Chapter 1
DESDEO: An Open Framework for Interactive Multiobjective Optimization

Vesa Ojalehto and Kaisa Miettinen

Abstract We introduce a framework for interactive multiobjective optimization methods called DESDEO released under an open source license. With the framework, we want to make interactive methods easily accessible to be applied in solving real-world problems. The framework follows an object-oriented software design paradigm, where functionalities have been divided to modular, self-contained components. The framework contains implementations of some interactive methods, but also components which can be utilized to implement more interactive methods and, thus, increase the applicability of the framework. To demonstrate how the framework can be used, we consider an example problem where the pollution of a river is controlled. To solve this problem with four objectives, we apply two interactive methods called NAUTILUS and NIMBUS and show how the method can be switched during the solution process.

1.1 Introduction

We describe an open source framework DESDEO devoted to interactive methods for solving multiobjective optimization problems. The main aim of the framework is to make interactive methods closer to researchers and practitioners by making their implementations readily available to be applied in solving optimization problems involving multiple (even nonlinear) conflicting objectives. The problem formulations may use different simulation and modelling tools. The framework is not only a source of implementations to be applied but one can also add new implementations

Vesa Ojalehto
University of Jyvaskyla, Faculty of Information Technology, P.O. Box 35 (Agora), FI-40014 University of Jyvaskyla, Finland, e-mail: vesa.ojalehto@jyu.fi

Kaisa Miettinen
University of Jyvaskyla, Faculty of Information Technology, P.O. Box 35 (Agora), FI-40014 University of Jyvaskyla, Finland, e-mail: kaisa.miettinen@jyu.fi

there. For this, the framework includes reusable components that can be utilized. The framework is open and released under a permissive open source license. The source code is in Python and more information about the framework is available at `https://desdeo.it.jyu.fi`.

Real-world problems typically contain several, conflicting objectives that should be simultaneously optimized. When we consider functions of variables, we call such problems multiobjective optimization problems. Because of the conflicting nature of the objectives, these problems usually have several solutions with different trade-offs among the objectives. These so-called Pareto optimal solutions cannot be compared and ordered without some additional information. Eventually, a single solution (or few of them) is (are) to be found to be realized. Additional information can be obtained in the form of preference information from a human decision maker (DM). Our aim is to support the DM in finding a solution that best corresponds to his/her preferences in the presence of different trade-offs between the conflicting objectives.

In the literature, there exists a wide array of methods based on different approaches for considering multiobjective optimization problems and taking the DM's preferences into account (see, e.g., [5, 16, 27] and references therein). Multiobjective optimization methods can be classified according to the role of the DM in the solution process [10, 16]. If there is no DM available, some no-preference method is to be used to find some neutral compromise among the objectives. In a priori methods, the DM is asked to first express hopes and desires and then a Pareto optimal solution best corresponding to them is found. Alternatively, a representative set of Pareto optimal solutions can first be generated and then the DM is expected to select the most preferred of them. Such methods are called a posteriori methods. In a priori methods, the DM may have too optimistic or pessimistic expectations and, thus, may not be satisfied with the solution found. On the other hand, it may be computationally expensive to generate a good representation of different Pareto optimal solutions and cognitively demanding for the DM to compare many solutions. Interactive methods aim at avoiding the above-mentioned shortcomings.

We concentrate on interactive methods, where the DM is asked to provide preference information in an iterative manner. As said, one of the concerns when solving multiobjective optimization problems is the cognitive load set on a DM in forming an understanding of the characteristics of the objective functions considered, especially, when dealing with a higher number of them. With interactive methods, the solution process is based on consecutive steps and a limited amount of information is exchanged at a time (i.e. per iteration). In each step, the DM specifies preference information, based on which new solutions are generated and shown to the DM. In this way, the DM can concentrate on a small set of solutions at a time and only solutions reflecting the DM's interests need to be generated. What is important, the DM can learn about the inter-dependencies among the objectives and the shape of the set of Pareto optimal solutions besides learning about the feasibility of one's preferences. In all, the DM can modify his/her preferences and gradually gain confidence on the suitability of the Pareto optimal solutions and iterate until the most preferred solution is found and selected as the final one at the end of the solution process.

Interactive methods have been utilized for solving multiobjective optimization problems in a wide variety of application areas, such as reservoir management [1], wastewater treatment management [9], optimal control in steel casting [18, 24], chemical engineering [20], construction of bridges [28] and analysing air pollution [38], etc. Even though there exist many different interactive methods, their implementations are scarce or not generally available, and typically application specific. To facilitate implementation related issues, it has been suggested to separate methodological aspects from technical ones [12] and, furthermore, separate the application, i.e., the multiobjective optimization problem formulation from the method [31].

Except the field of evolutionary algorithms (see, e.g., [7]), where releasing source codes is a common practice, openly accessible frameworks are rare in the field of multiobjective optimization. Even though methods considering preference information from a decision maker have been implemented in the evolutionary field (see, e.g., [14] and references therein), to our knowledge, there does not exist any openly accessible framework suitable for developing interactive multiobjective optimization methods. Furthermore, even proprietary implementations of interactive multiobjective optimizations methods are rare. With DESDEO, we want to fill a gap by making interactive methods more widely and easily available.

The DESDEO framework concentrates on interactive methods with the main focus on the structures and components needed in implementing them. With the DESDEO framework, we want to bring interactive algorithms available for a wider audience as well as facilitate their development. The framework proposed here follows an object-oriented architecture design and has been divided into several modules designed to be independently usable to fulfil different functionalities needed by methods implemented. Among the design goals have been simplicity and ease of use, as well as flexibility and extensibility. The framework has been implemented with the Python programming language [35] and its source code is publicly available at `https://desdeo.it.jyu.fi/` by following the "source code" link on top of the page.

Besides implementations of interactive methods, the DESDEO framework includes connections to different tools for modelling optimization problems as well as to other types of optimization methods (typically, single objective optimization methods) employed by the interactive methods. Currently, the framework does not include any graphical user interface as it closely follows the structure suggested in [31], where the model of the problem, the algorithm and the user interface are separated. It is obvious that interactive methods do require user interfaces for preference elicitation and communicating information to the DM. To this end, we have a web application as a supporting software for the framework, which is used to demonstrate the interactive methods. This accompanying web application is available at the same web address. However, as said, we focus on the algorithmic aspects of the methods and do not go into details of user interface design. In addition to the web application, it is possible to use the framework with previously developed user interfaces, such as IND-NIMBUS [17, 31]. It should be noted that the methods implemented in the framework are aimed for solving nonlinear problems. Naturally, extending the framework to handle different types of problems is possible and desirable. At the moment, the DESDEO framework contains implementations of the synchronous NIMBUS method [23] and methods of the NAUTILUS family [26].

The rest of this chapter is structured as follows. In Section 1.2 we describe the basic concepts of interactive methods as well as briefly describe the methods discussed and applied in this chapter. Next, in Section 1.3, we introduce the actual DESDEO framework. As the framework is being continuously developed, we do not include a detailed description of the framework but concentrate on a general level description of the structure and components of the framework. To demonstrate how the framework can be used, in Section 1.4, we utilize the web application with the DESDEO framework to solve a multiobjective optimization problem, i.e., a use case. Finally, in Section 1.5 we draw conclusions and discuss some future developments for the DESDEO framework.

1.2 Background

In what follows, we briefly discuss the background material used in this chapter. First, we introduce the main concepts and notation and a general structure of interactive methods considered. We then continue with brief summaries of the methods applied to solve the use case in Section 1.4, that is, the interactive methods NIMBUS and NAUTILUS.

1.2.1 Some Basics of Interactive Multiobjective Optimization

We consider multiobjective optimization problems of the general form

$$\begin{array}{cl} \text{minimize (or maximize)} & \{f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_k(\mathbf{x})\} \\ \text{subject to} & \mathbf{x} \in S, \end{array} \tag{1.1}$$

where $f_i : S \rightarrow R$ are k (≥ 2) (conflicting) objective functions and $\mathbf{x} = (x_1, x_2, \ldots, x_n)^T$ is the *decision (variable)* vector bounded by constraints that form a feasible set $S \subset \mathbb{R}^n$. Objective vectors $\mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}),\ f_2(\mathbf{x}), \ldots, f_k(\mathbf{x}))^T$ consist of *objective (function) values* calculated at any feasible $\mathbf{x}$.

A multiobjective optimization problem with conflicting objectives has so-called Pareto optimal solutions with different trade-offs. A decision vector $\hat{\mathbf{x}}$ and the corresponding objective vector $\mathbf{f}(\hat{\mathbf{x}})$ are called Pareto optimal if there does not exist any other feasible $\mathbf{x}$ so that $f_j(\mathbf{x}) \leq f_i(\hat{\mathbf{x}})$ for all $i = 1, ..., k$ and $f_j(\mathbf{x}) < f_j(\hat{\mathbf{x}})$ for least one $j = 1, ..., k$. Pareto optimal solutions and the corresponding objective vectors form a Pareto optimal set to problem (1.1) in the decision space $\mathbb{R}^n$ and in the objective space $\mathbb{R}^k$, respectively.

Our aim is to find the most preferred Pareto optimal solutions using a *DM's preferences*, i.e., for example, information on desired changes to previously obtained solutions in order to find more preferred solutions for the problem. This means finding trade-offs between objectives which are acceptable for the DM or desirable values for the objective functions. For simplicity, in what follows, we assume all objective functions to be minimized, even though the framework can handle both objective functions to be maximized and minimized.

When using interactive methods, the ranges of objective function values in the Pareto optimal set can be shown to the DM to give an understanding of the attainable solutions. The *ideal objective vector* $\mathbf{z}^{\star}$ consists of the best possible objective function values whereas the worst objective function values over the Pareto optimal set form a *nadir objective vector* $\mathbf{z}^{\text{nad}}$. A *utopian objective vector* $\mathbf{z}^{\star\star}$ is commonly used instead of the ideal objective vector. The components of the ideal objective vector can be obtained by minimizing each of the objective functions individually subject to S. The utopian vector is then created by subtracting a small, positive epsilon from each of the components of the ideal objective vector. Thus, the components of the utopian objective vector are strictly better than those of the ideal objective vector.

The nadir objective vector is typically estimated by using a pay-off table (see, e.g., [3, 13, 16]), as accurate information would require knowing the whole Pareto optimal set. Further estimation ideas are given, e.g., in [2, 6, 40]). When forming a pay-off table, the decision vectors obtained when finding the components of the ideal objective vector are stored and all objective functions are evaluated at these points. Thus, in a pay-off table, components of the ideal objective vector lie on the diagonal of the table. The estimate for the nadir objective value of the ith objective can be found by finding the maximum value of the ith column.

As said, interactive methods consist of a series of steps called iterations, where in each step, new solutions reflecting the preference information obtained are generated. This is typically done by solving *subproblems* involving a single objective function, see, e.g., [16, 27], which is often called a scalarizing function. Subproblems contain elements of the original multiobjective optimization problem and preference information. It should be noted that a single objective optimization method which is suitable for the characteristics of the problem concerned is needed. By selecting the subproblems in an appropriate way, we get Pareto optimal solutions to the original problem. In general, many interactive methods follow a *core structure* [31], which can be described as follows:

1. Initialize the solution process by e.g., calculating ideal and nadir objective vectors as well as other method specific information.
2. Solve a method-specific subproblem to generate an initial solution or solutions to be used as a starting solution(s) and denote as current solution(s).
3. Ask the DM to specify preference information regarding the current solution(s). The type of the preference information depends on the method.
4. Generate new solution(s) by solving appropriate subproblem(s) involving the preference information.
5. Ask the DM to select the most preferred solution or a set of solutions from the set of previously generated solutions and denote it as the new current solution(s).

6. If the selected solution is satisfactory to the DM, stop. Otherwise continue from step 3.

Next, we briefly describe the NIMBUS and NAUTILUS methods used in Section 1.4 for demonstrating how the DESDEO framework can be used.

1.2.2 The Synchronous NIMBUS Method

The type of preference information used in the NIMBUS method [16, 21, 23] is the classification of the objective functions. This means that at each iteration, the DM considers the objective function values of a current Pareto optimal solution and is asked to classify each objective function into one of five different classes. These classes indicate what kind of changes in the objective function values would provide a more preferred solution. The classes are for functions f_i whose values

should be improved ($i \in I^{<}$),
should be improved to some aspiration level $\hat{z}_i < f_i(\mathbf{x^c})$ ($i \in I^{\leq}$),
are satisfactory at the moment ($i \in I^{=}$),
are allowed to impair up to some bound $\varepsilon_i > f_i(\mathbf{x^c})$ ($i \in I^{\geq}$),
are allowed to change freely ($i \in I^{\diamond}$).

A classification is feasible if at least one objective function should be improved and at least one is allowed to be impaired from the current values. In the synchronous NIMBUS method [23], up to four subproblems are formed based on the classification information provided by the DM. Each subproblem follows the preference information in a slightly different way, thus, providing up to four different Pareto optimal solutions [22]. The DM decides how many solutions (s)he wants to see and compare. As per the core structure, the solutions are shown to the DM who selects one of them or one of the previously generated Pareto optimal solutions as a current solution to be classified or as the most preferred solution. The DM can also ask for intermediate solutions to be generated between any two solutions generated so far.

Next, we describe two subproblems of the four ones used by the synchronous NIMBUS method. See [23] for the formulations of the other two subproblems. The so-called standard NIMBUS subproblem is of the form

$$
\begin{aligned}
\text{minimize} \quad & \max_{\substack{i \in I^{<} \\ j \in I^{\leq}}} \left[\frac{f_i(\mathbf{x}) - z_i^{\star}}{z_i^{\text{nad}} - z_i^{\star\star}}, \frac{f_j(\mathbf{x}) - \hat{z}_j}{z_j^{\text{nad}} - z_j^{\star\star}} \right] + \rho \sum_{i=1}^{k} \frac{f_i(\mathbf{x})}{z_i^{\text{nad}} - z_i^{\star\star}} \\
\text{subject to} \quad & f_i(\mathbf{x}) \leq f_i(\mathbf{x}^c) \ \text{ for all } \ i \in I^{<} \cup I^{\leq} \cup I^{=}, \\
& f_i(\mathbf{x}) \leq \varepsilon_i \ \text{ forall } \ i \in I^{\geq}, \\
& \mathbf{x} \in S,
\end{aligned}
\tag{1.2}
$$

where $\mathbf{x}^c$ represents the decision vector of the current solution. The formulation uses an *augmentation term* guaranteeing Pareto optimality of the obtained solutions (see, e.g., [16, 23]). The term $\rho > 0$ is a so-called augmentation coefficient with a small positive value. The aspiration levels and bounds $\hat{z}_i$ and ε_i, respectively, are obtained from the classification information.

The classification information and ranges of the objective functions in the Pareto optimal set can be easily used for generating a corresponding reference point $\bar{z}$. This is done by setting $\bar{z}_i = z_i^\star$ for $i \in I^<$, $\bar{z}_i = \hat{z}_i$ for $i \in I^\leq$, $\bar{z}_i = f_i(\mathbf{x}^c)$ for $i \in I^=$, $\bar{z}_i = \varepsilon_i$ for $i \in I^\geq$ and $\bar{z}_i = z_i^{\mathrm{nad}}$ for $i \in I^\diamond$. The following subproblem uses such a reference point information in an achievement scalarizing function which also generates Pareto optimal solutions [42]

$$\begin{aligned} \text{minimize} \quad & \max_{i=1,\ldots,k} \left[\frac{f_i(\mathbf{x}) - \bar{z}_i}{z_i^{\mathrm{nad}} - z_i^{\star\star}} \right] + \rho \sum_{i=1}^{k} \frac{f_i(\mathbf{x})}{z_i^{\mathrm{nad}} - z_i^{\star\star}} \\ \text{subject to} \quad & \mathbf{x} \in S. \end{aligned} \tag{1.3}$$

One should note that any point in the objective space $\mathbb{R}^k$ can be projected to the Pareto optimal set by setting it as a reference point to problem (1.3). This is how the NIMBUS method generates the initial Pareto optimal solution and intermediate solutions as per the core structure described in Subsection 1.2.1. For more details of the NIMBUS method, see [21, 23, 31].

1.2.3 NAUTILUS Method

As discussed so far, typically, solving multiobjective optimization problems involve considering Pareto optimal solutions only. Thus, the DM must study different trade-offs between conflicting objectives and accept losses in at least objective function to gain in some other objective function. However, according to the prospect theory [11], humans do not react symmetrically to gains and losses and it has been suggested that making explicit trade-offs can hinder the decision-making process of finding the most preferred solutions. It was shown, e.g., in [34] that trade-offs implied negative reactions in DMs. Furthermore, it has been discussed in [4, 11] that our past experiences may limit our future expectations, causing the DMs to anchor near some particular solution, such as the initial solution. To avoid these shortcomings, a new approach called the NAUTILUS method was introduced in [19] and further variations and extensions of it in [25, 36].

What is common in all members of the NAUTILUS method family is that the solution process begins from the worst possible, i.e., a nadir objective vector, or from any point from where all objective function values can be simultaneously improved. From this point, the DM is iteratively progressing towards the Pareto optimal set in order to eventually find the most preferred solution for him/her. This is achieved by showing to the DM the current objective function values forming an iteration point and bounds indicating objective values which can be reached from that point without trade-offs.

In our use case, we apply the original NAUTILUS method [19]. For example, if the first iteration point is the nadir objective vector, from that point, the reachable set of solutions is limited by the ideal objective vector. The DM is then asked to give the number of steps to be taken and specify preferences in which direction (s)he wants to move from the current iteration point. With this information, a new iteration point is generated by taking a step towards a Pareto optimal solution found by solving problem (1.3). As this point is closer to the Pareto optimal set than the previous point, the reachable set of solutions shrinks. Information about ranges of reachable objective values is updated (by solving an ε-constraint problem [19]) and shown to the DM. The process is continued until the specified number of steps has been taken and, thus, a Pareto optimal solution reached. At any iteration of the process, the DM can change his/her preferences for the next iteration or return to any previous iteration and provide new preference information there. The DM is also shown the distance from the current iteration point to the closest Pareto optimal solution. For more details about the NAUTILUS family of methods, see [26]).

1.3 DESDEO Framework

The motivation behind the DESDEO framework has been to provide a tool set for researchers and practitioners, which can be utilized for applying and developing interactive multiobjective optimization methods. To this end, we next introduce the main components of the DESDEO framework, that is, the structures needed when implementing interactive multiobjective optimization methods. To be more specific, we describe the general design and architecture of the framework to give the reader an insight of how the framework can be extended. Then, we describe in Section 1.4 how the already implemented methods can be used in practice.

1.3.1 Structure of the DESDEO Framework

The main design of the DESDEO framework consists of different components which can be utilized for implementing interactive multiobjective methods. The aim of the structure is to facilitate both developing new methods as well as implementing previously published methods. To this end, we consider the underlying structures of interactive algorithms, described in Subsection 1.2.1 as a core structure (see also [31]). As mentioned earlier, we do not discuss user interfaces, but algorithms. Instead, the DESDEO framework is intended to be connected to an external user interface, such as the web based DESDEO application (or IND-NIMBUS [17, 31]). For an example of a user interface development see, e.g., [41]. As the framework is under constant development, no detailed descriptions on the use and class structures are given here, but they are available at the DESDEO web site with quick instructions how to use the framework with accompanying README -file.

The structure underlying the DESDEO framework is visualized in Figure 1.1. In essence, the framework consists of four different layers, each of which is com-

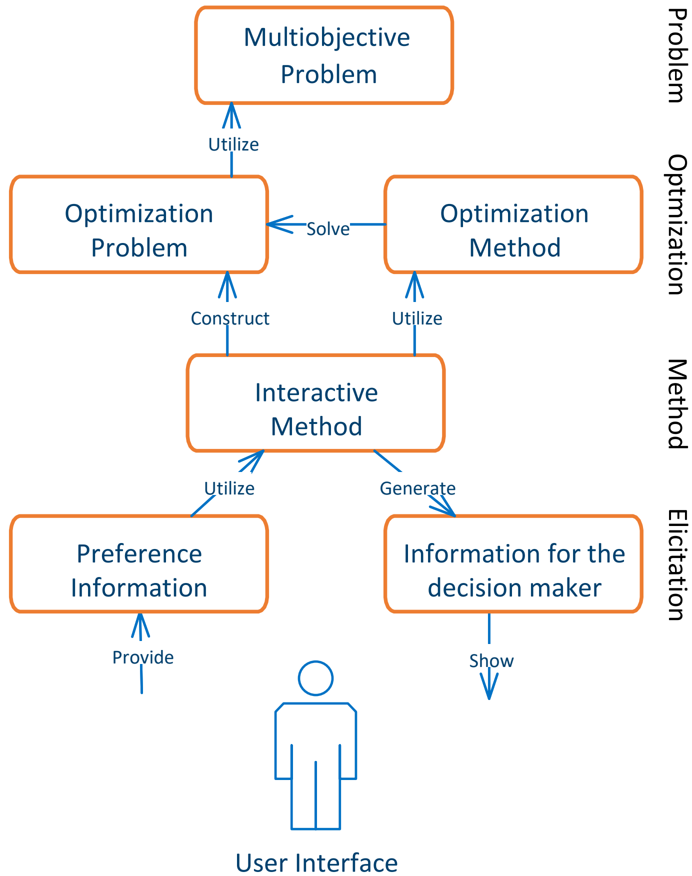

Fig. 1.1 The DESDEO framework overview

municating using predefined communication channels to allow reusability between the components on different layers. The layers are called problem, optimization, method, and elicitation, which follow closely the module structure of the DESDEO framework. In practice, the structure used means that the layers are separated from each other and can be changed as needed. As mentioned earlier, applying any interactive method requires a user interface as a fifth layer but the DESDEO framework does not consider the elicitation of preference information beyond storing different types of preferences.

In the DESDEO framework, the elicitation layer is the layer that the DM is directly interacting with, typically using an interface. It is used to obtain preference information from the DM and to convert this information into a format required by the interactive method in question. Furthermore, it collects the information generated during the interactive solution process and presents it in the format required by the user interface to be shown to the DM. In the method layer, the interactive method utilizes the preference information to construct an appropriate subproblem or subproblems to be solved in the optimization layer. As mentioned earlier, the subproblems are solved with a suitable single objective optimization method. The top-most layer in the figure, the problem layer, contains the problem model which defines the multiobjective optimization problem to be solved, such as the formulations of the objective functions and constraints.

The key point of the structure presented in Figure 1.1 is generality, that is, avoiding details of any particular method. Indeed, the framework is general and different components can be replaced as needed. For example, different interactive methods should be able to be applied for solving the same multiobjective optimization problem without changes to the method implementations. Naturally, there are some meaningful limitations to this and, for example, linear multiobjective optimization problems should not be solved with methods aimed at solving black-box problems, but methods appropriate to the characteristics of the problem in question should be applied.

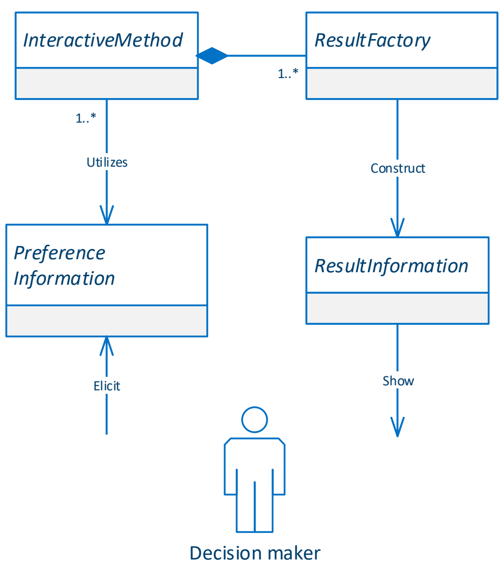

Fig. 1.2 The DESDEO base class structure

In practice, the separation of the layers is realized by using an object-oriented structure, where all functionalities are implemented within subclasses derived from abstract base classes. The four main base classes of the DESDEO framework are illustrated in Figure 1.2 using a pseudo Unified Modeling Language (UML) diagram (see [8] for further information about UML). The classes illustrated are *abstract base classes*, that is, they are never instantiated as class objects. Instead, they define the base functionality, which their *subclasses*, i.e., classes derived from them should implement. These subclasses offer the user of the DESDEO framework a concrete functionality needed when implementing different interactive methods. In other words, a DESDEO implementation of an interactive method consists of subclasses derived from the base classes shown in Figure 1.2. In what follows, we give examples of subclasses of the *PreferenceInformation* class for handling different types of preference information (obtained from the DM). We also show how the *InteractiveMethod* class can be extended when implementing the NIMBUS method.

1.3.2 Preference Handling in the DESDEO Framework

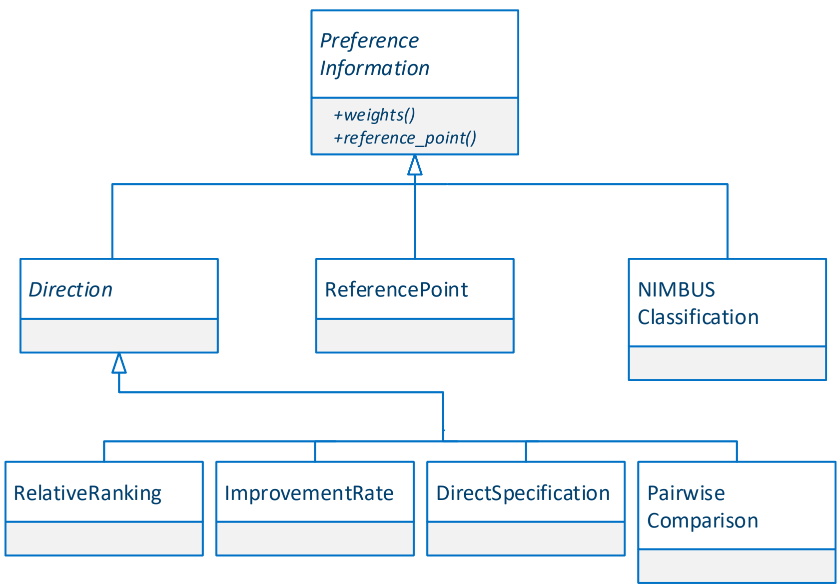

Fig. 1.3 DESDEO preference handling class structure

As mentioned previously, the separation of the layers is realized by using an object-oriented structure, where all functionalities are implemented within subclasses derived from abstract base classes. As an example, Figure 1.3 demonstrates how the framework handles preference information elicited from the DM. In the figure, the main abstract class *PreferenceInformation* defines and enforces the interface that allows the user interface to get the DM's preferences as well as the interactive method to gain access to the DM's preferences in a format required by the method. As an example of the latter, the *PreferenceInformation* class provides two abstract methods to access preferences, that is, a *weights* method which return a vector of weights if such preference information is required and a *reference_point* method which return the preference information as a reference point consisting of aspiration levels. These two methods are the main interface for accessing the preference information, and as abstract methods, they should be implemented in every subclass of the *PreferenceInformation* base class.

The classes defined in the DESDEO framework are self-contained, that is, after the initialization, they are assumed to contain or to have access to all necessary information to perform their functionalities. For example, when providing the *Classification* class with new preferences elicited from the DM, it is assumed that the class has access to information related to the problem being solved and can verify that the preference information is suitable for it, e.g., the number of objective functions is correct. Similarly, it is the responsibility of the subclasses of the *PreferenceInformation* class to implement the methods such as weights and reference_point and present the elicited preference information in the requested format. If such presentation is not possible, an exception should be raised and the interactive method requesting the preference information should either ask for another presentation of the preferences if possible or handle the situation as an error which is passed back to the user interface and to the DM. In this way, the implementation of the interactive method does not need to mind how preferences are given in the user interface and the responsibility that the preference information is suitable for the method in question lies within the user interface implementation.

In the DESDEO framework, we have three different examples on how preferences can be elicited and handled within the framework. Preference information can be provided as a desired direction of simultaneous improvements (used, e.g., by NAUTILUS [26]), classification of objective functions (used, e.g., by NIMBUS [23]), or specifying a reference point of aspiration levels (see, e.g., [43]).

Different ways of specifying the direction of simultaneous improvement are discussed e.g., [26]). For eliciting the direction of simultaneous improvement, we have a *Direction* base class, from which four subclasses are derived. That is, the direction can be specified by ranking each objective function based on the importance of being improved (*RelativeRanking*), providing an improvement rate as a percentage how much the DM would like to improve each objective (*ImprovementRate*), by direct specification (*DirectSpecification*) or by pairwise considerations of the objective functions to define improvement ratios between them (*PairwiseComparison*).

Other options to provide preference information can similarly be modified or extended, if needed. For example, in addition to classifying objectives into five classes using the *NIMBUSClassification* class described in Subsection 1.2.2, they could be classified as per the STOM or Step methods to three or two classes, see [3, 29], respectively. In this case, there would be a need to introduce a new *Classification* abstract class, from which a new *STOMClassification* class would be derived (along with the current Classification class named as *NIMBUSClassification*).

1.3.3 Extending the DESDEO Framework

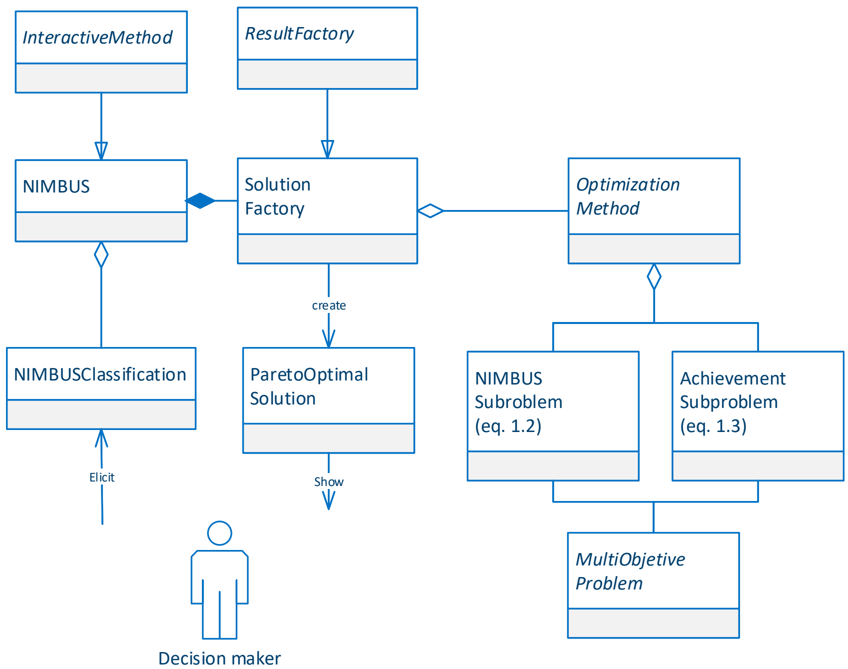

Fig. 1.4 Class structure of the NIMBUS method implementation

In Figure 1.4, we present the class structure of the NIMBUS method implemented within the DESDEO framework. The *NIMBUS* class, which is a subclass of *InteractiveMethod*, is where the algorithm of the NIMBUS method is implemented. That is, it defines what kind of preference information should be elicited from the DM and how that information is used to obtain new solutions. The preference information must be provided as a subclass of the *PreferenceInformation* class (shown in Figure 1.2), which in the case of the NIMBUS method is the *NIMBUSClassification* class. The *NIMBUSClassification* class stores the classification information described in Subsection 1.2.2 and is passed to a *SolutionFactory* class, which uses it to create new Pareto optimal solutions. For generating new solutions, *SolutionFactory* has an access to subproblems to be solved and appropriate single objective optimization methods. These are encapsulated in an *OptimizationMethod* class.

As an example of the four possible subproblems of the NIMBUS method, in Figure 1.4 we have the *NIMBUSSubroblem* corresponding to problem (1.2) and *AchievementSubroblem* corresponding to problem (1.3). Finally, these subproblems are associated with the multiobjective optimization problem (*MultiObjectiveProb-*

lem) being solved. (For further information about NIMBUS subproblems, see Subsection 1.2.2 and [23].)

Of the classes mentioned, only *NIMBUS* and *SolutionFactory* are directly specific to the NIMBUS method implementation and all other classes can be utilized when implementing another method. For example, when implementing the NAUTILUS method, the same *AchievementProblem* class can be used when generating new intermediate points with the same *OptimizationMethod* and *MultiObjectiveProblem* classes. Similarly, as mentioned earlier, even though the preference information is given as *NIMBUSClassification*, the NIMBUS classification information can be easily converted to a reference point and it could be given as a *ReferencePoint* as they both implement the same interface for preference handling. On the other hand, as described in Subsection 1.2.3, in the NAUTILUS method only the final solution shown to the DM is Pareto optimal and, thus, the *SolutionFactory* could not be utilized but the method requires its own subclass (i.e. *IterationPointFactory* class, which is not discussed here but is available in the DESDEO framework).

We do not go into details of the *OptimizationMethod* class. Naturally, when solving the subproblems to find new solutions, the selected single objective optimization method should be suitable for solving the scalarized subproblems, which consist of the objective functions and constraints of the underlying multiobjective optimization problem (described by the *OptimizationMethod* class). The subproblems (1.2) and (1.3) presented here are nondifferentiable as they involve min-max functions and, therefore, the single objective optimization methods applied should be suitable for such problems. If the multiobjective optimization problem in question is differentiable, these subproblems can be reformulated to their differentiable equivalents by adding a new decision variable and converting the min-max functions as constraints (see, e.g., [16]). Then, the interactive method implementation can be used as is, by changing the *SolutionFactory* to use the new differentiable subproblems and a suitable subclass of *OptimizationMethod*.

In the next section, we will consider the DESDEO framework from another angle. We give an example of how a multiobjective optimization problem can be formulated and solved with the framework.

1.4 Use Case: River Pollution Problem

In this section, we demonstrate how the DESDEO framework can be applied for solving a four-objective river pollution problem formulated in [30]. We first apply the NAUTILUS method to find a single Pareto optimal solution and then use it as the starting point of the NIMBUS method to refine that solution. As said, both of these methods have been implemented in the framework. The single objective subproblems related to the interactive methods are solved with differential evolution [39] available from the SciPy module [33]. Differential evolution is using the default values of the module.

The problem considers a river being polluted by a fishery and a city. The pollution is controlled by two treatment plants, one managed by the fishery and another managed by the city. The aim is to improve the quality of water in both the city and the fishery and also minimize the costs incurred.

To be more specific, there exist two treatment plants, one in the fishery and one in the city. The pollution is described in pounds of biochemical oxygen demanding material (BOD) and the two decision variables considered, x_1 and x_2, represent the proportional amounts of BOD removed from water in the two treatment plants, respectively. The first two objective functions are to be maximized and involve water quality: f_1 represents water quality of the fishery and objective f_2 of the city as pounds of BOD.

The third objective f_3 represents return on investment (ROI) at the fishery as a percentage to be maximized and the fourth objective function f_4 to be minimized is the increase of the tax rate in the city due to operating the treatment plant. The objective functions are formulated as follows

$$\begin{aligned}
&\text{maximize} && f_1(\mathbf{x}) = 4.07 + 2.27x_1\\
&\text{maximize} && f_2(\mathbf{x}) = 2.60 + 0.03x_1 + 0.02x_2 + \frac{0.01}{1.39 - x_1^2} + \frac{0.30}{1.39 - x_2^2}\\
&\text{maximize} && f_3(\mathbf{x}) = 8.21 - \frac{0.71}{1.09 - x_1^2}\\
&\text{minimize} && f_4(\mathbf{x}) = -0.96 + \frac{0.96}{1.09 - x_2^2}\\
&\text{subject to} && 0.3 \leq x_1, x_2 \leq 1.0.
\end{aligned}$$

The DESDEO model of the problem is given in Appendix 1.5 with some further information.

1.4.1 DESDEO Model of the Problem

The problem class structure of the DESDEO framework is shown in Figure 1.5. As can be seen, all problems solved with the DESDEO framework must be derived from the *MultiObjectiveProblem* base class. The *MultiObjectiveProblem* class is in the problem layer, as seen in Figure 1.1. This means that it offers an interface for how the other components of the DESDEO framework can have an access to a model of a multiobjective optimization problem. As other layers, *MultiObjectiveProblem* is an abstract base class, and concrete functionalities must be implemented with its subclasses. It should be noted that the DESDEO framework assumes that all objectives are to be minimized, i.e., if an objective is maximized, its values should be negated when shown to the DM (so that the values are understandable to the DM).

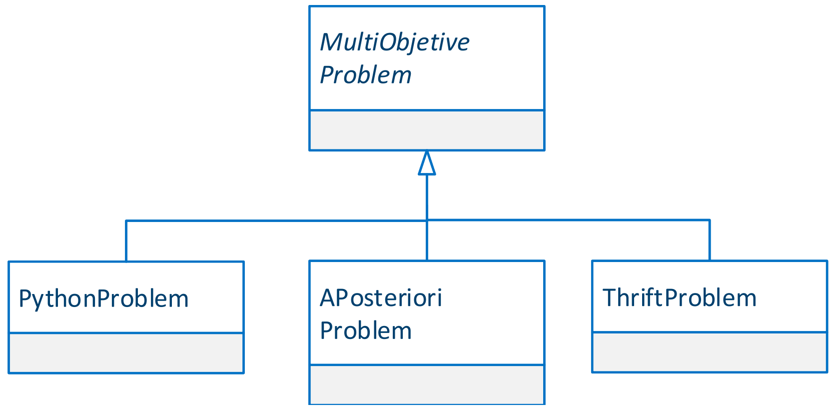

Fig. 1.5 DESDEO problem formulation class structure

Currently, *MultiObjectiveProblem* has three subclasses. Firstly, objective functions and constraints of the problem in question are formulated in *PythonProblem* with the Python language. Secondly, *APosterioriProblem* contains the previously generated set of Pareto optimal solutions approximating the Pareto optimal set. *APosterioriProblem* can access this information without any calculations. Finally, with *ThriftProblem*, the multiobjective optimization problem is formulated with an external modelling or simulation tool and accessed via an Apache Thrift protocol [37]. The problems modelled using the *ThriftProblem* class are typically referred to as black-box problems, where there may not exist analytical formulations of the objective functions and constraints, but their values are evaluated by calling a simulator. Alternatively, the ThriftProblem can also be modelled with a modelling tool like MATLAB or GAMS, where there can exist function formulations and the ThriftProblem problem gives access to DESDEO to the problem.

As with other classes of the framework, the base class defines a common interface for obtaining the information specific to a particular problem, allowing different problems to be solved with any interactive method implemented with the framework, provided that the underlying subproblems and single objective optimization methods are suitable for the problem. For example, if an interactive method is suitable for solving only linear problems, it should not be utilized for solving black-box problems. At the moment, all problem classes are for black-box problems, and extending the framework to e.g., linear problems would require adding additional classes derived from the *MultiObjectiveProblem* class, such as *BlackBoxProblem* and *LinearProblem*.

The river pollution problem is implemented as a subclass of the *PythonProblem* class and is available in the example directory of the DESDEO website as NarulaWeistroffer.py. There exists similar examples for other types of problems, and using the provided examples, building models for other multiobjective problems should be a straightforward task.

1.4.2 Interactive Solution Process

In what follows, we solve the river pollution problem using first the NAUTILUS method and then the NIMBUS method to find a single Pareto optimal solution as the final solution. As the user interface, we utilize the web-based application.

Among the advantages of the NAUTILUS method are that it allows finding a Pareto optimal solution corresponding to the DM's preferences while avoiding an anchoring bias as well as avoiding the need of trading off between objectives (see [19] and references within). On the other hand, when the DM has learned the main characteristics of the problem and wishes to explore some specific areas of the Pareto optimal set, it can be more intuitive to specify preferences in the form of a classification of objectives as in the NIMBUS method. With the classification, desired levels and bounds for the objectives can be directly specified and a new Pareto optimal solution is found without generating intermediate iteration points. As mentioned in [23], the starting point for the NIMBUS method can be any Pareto optimal solution given by the DM. Therefore, we first utilize the NAUTILUS method to find a Pareto optimal solution and then continue by further refining it with the NIMBUS method.

In NAUTILUS, the DM decided to use relative ranking of the objectives to express preferences. The DM specified ranks for each objective to indicate how important their improvement from the current iteration point was. This means that in each iteration, the method was expected to improve most the highest ranked objective functions. Several objectives could have the same rank. These ranks were then used to formulate the scalarized subproblem used for finding a new iteration point (for details, see e.g., [19]).

The solution process with the NAUTILUS method is summarized in Table 1.1. The first column of the table shows the iteration number and the second column indicates what information the row contains. The following columns contains values for the objective functions, that is, water quality at the fishery (WQ fishery), water quality at the city (WQ city), return on investment at the fishery (Fishery ROI) and the increase of the tax rate at the city (Tax increase), respectively. Information shown to the DM includes the iteration point (It. point) which indicates the current (not Pareto optimal) point from where the solution process is continued and bounds give information to the DM about the ranges of the objective function values in the set of Pareto optimal solutions that can be reached from the current iteration point without trading off (the range for each objective function is between the current iteration point and the bound values). Finally, the rows called Ranking contain ranks that the DM specified as preference information.

One should remember that the first three objective functions are to be maximized, i.e., for them the components of the ideal objective vector are the biggest values and for the nadir the smallest values. Naturally, the opposite is true for the fourth objective to be minimized. Therefore, the ideal and the estimated nadir objective vectors are $\mathbf{z}^{\star} = (6.34, 3.44, 7.50, 0.0)^T$ and $\mathbf{z}^{\mathrm{nad}} = (4.07, 2.89, 0.32, 9.71)^T$, respectively. As mentioned in Subsection 1.2.3, the NAUTILUS method is initialized with the nadir objective vector as the initial iteration point. Then, the ideal objective vec-

Iter	Issue	WQ fishery	WQ city	Fishery ROI	Tax increase
1	It. point	4.07	2.87	0.32	9.71
	Bound	6.34	3.44	7.50	0.00
	Ranking	2	2	1	1
2	It. point	4.60	2.97	1.58	8.70
	Bound	6.34	3.44	7.50	0.00
	Ranking	2	2	1	1
2	It. point	4.60	2.97	1.58	8.70
	Bound	6.34	3.44	7.50	0.00
	Ranking	2	2	1	1
3	It. point	5.10	3.10	2.88	7.67
	Bound	6.32	3.42	7.50	0.45
	Ranking	2	3	1	4
4	It. point	5.59	3.15	4.40	5.30
	Bound	6.30	3.40	7.40	1.42
	Ranking	1	1	2	2
	PO Solution	**6.03**	**3.23**	**6.15**	**3.21**

Table 1.1 Solving river pollution problem with the NAUTILUS method

tor gives the bounds, i.e., solutions that can be reached from the initial iteration point without trading-off. (The DM could also specify some other point as the initial iteration point, from which the reachable region would then be calculated by the method.)

In the beginning, the DM decided to take four steps to find a desired Pareto optimal solution. The DM was shown the nadir values of objectives and he decided to prioritize the environmental aspects over the economical issues by giving both objectives related to the water quality a higher importance of 2 and a lower importance of 1 to the return on investment at the fishery and the increase of the city taxes. As the best reachable objective values were still the ideal values, the DM concluded that each objective should still be improved in this direction. Therefore, the DM decided to continue with the same preferences, i.e., take one more step in the current direction.

In the second iteration, the DM obtained the iteration point $(5.1, 3.1, 2.88, 7.67)^T$ with the best reachable values $(6.32, 3.42, 7.5, 0.45)^T$. So far, the DM had set two objectives on the same rank of priorities but now he decided to concentrate on the water quality in the city in the third iteration by increasing its importance rank to 3. The DM also noticed that a significantly lower increase in the tax could still be gained and increased the importance rank of the corresponding objective to 4. Now each objective had a different rank and the obtained iteration point was $(5.59, 3.15, 4.4, 5.3)^T$ with reachable best values as $(6.3, 3.4, 7.4, 1.42)^T$. The DM decided that there is no need to improve the water quality further and gave an importance rank of 1 to the corresponding objectives. On the other hand, both return on investment at the fishery as well as city tax increase could still be improved. As the DM had regarded decreasing the tax increase significantly more importance in the previous iteration, he now decided to give the same importance rank of 2 to both objectives. Because this was the final iteration, this led to a Pareto optimal solution

$(6.03, 3.23, 6.15, 3.21)^T$. It should be noted that at any iteration the DM could have returned to any of the previous iteration points, change the number of intermediate iteration points generated or how the preferences are expressed, but in this solution process these options were not used. As mentioned earlier, NAUTILUS also gives information about the distance of the iteration points to the Pareto optimal set but in this case, the DM was not willing to use this information.

The DM was rather happy with the obtained Pareto optimal solution as he had been able to find it without trading off. However, he was still hoping for a somewhat smaller increase in the city taxes and a bit better return on investment for the fishery. The DM could now have returned, for example, to iteration 4 of NAUTILUS and give a higher importance rank to the fourth objective. On the other hand, he considered water quality to be satisfactory in the fourth iteration, which might mean that the DM should change also the importance ranks of those objectives. Therefore, instead of continuing with NAUTILUS, he decided to switch the method and proceed by refining the obtained Pareto optimal solution with the NIMBUS method starting from the final solution of NAUTILUS.

The solution process with the NIMBUS method is summarized in Table 1.2. The main difference to the solution process with the NAUTILUS method shown in Table 1.1 is that in the NIMBUS method, all solutions shown are Pareto optimal, whereas only the final solution of NAUTILUS is Pareto optimal. During the NIMBUS solution process, the DM is shown the ranges of the objective functions, i.e. ideal and nadir values, here depicted in the first two rows of Table 1.2. The ranges stay the same for duration of the solution process, unless some Pareto optimal solution found has better or worse objective values than found with the pay-off table (in which case they are updated accordingly). Because we are solving the same problem, the ideal and nadir values are the same as when using the NAUTILUS method.

Iter	Issue	WQ fishery	WQ city	Fishery ROI	Tax increase
	Ideal	6.34	3.44	7.50	0.00
	Nadir	4.07	2.89	0.32	9.71
1	NAUTILUS sol.	6.03	3.23	6.15	3.21
	Classif	$I^{\geq=5.5}$	$I^{\geq=3.0}$	$I^{\leq=6.5}$	$I^{\leq=2.0}$
		5.63	3.05	7.07	1.20
		5.84	3.09	6.74	1.57
		5.58	3.05	7.12	1.23
2	Cur. Sol.	5.84	3.09	6.74	1.57
	Classif	$I^{\leq=6.0}$	$I^{\leq=3.1}$	$I^{\geq=6.5}$	$I^{\geq=2.0}$
		6.33	3.34	0.9	5.35
		5.97	3.15	6.4	2.09
		5.97	3.15	6.37	2.16
	Final Sol.	**5.97**	**3.15**	**6.40**	**2.09**

Table 1.2 Refining the Pareto optimal solution found by the NAUTILUS method with the NIMBUS method

Typically, the NIMBUS method generates an initial solution to be shown for the DM, but in this case the DM wished to continue the solution process from the Pareto optimal solution found with NAUTILUS. Therefore, the solution process started from the Pareto optimal solution $(6.03, 3.23, 6.15, 3.21)^T$ (denoted by NAUTILUS sol. in the table). The classification information provided by the DM is shown on the row names as "Classif". The notation corresponds to the one given in Subsection 1.2.2 and the aspiration level or bound specified by the DM in connection with the class is given after the symbol of the class. For example, in the first iteration, the DM decided to allow the water quality at the fishery to decrease till the bound 5.5 indicated by $I^{\geq=5.5}$. The main aim of the DM was to improve, that is, to decrease the tax increase in the city, but at the same time to maintain satisfactory values for the other objectives. Therefore, in the first NIMBUS iteration, the DM wanted to improve the tax increase till an aspiration level of 2.0 as well get a relatively smaller improvement on the return on investment by giving it an aspiration level of 6.5. At the same time, the DM did not wish to impair the water quality too much and, therefore, gave bounds of 5.5 and 3.0. As mentioned earlier, the synchronous NIMBUS method uses four different single objective subproblems and it is possible to generate up to four new solutions in each iteration, but on both of the NIMBUS iterations, the method was able to provided only three different Pareto optimal solutions to be shown to the DM as two of the solutions were too similar to each other. All of them were able to achieve the aspiration level that the DM desired and of these, the DM decided to select the one with the tax increase closest to his preferences, i.e., $(5.84, 3.09, 6.74, 1.57)^T$.

For the second NIMBUS iteration, he wanted to see whether it would be possible to obtain satisfactory levels of water quality while maintaining good values for the economic aspects. Therefore, he gave the previously given aspiration level of 2.0 as a boundary for the impairment of the city tax increase and 6.5 as a lower bound of the impairment at the fishery's return on investment. Based on the previous results, he thought that it could be possible to find Pareto optimal solutions with better water quality than the bounds he specified in the first iteration and gave aspiration levels of 6.0 to the water quality in the fishery and 3.1 to the water quality in the city. Again, the NIMBUS method generated three different Pareto optimal solutions (and the bounds could not be strictly obeyed). The DM selected $(5.97, 3.15, 6.40, 2.09)^T$ since the desired water quality for the fishery was achieved and the water quality in the city was also close to the desired level. Even though both the fishery's return on investment and the city tax increase were somewhat worse than the given bounds, the obtained values were satisfactory for the DM. Based on the results obtained so far and the learning that had taken place, the DM decided that it would be unlikely to obtain solutions with significantly better objective function values, and selected $(5.97, 3.15, 6.40, 2.09)^T$ the final, most preferred Pareto optimal solution for the problem.

1.5 Conclusions

We have introduced the DESDEO framework that is aimed at providing openly available implementations of different interactive methods for multiobjective optimization. The framework is published under a permissive open source license and is freely available at `https://desdeo.it.jyu.fi`. By utilizing the methods implemented in the framework with the accompanying user interface, interactive multiobjective optimization methods can be used without having a strong technical background. The framework has a modular structure of self-contained components. The framework contains implementations of several methods as well as components that can be used when implementing new methods. By following the information provided at the DESDEO web site, one can extend the methods already implemented and implement other methods in the framework. The framework documentation also provides examples on how to use different single objective optimization methods and how to model multiobjective optimization problems in the framework. A user with some experience on programming should be able to extend them for his/her own needs.

To demonstrate the applicability of the framework, we have applied two methods, namely the original NAUTILUS and the synchronous NIMBUS methods to solve a multiobjective river pollution problem with four objectives. This example demonstrates the benefits of having several methods implemented in the same framework as the DM could conveniently switch the method during the solution process without any additional effort. Naturally, methods can also be utilized separately, i.e., one can apply different methods to solve the same problem and, for example, compare the results if so desired.

The development of the DESDEO framework is ongoing work. At the moment, the framework is best suited for solving nonlinear problems with continuous variables (because of the single objective optimization methods available), but we plan to extend it with options for solving e.g. linear problems and problems with mixed integer variables. We did not here discuss graphical components related to interactive methods, but naturally the work with the DESDEO web application will continue. Currently, there exist several research lines on comparing different interactive methods (see, e.g., [15, 32]) and our aim is to include such methods in the DESDEO framework in order to build a corresponding tool set for interactive multiobjective optimization as already exist for evolutionary multiobjective optimization methods, such as JMetal [7]. Furthermore, implementation of the *ThriftProblem* class used for black-box optimization should be restructured, as it has potential security issues and it is not suitable to be used over public network.

Acknowledgements This work was supported on the part of Vesa Ojalehto by Academy of Finland (grant number 287496).

Appendix

In what follows, we give source listings of the river pollution problem solved in Subsection 1.4.2. As mentioned earlier, we consider a river that is polluted by a city and a fishery. The aim of the DM is to improve the quality of water and to minimize the costs incurred. The problem has four objectives and two variables, three of the objectives to be maximized and the fourth one to be minimized.

To understand how the problem can be solved, we provide the source code listing in the *examples* directory of the DESDEO framework as a file named NarulaWeistroffer.py. Note that the framework is under an active development.

The initialization of the River Pollution problem can be seen in Listing 1.1. As mentioned earlier, all concrete problems must be derived from some base class. As the problem is formulated with Python, we import the *PythonProblem* class from the problem module, from which a *RiverPollution* subclass is derived. In the *RiverPollution* class, we first provide a description of the problem (with the relevant reference as a documentation string). We then proceed with the problem formulation.

Problem dimensions, variables and other characteristics are defined in the *__init__* method of the problem class. Of those, only the number of objective functions and the box constraints of the decision variables are required, other parameters are optional. For the river pollution problem, the number of objective (nobj) is four, and it does not have other constraints (nconst) besides box constraints. As the ideal and nadir objective vectors are known they can be provided, but if they are not known, the method calculates them if needed. Even though the DESDEO framework assumes that all objective functions are to be minimized, it is possible to provide a value for the parameter *maximized* in order the DESDEO framework to convert objective function values whenever communicating from and to the user interface. If this value is not set, such conversions cannot be made, and they must be handled by the user interface. The value True of the parameter indicates that the objective is to be maximized and with False, it is to be minimized. It is also possible to give names to objective functions as well as a name for the problem as shown in the listing.

Finally, the problem requires decision variables provided with the *add_variable* method of the class with an instance of the *Variable* class. The *Variable* class is initialized with three parameters, namely box constraints giving upper and lower bound value for the decision variables, starting point to be used when solving the problem and name of the variable. Of these, only the first, box constraints, is required and the others are optional. If the starting point is not given but the single objective optimization method used requires a starting point, the lower bounds of the variables are used as the starting point. It should be noted that when the DESDEO framework is extended to handle problems with discontinuous variables, the variable handling must be changed.

In Listing 1.2, we show how objective functions are given by overloading the *evaluate* method of the *PythonProblem* class. As an input parameter, the *evaluate* method takes a population, which is a set of decision variable vectors each representing a new objective vector to be calculated. This means that when called, the *evaluate* method evaluates objective function values for each decision variable vec-

Listing 1.1 Problem initialization

```
import math
from pyDESDEO.problem import PythonProblem, Variable

class RiverPollution(PythonProblem):
   """
    River pollution problem by Narula and Weistroffer [1]

    The problem has four objectives and two variables

    The problem describes a (hypothetical) pollution problem
    of a river, where a fishery and a city are polluting
    water. The decision variables represent the proportional
    amounts of biochemical oxygen demanding material removed
    from water in two treatment plants located after the
    fishery and after the city.

    The first and second objective functions describe the
    quality of water after the fishery and after the city,
    respectively, while objective functions three and four
    represent the percent return on investment at the fishery
    and the addition to the tax rate in the
    city. respectively.


    References
    ----------

    [1] Narula, S. & Weistroffer, H. A flexible method for
      nonlinear multicriteria decision-making problems
      Systems, IEEE Transactions on Man and Cybernetics,
      1989, 19, 883-887.
    """

    def __init__(self):
        super(RiverPollution, self).__init__(
            nobj=4,
            nconst=0,  # Optional
            ideal=[-6.34, -3.44, -7.5, 0.1],      # Optional
            nadir=[-4.07, -2.87, -0.32, 9.71],    # Optional
            maximized=[True, True, True, False],  # Optional
            objectives=['Water Quality Fishery',  # Optional
                        'Water Quality City',
                        'Fishery ROI',
                        'City Tax Increase'],
            name='River pollution problem',       # Optional

        )
        self.add_variables(Variable([0.0, 1.0],
                                  starting_point=0.5, # Optional
                                  name='BOD City'))   # Optional
        self.add_variables(Variable([0.0, 1.0],
                                  starting_point=0.5, # Optional
                                  name='BOD City'))   # Optional
```

Listing 1.2 Problem initialization

```
    def evaluate(self, population):
        objectives = []

        for values in population:
            res = []
            x0_2 = math.pow(values[0], 2)
            x1_2 = math.pow(values[1], 2)

            res.append(-1.0 * (4.07 + 2.27 * values[0]))

            res.append(-1.0 * (2.6 + 0.03 * values[0] + 0.02 *
                                  values[1] + 0.01 / (1.39 - x1_2)
                                   + 0.3 / (1.39 - x1_2)))

            res.append(-1.0 * (8.21 - 0.71 / (1.09 - x0_2)))

            res.append(-1.0 * (0.96 - 0.96 / (1.09 - x1_2)))

            objectives.append(res)

        return objectives
```

Listing 1.3 Solving the problem with NAUTILUS

```
    from pyDESDEO.utils import tui
    from pyDESDEO.method import NAUTILUSv1
    from pyDESDEO.optimization import SciPyDE

    method = NAUTILUSv1(RiverPollution(), SciPyDE)

    NAUTILUS_solution = tui.iter_nautilus(method)[0]

    print(method.problem.to_ui(NAUTILUS_solution))
   # Output:
   # [-6.2927077117830965, -3.4038593790999485,
   #    -7.401394350956817, 1.6201876469013787]
```

tor in the population. If objective function values of a single decision vector are to be evaluated, the population should have only this single decision variable vector. As a return value, the *evaluate* method gives the objective vector as a list of objective function values.

As mentioned, the DESDEO framework does not include a graphical user interface for solving problems. It does include a text-based framework for building iterative solution processes called *tui*. In Listing 1.3, we give an example on how the framework can be used with the *tui* module to solve a problem via a text-based interface using the first variant of the NAUTILUS method family (which was the first method used in the interactive solution process reported in Subsection 1.4.2).

In the first three lines of the Listing 1.3, we import additional classes and modules needed. That is, the *tui* module containing the text-based user interface, the class *NAUTILUSv1* corresponding to the method and finally *SciPyDE* as the single objective optimization method to be used when solving the scalarized subproblems.

On the last three lines, we first initialize the method class by providing it with an instance of a multiobjective optimization problem formulated earlier and with the single objective optimization method *SciPyDE* to be used. The call to *tui.iter_nautilus* function in the line 7 starts the interactive solution process asking the DM to specify the preference information and returning the final solution obtained. On the last line, we print out the obtained solution which are converted from the minimized values to maximized using method *problem.to_ui*.

Listing 1.4 Interactive solution process with the NIMBUS method

```
1  from pyDESDEO.method import NIMBUS
2  from pyDESDEO.preference import NIMBUSClassification
3
4  method = NIMBUS(RiverPollution(), SciPyDE)
5
6  method.selected_solution = NAUTILUS_solution
7
8  class1 = NIMBUSClassification(method.problem,
9                                [(">=", -5.5),
10                                (">=", -3.0),
11                                ("<=", -6.5),
12                                ("<=", -2.0)])
13 iter1 = method.nextIteration(preference = class1)
14 print(method.problem.to_ui(iter1))
15 # Output
16 # [[5.63,3.05,7.07,1.20],
17 # [5.84,3.09,6.74,1.57],
18 # [5.58,3.05,7.12,1.23]]
```

In Listing 1.4, we give more details on how an interactive solution process can be implemented using the NIMBUS method as an example. The basic idea is same for all interactive methods currently implemented in the framework. As with the NAUTILUS example, we first import new classes needed when using the NIMBUS method, namely the *NIMBUS* class itself, and the *NIMBUSClassification* class to store the preference information obtained from the DM. We use the same *RiverPollution* and *SciPyDE* classes to initialize the method used on line 4. In contrast to the NAUTILUS example, here we do not use the *tui* module to get the DM's preferences and to show solutions. Instead, they are given directly to show how the interactive solutions process is constructed.

On line 6, we set the previously obtained *NAUTILUS_solution* as the current, selected solution. On lines 8 to 12 new preference information is given as a NIM-

BUS classification and initialized as the *class1* object of *NIMBUSClassification*. The preference information is the same as the information given be the DM in Table 1.2. New solutions are generated on line 13 with the *nextIteration* method of the *NIMBUS* object and printed out in line 14. This can then be continued until the final solution is found.

References

1. Agrell, P.J., Lence, B.J., Stam, A.: An interactive multicriteria decision model for multipurpose reservoir management: The Shellmouth Reservoir. Journal of Multi-Criteria Decision Analysis **7**(2), 61–86 (1998)
2. Bechikh, S., Ben Said, L., Ghedira, K.: Estimating nadir point in multi-objective optimization using mobile reference points. In: IEEE Congress on Evolutionary Computation (CEC), pp. 1–9 (2010)
3. Benayoun, R., de Montgolfier, J., Tergny, J., Laritchev, O.: Linear programming with multiple objective functions: Step method (STEM). Mathematical Programming **1**, 366–375 (1971)
4. Buchanan, J.T., Corner, J.: The effects of anchoring in interactive MCDM solution methods. Computers & Operations Research **24**(10), 907–918 (1997)
5. Chankong, V., Haimes, Y.Y.: Multiobjective Decision Making Theory and Methodology. North-Holland, New York (1983)
6. Deb, K., Miettinen, K., Chaudhuri, S.: Towards an estimation of nadir objective vector using a hybrid of evolutionary and local search approaches. IEEE Transactions on Evolutionary Computation **14**(6), 821–841 (2010)
7. Durillo, J.J., Nebro, A.J.: jmetal: A java framework for multi-objective optimization. Advances in Engineering Software **42**, 760–771 (2011)
8. Fowler, M.: UML distilled: a brief guide to the standard object modeling language. Addison-Wesley Professional (2004)
9. Hakanen, J., Sahlstedt, K., Miettinen, K.: Wastewater treatment plant design and operation under multiple conflicting objective functions. Environmental Modelling & Software **46**(1), 240–249 (2013)
10. Hwang, C.L., Masud, A.S.M.: Multiple Objective Decision Making, Methods and Applications: A State-of-the-art Survey. Springer, Berlin, Heidelberg (1979)
11. Kahneman, D., Tversky, A.: Prospect theory: An analysis of decision under risk. Econometrica pp. 263–291 (1979)
12. Kaliszewski, I.: Out of the mist–towards decision-maker-friendly multiple criteria decision making support. European Journal of Operational Research **158**(2), 293–307 (2004)
13. Korhonen, P., Salo, S., Steuer, R.E.: A heuristic for estimating nadir criterion values in multiple objective linear programming. Operations Research **45**(5), 751–757 (1997)
14. Li, L., Yevseyeva, I., Basto-Fernandes, V., Trautmann, H., Jing, N., Emmerich, M.: Building and using an ontology of preference-based multiobjective evolutionary algorithms. In: H. Trautmann, G. Rudolph, K. Klamroth, O. Schütze, M. Wiecek, Y. Jin, C. Grimme (eds.) Proceedings of the 9th International Conference on Evolutionary Multi-Criterion Optimization, pp. 406–421. Springer (2017)
15. López-Ibáñez, M., Knowles, J.: Machine decision makers as a laboratory for interactive EMO. In: A. Gaspar-Cunha, C. Henggeler Antunes, C.C. Coello (eds.) Evolutionary Multi-Criterion Optimization, Lecture Notes in Computer Science, pp. 295–309. Springer International Publishing (2015)
16. Miettinen, K.: Nonlinear Multiobjective Optimization. Kluwer Academic Publishers (1999)
17. Miettinen, K.: IND-NIMBUS for demanding interactive multiobjective optimization. In: T. Trzaskalik (ed.) Multiple Criteria Decision Making '05, pp. 137–150. The Karol Adamiecki University of Economics in Katowice, Katowice (2006)

18. Miettinen, K.: Using interactive multiobjective optimization in continuous casting of steel. Materials and Manufacturing Processes **22**(5), 585–593 (2007)
19. Miettinen, K., Eskelinen, P., Ruiz, F., Luque, M.: NAUTILUS method: An interactive technique in multiobjective optimization based on the nadir point. European Journal of Operational Research **206**(2), 426–434 (2010)
20. Miettinen, K., Hakanen, J.: Why use interactive multi-objective optimization in chemical process design. In: G.P. Rangaiah (ed.) Multi-objective Optimization: Techniques and Applications in Chemical Engineering, pp. 153–188. World Scientific (2008)
21. Miettinen, K., Mäkelä, M.M.: Interactive multiobjective optimization system WWW-NIMBUS on the Internet. Computers & Operations Research **27**(7-8), 709–723 (2000)
22. Miettinen, K., Mäkelä, M.M.: On scalarizing functions in multiobjective optimization. OR Spectrum **24**(2), 193–213 (2002)
23. Miettinen, K., Mäkelä, M.M.: Synchronous approach in interactive multiobjective optimization. European Journal of Operational Research **170**(3), 909–922 (2006)
24. Miettinen, K., Mäkelä, M.M., Männikkö, T.: Optimal control of continuous casting by nondifferentiable multiobjective optimization. Computational Optimization and Applications **11**, 177–194 (1998)
25. Miettinen, K., Podkopaev, D., Ruiz, F., Luque, M.: A new preference handling technique for interactive multiobjective optimization without trading-off. Journal of Global Optimization **63**(4), 633–652 (2015)
26. Miettinen, K., Ruiz, F.: NAUTILUS framework: towards trade-off-free interaction in multiobjective optimization. Journal of Business Economics **86**(1), 5–21 (2016)
27. Miettinen, K., Ruiz, F., Wierzbicki, A.P.: Introduction to multiobjective optimization: Interactive approaches. In: J. Branke, K. Deb, K. Miettinen, R. Slowinski (eds.) Multiobjective Optimization: Interactive and Evolutionary Approaches, pp. 27–57. Springer-Verlag (2008)
28. Nakayama, H., Kaneshige, K., Takemoto, S., Watada, Y.: Application of a multi-objective programming technique to construction accuracy control of cable-stayed bridges. European Journal of Operational Research **87**(3), 731–738 (1995)
29. Nakayama, H., Sawaragi, Y.: Satisficing trade-off method for multiobjective programming. In: M. Grauer, A.P. Wierzbicki (eds.) Interactive Decision Analysis, pp. 113–122. Springer, Berlin (1984)
30. Narula, S., Weistroffer, H.: A flexible method for nonlinear multicriteria decision-making problems. IEEE Transactions on Systems, Man and Cybernetics **19**(4), 883–887 (1989)
31. Ojalehto, V., Miettinen, K., Laukkanen, T.: Implementation aspects of interactive multiobjective optimization for modeling environments: The case of GAMS-NIMBUS. Computational Optimization and Applications **58**(3), 757–779 (2014)
32. Ojalehto, V., Podkopaev, D., Miettinen, K.: Towards automatic testing of reference point based interactive methods. In: J. Handl, E. Hart, R.P. Lewis, M. López-Ibáñez, G. Ochoa, B. Paechter (eds.) Proceedings of the 14th International Conference on Parallel Problem Solving from Nature, pp. 483–492. Springer (2016)
33. Oliphant, T.E.: SciPy: Open source scientific tools for Python. Computing in Science and Engineering **9**, 10–20 (2007)
34. Ravaja, N., Korhonen, P., Köksalan, M., Lipsanen, J., Salminen, M., Somervuori, O., Wallenius, J.: Emotional–motivational responses predicting choices: The role of asymmetrical frontal cortical activity. Journal of Economic Psychology **52**, 56–70 (2016)
35. van Rossum, G.: Python tutorial. Tech. rep., Centrum voor Wiskunde en Informatica (CWI) (1995)
36. Ruiz, A.B., Sindhya, K., Miettinen, K., Ruiz, F., Luque, M.: E-NAUTILUS: a decision support system for complex multiobjective optimization problems based on the NAUTILUS method. European Journal of Operational Research **246**(1), 218–231 (2015)
37. Slee, M., Agarwal, A., Kwiatkowski, M.: Thrift: Scalable cross-language services implementation. Facebook White Paper **5**(8) (2007)
38. Stam, A., Kuula, M., Cesar, H.: Transboundary air pollution in Europe: An interactive multicriteria tradeoff analysis. European Journal of Operational Research **56**(2), 263–277 (1992)

39. Storn, R., Price, K.: Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. Journal of Global Optimization **11**(4), 341–359 (1997)
40. Szczepański, M., Wierzbicki, A.: Application of multiple criteria evolutionary algorithms to vector optimisation, decision support and reference point approaches. Journal of Telecommunications and Information Technology pp. 16–33 (2003)
41. Tarkkanen, S., Miettinen, K., Hakanen, J., Isomäki, H.: Incremental user-interface development for interactive multiobjective optimization. Expert Systems with Applications **40**, 3220–3232 (2013)
42. Wierzbicki, A.: A mathematical basis for satisficing decision making. Mathematical Modelling **3**, 391–405 (1982)
43. Wierzbicki, A.P.: The use of reference objectives in multiobjective optimization. In: G. Fandel, T. Gal (eds.) Multiple Criteria Decision Making Theory and Application, pp. 468–486. Springer (1980)