide a whole architecture design for the metaheuristic approach to be used. Moreover, the programmer should need to redo as little code as possible. This aim requires a clear and maximal conceptual separation of the solution methods and the problem to be solved. The user might only write the minimal problem-specific code and the development process might be done in an incremental way, what will considerably simplify the implementation and reduce the development time and cost.
- *Flexibility and adaptability.* It must be possible to easily add new features or to modify existing ones without involving other components. Users must have access to source code and use inheritance or specialization concepts of object-oriented programming to derive new components from base or abstract classes. Furthermore, as existing problems evolve and new others arise, the framework components must be conveniently specialized and adapted.
- *Utility.* The framework must cover a broad range of metaheuristics, fine-grained components, problems, parallel and distributed models, hybridization mechanisms, etc.
- *Transparent and easy access to performance and robustness.* As the optimization applications are often time-consuming, the performance issue is crucial. Parallelism and distribution are two important ways to achieve high performance execution. Moreover, the execution of the algorithms must be robust in order to guarantee the reliability and the quality of the results. Hybridization mechanisms generally allow to obtain robust and better solutions.
- *Portability.* In order to satisfy a large number of users, the framework must support many physical architectures (sequential, parallel, distributed) and their associated operating systems (Windows, Linux, MacOS).
- *Usability and efficiency.* The framework must be easy to use and must not contain any additional cost in terms of time or space complexity in order to keep the efficiency of a special-purpose implementation. On the contrary, the framework is intented to be less error-prone than a specifically developed metaheuristic.

The ParadisEO platform honors all the above-mentioned criteria and aims to be used by both non-specialists and optimization experts. Furthermore, the ParadisEO-MOEO module must cover additional goals related to multi-objective optimization. Thus, in terms of design, it might, for instance, be a commonplace to extend a single-objective optimization problem to the multi-objective case without modifying the whole metaheuristic implementation.

5.3.4 Existing Software Frameworks for Evolutionary Multi-Objective Optimization

Many frameworks dedicated to the design of metaheuristics have been proposed so far. However, very few are able to handle MOPs, even if some of them provide components for a few particular EMO strategies, such as ECJ [1], JavaEVA [42] or Open BEAGLE [20]. Table 5.1 gives a non-exhaustive comparison between a number of existing software frameworks for multi-objective metaheuristics, including jMetal [17], the MOEA toolbox for Matlab [45], MOMHLib++ [2], PISA [7]

Table 5.1. Main characteristics of some existing frameworks for multi-objective metaheuristics

Framework	Problems		Statistical tools		Hybrid.	Parallel	Type	Lang.	License
	Cont.	Comb.	Off-line	On-line					
jMetal	yes	yes	yes	no	yes	no	white	java	free
MOEA for Matlab	yes	no	no	no	no	yes	black	matlab	free / com.
MOMHLib++	yes	yes	no	no	yes	no	white	c++	free
PISA	yes	yes	yes	no	no	no	black	any	free
Shark	yes	no	no	no	yes	no	white	c++	free
ParadisEO	yes	yes	yes	yes	yes	yes	white	c++	free

and Shark [3]. Note that other software exists for multi-objective optimization [38], but some of them cannot be considered as frameworks and others do not deal with metaheuristics. The frameworks presented in Table 5.1 are distinguished according to the following criteria: the kind of MOPs they are able to tackle (continuous and/or combinatorial problems), the availability of statistical tools (including performance metrics), the availability of hybridization or parallel features, the framework type (black box or white box), the programming language and the license type (free or commercial).

First, let us mention that every listed software framework is free of use, except for the MOEA toolbox, which requires the commercial software Matlab. They can all handle continuous problems, but only a subset of them are able to deal with combinatorial MOPs. Moreover, some cannot be considered as white-box frameworks since their architecture is not decomposed into components. For instance, to design a new algorithm under PISA, it is necessary to implement it from scratch, as no existing element can be reused. Similarly, even if Shark can be considered as a white-box framework, its components are not as fine-grained as the ones of ParadisEO. On the contrary, ParadisEO is an open platform where anyone can contribute and add his/her own features. Finally, only a few ones are able to deal with hybrid and parallel metaheuristics at the same time. Hence, in opposition to jMetal and MOMHLib++, ParadisEO offers easy-to-use model for the design of parallel and distributed features. Therefore, in comparison to other existing software frameworks dedicated to multi-objective metaheuristics design, ParadisEO is the only one that achieves all the aforementioned goals.

5.4 Design and Implementation of Evolutionary Multi-Objective Metaheuristics with ParadisEO-MOEO

This section gives a detailed description of the base classes provided within the ParadisEO framework to design an EMO algorithm.[4] The flexibility of the framework and its modular architecture, based on the three main multi-objective metaheuristic design issues (fitness assignment, diversity preservation and elitism), allows to implement efficient algorithms in solving a large diversity of MOPs. The granular

[4] The classes presented in this paper are described as in version 1.2 of ParadisEO.

decomposition of ParadisEO-MOEO is based on the unified model proposed in the previous section.

As an EMO algorithm differs of a single-objective one only in a number of points, some ParadisEO-EO components are directly reusable. Therefore, in the following, note that the names of ParadisEO-EO classes are all prefixed by `eo` whereas the names of ParadisEO-MOEO classes are prefixed by `moeo`. ParadisEO is an object-oriented platform, so that its components will be specified by the UML standard [4]. But, due to space limitations, only a subset of the UML diagrams is provided, but the whole inheritance diagram as well as the classes documentation and many examples of use are available at the ParadisEO website. Moreover, a large part of the ParadisEO components are based on the notion of template and are defined as class templates. This concept and many related functions are featured within the C++ programming language and allows the classes to handle generic types, so that they can work with many different data types without having to be rewritten for each one.

In this section, both problem-dependent and problem-independent components are detailed. First, basic elements (representation, evaluation, initialization and stopping criteria) are outlined. Then comes the EMO-specific (fitness, diversity and elitism) and EA-related (variation, selection, replacement) components. Finally, the way to build a whole EMO algorithm is presented and a brief discussion concludes the section.

5.4.1 Basic Components

In this section, basic components are presented: solution representation, evaluation, initialization and stopping criteria.

Representation

Solution representation is the starting point for anyone who plans to design any kind of metaheuristic. Successful applications of metaheuristics strongly requires a proper solution representation. Various encodings may be used such as binary variables, real-coded vectors, permutations, discrete vectors, and more complex representations. Note that the choice of a representation will considerably influence the way solutions will be initialized and evaluated in the objective space, and the way variation operators will be applied. A solution needs to be represented both in the decision space and in the objective space. While the representation in the objective space can be seen as problem-independent, the representation in the decision space must be relevant to the tackled problem. In the single-objective case, a single value is usually used for the representation in the unidimensional objective space. For MOPs, where the objective space is multi-dimensional, a tuple of n values, called *objective vector*, might be used for such a representation. Using ParadisEO-MOEO, the first thing to do is to set the number of objectives for the problem under consideration and, for each one, if it has to be minimized or maximized. This can be done by using the `moeoObjectiveVectorTraits` static class. Then, a class templatized with the later one and inheriting of `moeoObjectiveVector` has to

be created for the representation of an objective vector, as illustrated in Fig. 5.1. Since a big majority of MOPs deal with real-coded objective values, a class modelling real-coded objective vectors is already provided within ParadisEO-MOEO. Note that this class can be used for any MOP without loss of generality.

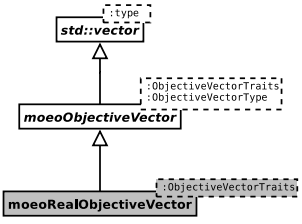

Fig. 5.1. UML diagram for the representation of a solution in the objective space

The class used to represent a whole solution within ParadisEO-MOEO is then templatized within a given objective vector type, and must define its representation in the decision space, which fully depends of the tackled problem. In the implementation-level, the way to do so is to extend the `MOEO` class in order to be used for a specific problem. This modeling is applicable for every kind of problem with the aim of being as general as possible. Nevertheless, ParadisEO-MOEO also provides easy-to-use classes for standard vector-based representations and, in particular, implementations for vectors composed of bits, of integers or of real-coded values, that can thus directly be used in a ParadisEO-MOEO-designed application. These classes are summarized in Fig. 5.2.

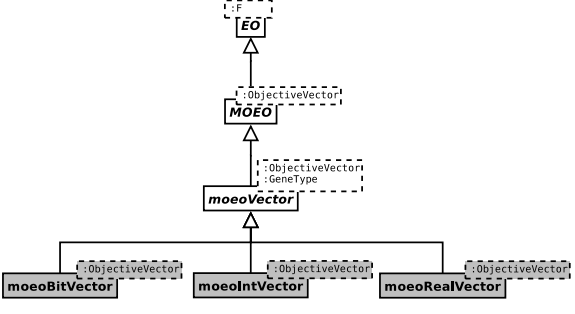

Fig. 5.2. UML diagram for the representation of a solution

Evaluation

The problem at hand is to optimize a set of objective functions simultaneously over a given search space. Then, each time a new solution integrates the population,

its objective vector must be evaluated, *i.e.* the value corresponding to each objective function must be set. ParadisEO-MOEO stores an objective vector within any `MOEO` object, and the way it is computed is ensured by components inheriting of the `eoEvalFunc` abstract class which is illustrated in Fig. 5.3. It basically takes a `MOEO` object and sets its objective vector. Moreover, note that a C++ function can be embedded into an `eoEvalFuncPtr` object in order to apply it to the individual and to set its objective values. Similarly, the `eoExternalEvalFunc` class provides a component able to embed an external evaluation function which is then considered as a black-box function by the problem solver. Finally, the `eoEvalFuncCounter` class allows to count the number of evaluations performed until the end of the algorithm. The resulting counter can either serve as a stopping criteria or provide a statistical resource to the user.

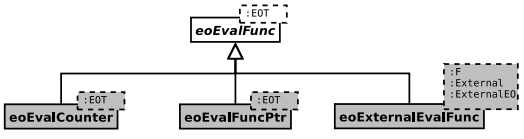

Fig. 5.3. UML diagram for evaluation

Generally speaking, for real-world optimization problems, the evaluation of a solution in the objective space is by far the most computationally expensive step of the chosen metaheuristic approach. A possible way to overcome this trouble is the use of parallel and distributed models, that can largely be simplified in the context of ParadisEO thanks to the ParadisEO-PEO module of the software library. The reader is referred to [10] for more information on how to design parallel and distributed metaheuristics within ParadisEO-PEO.

Initialization

Whatever the algorithmic solution to be designed, a way to initialize a solution (or a population of solutions) is expected. While dealing with any population-based metaheuristic, one has to keep in mind that the initial population must be diversified in order to prevent a premature convergence. This remark is even more true for MOPs where the goal is to find a well-converged and a well-spread approximation. The way to initialize a solution is closely related to the problem under consideration and to the representation at hand. In most approaches, the initial population is generated randomly or according to a given diversity function. A number of initialization schemes already exist in a lot of libraries for standard representations, which is also the case within ParadisEO. But some situations could require a combination of many operators or a specific implementation. Indeed, as shown in Fig. 5.4, the framework provides a range of initializers all inheriting of `eoInit`, as well as an easy way to combine them thanks to an `eoCombinedInit` object.

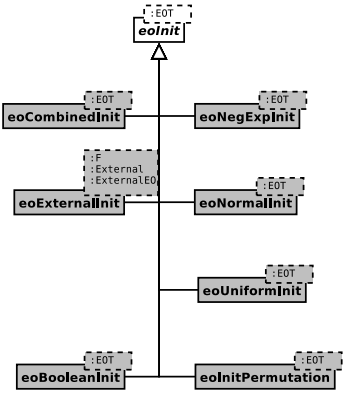

Fig. 5.4. UML diagram for initialization

Stopping Criteria, Checkpointing and Statistical Tools

Since an iterative method computes successive approximations, a practical test is required to determine when the process must stop. As illustrated in Fig. 5.5, in the frame of ParadisEO, many stopping criteria extending `eoContinue` are provided. For instance, the algorithm can stop after a given number of iterations (`eoGenContinue`), a given number of evaluations (`eoEvalContinue`), a given run time (`eoTimeContinue`) or in an interactive way, as soon as the user decides to (`eoCtrlCContinue`). Moreover, note that different stopping criteria can be combined thanks to an `eoCombinedContinue` object, in which case the process stops once one of the embedded criteria is satisfied.

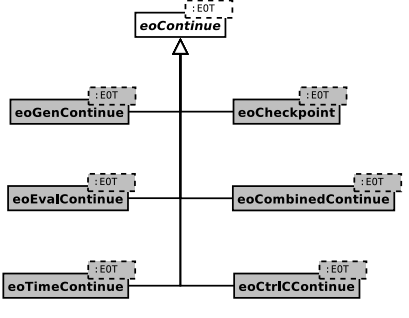

Fig. 5.5. UML diagram for stopping criteria

In addition, many other procedures may be called at each iteration of the main algorithm. The `eoCheckPoint` class allows to perform some systematic actions at each algorithm iteration in a transparent way by being embedded in the global

`eoContinue` object. The checkpointing engine is particularly helpful for fault tolerance mechanisms and to compute statistics. Indeed, some useful statistical tools are also provided within ParadisEO-MOEO. Then, it is for instance possible to save the contents of the current approximation set at each iteration, so that the evolution of the current non-dominated front can be observed or studied using graphical tools such as Guimoo (Graphical User Interface for Multi-Objective Optimization)[5]. Furthermore, as pointed out in Sect. 5.2, an important issue in the EMO field relates to the algorithm performance analysis and to set quality metrics [52]. As shown in Fig. 5.6, a couple of metrics are featured within ParadisEO-MOEO. Unary metrics are used to quantify the quality of a non-dominated set (or of a single solution), while binary metrics are used for pairwise comparisons (between two non-dominated sets or solutions). Thus, the hypervolume metric is available both in its unary [51] and its binary [52] form. Moreover, the entropy metric [5], the contribution metric [35] as well as the additive and the multiplicative ε-indicators [52] are all implemented and can thus be used to compare two sets of solutions. Besides, some implementations for pairwise comparison of solutions (that are then usable within the binary indicator-based fitness assignment schemes, see Sect. 5.4.2) are also proposed. Of course, other metrics can easily be implemented by inheritance. Another interesting feature is the possibility to compare the current archive with the archive of the previous iteration by using a given binary metric, and to print the progression of this measure iteration after iteration.

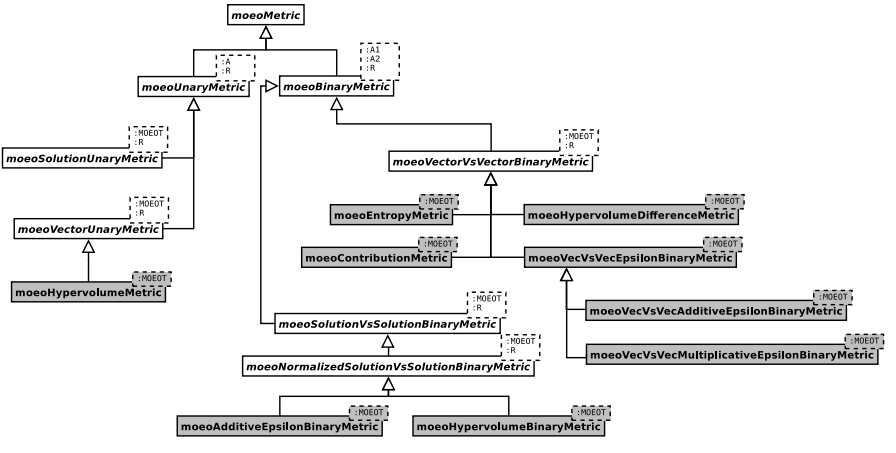

Fig. 5.6. UML diagram for metrics

5.4.2 EMO-Related Components

Here, we give a detailed description of EMO-specific components: fitness, diversity and elitism.

[5] `http://guimoo.gforge.inria.fr/`

Fitness Assignment Schemes

The most common fitness assignment strategies are implemented within ParadisEO-MOEO: scalar approaches, dominance-based approaches and indicator-based approaches. Following the taxonomy introduced in Sect. 5.2, the fitness assignment schemes are classified into four categories, as illustrated in the UML diagram of Fig. 5.7:

- Scalar approaches: `moeoScalarFitnessAssignment`
- Criterion-based approaches: `moeoCriterionBasedFitnessAssignment`
- Dominance-based approaches: `moeoDominanceBasedFitnessAssignment`
- Indicator-based approaches: `moeoIndicatorBasedFitnessAssignment`

A detailed description of existing fitness assignment schemes provided within the framework are listed below. Moreover, note that there also exists a dummy fitness assignment strategy in case it would be useful for some specific implementation.

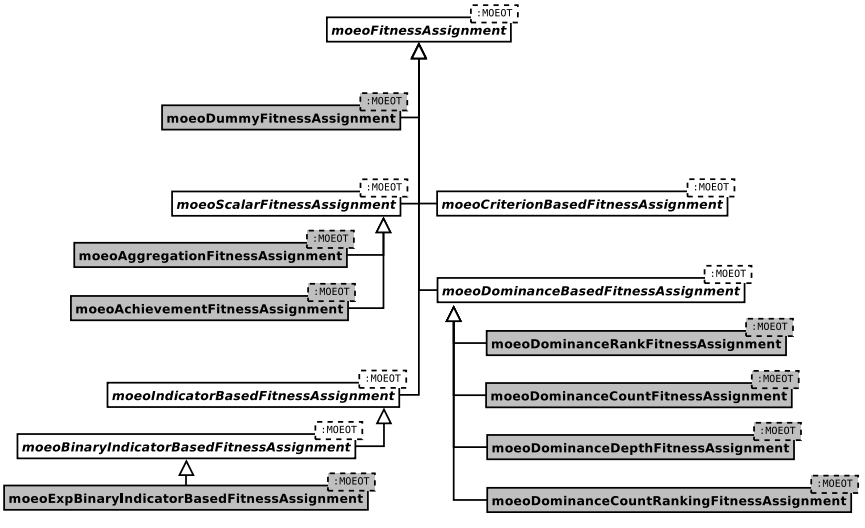

Fig. 5.7. UML diagram for fitness assignment

Achievement Fitness Assignment Scheme

One of the provided fitness assignment schemes is the family of *achievement scalarizing functions*, proposed by Wierzbicki [47]. This scalar approach is based on an arbitrary reference point R, generally given by a decision maker, and consists in projecting R onto the set of Pareto optimal solutions.

Dominance-Rank Fitness Assignment Scheme

In this strategy, the fitness value associated to a given solution x corresponds to the number of population items that dominate x. This scheme has been proposed in [18]

and is, for instance, used in the Fonseca and Fleming MOGA (Multi-Objective GA) [18] and in the Horn et al. NPGA (Niched-Pareto GA) [26].

Dominance-Count Fitness Assignment Scheme

This approach consists in assigning, to a solution x, a fitness value equal to the number of population items that are dominated by x. For instance, it is combined to the dominance rank scheme in the frame of SPEA (Strength Pareto EA) [51] and SPEA2 [50].

Dominance-Depth Fitness Assignment Scheme

Another implemented fitness assignment scheme is the dominance depth approach proposed by Goldberg [21] and used, for instance, in NSGA (Non-dominated Sorting GA) [41] and NSGA-II [14]. This strategy consists in classifying a set of solutions into several classes (or fronts). A solution that belongs to a class does not dominate another one from the same class. Then, individuals from the first front all belong to the best non-dominated set of the population; individuals from the second front all belong to the second best non-dominated set; and so on.

Dominance Count Ranking Fitness Assignment Scheme

In this strategy, the dominance-count and dominance-rank schemes are combined. The fitness value of a solution x corresponds to the sum of ranks of all solutions dominated by x. This technique is used in SPEA2 [50].

Binary Indicator-Based Fitness Assignment Scheme

In this strategy, the fitness values are computed by comparing individuals on the basis of an arbitrary binary quality indicator I (or binary performance metric). Thereby, no particular diversity preservation mechanism is generally necessary, with regards to the indicator being used. The chosen indicator represents the overall goal of the search process. Thus, the fitness value of a solution reflects its usefulness according to this goal. As discussed earlier in the chapter (see Sec. 5.4.1), several binary quality indicators to be used in the frame of this scheme are proposed within ParadisEO-MOEO.

Diversity Assignment Schemes

As illustrated in the UML diagram of Fig. 5.8, diversity preservation strategies must inherit of the `moeoDiversityAssignment` class. In addition to a dummy technique, a number of other diversity assignment schemes are available and are listed below.

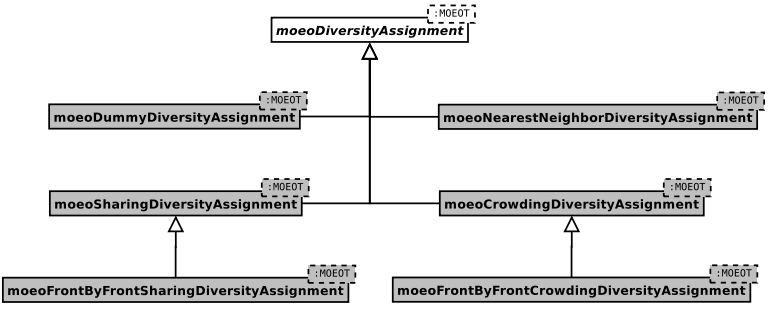

Fig. 5.8. UML diagram for diversity assignment

Sharing Diversity Assignment Scheme

The notion of *sharing* (or *fitness sharing*) was originally suggested by Goldberg and Richardson [22] to preserve diversity among solutions in an EA population. It was first employed by Fonseca and Fleming [18] in the frame of EMO. This *kernel* method consists in estimating the distribution density of a solution using a so-called *sharing function* that is related to the sum of distances to its neighborhood solutions. A sharing distance parameter specifies the similarity threshold, *i.e.* the size of *niches*. The distance measure between two solutions can be defined in the decision space, in the objective space or can combine both. A distance metric partly or fully defined in the parameter space strongly depends of the tackled problem. But, standard normalized distance metrics defined in the objective space are already provided within ParadisEO-MOEO for real-coded objective vectors. Sharing is one of the most popular technique and is commonly used in a large number of EMO algorithms such as MOGA [18], NPGA [26], NSGA [41] and more. Note that a 'front by front' procedure is also proposed as, in some cases [18, 41], sharing only occurs between solutions of same rank.

Nearest Neighbor Diversity Assignment Scheme

The nearest neighbor diversity maintaining strategy computes the distance between a given objective vector and its nearest neighbors in order to estimate the density of its neighborhood. The density estimator is generally based on the volume of the hyper-rectangle defined by these nearest neighbors. For instance, the SPEA2 [50] diversity preservation mechanism is based on this technique.

Crowding Diversity Assignment Scheme

Another diversity assignment scheme is the concept of *crowding*, firstly suggested by Holland [25] and used by De Jong [28] to prevent *genetic drift* [28]. It is employed by Deb et al. [14] in the NSGA-II. Contrary to the sharing diversity preservation scheme, this one allows to maintain diversity without specifying any parameter. It consists in estimating the density of solutions surrounding a particular

point of the objective space. As before, a similar mechanism working on sub-classes of solutions is also provided within ParadisEO-MOEO.

Elitism

As shown in Fig. 5.9, an archive is represented by the `moeoArchive` abstract class and is a population using a particular dominance relation to update its contents. An abstract class for fixed-size archives is given: `moeoFixedArchive`. But implementations of an unbounded archive (`moeoUnboundedArchive`), a general-purpose bounded archive based on a fitness and/or a diversity assignment scheme(s) (`moeoBoundedArchive`) as well as the SPEA2 archive (`moeoSPEA2Archive`) are provided. Generally speaking, the dominance relation used to update the archive contents is the Pareto-dominance relation, which is employed by default. But, other dominance criteria are found in the literature. Therefore, the framework offers the opportunity to use any dominance relation for that purpose by means of a `moeoObjectiveVectorComparator` object. As shown in Fig. 5.10, implemented criteria consist of Pareto-dominance, weak-dominance, strict-dominance, ε-dominance [24], and g-dominance [37]. Usually, an archive is used as an external

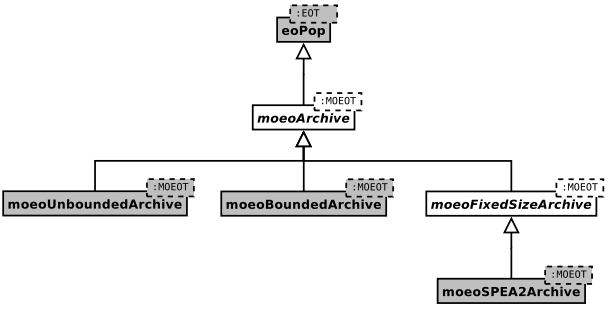

Fig. 5.9. UML diagram for archiving

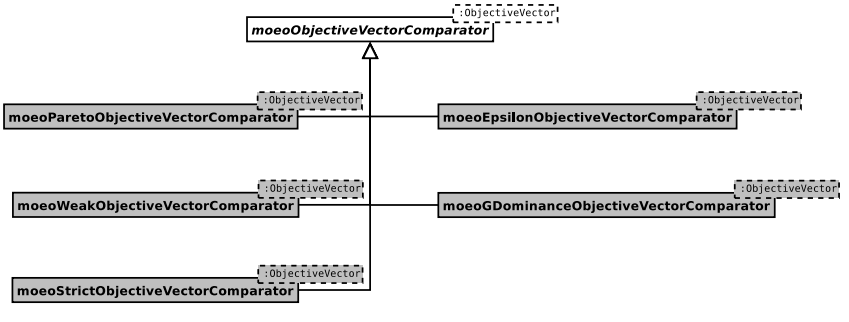

Fig. 5.10. UML diagram for for dominance relation (used for pairwise objective vector comparison)

storage only. However, we will see in the next section that archive members can also be used during the selection phase of an EMO algorithm.

5.4.3 EA-Related Components

EA-related components are presented below: variation operators as well as selection and replacement mechanisms.

Variation

The purpose of variation operators is to modify the representation of solution in order to move them in the search space. Generally speaking, while dealing with EAs, these problem-dependent operators are stochastic. They can be classified according to the number of arguments they use or modify, *i.e.*:

- Variation operators involving two individuals are called *recombination operators*. They can either modify one parent according to the material of the other one, or modify both parents. At the implementation level, the former are called *binary operators* and the latter *quadratic operators*.
- Variation operators involving a single individual are *mutation operators*.

Note that straightforward extensions allow to combine these simple operators. For instance, in a standard proportional combination, a given operator is selected from among a set of operators based on some user-defined rates. Furthermore, other variation operators generating any number of offspring from any number of parents, called *general operators*, can also be defined.

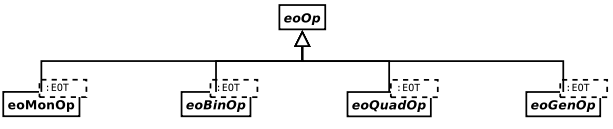

Fig. 5.11. UML diagram for variation

As shown in Fig. 5.11, in terms of implementation, all variation operators must derive from the `eoOp` base class. Four abstract classes inherit of `eoOp`, namely `eoMonOp` for mutation operators, `eoBinOp` and `eoQuadOp` for recombination operators and `eoGenOp` for other kinds of variation operators. Various operators of the same arity can be combined using some helper classes. Note that variation mechanisms for some classical (real-coded, vector-based or permutation-based) representations are already provided in the framework. Moreover, a hybrid mechanism can easily be designed by using a single-objective local mechanism search as a mutation operator, as they both inherit from `eoMonOp`, see ParadisEO-MO [8]. All variation operators designed for a given problem must be embedded into a `eoTranform` object.

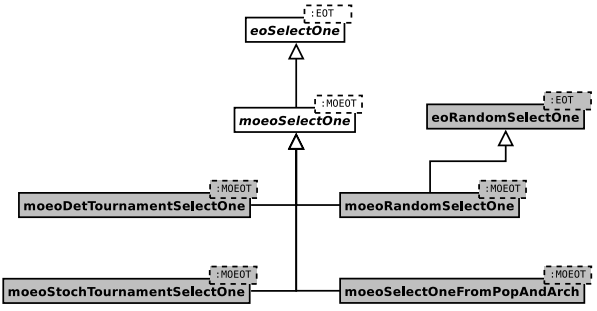

Fig. 5.12. UML diagram for selection

Selection Schemes

The selection step is one of the main search operators of EAs. It consists in choosing some solutions that will be used to generate the offspring population. In general, the better is an individual, the higher is its chance of being selected, so that fitness and/or the diversity value(s) are normally used. There exists a large number of selection strategies in the frame of EMO. Four ones are provided within ParadisEO-MOEO (see Fig. 5.12):

- A *random selection* (`moeoRandomSelectOne`), that consists in selecting a parent randomly among the population members, without taking fitness nor diversity information into account.
- A *deterministic tournament selection* (`moeoDetTournamentSelectOne`), that consists in performing a tournament between m randomly chosen population members and in selecting the best one.
- A *stochastic tournament selection* (`moeoStochTournamentSelectOne`), that consists in performing a binary tournament between randomly chosen population members and in selecting the best one with a probability p or the worst one with a probability $(1-p)$.
- An *elitist selection* (`moeoSelectOneFromPopAndArch`), that consists in selecting a population member based on some selection scheme with a probability p, or in selecting an archive member using another selection scheme with a probability $(1-p)$. So, elite solutions also contribute to the evolution engine by being used as parents. This scheme has been integrated in various elitist EMO algorithms including SPEA [51], SPEA2 [50] or PESA [12].

All these selection methods are of the `moeoSelectOne` type and need to be embedded into an `eoSelect` object to be properly used. Of course, everything is done to easily implement a new selection scheme with a minimum programming effort.

Replacement Schemes

Selection pressure is also affected at the replacement step where survivors are selected from both the current and the offspring population. A large majority

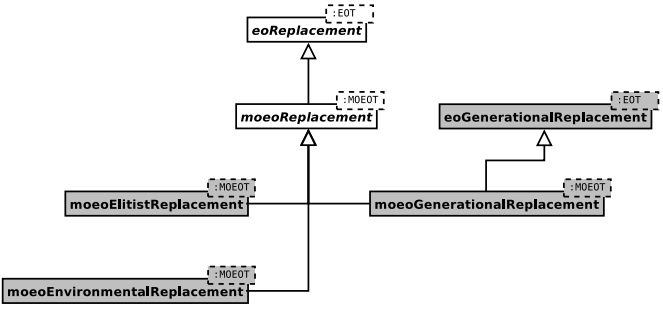

Fig. 5.13. UML diagram for replacement

of replacement strategies depend on the fitness and/or the diversity value(s) and are, somehow, EMO-specific. Three replacement schemes are provided within ParadisEO-MOEO (see Fig. 5.13), but this list is not exhaustive as new ones can easily be implemented due to the genericity of the framework.

- A *generational replacement* (`moeoGenerationalReplacement`), that consists in keeping the offspring population only, while all parents are deleted.
- An *elitist replacement* (`moeoElitistReplacement`), that consists in choosing the N best solutions (where N stands for the population size).
- An *environmental replacement* (`moeoEnvironmentalReplacement`), that consists in deleting one-by-one the worst individuals, and in updating the fitness and the diversity values of the remaining solutions each time there is a deletion. The process ends once the required population size is reached.

5.4.4 Evolutionary Multi-Objective Optimization Algorithms

Now that all the basic, EMO-specific and EA-related components are defined, an EMO algorithm can easily be designed using the fine-grained classes of ParadisEO. As the implementation is conceptually divided into components, different operators can be experimented without engendering significant modifications in terms of code writing. As seen before, a wide range of components are already provided. But, keep in mind that this list is not exhaustive as the framework perpetually evolves and offers all that is necessary to develop new ones with a minimum effort. Indeed, ParadisEO is a white-box framework that tends to be flexible while being as user-friendly as possible. Fig. 5.14 illustrates the use of the `moeoEasyEA` class that allows to define an EMO algorithm in a common fashion, by specifying all the particular components required for its implementation. All classes use a template parameter *MOEOT (Multi-Objective Evolving Object Type)* that defines the representation of a solution for the problem under consideration. This representation might be implemented by inheriting of the `MOEO` class as described in Sect. 5.4.1. Note that archive-related components do not appear in the UML diagram, as we chose to let the use of an archive as optional. The archive update can easily be

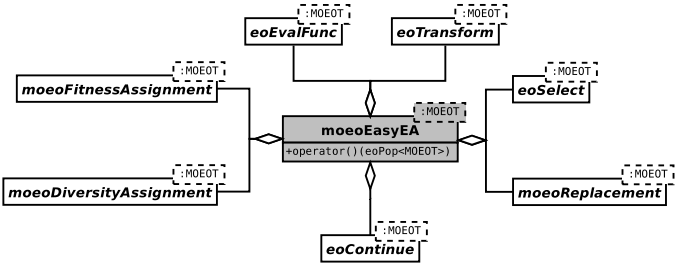

Fig. 5.14. UML diagram for the design of an EMO algorithm

integrated into the EA by means of the checkpointing process. Similarly, the initialization process does not appear either, since an instance of `moeoEasyEA` starts with an already initialized population.

Easy-to-use EMO Algorithms

In order to satisfy both the common user and the more experimented one, ParadisEO-MOEO also provides even more easy-to-use EMO algorithms (see Fig. 5.15). These classes propose different implementations of some state-of-the-art algorithms by using the fine-grained components of ParadisEO. Hence, MOGA [18], NSGA [41], NSGA-II [14], SPEA2 [50], IBEA [48] and SEEA [34] are proposed in a way that a minimum number of problem- or algorithm-specific parameters are required. These easy-to-use algorithms also tend to be used as references for a fair performance comparison in the academic world, even if they are also well-suited for a straight use to solve real-world MOPs. In a close future, other easy-to-use multi-objective metaheuristics will be proposed while new fined-grained components will be implemented into the frame of ParadisEO-MOEO.

5.4.5 Discussion

ParadisEO-MOEO has been used and experimented to solve a large range of MOPs from both academic and real-world fields, which evidences its high flexibility. Indeed, various academic MOPs have been tackled within ParadisEO-MOEO, including continuous test functions (like the ZDT and DTLZ functions family defined in [16]), scheduling problems (permutation flow-shop scheduling problem [32]), routing problems (multi-objective traveling salesman problem, bi-objective ring star problem [34]), etc. Moreover, it has been successfully employed to solve real-world applications in structural biology [9], feature selection in cancer classification [44], data-mining [29], materials design in chemistry [40], etc. Besides, a detailed documentation as well as some tutorial lessons and problem-specific implementations are freely available on the ParadisEO website[6]. And we expect the number of MOP

[6] `http://paradiseo.gforge.inria.fr`

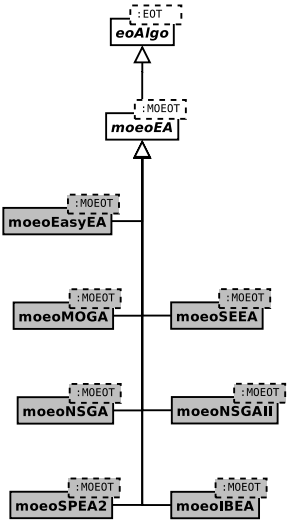

Fig. 5.15. UML diagram for easy-to-use EMO algorithms

contributions to largely grow in a near future. Furthermore, note that the implementation of EMO algorithms is just an aspect of the features provided by ParadisEO. Indeed, the whole framework allows to conveniently design hybrid as well as parallel and distributed metaheuristics, including EMO methods. Hence, hybrid mechanisms can be exploited in a natural way to make cooperating metaheuristics belong to the same or to different classes. Moreover, the three main parallel models are concerned: algorithmic-level, iteration-level and solution-level and are portable on different types of architecture. For instance, in the frame of ParadisEO, hybrid EMO algorithms have been experimented in [34], a multi-objective cooperative island model has been designed in [43], and costly evaluation functions have been parallelized in [9]. The reader is referred to [10] for more information about ParadisEO's hybrid and parallel models.

5.5 Case Study: An EMO Algorithm for a Bi-objective Scheduling Problem

The Flow-shop Scheduling Problem (FSP) is one of the most well-known scheduling problems and has been widely studied in the literature. The majority of works dedicated to the FSP considers it on a single-objective form and mainly aim at minimizing the makespan (*i.e.* the total completion time). However, many objective functions, varying according to the particularities of the tackled problem, may be considered and some multi-objective approaches have also been proposed. For a survey, see for instance [31, 46].

5.5.1 Problem Definition

Solving the FSP consists in scheduling a set of N jobs $J_1, J_2, \ldots, J_N$ on M machines $M_1, M_2, \ldots, M_M$. Machines are critical resources, *i.e.* one machine cannot process more than one job at a time. Each job J_i is composed of M consecutive tasks $t_{i1}, t_{i2}, \ldots, t_{iM}$, where t_{ij} represents the j^{th} task of the job J_i, requiring the machine M_j. A processing time p_{ij} is associated to each task t_{ij}; and a due date d_i is given to each job J_i (the deadline of the job). In this study, we focus on the permutation FSP, where the operating sequences of the jobs are identical and unidirectional on every machine, as illustrated in Fig. 5.16. Many objective functions may be

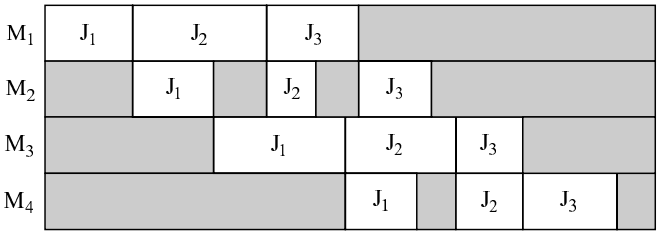

Fig. 5.16. An example of solution for a permutation flow-shop problem where 3 jobs (J_1, J_2, J_3) have to be scheduled on 4 machines (M_1, M_2, M_3, M_4).

tackled while scheduling tasks on several machines. The FSP that we consider here aims at minimizing the makespan (C_{max}) and the total tardiness ($\overline{T}$). These objectives are among the most widely investigated in the literature. For each task t_{ij} being scheduled at the time s_{ij}, they are computed as follows:

$$C_{max} = \max_{i \in \{1,\ldots,N\}} \{s_{iM} + p_{iM}\} \tag{5.1}$$

$$\overline{T} = \sum_{i=1}^{N} \Big\{ \max\{0, s_{iM} + p_{iM} - d_i\} \Big\} \tag{5.2}$$

According to the Graham et al. notation [23], the problem under consideration can be denoted by $F/perm, d_i/(C_{max}, \overline{T})$.

5.5.2 Implementation

In this section, we focus on the implementation of an EMO algorithm to approximate the efficient set for the FSP presented above. First, the design and the implementation of problem-dependent components are discussed. Then, the choice of problem-independent components is presented. And finally, the implementation of the EMO algorithm is given. Note that this case study is closely related to a ParadisEO-MOEO tutorial available on the website, so that detailed source code can easily be retrieved.

Problem-Dependent Components Design

Below are presented the problem-dependent components designed for the problem under consideration: solution representation, evaluation, initialization and variation operators.

Representation

First of all, let us define the number of objectives for the problem under consideration, and if they are to be minimized or maximized. This is done by specializing the `moeoObjective VectorTraits` class. In our case, let denote the specialized class by `fspObjVecTraits`. Then, the representation in the objective space can be defined as a real-coded objective vector:

```
typedef moeoRealObjectiveVector<fspObjVecTraits> fspObjVec;
```

Now, for the representation in the decision space, we use a permutation-based encoding. So, let us define our solution type `FSP` by a vector of integer:

```
typedef moeoIntVector<fspObjVec> FSP;
```

Evaluation

The evaluation class has to evaluate the values of a given solution for every objective, *i.e.* the makespan and the total tardiness for the problem under consideration. We here define a `fspEval` class inheriting of `eoEvalFunc`:

```
class fspEval : public eoEvalFunc<FSP>
{
public:
    void operator() (FSP & _fsp)
    {
        fspObjVec objVec;
        objVec[0] = makespan(_fsp);
        objVec[1] = tardiness(_fsp);
        _fsp.objectiveVector(objVec);
    }
    // ...
}
```

Initialization

ParadisEO already provides an implementation for initializing permutations. Therefore, let us define a `fspInit` class of the `eoInitPermutation` type:

```
typedef eoInitPermutation<FSP> fspInit;
```

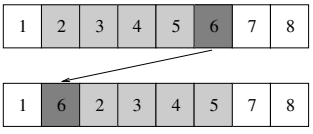

Fig. 5.17. Shift mutation.

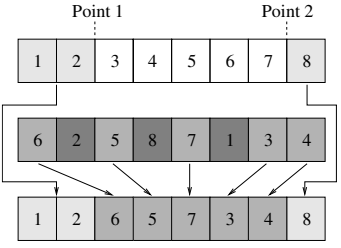

Fig. 5.18. Two-point crossover

Variation

Regarding variation operators, we choose to use a shift mutation and to implement a two-point crossover as described in [27]. These operators are respectivelly illustrated in Fig. 5.17 and Fig. 5.18.

```
typedef eoShiftMutation<FSP> fspMutation;


class fspCrossover : public eoQuadOP<FSP>
{
public:
    bool operator() (FSP & _fsp1, FSP & _fsp2)
    {
        // ...
    }
}
```

Problem-Independent Components Decision

We arbitrary choose to use the following problem-independent components in the frame of the EMO algorithm designed in this section. Of course, different operators can easily be experimented with a very low coding effort.

- *Fitness assignment*: dominance-rank.
- *Diversity assignment*: sharing.
- *Elitism*: unbounded archive.
- *Selection*: deterministic tournament.
- *Replacement*: elitist.
- *Stopping criteria*: maximum number of generations.

EMO Algorithm Design

The source code of the main program file is given below. First, some parameters are given. Then, problem-dependent and problem-independent components are instantiated. Finally, the algorithm is built and is applied to the initialized population.

```
/* parameters */
// population size
int _popSize = 100;
// crossover probability
double _pCross = 0.25;
// mutation probability
double _pMut = 1.0;
// tournament size for selection
int _tourSize = 2;
// maximum number of generations
int _maxGen = 1000;

/* representation-dependent components */
// evaluation
fspEval eval;
// initialization
fspInit init;
// variation operators
fspCrossover cross;
fspMutation mut;
eoSGATransform<FSP> op(cross,_pCross,mut,_pMut);

/* representation-independent components */
// initial population
eoPop<FSP> pop(_popSize,init);
// unbounded archive
moeoUnboundedArchive<FSP> arch;
// fitness assignment
moeoDominanceRankFitnessAssignment<FSP> fitness;
// diversity assignment
moeoSharingDiversityAssignment<FSP> diversity;
// selection
moeoDetTournamentSelect<FSP> select(_tourSize);
// replacement
moeoElitistReplacement<FSP> replace(fitness,diversity);
// stopping criteria
eoGenContinue<FSP> stop(_maxGen);
// checkpoint
eoCheckPoint<FSP> check(stop);
// archive updater
moeoArchiveUpdater<FSP> updater(arch,pop);
check.add(updater);
// algorithm
moeoEasyEA<FSP> algo
      (check,eval,select,op,replace,fitness,diversity);

/* apply the algorithm to the population */
algo(pop);
```

5.6 Conclusion

In this chapter, we first presented a unified view of evolutionary algorithms for solving multi-objective problems of both continuous and combinatorial optimization. The resulting flexible model, based on the fundamental issues of fitness, diversity and elitism, has been used as a starting point for the implementation of a general purpose software framework called ParadisEO-MOEO. Base-class components follow the fine-grained decomposition of the model and allow to design many resolution methods in a modular way, by combining different strategies at each stage of its conception, with a minimum programming effort. Many classical strategies for problem-independent components are already provided. ¿From this set of mechanisms, state-of-the-art algorithms such as NSGA-II, SPEA2 and IBEA have been implemented and are available. Nevertheless, new components and algorithms will be integrated in a near future, as we hope the framework to constantly evolve in order to follow the most recent advances of the literature. Furthermore, a clear conceptual separation of the problem-specific part and problem-independent part of the metaheuristic is provided, so that the representation, the initialization and the evaluation of a solution as well as variation operators are the only components that must be specifically implemented for the problem to be solved. However, the framework also proposes standard techniques for the most common representation encodings, in which case the user only has to implement the objective functions associated to his/her problem. In addition, the platform also includes the most well-known parallel and distributed models for metaheuristics and their hybridization.

A large part of components involved in evolutionary multi-objective optimization are shared by many other search methods. Hence, we plan to generalize the unified view presented in this chapter to additional population-based multi-objective metaheuristics, including local search, scatter search and particle swarm optimization approaches. Afterward, the resulting general purpose model will be implemented in a modular way in order to be integrated into the ParadisEO-MOEO software framework. As well, there is a growing need in the MCDM community to provide a powerful tool devoted to interactive multi-objective optimization. And we believe that such a strong and reliable framework like ParadisEO-MOEO is the ideal platform to provide base-class components to an higher level software where metaheuristics would be involved. Another interesting extension would be to add components to deal with stochastic and dynamic multi-objective optimization problems.

Acknowledgement. This work was supported by the ANR DOCK project. The authors would like to gratefully acknowledge Sébastien Cahon and Nouredine Melab for their work on the preliminary version of ParadisEO-MOEO, as well as Abdel-Hakim Deneche for his precious contribution on the implementation of some components presented in this chapter.

References

1. http://cs.gmu.edu/~eclab/projects/ecj/
2. http://home.gna.org/momh/
3. http://shark-project.sourceforge.net/

4. OMG unified modeling language specification. Object Management Group (2000)
5. Basseur, M., Seynhaeve, F., Talbi, E.G.: Design of multi-objective evolutionary algorithms: Application to the flow-shop scheduling problem. In: Congress on Evolutionary Computation (CEC 2002), Honolulu, Hawai, USA, vol. 2, pp. 1151–1156 (2002)
6. Beume, N., Naujoks, B., Emmerich, M.: SMS-EMOA: Multiobjective selection based on dominated hypervolume. European Journal of Operational Research 181(3), 1653–1669 (2007)
7. Bleuler, S., Laumanns, M., Thiele, L., Zitzler, E.: PISA — a platform and programming language independent interface for search algorithms. In: Fonseca, C.M., Fleming, P.J., Zitzler, E., Deb, K., Thiele, L. (eds.) EMO 2003. LNCS, vol. 2632, pp. 494–508. Springer, Heidelberg (2003)
8. Boisson, J.C., Jourdan, L., Talbi, E.G.: ParadisEO-MO. Tech. rep. (2008)
9. Boisson, J.C., Jourdan, L., Talbi, E.G., Horvath, D.: Parallel multi-objective algorithms for the molecular docking problem. In: IEEE Symposium on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB 2008), Sun Valley Resort, Idaho, USA (2008)
10. Cahon, S., Melab, N., Talbi, E.G.: ParadisEO: A framework for the reusable design of parallel and distributed metaheuristics. Journal of Heuristics 10(3), 357–380 (2004)
11. Coello Coello, C.A., Lamont, G.B., Van Veldhuizen, D.A.: Evolutionary Algorithms for Solving Multi-Objective Problems, 2nd edn. Springer, New York (2007)
12. Corne, D., Knowles, J.D., Oates, M.J.: The pareto envelope-based selection algorithm for multi-objective optimisation. In: Deb, K., Rudolph, G., Lutton, E., Merelo, J.J., Schoenauer, M., Schwefel, H.-P., Yao, X. (eds.) PPSN 2000. LNCS, vol. 1917, pp. 839–848. Springer, Heidelberg (2000)
13. Deb, K.: Multi-Objective Optimization using Evolutionary Algorithms. John Wiley & Sons, Chichester (2001)
14. Deb, K., Agrawal, S., Pratap, A., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Transactions on Evolutionary Computation 6(2), 182–197 (2002)
15. Deb, K., Mohan, M., Mishra, S.: Evaluating the ϵ-domination based multi-objective evolutionary algorithm for a quick computation of pareto-optimal solutions. Evolutionary Computation 13(4), 501–525 (2005)
16. Deb, K., Thiele, L., Laumanns, M., Zitzler, E.: Scalable test problems for evolutionary multi-objective optimization. In: Abraham, A., Jain, R., Goldberg, R. (eds.) Evolutionary Multiobjective Optimization: Theoretical Advances and Applications, ch. 6, pp. 105–145. Springer, Heidelberg (2005)
17. Durillo, J.J., Nebro, A.J., Luna, F., Dorrosoro, B., Alba, E.: jMetal: A java framework for developing multi-objective optimization metaheuristics. Tech. Rep. ITI-2006-10, University of Málaga (2006)
18. Fonseca, C.M., Fleming, P.J.: Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization. In: Forrest, S. (ed.) Proceedings of the 5th International Conference on Genetic Algorithms (ICGA 1993), pp. 416–423. Morgan Kaufmann, Urbana-Champaign (1993)
19. Fourman, M.P.: Compaction of symbolic layout using genetic algorithms. In: Grefensette, J.J. (ed.) Proceedings of the 1st International Conference on Genetic Algorithms (ICGA 1985), pp. 141–153. Lawrence Erlbaum Associates, Pittsburgh (1985)
20. Gagné, C., Parizeau, M.: Genericity in evolutionary computation software tools: Principles and case study. International Journal on Artificial Intelligence Tools 15(2), 173–194 (2006)

21. Goldberg, D.E.: Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley, Boston (1989)
22. Goldberg, D.E., Richardson, J.: Genetic algorithms with sharing for multimodal function optimization. In: Second International Conference on Genetic Algorithms and their application, pp. 41–49. Lawrence Erlbaum Associates, Inc., Mahwah (1987)
23. Graham, R.L., Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G.: Optimization and approximation in deterministic sequencing and scheduling: A survey. Annals of Discrete Mathematics 5, 287–326 (1979)
24. Helbig, S., Pateva, D.: On several concepts for ε-efficiency. OR Spektrum 16(3), 179–186 (1994)
25. Holland, J.H.: Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Abor (1975)
26. Horn, J., Nafpliotis, N., Goldberg, D.E.: A niched pareto genetic algorithm for multiobjective optimization. In: IEEE Congress on Evolutionary Computation (CEC 1994), pp. 82–87. IEEE Press, Piscataway (1994)
27. Ishibuchi, H., Murata, T.: A multi-objective genetic local search algorithm and its application to flowshop scheduling. IEEE Transactions on Systems, Man and Cybernetics 28, 392–403 (1998)
28. Jong, K.A.D.: An analysis of the behavior of a class of genetic adaptive systems. Ph.D thesis, Ann Arbor, University of Michigan (1975)
29. Jourdan, L., Khabzaoui, M., Dhaenens, C., Talbi, E.G.: A hybrid evolutionary algorithm for knowledge discovery in microarray experiments. In: Olariu, S., Zomaya, A.Y. (eds.) Handbook of Bioinspired Algorithms and Applications, ch. 28, pp. 489–505. CRC Press, Boca Raton (2005)
30. Keijzer, M., Merelo, J.J., Romero, G., Schoenauer, M.: Evolving objects: A general purpose evolutionary computation library. In: Collet, P., Fonlupt, C., Hao, J.-K., Lutton, E., Schoenauer, M. (eds.) EA 2001. LNCS, vol. 2310, pp. 231–244. Springer, Heidelberg (2002)
31. Landa Silva, J.D., Burke, E., Petrovic, S.: An introduction to multiobjective metaheuristics for scheduling and timetabling. In: Gandibleux, X., Sevaux, M., Sörensen, K., T'kindt, V. (eds.) Metaheuristics for Multiobjective Optimisation. LNEMS, vol. 535, pp. 91–129. Springer, Berlin (2004)
32. Liefooghe, A., Basseur, M., Jourdan, L., Talbi, E.G.: Combinatorial optimization of stochastic multi-objective problems: an application to the flow-shop scheduling problem. In: Obayashi, S., Deb, K., Poloni, C., Hiroyasu, T., Murata, T. (eds.) EMO 2007. LNCS, vol. 4403, pp. 457–471. Springer, Heidelberg (2007)
33. Liefooghe, A., Basseur, M., Jourdan, L., Talbi, E.G.: ParadisEO-MOEO: A framework for evolutionary multi-objective optimization. In: Obayashi, S., Deb, K., Poloni, C., Hiroyasu, T., Murata, T. (eds.) EMO 2007. LNCS, vol. 4403, pp. 386–400. Springer, Heidelberg (2007)
34. Liefooghe, A., Jourdan, L., Talbi, E.G.: Metaheuristics and their hybridization to solve the bi-objective ring star problem: a comparative study. Tech. Rep. RR-6515, Institut National de Recherche en Informatique et Automatique, INRIA (2008)
35. Meunier, H., Talbi, E.G., Reininger, P.: A multiobjective genetic algorithm for radio network optimization. In: IEEE Congress on Evolutionary Computation (CEC 2000), pp. 317–324. IEEE Press, San Diego (2000)
36. Miettinen, K.: Nonlinear Multiobjective Optimization. International Series in Operations Research and Management Science, vol. 12. Kluwer Academic Publishers, Boston (1999)

37. Molina, J., Santana, L.V., Hernández-Díaz, A.G., Coello Coello, C.A., Caballero, R.: g-dominance: Reference point based dominance for multiobjective metaheuristics. European Journal of Operational Research 197(2), 685–692 (2009)
38. Poles, S., Vassileva, M., Sasaki, D.: Multiobjective optimization software. In: Branke, J., Deb, K., Miettinen, K., Słowiński, R. (eds.) Multiobjective Optimization. LNCS, vol. 5252, pp. 329–348. Springer, Heidelberg (2008)
39. Schaffer, J.D.: Multiple objective optimization with vector evaluated genetic algorithms. In: Grefensette, J.J. (ed.) Proceedings of the 1st International Conference on Genetic Algorithms (ICGA 1985), pp. 93–100. Lawrence Erlbaum Associates, Pittsburgh (1985)
40. Schuetze, O., Jourdan, L., Legrand, T., Talbi, E.G., Wojkiewicz, J.L.: New analysis of the optimization of electromagnetic shielding properties using conducting polymers and a multi-objective approach. Polymers for Advanced Technologies 19(7), 762–769 (2008)
41. Srinivas, N., Deb, K.: Multiobjective optimization using nondominated sorting in genetic algorithms. Evolutionary Computation 2(3), 221–248 (1994)
42. Streichert, F., Ulmer, H.: JavaEvA: a java based framework for evolutionary algorithms. Tech. Rep. WSI-2005-06, Centre for Bioinformatics Tübingen (ZBIT) of the Eberhard-Karls-University, Tübingen (2005)
43. Talbi, E.G., Cahon, S., Melab, N.: Designing cellular networks using a parallel hybrid metaheuristic on the computational grid. Computer Communications 30(4), 698–713 (2007)
44. Talbi, E.G., Jourdan, L., Garcia-Nieto, J., Alba, E.: Comparison of population based metaheuristics for feature selection: Application to microarray data classification. In: IEEE/ACS International Conference on Computer Systems and Applications (AICCSA 2008), pp. 45–52. IEEE, Los Alamitos (2008)
45. Tan, K.C., Lee, T.H., Khoo, D., Khor, E.F.: A multi-objective evolutionary algorithm toolbox for computer-aided multi-objective optimization. IEEE Transactions on Systems, Man and Cybernetics: Part B (Cybernetics) 31(4), 537–556 (2001)
46. T'Kindt, V., Billaut, J.C.: Multicriteria Scheduling: Theory, Models and Algorithms. Springer, Berlin (2002)
47. Wierzbicki, A.: The use of reference objectives in multiobjective optimization. In: Fandel, G., Gal, T. (eds.) Multiple Objective Decision Making, Theory and Application. LNEMS, vol. 177, pp. 468–486. Springer, Heidelberg (1980)
48. Zitzler, E., Künzli, S.: Indicator-based selection in multiobjective search. In: Yao, X., Burke, E.K., Lozano, J.A., Smith, J., Merelo-Guervós, J.J., Bullinaria, J.A., Rowe, J.E., Tiňo, P., Kabán, A., Schwefel, H.-P. (eds.) PPSN 2004. LNCS, vol. 3242, pp. 832–842. Springer, Heidelberg (2004)
49. Zitzler, E., Laumanns, M., Bleuler, S.: A tutorial on evolutionary multiobjective optimization. In: Gandibleux, X., Sevaux, M., Swrensen, K. (eds.) Metaheuristics for Multiobjective Optimisation. LNEMS, vol. 535, pp. 3–38. Springer, Heidelberg (2004)
50. Zitzler, E., Laumanns, M., Thiele, L.: SPEA2: Improving the strength pareto evolutionary algorithm. Tech. Rep. 103, Computer Engineering and Networks Lab (TIK), Swiss Federal Institute of Technology (ETH), Zurich, Switzerland (2001)
51. Zitzler, E., Thiele, L.: Multiobjective evolutionary algorithms: A comparative case study and the strength pareto approach. IEEE Transactions on Evolutionary Computation 3(4), 257–271 (1999)
52. Zitzler, E., Thiele, L., Laumanns, M., Foneseca, C.M., Grunert da Fonseca, V.: Performance assessment of multiobjective optimizers: An analysis and review. IEEE Transactions on Evolutionary Computation 7(2), 117–132 (2003)