

**This is a self-archived version of an original article. This version may differ from the original in pagination and typographic details.**

**Author(s):** Misitano, Giovanni; Saini, Bhupinder Singh; Afsar, Bekir; Shavazipour, Babooshka; Miettinen Kaisa

**Title:** DESDEO: The Modular and Open Source Framework for Interactive Multiobjective Optimization

**Year:** 2021

**Version:** Published version

**Copyright:** © 2021 the Authors

**Rights:** CC BY 4.0

**Rights url:** <https://creativecommons.org/licenses/by/4.0/>

**Please cite the original version:**

Misitano, Giovanni, Saini, Bhupinder Singh, Afsar, Bekir, Shavazipour, Babooshka, Miettinen Kaisa. (2021). DESDEO: The Modular and Open Source Framework for Interactive Multiobjective Optimization. IEEE Access, 9, 148277-148295. <https://doi.org/10.1109/ACCESS.2021.3123825>

Received October 1, 2021, accepted October 20, 2021, date of publication October 27, 2021, date of current version November 8, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3123825

# DESDEO: The Modular and Open Source Framework for Interactive Multiobjective Optimization

G. MISITANO , B. S. SAINI , B. AFSAR , B. SHAVAZIPOUR , AND K. MIETTINEN 

Faculty of Information Technology, University of Jyväskylä, 40014 Jyväskylä, Finland

Corresponding author: G. Misitano (giovanni.a.misitano@jyu.fi)


This work was supported by the Academy of Finland under Grant 322221.

**ABSTRACT** Interactive multiobjective optimization methods incorporate preferences from a human decision maker in the optimization process iteratively. This allows the decision maker to focus on a subset of solutions, learn about the underlying trade-offs among the conflicting objective functions in the problem and adjust preferences during the solution process. Incorporating preference information allows computing only solutions that are interesting to the decision maker, decreasing computation time significantly. Thus, interactive methods have many strengths making them viable for various applications. However, there is a lack of existing software frameworks to apply and experiment with interactive methods. We fill a gap in the optimization software available and introduce DESDEO, a modular and open source Python framework for interactive multiobjective optimization. DESDEO's modular structure enables implementing new interactive methods and reusing previously implemented ones and their functionalities. Both scalarization-based and evolutionary methods are supported, and DESDEO allows hybridizing interactive methods of both types in novel ways and enables even switching the method during the solution process. Moreover, DESDEO also supports defining multiobjective optimization problems of different kinds, such as data-driven or simulation-based problems. We discuss DESDEO's modular structure in detail and demonstrate its capabilities in four carefully chosen use cases aimed at helping readers unfamiliar with DESDEO get started using it. We also give an example on how DESDEO can be extended with a graphical user interface. Overall, DESDEO offers a much-needed toolbox for researchers and practitioners to efficiently develop and apply interactive methods in new ways – both in academia and industry.

**INDEX TERMS** Data-driven multiobjective optimization, evolutionary computation, interactive methods, multi-criteria decision making, nonlinear optimization, open source software, Pareto optimization.

## I. INTRODUCTION

Optimization in many real-life problems is typically characterized by several conflicting objectives to be considered simultaneously. In these multiobjective optimization problems, the presence of conflicting objectives results in many so-called Pareto optimal solutions with different trade-offs instead of a single optimal solution. These solutions are incomparable without additional information. Therefore, there is a need for a domain expert, referred to as a decision maker (DM), to ultimately choose one of the Pareto optimal solutions as the final one based on his/her preferences.

The associate editor coordinating the review of this manuscript and approving it for publication was Huaqing Li .

Different types of methods have been developed for solving multiobjective optimization problems in the multiple criteria decision making (MCDM) (e.g., [1]–[3]) and evolutionary multiobjective optimization (EMO) (e.g., [4], [5]) communities. Most MCDM methods incorporate a DM's preferences to focus on subsets of the Pareto optimal solutions reflecting the interests of the DM. These methods have a strong theoretical background and can guarantee Pareto optimality (see, e.g., [3]). Most MCDM methods use so-called scalarization or scalarizing functions to transform the original multiobjective optimization problem with the preference information into a scalarized problem (with a single objective) to be optimized. After this transformation, an appropriate single-objective optimization method is to be used to solve the scalarized problem. By carefully selecting the scalarizing

function, one can guarantee getting a Pareto optimal solution for the original problem so that the DM's preferences are considered. With different preferences, one can typically get different Pareto optimal solutions. For comparisons of different scalarizing functions, see, e.g., [6], [7]. In contrast, EMO methods handle a population of solutions at a time and generate several approximated Pareto optimal solutions to represent different Pareto optimal solutions. They often start from a random set of solutions and use different selection, mutation and recombination operators to create the next generation of solutions. Because of their heuristic nature, they cannot guarantee Pareto optimality, but they can be applied to challenging problems with, e.g., discontinuous or nonconvex functions.

One can classify different multiobjective optimization methods based on when a DM with preference information takes part in the solution process [1], [3]. A *no preference method* is applied in absence of preferences. The DM may provide his/her preferences before or after the solution process in *a priori* or *a posteriori methods*, respectively. In *a priori* methods, the DM provides hopes and expectations, and the method tries to find the best matching solution. In contrast, a representative set of Pareto optimal solutions is generated in *a posteriori* methods for the DM to choose from.

The fourth class of methods, known as interactive multi-objective optimization methods, involves the DM during the solution process. In this way, the DM iteratively provides his/her preferences while gradually gaining further insight into the problem and learning about hidden limitations such as the feasibility of the preferences and attainable solutions [8]. Therefore, the DM has a chance to modify his/her preferences based on new insight and learning. Moreover, the cognitive load set on the DM (at a time) is usually low compared to other methods, e.g., *a posteriori* methods. Indeed, the DM can focus the search on a subset of solutions and only consider Pareto optimal solutions of interest. This also saves computational resources. Because of these reasons, we consider here interactive methods. As mentioned, they consist of iterations. At each iteration, the DM sees a solution or some solutions reflecting the provided preferences and can adjust the preferences to eventually find the most preferred solution. Thanks to learning, the confidence of the DM grows during the solution process.

The DM can provide various types of preference information. Examples of them include so-called reference points whose components represent desired values for objective functions (also called aspiration levels), ranges for acceptable objective function values, classification, pairwise comparisons and selecting desired or undesired solutions out of a subset, to name a few (see, e.g., [3], [9]).

Over the years, different interactive methods have been developed in the literature, and they have shown their potential in various applications, see, e.g., [10]. They differ from each other mainly in terms of preference information used, how solutions reflecting preferences are generated, and what kind of information is provided to the

DM [3], [8], [11]. However, their implementations are done in isolation, and they are not readily available. Even though most interactive methods utilize similar components (such as types of preference information, scalarizing functions, sampling techniques), each method has a different way of implementation. These issues slow down the practical usage of interactive methods from different perspectives and introduces various challenges, which we have listed as follows:

- 1) It is not easy to find implementations of different interactive methods to be applied.
- 2) Identifying the most suitable interactive methods to be used in various real-life applications is challenging.
- 3) Comparing interactive methods is difficult because of the lack of having various interactive methods within the same framework.
- 4) Utilizing the implemented methods or some parts of their implementation in new developments is hard, so every new construction needs to be started from scratch.
- 5) The lack of openness limits applicability.
- 6) The iterative nature of the interactive methods, together with some standard components, enables switching between methods in different iterations of the solution process, at least in theory. Nonetheless, separate implementations have been preventing the chance of testing this exciting idea.

To the best of our knowledge, only one framework has been developed for interactive methods, which, to some extent, aims to address the listed issues. It is the so-called DESDEO framework [12]. However, the version discussed in [12] had practical issues in its implementation, overall structure, and modularity and was, thus, not ready for broader usage and extensions. For these reasons, there was a need to first re-structure and then to re-implement a new DESDEO framework, which is introduced in this paper. The new framework has a clear potential in addressing all the six listed challenges.

The new DESDEO framework implemented in Python [13] has a modular structure and, thus, involves reusable modules that can be utilized for implementing new interactive methods or modifying the existing ones. DESDEO enables solving computationally expensive simulation-based and data-driven problems using surrogate models, including uncertainty considerations. It contains implementations of several old and new interactive methods by various developers covering methods of both MCDM and EMO types. Thanks to the modular structure, new or revised methods can be conveniently included in the framework.

DESDEO consists of packages and modules. We introduce them and also demonstrate how DESDEO can be applied to solve problems with analytical expressions as well as data-driven and simulation based problems. The strengths of DESDEO include the option to hybridize scalarization based and evolutionary methods and the convenience of comparing different methods in the same environment. For instance,

there is no need to specify the problem to be solved for each method separately.

The modular structure enables hybridization between different types of methods. By hybridization, we mean the ability to use final or intermediate results of one method in another method, such as generating approximated Pareto optimal solutions utilizing an EMO method and using the solutions in an MCDM method or switching the method during the solution process, e.g., when the DM wants to change the type of preference information. This opens up new opportunities for utilizing different features of various methods while the DM is not limited to using only one method or one type of preferences. Various visualizations and a graphical user interface are also being developed with a similar modular structure in mind. The methods implemented in DESDEO can be utilized by anyone who has basic programming skills in Python, which has become a widely-used programming language in data and business analytics. Since the framework is open source, it is readily available for various applications and can be conveniently tailored for different problems, if needed. It is naturally also open to new contributions and anybody interested is welcome to contribute.

The rest of the paper is structured as follows. In Section II, we outline the general concepts and notations of multiobjective optimization that we use in this paper, briefly review the related open source frameworks for multiobjective optimization in the literature, and overview some interactive methods (referred to in this paper). Section III is targeted at readers interested in contributing to the development of DESDEO. For this, the framework structure introducing packages, modules, and external dependencies is described in detail. Those who only wish to apply the framework for solving multiobjective optimization problems can skip Section III and focus on four diverse illustrative use cases outlined in Section IV. In Section IV, we also give a basic example of a graphical user interface that can be implemented to ease interaction between the interactive methods in DESDEO and the DM. In Section V, we discuss the potential of the modular framework, such as adjusting or hybridizing methods and creating user interfaces for various interactive methods. Finally, we conclude in Section VI.

## II. BACKGROUND

In this section, we first introduce the main notation and concepts used in this paper. We then survey the state-of-the-art of open source software frameworks available for multiobjective optimization. Finally, we very briefly outline some of the interactive methods referred to in the use cases considered in Section IV.

### A. MULTIOBJECTIVE OPTIMIZATION

We consider the following form of multiobjective optimization problems minimizing  $k \geq 2$  objective functions [3]:

$$\begin{aligned} \min \quad & \mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), \dots, f_k(\mathbf{x})) \\ \text{s.t.} \quad & \mathbf{x} \in S, \end{aligned} \quad (1)$$

where  $f_i: S \rightarrow \mathbb{R}$  ( $i = 1, \dots, k$ ) are objective functions and  $\mathbf{x} = (x_1, \dots, x_n)^T$  is a vector of  $n$  decision variables in the feasible region  $S \subset \mathbf{R}^n$  defined by constraint functions. Without loss of generality, we here assume that all functions are to be minimized. If some function  $f_i$  is to be maximized, it is equivalent to minimize  $-f_i$ .

A decision (variable) vector  $\mathbf{x}^* \in S$  is called Pareto optimal if there exists no  $\mathbf{x} \in S$ , so that for all  $i, f_i(\mathbf{x}) \leq f_i(\mathbf{x}^*)$  and for some  $j, f_j(\mathbf{x}) < f_j(\mathbf{x}^*)$ . The image of Pareto optimal decision vectors in the objective space  $\mathbf{R}^k$  is called a Pareto front and it consists of Pareto optimal objective vectors. In the definition of Pareto optimality, a solution is not dominated by any other feasible solution. As we deal with evolutionary methods that cannot guarantee Pareto optimality, we also use the term nondominated solutions. They are not dominated by any solution in the solution set considered (typically referred to as a population), but are not necessarily Pareto optimal.

The best and the worst possible values of objective functions in the Pareto front are represented by an ideal and a nadir point, respectively. The components of the ideal point can be calculated by optimizing each objective function subject to  $S$  as a single-objective optimization problem. In contrast, computing the nadir point is difficult in practice as the set of all Pareto optimal solutions is unknown. However, some methods (e.g., a payoff table [14]) are available that can approximate the nadir point (see e.g., [3] and references therein).

### B. DATA-DRIVEN MULTIOBJECTIVE OPTIMIZATION

As mentioned in the introduction, DESDEO can be applied to solve different types of multiobjective optimization problems. Typically, the analytical forms of objective functions and constraints cannot be formulated in most real-life problems. In some cases, simulation models can be used to evaluate function values. In other cases, the objective or constraints values must be gained from some real experiences (or laboratory experiments). In either case, evaluating the function values is usually expensive from different perspectives. Therefore, so-called surrogate models can be utilized instead of the original expensive models or experiments.

On the other hand, in today's digital societies, various data from different sources are continuously recorded, which can be used as a new source of information in decision making. Making the most of the data available can lead to data-driven optimization problems. In this case, no other information than the data is available, giving no other option than fitting surrogate models to formulate functions for optimization problems based on the data. Then, surrogate models approximate the objective or constraint values.

Different types of surrogate models, such as probabilistic (e.g., Bayesian network [15] and Markov chain Monte Carlo) or machine learning techniques (e.g., radial basis functions [16], Kriging or Gaussian processes [17], [18], support vector regression [19], and neural networks [20], [21]) exist and can be utilized to derive functions for multiobjective optimization problems. Most of these techniques are freely

available in different Python packages and libraries, which can be used within the DESDEO framework.

### C. LITERATURE REVIEW ON OPEN SOURCE FRAMEWORKS FOR MULTIOBJECTIVE OPTIMIZATION

We have surveyed open source frameworks for multiobjective optimization problems. We do not consider closed source and commercial software implementations because they do not provide an opportunity to adjust the methods to one's needs in the way open source software does. Several open source software frameworks have been proposed in the literature. Each of them has its own strengths and limitations and differs in some nuances from the others. In general, many aspects should be considered when selecting an appropriate framework for one's needs. For example, familiarity with the programming language used to implement the framework, the characteristics of the problem to be solved, the availability of visualization tools, and an exemplary user interface can influence the selection of a framework.

Table 1 summarizes well-known open source frameworks proposed for solving multiobjective optimization problems. We also list some common frameworks with a modular structure, where multiobjective optimization methods can be created. Besides the name and the programming language used, the table lists whether the frameworks focus on multiobjective optimization, include MCDM or EMO types of methods, provide a decision-making mechanism where a DM can provide his/her preference information and choose the most preferred solution, visualization tools, and a user interface. The table also states whether the framework has a modular structure or not. In the following, we briefly describe each of the frameworks.

DEAP [22] and Inspyred [23] do not focus specifically on multiobjective optimization but provide Python implementations of e.g., genetic algorithms, simulated annealing, and differential evolution. The (a posteriori) EMO method NSGA-II for multiobjective optimization is also included. Since these two frameworks have been developed with a modular structure, more multiobjective optimization methods can be developed by using the modules available in the framework. Inspyred includes further nature-inspired optimization algorithms such as particle swarm optimization and ant colony optimization.

vOptSolver [24] has been implemented in the Julia language. It integrates several exact algorithms for multiobjective linear optimization problems (including mixed-integer problems).

Platypus [25] involves Python implementations of several well-known EMO methods concentrating, thus, on multiobjective optimization. It also includes an analysis tool for visual comparison of EMO methods by applying some performance indicators.

MOEA [26] is a Java-based framework that enables automatic parallelization of methods across multiple processor

cores. It includes most of the state-of-the-art a posteriori EMO methods.

PyGMO [27] is a Python extension of PaGMO (C++) [28] which has implementations of a variety of single- and multiobjective optimization methods and real-life engineering problems in an object-oriented architecture. Automatic parallelization of the implemented methods enables using the underlying multicore architecture efficiently.

jMetalPy [29] extends the Java-based framework jMetal [30] (which contains metaheuristic methods like evolutionary methods) for multiobjective optimization to be used in Python. jMetalPy provides improved data analysis, interactive visualization of Pareto optimal solutions, and increased computational performance by applying libraries available in Python. Additionally, jMetalPy facilitates parallel computing for computationally expensive problems.

Pymoo [31] is a multiobjective optimization framework in Python and offers evolutionary methods for single- and multiobjective optimization problems. It involves several visualization techniques for illustrating results and well-known indicators to compare the performance of the methods.

Finally, PlatEMO [32] is an open source framework developed in MATLAB including many EMO methods, widely used performance indicators, and benchmark problems. It also has a graphical user interface. However, one should note that even though the implementation is openly available, a MATLAB license is required to use it. Therefore, while being commercial software, PlatEMO still allows adjusting its implementation to meet specific needs.

The frameworks mentioned so far do not contain interactive methods. They include either MCDM or EMO types of methods, but not both, and only one of the frameworks comes with a user interface. As this summary shows, overall, DESDEO is unique since it is the only open source framework including interactive methods. Thus, DESDEO fills a gap in the software available in the multiobjective optimization community. DESDEO has a clear modular structure making it easy for users and developers to contribute new contents. Importantly, DESDEO involves both MCDM and EMO types of methods, enabling hybridizing and switching between methods depending on needs and application areas. Moreover, elements for building custom graphical user interfaces for efficient interaction between the DM and interactive methods are a planned future inclusion in DESDEO. These elements are currently under active development and are to be included as additional packages in DESDEO eventually. Therefore, visualization and user interface (UI) items for DESDEO are in parentheses in Table 1 for the time being. However, specialized non-modular graphical user interfaces have been developed for DESDEO in the past as seen in Section IV-F.

### D. SOME INTERACTIVE METHODS IMPLEMENTED

As mentioned earlier, different interactive multiobjective optimization methods have been implemented in DESDEO. In this section, we briefly introduce a few that are utilized

**TABLE 1.** Summary of open source optimization frameworks. In the table, MO stands for multiobjective optimization.

Name	Programming language	MO focus	Method type		Decision making	Interactive methods	Visualization	UI	Modularity
			EMO	MCDM					
ine DEAP	<i>Python</i>								✓
Inspyred	<i>Python</i>								✓
vOptSolver	<i>Julia</i>	✓		✓					
Platypus	<i>Python</i>	✓	✓				✓		✓
MOEA	<i>Java</i>	✓	✓				✓		✓
PaGMO/PyGMO	<i>C++/Python</i>	✓	✓				✓		✓
jMetal/jMetalPy	<i>Java/Python</i>	✓	✓		✓		✓		✓
Pymoo	<i>Python</i>	✓	✓		✓		✓		✓
PlatEMO	<i>Matlab</i>	✓	✓		✓		✓	✓	✓
<b>DESDEO</b>	<i>Python</i>	✓	✓	✓	✓	✓	(✓)	(✓)	✓

later in Section IV: the reference point method [33], the synchronous NIMBUS method [34] and the NAUTILUS family [35] (particularly E-NAUTILUS [36]) from MCDM methods, and RVEA [37] and NSGA-III [38] from EMO methods.

The reference point method [33] is a popular interactive multiobjective optimization method in which the DM provides preferences as desired objective function values constituting a reference point. Then, at each iteration,  $k + 1$  Pareto optimal solutions reflecting the reference point are found by utilizing an achievement scalarizing function. The DM can iterate (i.e., compare solutions and provide new reference points) until the most preferred solution is found.

In NIMBUS, starting from a Pareto optimal solution, a DM expresses his/her preferences by classifying the objective functions corresponding to the Pareto optimal solution into up to five preference classes to indicate how the current objectives should change to be more preferable to the DM. In each iteration of NIMBUS, based on the DM's preferences, 1–4 Pareto optimal solutions are generated and shown to the DM (the DM decides how many new solutions (s)he wants to see). Besides classification, the DM can ask for the desired number of solutions generated between any two Pareto optimal ones. Like other interactive methods, the solution process continues until the DM has found his/her most preferred solution.

The NAUTILUS family [35] contains interactive trade-off-free methods. This means that the DM does not deal with Pareto optimal solutions but gradually approaches the Pareto front starting from an inferior solution (like a nadir point). Then, following the DM's preferences, all objectives are simultaneously improved until a Pareto optimal solution is reached. During the solution process, the ranges of objective function values that still can be reached without trading-off naturally shrink. Once a Pareto optimal solution is reached, the solution process stops since it is no longer possible to proceed without trading-off. NAUTILUS variants vary regarding the types of preference information used and how solutions are generated in each iteration (see [35] for a comparison of the differences). For example, in each iteration of the original NAUTILUS [39] method, the DM ranks the objective

functions based on the preferred improvement of the current objective values. In contrast, in NAUTILUS 2 [40], ratios of improvement are provided by the DM. In E-NAUTILUS [36], which is particularly developed for handling computationally expensive problems, the DM can compare multiple solutions (referred to as *intermediate points*) at each iteration. Finally, NAUTILUS Navigator [41] integrates NAUTILUS with navigation ideas [42], where the DM sees ranges of objective function values that are still reachable from the current iteration point shrinking in real-time and provides preferences as desired aspiration levels and bounds not to be exceeded.

Besides MCDM type of methods, various interactive EMO methods have also been developed and implemented in DESDEO. They include interactive versions [43] of the reference vector-guided evolutionary algorithm (RVEA) [37] and NSGA-III [38]. RVEA and NSGA-III are originally a posteriori methods. The interactive version of NSGA-III has been implemented, corresponding to how RVEA was made interactive in [43]. The main type of preference information used in both is a reference point, but other preference types are also available for RVEA.

### III. STRUCTURE OF THE DESDEO FRAMEWORK

In this section, we describe the structure of the DESDEO framework, including packages of the framework and the modules in each package. In addition, we discuss the purpose of each package and its dependencies. We also consider the implementation of the DESDEO framework and its external dependencies. Lastly, we discuss the architectural choices made in DESDEO that any aspiring developer and user of the frameworks should be aware of. This section is intended mostly for those interested in contributing to the framework's development. Those interested only in utilizing the framework for solving multiobjective optimization problems may proceed to Section IV.

#### A. PACKAGES AND MODULES

In the modular structure of DESDEO, each package is a collection of modules, which contain class and function definitions to tackle specific tasks in modeling and solving multiobjective optimization problems interactively. The main

packages, called *core packages*, and their individual *modules* are presented in Figure 1. Each package has a well-defined purpose and is built to address a certain set of tasks in interactive multiobjective optimization methods. The modules may depend on other packages lower in the structure, as shown in Figure 2.

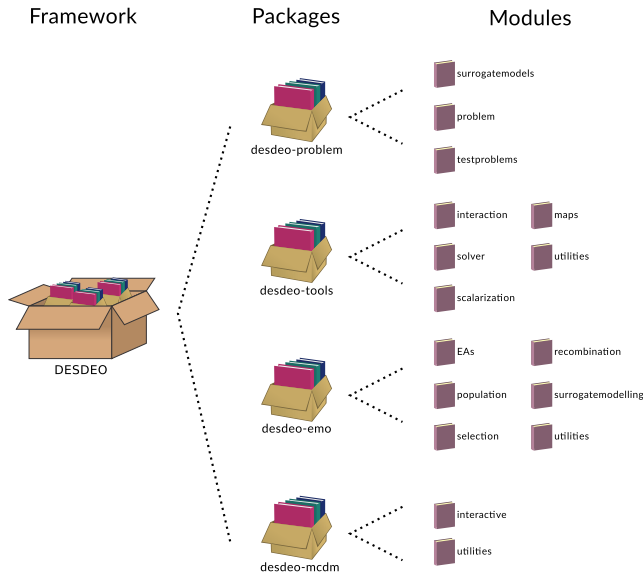


FIGURE 1. The main structure of the DESDEO framework with packages and modules included in each package. Further packages and modules can be added as needed.

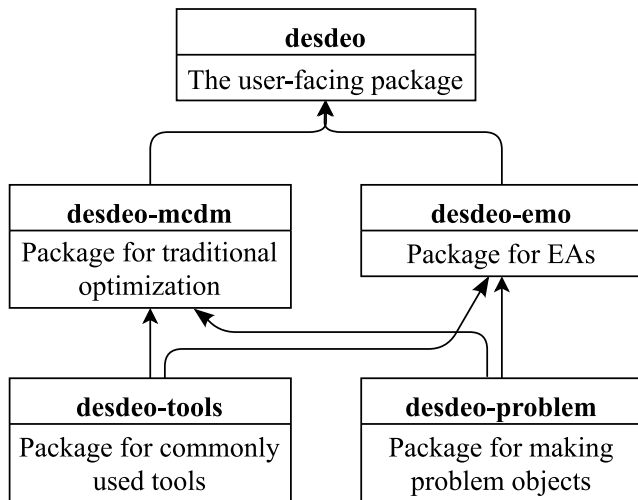


FIGURE 2. The packages of DESDEO and their dependencies on each other.

In Figure 2, the arrows represent the internal dependencies of packages in DESDEO, e.g., the package *desdeo-emo* depends on both the packages *desdeo-tools* and *desdeo-problem*. A modular structure allows users to choose which parts of the framework to use. For example, to model a multiobjective optimization problem, one can use the *desdeo-problem* package and avoid the needless inclusion of the other

packages. Additionally, having the framework structured in a modular fashion eases the development of the framework by encapsulating features and functionalities related to interactive multiobjective optimization in their own respective packages.

In what follows, we describe packages included in DESDEO and their dependencies on other packages. The packages also depend on existing popular Python packages, which are discussed further at the end of this section.

The *desdeo-problem* package contains features related to the formulation and modeling of multiobjective optimization problems. Problems can be analytical expressions of functions depending on decision variables, or modeled based on collected data related to the multiobjective optimization problem (either utilizing data available or data obtained by running a problem-specific simulator). The problem can naturally also have constraint functions defining a feasible region. Tools for problem formulation can be found in the module *problem*. As already mentioned, surrogate models may be trained and used to model functions of a multiobjective optimization problem based on data. For example, Gaussian regression is available as a surrogate model but any other machine learning-focused package can be used to train surrogate models. The tools for building surrogates can be found in the *surrogatemodels* module. Moreover, commonly utilized test problems in multiobjective optimization can be found in the *testproblems* module. Such problems include, for example, the DTLZ problems [44]. The *desdeo-problem* package does not depend on any other package in the DESDEO framework.

The *desdeo-tools* package contains utility tools that are expected to be used during any phase of the optimization process, irrespective of the method type (MCDM or EMO) used for optimization. Such tools include abstractions for various preference elicitation techniques, scalarizing functions, and nondominated sorting. The *interaction* module contains methods to ease interaction between a DM and an interactive multiobjective optimization method. The *scalarization* module contains scalarization tools for transforming multiobjective optimization problems into single-objective problems (incorporating preference information). As mentioned in the introduction, we can get Pareto optimal solutions by using appropriate scalarization functions, such as achievement scalarizing functions [45] and the scalarization function of the  $\epsilon$ -constraint method [2]. The *maps* module contains tools for transforming objectives from one space to another, such as e.g., the so-called preference incorporated space [46]. Finally, the *solver* module contains tools for solving scalarized problems. These solvers must be appropriate for the characteristics of the problem in question (considering, e.g., the type of variables and the nature of functions involved). The *desdeo-tools* package does not depend on any other package in the DESDEO framework.

The *desdeo-emo* package is the repository of evolutionary algorithms (EAs) and tools which are specifically used with EMO methods. Besides interactive EMO methods, it has

implementations of basic (a posteriori) EMO methods and some a priori methods as they can be used as elements of interactive ones. The package contains the following modules: *population*, *recombination*, *selection*, *EAs*, *surrogatemodelling*, and *utilities*. The first three modules contain abstractions representing the population, crossover and mutation operators as well as selection operators. We use these abstractions as building blocks to implement various evolutionary algorithms in the *EAs* module. New EAs can be implemented by either modifying the implementations in the *EAs* module or by using the building blocks in other modules in entirely new ways. The *surrogatemodelling* module implements certain EA based methods which are specifically designed to train surrogate models. Finally, *utilities* contains miscellaneous tools that are used by one or more EMO methods, but do not fit in the other modules. The *desdeo-emo* package depends on the *desdeo-problem* and *desdeo-tools* packages.

As the name suggests, the *desdeo-mcdm* package contains implementations of interactive multiobjective optimization methods of the MCDM type (involving scalarization functions to generate Pareto optimal solutions). The methods themselves are in the *interactive* module. For example, the synchronous NIMBUS and methods belonging to the NAUTILUS family are implemented in this module. The *utilities* module contains various utilities often needed in MCDM methods. For instance, the utilities include a payoff table method for computing an ideal and an approximation of the nadir point. The *desdeo-mcdm* package depends on the *desdeo-problem* and *desdeo-tools* packages.

Besides the core packages of DESDEO discussed so far, other packages can be, and have also been, developed based on the packages discussed. Examples of these packages consist of specialized graphical UIs and new experimental interactive multiobjective optimization methods not mature enough to be included in DESDEO yet. Due to their experimental nature, we will not discuss these additional packages further here.

As mentioned, the DESDEO framework has been implemented in Python and is available online as open source software on GitHub.<sup>1</sup> The framework makes use of existing Python libraries in the SciPy ecosystem, most notably NumPy [47], SciPy (the library) [48] and Pandas [49]. NumPy offers numerically efficient data structures, which enables an efficient handling of array-like structures present everywhere in the DESDEO framework. SciPy offers existing computational routines. For example, it offers excellent optimization routines for optimizing constrained problems with a single objective. As mentioned, this kind of problem emerges, for instance, when scalarizing a multiobjective optimization problem. In turn, Pandas has excellent and efficient data manipulation routines. They are needed especially when representing data-driven multiobjective optimization problems, which may sometimes consist of large amounts of data

requiring extensive feature engineering before modeling a multiobjective optimization problem.

For a more detailed description of each package and module found in DESDEO, the reader is encouraged to check DESDEO's main documentation. The documentation is found online (<https://desdeo.readthedocs.io/en/latest/>) where the individual documentation of each core package can be found with additional details about implemented classes and functions.

## B. ARCHITECTURAL DECISIONS IN DESDEO

A couple of choices have been made during the development of DESDEO. The user of the framework should keep them in mind while developing or using the framework.

As mentioned in Section II, objective functions in multiobjective optimization problems can either be minimized or maximized, but within the optimization methods in DESDEO, functions are always assumed to be minimized. This means that we convert functions to be maximized to functions to be minimized and internally only deal with minimization problems. This choice has been made to remove any possible software bugs, confusion, and guesswork related to keeping track whether an objective is to be minimized or maximized, transforming problems from one type to another, and parsing preference information. Naturally, when displaying information related to a multiobjective optimization problem and its solutions to a DM, the objectives are presented in their original form. The task of making the conversion whenever needed (also in the preference information) is the responsibility of the UI.

As interactive multiobjective optimization methods vary in the type of interaction and preference information required from the DM, abstraction of interaction has been kept simple and non-restrictive in DESDEO. Each interactive method has at least two (object) methods: *start* and *iterate*. As the name suggests, the former is always used to start a method after it has been instantiated. Likewise, the *iterate* method is then used for any subsequent interactions after starting the method. Both the *start* and *iterate* methods return at least one *request* (Python) object. These objects contain all the necessary information to carry out a required interaction with the interactive method in their *content* attribute, which is a Python dictionary. The contents of a *request* may vary depending on the interactive method, but each *content* dictionary in DESDEO comes at least with a *message* entry meant to give a hint to the user of what is expected of them interaction-wise. Each *request* object has a *response* attribute, which is also a dictionary. The *response* dictionary has its own entries, which the user must define to continue iterating the interactive method. After the entries of the *response* have been defined, the *iterate* method can be invoked by giving it the *request* containing the *response* with defined entries as an argument. The *iterate* method will then return a new *request*. Examples of this *request-response* structure can be found in the use cases in Section IV. However,

<sup>1</sup><https://github.com/industrial-optimization-group/DESDEO>



it is not expected that a DM oneself would directly handle these Python dictionaries. Instead, it is expected that some external interface is used to facilitate interaction between a DM and DESDEO. The `request` and `response` should be mainly used to store and communicate information to and from interactive methods available in DESDEO.

#### IV. USE CASES

In this section, we demonstrate how one can use the DESDEO framework to define different types of problems and solve them by applying interactive multiobjective optimization methods. For simplicity, we use a river pollution problem with five objective functions and two decision variables presented in Section IV-A. In Section IV-B, we describe how to define a problem with an analytical formulation and solve it using the synchronous NIMBUS method [34]. This method is in the *desdeo-mcdm* package and incorporates classification types of preferences. Section IV-C is devoted to defining and solving a data-driven problem. The interactive RVEA [43] method in the *desdeo-emo* package is applied, where preference information is given as a reference point. In Section IV-D, we consider some challenges of computationally expensive problems and follow the three-stage approach [50], where first, NSGA-III is utilized in a pre-decision-making stage to generate nondominated solutions. Then, a computationally inexpensive surrogate problem is formed. In the decision-making stage, the DM applies the interactive E-NAUTILUS [36] method to solve the surrogate problem. There can also be a post-decision-making stage to assure the Pareto optimality of the final solution. Here we consider the first two stages as a hybrid way of using methods within the DESDEO framework. Lastly, in Section IV-E we demonstrate how the DM can switch interactive methods in DESDEO to express his/her preferences in different ways. This example also illustrates some of the advantages the DESDEO environment provides. Finally, we discuss some graphical user interfaces in Section IV-F.

As mentioned, the main documentation of DESDEO can be found online (<https://desdeo.readthedocs.io/en/latest/>). The documentation of each core package discussed in Section III, can be readily accessed through the main documentation. We advice the reader to check the documentation for any additional details related to the use cases considered in Sections IV-B, IV-C, IV-D, and IV-E. The examples shown in these sections can also be found online in a Jupyter Notebook.<sup>2</sup>

##### A. THE RIVER POLLUTION PROBLEM

The river pollution problem [51] considers a river close to a city. There are two sources of pollution: industrial pollution from a fishery and municipal waste from the city and two treatment plants (in the fishery and the city). The pollution is reported in pounds of biochemical oxygen demanding

<sup>2</sup>[https://desdeo.readthedocs.io/en/latest/notebooks/four\\_simple\\_use\\_cases.html](https://desdeo.readthedocs.io/en/latest/notebooks/four_simple_use_cases.html)

material (BOD), and water quality is measured in dissolved oxygen concentration (DO).

Cleaning water in the city increases tax rate, and cleaning in the fishery reduces the return on investment. The problem is to improve the DO level in the city and at the municipality border ( $f_1$  and  $f_2$ , respectively) while, at the same time, maximizing the percent return on investment at the fishery ( $f_3$ ) and minimizing addition to the city tax ( $f_4$ ). We consider a variant of the problem [52] with one more objective to ensure the treatment plants' efficiency by keeping the proportional amount of BOD removed from the water close to the ideal value of 0.65 ( $f_5$ ). The corresponding multiobjective optimization problem where all objectives have been converted to be minimized is as follows:

$$\begin{aligned}
 \min \quad & f_1(\mathbf{x}) = -4.07 - 2.27x_1 \\
 \min \quad & f_2(\mathbf{x}) = -2.60 - 0.03x_1 - 0.02x_2 \\
 & \quad \quad \quad - \frac{0.01}{1.39 - x_1^2} - \frac{0.30}{1.39 - x_2^2} \\
 \min \quad & f_3(\mathbf{x}) = -8.21 + \frac{0.71}{1.09 - x_1^2} \\
 \min \quad & f_4(\mathbf{x}) = -0.96 + \frac{0.96}{1.09 - x_2^2} \\
 \min \quad & f_5(\mathbf{x}) = \max\{|x_1 - 0.65|, |x_2 - 0.65|\} \\
 \text{s.t.} \quad & 0.3 \leq x_1, x_2 \leq 1.0,
 \end{aligned} \tag{2}$$

where the proportional amounts of BOD removed from water in the two treatment plants are, respectively, the decision variables  $x_1$  and  $x_2$ .

##### B. USE CASE 1: PROBLEM WITH AN ANALYTICAL FORMULATION

DESDEO has good support for defining and optimizing problems with analytical formulations. DESDEO provides individual classes to define components of problem (1), i.e., objective functions, variables, and constraint functions separately. Box-constraints for variables are also supported.

Here we analytically define (2) and use modules of the *desdeo-problem* package and NumPy. The imports needed are shown in Source code 1. Notice that this problem has only box-constraints.

```

1 import numpy as np
2
3 from desdeo_problem import MOProblem, Variable,
  ↳ ScalarObjective

```

**SOURCE CODE 1.** Needed imports for a problem defined analytically. The class `MOProblem` is used to define a problem, the class `Variable` its decision variables, and the class `ScalarObjective` the objective functions.

We define the five objective functions as shown in Source code 2 as individual functions. These functions are expected to return a 1-dimensional NumPy array with each element representing the respective objective value when evaluated with one or more decision variable vectors. These decision

variable vectors are stored in 2-dimensional NumPy arrays, with each row representing a single vector. The defined functions are then used in the `ScalarObjective` class to instantiate new objects. Finally, each of the objects is stored in a list.

```

1 def f_1(x: np.ndarray) -> np.ndarray:
2     x = np.atleast_2d(x) # This step is to guarantee that the
   ↪ function works when called with a single decision variable vector as
   ↪ well.
3     return -4.07 - 2.27*x[:, 0]
4
5 def f_2(x: np.ndarray) -> np.ndarray:
6     x = np.atleast_2d(x)
7     return -2.60 - 0.03*x[:, 0] - 0.02*x[:, 1] - 0.01
   ↪ / (1.39 - x[:, 0]**2) - 0.30 / (1.39 + x[:,
   ↪ 1]**2)
8
9 def f_3(x: np.ndarray) -> np.ndarray:
10    x = np.atleast_2d(x)
11    return -8.21 + 0.71 / (1.09 - x[:, 0]**2)
12
13 def f_4(x: np.ndarray) -> np.ndarray:
14    x = np.atleast_2d(x)
15    return -0.96 - 0.96 / (1.09 - x[:, 1]**2)
16
17 def f_5(x: np.ndarray) -> np.ndarray:
18    return np.max([np.abs(x[:, 0] - 0.65),
   ↪ np.abs(x[:, 1] - 0.65)], axis=0)
19
20 objective_1 = ScalarObjective(name="f_1",
   ↪ evaluator=f_1)
21 objective_2 = ScalarObjective(name="f_2",
   ↪ evaluator=f_2)
22 objective_3 = ScalarObjective(name="f_3",
   ↪ evaluator=f_3)
23 objective_4 = ScalarObjective(name="f_4",
   ↪ evaluator=f_4)
24 objective_5 = ScalarObjective(name="f_5",
   ↪ evaluator=f_5)
25
26 objectives = [objective_1, objective_2, objective_3,
   ↪ objective_4, objective_5]

```

**SOURCE CODE 2.** Defining the objective functions of problem (2).

The variables of the problem are defined similarly in Source code 3. Each `Variable` object is instantiated by providing the variable's name, initial value, and lower and upper bound (i.e., box-constraints). The objects are then stored in a list.

```

1 x_1 = Variable("x_1", 0.5, 0.3, 1.0)
2 x_2 = Variable("x_2", 0.5, 0.3, 1.0)
3
4 variables = [x_1, x_2]

```

**SOURCE CODE 3.** Defining the variables and their bounds for problem (2).

Finally, we define the multiobjective optimization problem by instantiating an `MOPProblem` object in Source code 4 using the lists of `ScalarObjectives` and `Variables` defined earlier. If the problem had additional constraints, they would be defined in a similar way to objective functions and provided as a third argument (`constraints`) to the initialization method of the `MOPProblem` class. However, here we only have box-constraints, which were accounted for when defining the variables in Source code 3.

As mentioned, we solve problem (2) in this case with the synchronous NIMBUS method [34]. Since it needs the ideal

```

1 mo_problem = MOPProblem(variables=variables,
   ↪ objectives=objectives)

```

**SOURCE CODE 4.** Defining the multiobjective optimization problem object of problem (2).

and nadir points, we approximate them with the payoff table method found in the *desdeo-mcdm* package's *utilities* module in Source code 5. We store them inside the object defining our problem to have easy access to them later.

```

1 from desdeo_mcdm.utilities import payoff_table_method
2
3 ideal, nadir = payoff_table_method(mo_problem)
4
5 mo_problem.ideal = ideal
6 mo_problem.nadir = nadir

```

**SOURCE CODE 5.** Applying the payoff table method to get approximations of the ideal and nadir points.

```

1 from desdeo_mcdm.interactive.NIMBUS import NIMBUS
2
3 nimbus = NIMBUS(mo_problem)
4
5 classification_request, _ = nimbus.start()

```

**SOURCE CODE 6.** Instantiating a NIMBUS object and invoking its `start` method. The `start` method returns two requests of which the second one is irrelevant to this example and is therefore matched to an underscore on line 5.

We can now start solving problem (2) using NIMBUS as shown in Source code 6. After importing the NIMBUS class, we instantiate an object of it by providing it `mo_problem`, which was defined earlier. The `start` method returns a `classification_request`, which is used to interact with the method as described in Section III-B. The message-entry found in the `content` attribute of `classification_request` is printed in Console 1. We remind the reader that in practice, a UI should handle requests. An example of such can be found in Section IV-F.

```

>>>print(classification_request.content["message"])
Please classify each of the objective values in one of
↪ the following categories:
1. values should improve '<'
2. values should improve until some desired
   ↪ aspiration level is reached '<='
3. values with an acceptable level '='
4. values which may be impaired until some upper
   ↪ bound is reached '>='
5. values which are free to change '0'
Provide the aspiration levels and upper bounds as a
↪ vector. For categories 1, 3,
and 5, the value in the vector at the objective's
↪ position is ignored. Supply also
the number of maximum solutions to be generated.

```

**CONSOLE 1.** The message printed in the `request` returned by NIMBUS.

As seen in Console 1, we have been provided with instructions on how to proceed. The content of the `response`,

```
>>>print(
... classification_request.content["objective_values"]
objective values [ -5.746, -2.779, -6.906, -11.626,
↳ 0.349 ]
>>>print("ideal", ideal)
ideal [ -6.339, -2.864, -7.499, -11.626, 0 ]
>>>print("nadir", nadir)
nadir [ -4.751, -2.767, -0.321, -1.920, 0.349 ]
```

**CONSOLE 2.** The objective values of the initial solution computed by NIMBUS, and the ideal and nadir points of the problem.

in the case of NIMBUS, also contains objective vectors, which we can inspect by printing them as done in Console 2.

```
1 response = {
2   "classifications": ["<=", "0", "=", ">=", "<"],
3   "levels": [-6.2, 0, 0, -3.0, 0],
4   "number_of_solutions": 2,
5 }
6
7 classification_request.response = response
8
9 save_request, _ =
↳ nimbus.iterate(classification_request)
```

**SOURCECODE 7.** Defining a response with preference information required by NIMBUS, and then iterating. Newly computed objective vectors are then printed.

We then define a response in Source code 7 containing preference information, in this case, classifications, and continue iterating by invoking the `iterate` method of NIMBUS. The new objective vectors computed in the first iteration of NIMBUS are shown in Console 3.

```
>>>print("objective vectors",
↳ save_request.content["objectives"])
objective vectors [
array([ -5.746, -2.799, -6.906, -3.000, 0.137 ]),
array([ -5.545, -2.808, -7.146, -2.398, 5.508 ])]
```

**CONSOLE 3.** The objective vectors computed by NIMBUS based on the classification given by the DM.

Iterations of the NIMBUS method may continue by defining new responses to the requests returned by subsequent invocations of the `iterate` method. According to the definition of NIMBUS [34] the subsequent requests can also prompt the DM to choose previously computed solutions between which to compute additional solutions, or to select previously computed solutions to be saved into an archive for later viewing, for example. How each of the requests is handled in practice and how information is displayed to the DM depends on the choice of a UI, as stated before. Here, we have only shown the information in textual format to showcase the concepts of requests and responses, which can be found in other interactive methods defined in DESDEO as well.

### C. USE CASE 2: DATA-DRIVEN PROBLEM

As mentioned, the DESDEO framework can be used to solve data-driven problems by fitting surrogate models to the data.

This means that the data is assumed to contain samples of decision variable values, and corresponding objective vectors and surrogate models are fitted to represent each objective function individually. To demonstrate this, in this use case, we assume that we only have access to a small number of data points generated before the initiation of the solution process. We have generated data points for problem (2) by sampling the feasible region in the decision space using Latin hypercube sampling [53], and evaluated them using the analytical functions to obtain the corresponding objective vectors. A total of 100 points were sampled and the resulting data set saved on disk. The structure of the dataset is shown in Table 2, where the first row contains the names of the columns (decision variables or objectives).

**TABLE 2.** Format of the raw data used for surrogate-assisted optimization.

x_1	x_2	f_1	f_2	f_3	f_4	f_5
0.8211	0.7949	-5.9339	-3.0502	-6.5023	-3.0557	0.1711
0.7328	0.4363	-5.7336	-2.8925	-6.9258	-2.0271	0.2136
...	...	...	...	...	...	...

```
1 import pandas as pd
2 from desdeo_problem.problem import DataProblem
3
4
5 training_data =
↳ pd.read_csv("./data/River_pollution.csv",
↳ comment="#")
6
7 problem = DataProblem(
8   data=training_data,
9   variable_names=["x_1", "x_2"],
10  objective_names=["f_1", "f_2", "f_3", "f_4",
↳ "f_5"],
11  bounds=pd.DataFrame(
12    [[0.3, 0.3], [1.0, 1.0]],
13    columns=["x_1", "x_2"],
14    index=["lower_bound", "upper_bound"]),
15 )
```

**SOURCECODE 8.** Formulating the problem using data.

We formulate the problem as shown in Source code 8. The Pandas package is used for importing and handling the data as shown in line 5 with the variable `training_data`. We can now define the problem by instantiating a `DataProblem` object. This is done by passing training data, names of the decision variables and the objective functions, and the lower and upper bounds of the decision variables. If these bounds are not provided, the infimum and supremum of the dataset are assumed to be the bounds.

In Source code 9, we show how the newly created `DataProblem` object can be used to train surrogate models for the objectives. We begin by importing the surrogate modeling technique of choice. Here, we use the `GaussianProcessRegressor` class from `desdeo_problem`, which is a wrapper around the `scikit-learn` class of the same name. Similar wrappers can be defined for other options of existing surrogate models. The modeling

```

1 from desdeo_problem.surrogatemodels.SurrogateModels
  ↪ import GaussianProcessRegressor
2
3 problem.train(
4     models=GaussianProcessRegressor,
5     model_parameters={"optimizer": "fmin_l_bfgs_b"}
6 )

```

**SOURCECODE 9.** Training Gaussian process regression surrogate models for the objectives.

algorithm and model parameters can be passed to the `train` method of the `DataProblem` object, which automatically trains the surrogate models for all objectives. Details about advanced use cases of the `train` method, such as training different kinds of surrogate models for different objectives, can be found in the documentation of *desdeo-emo*. More information about the model parameters is available in the documentation of the package of the model used, in this case, `scikit-learn`.

Once the surrogate models have been trained, we apply here the interactive RVEA method from the *desdeo-emo* package to solve the resulting problem using reference points as preference information. In Source code 10, we pass the `problem` variable as the first argument to the RVEA instance. This is followed by two Boolean arguments: `interact=True` and `use_surrogates=True`. The first argument enables the use of an interactive version of RVEA as presented in [43]. The second argument enables RVEA to use the surrogate models as objectives in place of analytical functions. Details about other arguments which control various aspects of the evolutionary method can be found in the documentation of *desdeo-emo*.

```

1 from desdeo_emo.EAs import RVEA
2
3 evolver = RVEA(
4     problem,
5     interact=True,
6     use_surrogates=True
7 )
8
9 (_, _, refp_request, _), _ = evolver.requests()
10
11 # references given to refp_request
12
13 (_, _, refp_request, _), _ =
  ↪ evolver.iterate(refp_request)

```

**SOURCECODE 10.** Using interactive RVEA to solve the surrogate problem.

We begin the interactive solution process by providing the first reference point to `evolver`. This is done by first calling the `request` method of `evolver`, which returns `refp_request` and additional requests, irrelevant in this example, matched to underscores. This is similar to the requests returned by the `start` method of `nimbus` in the previous subsection. The `refp_request` variable accepts preferences as a reference point. Similar to `classification_request` in the previous subsection, this object also has a `content` method and a `response` attribute. The comment on line 11 in Source code 10

signifies the DM providing preferences to `refp_request`. As mentioned in the previous subsection, this can be achieved by a command-line interface, a graphical UI, or by using a console environment, like IPython or Jupyter Notebook.

In Console 4, we show how preferences can be defined. The `content` attribute of `refp_request` can be shown to the DM to describe the acceptable ranges of the preferences (here, ranges for aspiration levels as components of the reference point) and how the preference information will be utilized in the method used. The DM then provides preferences to the `response` attribute of `refp_request` using a Pandas data frame. The names of the columns of this data frame have to be the same as the objective function names, and the values contained in the data frame reflect the preferences of the DM in the form of a reference point. The preference information can then be submitted to the `iterate` method of the `evolver` object to run one iteration of interactive RVEA. This involves running the evolutionary method for a number of generations. This number can be changed by the user using arguments of the RVEA class, and the details can be found in the documentation.

```

>>>print (refp_request.content["message"])
Provide a reference point worse than or equal to the
↪ ideal point:
f_1      -6.34
f_2     -3.4391
f_3     -7.69844
f_4    -11.1186
f_5     0.0495768
Name: ideal, dtype: object
The reference point will be used to focus the reference
↪ vectors towards the preferred region.
If a reference point is not provided, the previous state
↪ of the reference vectors is used.
If the reference point is the same as the ideal point,
↪ the reference vectors are spread uniformly in the
↪ objective space.

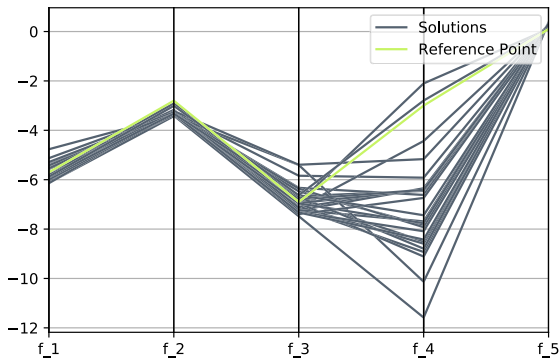
>>>refp_request.response = pd.DataFrame(
...     [[-5.7, -2.8, -6.9, -3.0, 0.1]],
...     columns=["f_1", "f_2", "f_3", "f_4", "f_5"])

>>>requests, _ = evolver.iterate(refp_request)

```

**CONSOLE 4.** Checking the contents of the preference request object and saving the DM's preferences using a console environment.

After each iteration, the solutions generated can be accessed through the `individuals` and `objectives` attributes of `evolver.population`. The former contains the decision variable vectors of the set of solutions, whereas the latter contains the corresponding set of objective vectors. The solutions received after one iteration of RVEA are shown in Figure 3 in the parallel coordinates plot. As can be seen, many solutions were found that follow the reference point of the DM (denoted in green color) closely. If, however, the DM is not satisfied with the results or wants to see solutions in a different region of the objective space, the steps shown in Console 4 can be repeated as many times as desired with different preference information.



**FIGURE 3.** Solutions obtained for the data-driven river pollution problem after using RVEA for one iteration.

#### D. USE CASE 3: COMPUTATIONALLY EXPENSIVE PROBLEM

Multiobjective optimization problems can involve expensive function evaluations. In such cases, computing new solutions in each iteration of an interactive method is not feasible because of the long periods of time a DM would have to wait to see new solutions. Instead, we can use an interactive multiobjective optimization method that works on a computationally less expensive surrogate problem based on a pre-computed representation of Pareto optimal solutions. To get a representation, we can use, e.g., some a posteriori methods like EMO methods. Here we use this simple, yet quite effective, way as an example of combining methods from the *desdeo-mcdm* and *desdeo-emo* packages.

```

1 from desdeo_emo.EAs import NSGAIII
2
3 evolver = NSGAIII(mo_problem, interact=False)
4
5 while evolver.continue_evolution():
6     evolver.iterate()
7
8 individuals, pareto_set = evolver.end()

```

**SOURCECODE 11.** Generating a representation of Pareto optimal solutions for problem (2) using the *desdeo-emo* package.

We first generate a representative set approximating Pareto optimal solutions for the river pollution problem (2) as shown in Source code 11. For this, we apply NSGA-III (activated by using the `interact=False` argument) as an a posteriori method to get solutions for the problem as implemented in Source code 4. The method is run until a pre-determined termination criterion is met (the default being 1000 generations). After this, we can use the `end` method of the `evolver` object to extract a representation of the Pareto front (i.e., non-dominated solutions) from the population as `individuals` (decision vectors) and `pareto_set` (objective vectors).

We then apply the E-NAUTILUS method [36] with the generated set of nondominated solutions as its input. It also needs estimates of the ideal and nadir points, typically estimated from the available solutions. However, because we have previously computed (in Section IV-B) the ideal

and (estimated) nadir points, we apply them. As mentioned in Section II-D, the solution process starts with an inferior solution and gradually approaches the Pareto front.

In Source code 12, we set up the E-NAUTILUS method using the `ENautilus` class from the *desdeo-mcdm* package. We invoke the `start` method as we did in the case of NIMBUS in Section IV-B to start the solution process. We can get a hint on how to progress by printing the message stored in the request as done in Console 5.

```

1 from desdeo_mcdm.interactive import ENautilus
2
3 # 'pareto_set' stores the set of solutions computed using NSGA-III
4 method = ENautilus(pareto_set, ideal, nadir)
5
6 enautilus_request = method.start()

```

**SOURCECODE 12.** Initializing the E-NAUTILUS method using the set of solutions computed using NSGA-III and the previously computed ideal and nadir points of problem (2).

```

>>>print(enautilus_request.content["message"])
Please specify the number of iterations as
↪ 'n_iterations'
to be carried out, and how many intermediate points to
↪ show as 'n_points'.

```

**CONSOLE 5.** The help message returned by starting the E-NAUTILUS method.

```

1 response = {"n_iterations": 5, "n_points": 3}
2
3 enautilus_request.response = response
4
5 enautilus_request = method.iterate(request)

```

**SOURCECODE 13.** Specifying the number of iterations to be carried out and the number of points to be shown in each iteration of the E-NAUTILUS method.

A response to the request returned by the `start` method is then defined in Source code 13. We choose five iterations and want to see three intermediate points after each iteration. We then continue iterating and get a new request from invoking the `iterate` method, which contains the message displayed in Console 6.

```

>>>print(enautilus_request.content["message"])
Please select the most preferred point by index as
↪ 'preferred_point_index'

```

**CONSOLE 6.** The help message in a request returned from iterating the E-NAUTILUS method after it has been started.

To address the message shown in Console 6, we first inspect the intermediate points and bounds of the reachable solutions computed in the first iteration of E-NAUTILUS in Console 7.

In Source code 14, we define a response to the current request and continue iterating by invoking the `iterate` method. Subsequent iterations are carried out as shown in

```
>>>print("Points:", enautilus_request.content["points"])
Points:
[[ -4.985, -2.911, -1.562, -3.470, 0.347 ]
 [ -5.012, -2.853, -1.460, -2.229, 0.325 ]
 [ -5.067, -2.915, -0.395, -3.489, 0.349 ]]
>>>print("Upper bounds:",
↳ enautilus_request.content["upper_bounds"])
Upper bounds:
[[ -4.756, -3.100, -0.691, -2.094, 0.35 ]
 [ -4.808, -2.927, -3.204, -1.967, 0.35 ]
 [ -4.752, -3.100, -0.406, -2.094, 0.35 ]]
>>>print("Lower bounds:",
↳ enautilus_request.content["lower_bounds"])
Lower bounds:
[[ -6.315, -3.430, -7.443, -11.158, 0.196 ]
 [ -6.280, -3.347, -7.425, -7.785, 0.009 ]
 [ -6.338, -3.440, -7.401, -11.511, 0.196 ]]
```

**CONSOLE 7.** Printing the intermediate points, upper bounds, and lower bounds computed in the first iterations of E-NAUTILUS.

```
1 response = {"preferred_point_index": 2}
2
3 enautilus_request.response = response
4
5 enautilus_request = method.iterate(enautilus_request)
```

**SOURCECODE 14.** Expressing preference to be the second point shown in Console 7 and iterating.

Source code 14. We show an example of this using a graphical UI in the next subsection.

This way of exploring an existing representation of a Pareto front is well suited for computationally expensive problems since no expensive function evaluations are needed when the DM is involved. Even though the problem in this example is not really computationally expensive, the process of first using an EMO method to compute a representation of the Pareto optimal front, and then exploring it using the E-NAUTILUS method is identical in a computationally expensive case.

#### E. USE CASE 4: SWITCHING METHODS

As interactive multiobjective optimization methods vary in the type of preference information they require from a DM and the type of information they provide to the DM, it is sometimes desirable to switch between iterations to a method that is better suited to the changing needs of the DM. The DESDEO environment enables this kind of a switch even between different types of methods. To illustrate this, we consider an example, where we have finished iterating with the E-NAUTILUS method as described in Subsection IV-D and arrived at the solution  $[-6.27116931, -2.80042652, -3.46795271, -6.57327201, 0.31967811]$ . Since this method uses a set of solutions approximating the Pareto front, we can improve the solution by utilizing the synchronous NIMBUS method and considering the original problem (2) in its analytical form. We can also think that we have applied E-NAUTILUS as a trade-off-free method to find a good starting point for NIMBUS and avoided anchoring at a randomly selected starting point. From NIMBUS, we then switch to

applying the reference point method [33], that is, change the preference information type from classifying the objectives to providing a reference point.

To begin, we instantiate a NIMBUS object with the solution we arrived at with E-NAUTILUS. We use this solution to derive a Pareto optimal solution that NIMBUS starts with in Source code 15. This solution is shown in Console 8. Small improvements were made in the values of the third and fourth objectives while the other objective values remained unchanged. This is also an example of a possible realization of the post-decision-making stage (mentioned at the beginning of Section IV) to assure the Pareto optimality of the solution found using E-NAUTILUS since EMO methods (used here to generate the input set for E-NAUTILUS) cannot guarantee Pareto optimality.

```
1 nimbus_method = NIMBUS(mo_problem,
↳ starting_point=np.array([-6.27116931,
↳ -2.80042652, -3.46795271, -6.57327201,
↳ 0.31967811]))
2 classification_request, _ = nimbus_method.start()
3 solution =
↳ classification_request.content["objective_values"]
```

**SOURCECODE 15.** Instantiating a NIMBUS object with a specified starting point and starting the method.

```
>>>print("Improved starting point:", solution)
Improved starting point: [-6.27116931 -2.80042652
↳ -3.46795286 -6.57327759 0.31967811]
```

**CONSOLE 8.** Printing the solution to be classified in the first iteration of synchronous NIMBUS in Source code 15.

```
1 response = {
2     "classifications": [ ">=", ">=", "=", "<=", "<=" ],
3     "levels": [-6.0, -2.78, 0, -5.5, 0.28],
4     "number_of_solutions": 4,
5 }
6
7 classification_request.response = response
8
9 save_request, _ =
↳ nimbus_method.iterate(classification_request)
10
11 alternatives = np.array(save_request["objectives"])
```

**SOURCECODE 16.** Providing classification to synchronous NIMBUS and iterating the method further.

Next, we take an iteration with the synchronous NIMBUS using the classification shown in Source code 16. Based on this preference information, the method provides four new Pareto optimal solutions shown in Console 9. While inspecting the solutions, we find the first to our liking, but we would next like to provide a reference point instead of a classification. Thus, we switch to using the reference point method in the *desdeo-mcdm* module. We initialize the reference point method by instantiating a `ReferencePointMethod` object as done in Source code 17.

We provide the best (i.e., the solution we like the most) NIMBUS solution  $[-6.06739, -2.79173,$

```
>>>print("New solutions based on the classification:",
↪ alternatives)
New solutions based on the classification:
[[-6.06739 -2.79173 -5.96145 -6.57333 0.30863]
 [-6.17792 -2.79803 -5.09184 -5.39729 0.28469]
 [-6.17191 -2.79781 -5.15768 -5.38138 0.28427]
 [-6.15075 -2.79704 -5.36753 -5.33890 0.28314]]
```

**CONSOLE 9.** Printing the new solutions computed in Sourcecode 16.

```
1 from desdeo_mcdm.interactive import
↪ ReferencePointMethod
2
3 rp_method = ReferencePointMethod(mo_problem,
↪ mo_problem.ideal, mo_problem.nadir)
4 initial_request = rp_method.start()
```

**SOURCECODE 17.** Instantiating a reference point method object and starting it.

-5.96145, -6.57333, 0.30863]) as the reference point in Source code 18 by defining the reference point as a part of the response. We get the solutions shown in Console 10 and since the first one is so similar to the reference point, we stop the solution process and select the first solution as the final one.

```
1 response = {
2     "reference_point": np.array([-6.06739, -2.79173,
↪ -5.96145, -6.57333, 0.30863]),
3 }
4
5 initial_request.response = response
6
7 rp_request = rp_method.iterate(initial_request)
8
9 alternatives =
↪ np.array(rp_request.content["additional_solutions"])
```

**SOURCECODE 18.** Iterating with the reference point method by providing a reference point.

```
>>>print("Alternative solutions:", alternatives)
Alternative solutions:
[[-6.06738 -2.79173 -5.96151 -6.57720 0.30869]
 [-6.06739 -2.78816 -5.96146 -11.62453 0.34999]
 [-6.06827 -2.79173 -5.95658 -6.59358 0.30895]
 [-6.06739 -2.79173 -5.96146 -6.57853 0.30871]
 [-6.06810 -2.79162 -5.95754 -6.67149 0.31016]]
```

**CONSOLE 10.** Printing the alternative solutions computed by the reference point method after providing the reference point as done in Source code 18.

If we were not satisfied yet, we could continue iterating by providing a new reference point in a similar way to what was done in Source code 18. We could also switch back to the synchronous NIMBUS method by initializing a NIMBUS object once more, as was done in Source Code 15. In principle, we could also switch to an EMO method, for example, by providing one of the solutions as a reference point to RVEA similar to how it was done in Source code 10, while also switching back to the surrogate version of problem (2). But in this case it makes no sense to switch from Pareto optimal to approximated solutions. Naturally,

we are not limited to the interactive methods considered in the use cases, but any method in DESDEO is applicable. Without DESDEO, we would be forced to resort to switching our whole working environment, which may require needless repetition, for example, redefining the same problem multiple times (and possibly in a different syntax). Thanks to DESDEO, we have all the methods, problems, and other relevant information (e.g., solutions computed with different methods) in the same environment, which allows to readily switch methods and re-use already created information.

## F. SOFTWARE APPLICATIONS BUILT UTILIZING DESDEO

So far, we have not really discussed UIs in the DESDEO framework apart from using a console environment. However, DESDEO is easy to extend to build more advanced software applications, such as graphical user interfaces (GUIs), which facilitate interaction between DMs and interactive methods. In this section, we explore an example of such a GUI implemented for a method in the *desdeo-mcdm* package. It is naturally possible to implement similar GUIs for methods in the *desdeo-emo* package as well.

We consider an interface implemented for E-NAUTILUS. We have furthermore chosen a web interface because they are accessible to anyone through any modern web browser. The interface has been developed using the Python libraries *plotly* and *plotly-dash* (<https://plotly.com/>) due to their ease of use and versatility for developing web interfaces. However, there is a significant lack in support for interactive visualizations in these libraries, which we had to circumvent, leading to a lack in general usability.

The web interface for E-NAUTILUS can be seen in Figure 4. At the top of the interface, we have controls for the DM to engage with E-NAUTILUS: the DM can choose the most preferred intermediate point (labeled as ‘candidate’ in the figure) by using the radio buttons and click on the ‘ITERATE’-button to continue iterating. Below the controls, there are three different ways to visualize information about the intermediate points calculated: at the top left, a spider plot showing the intermediate points (solid lines) and the best reachable objective function values from each point (dashed lines); at the top right, a parallel coordinate plot showing only the intermediate points with the currently selected point (using radio buttons) being highlighted in red; and at the very bottom, the values of each intermediate point and their best reachable values in a table with the currently selected intermediate point highlighted in blue.

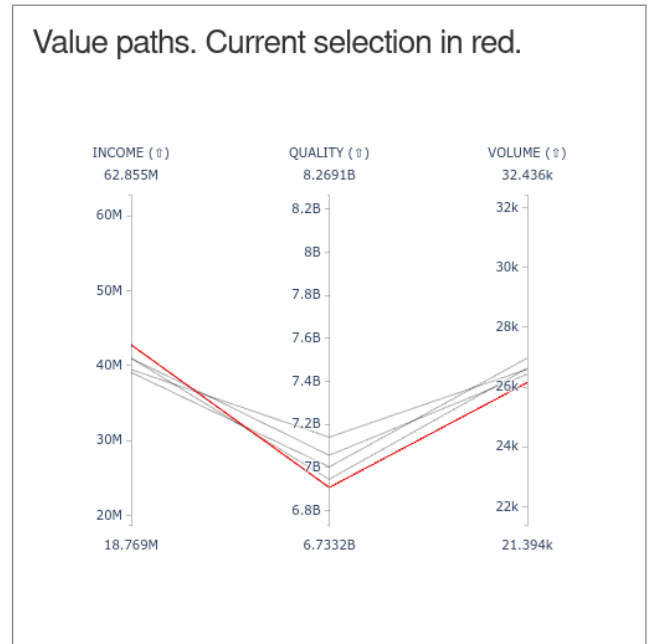
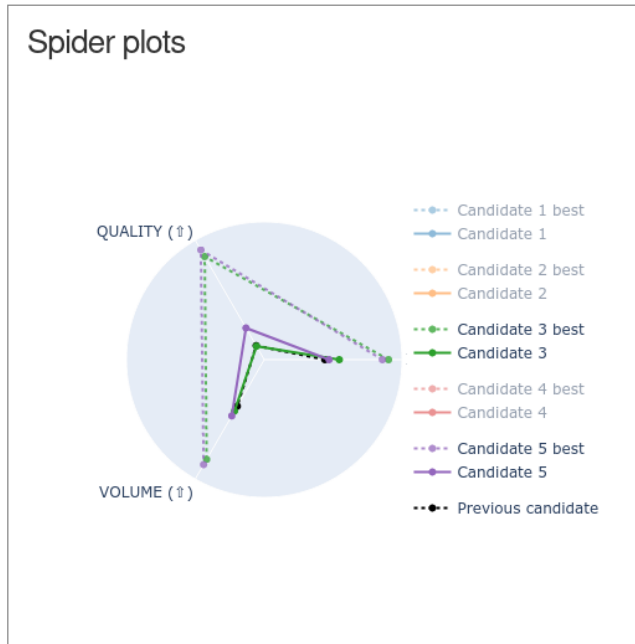
The spider plot in Figure 4 is worth a closer look. First, each intermediate point can be explored by clicking the respective point on the legend to the right of the plot. Second, the plot also shows in black the intermediate point chosen by the DM in the previous iteration. Showing the previously chosen point was desired by a real DM in a practical application where this interface was used. This is an example of a subjective need that may arise when interacting with real DMs. Lastly, it is worth comparing the information in Figure 4 to the information outputted in Console 7 to see that the information

## E-NAUTILUS: Iterations left 4

Select the most preferred candidate and continue iterating.

- Candidate 1
- Candidate 2
- Candidate 3
- Candidate 4
- Candidate 5

ITERATE



### Tabled candidate objective values

Candidate	INCOME (€)	QUALITY (€)	VOLUME (€)
1	41,011,177.17	6,944,671,947.74	26,627.11
2	40,988,980.81	7,056,474,462.39	26,431.18
3	42,788,884.27	6,906,564,678.96	26,151.17
4	39,107,181.60	7,002,406,061.35	26,967.98
5	39,494,600.68	7,140,768,731.69	26,594.86

### Tabled candidate best reachable values

Best reachable	INCOME (€)	QUALITY (€)	VOLUME (€)
1	57,829,414.62	8,145,971,745.04	30,867.17
2	57,486,543.70	8,145,971,745.04	30,867.17
3	58,521,063.17	8,059,687,621.67	30,623.93
4	57,049,818.26	8,145,971,745.04	31,147.81
5	56,558,622.26	8,145,971,745.04	31,105.08

[Back to method index](#)

**FIGURE 4.** The GUI of the E-NAUTILUS method implemented in plotly-dash. A toy multiobjective optimization problem with three objectives (INCOME, QUALITY, VOLUME) to be maximized is shown.

shown for a single iteration in the web interface and the console are virtually the same. In practice, the presented interface simply handles the requests and responses

(discussed in Section III-B) as was done in Section IV-D. In the E-NAUTILUS GUI, we have a different multiobjective optimization problem with three objectives to be maximized



instead of problem (2). We have chosen a problem with fewer objectives for simplicity. Note that the arrows after the function names remind of the maximization.

The interface described for E-NAUTILUS is available online (<https://desdeo.it.jyu.fi/dash>) alongside an interface implementation for NAUTILUS Navigator as well. The source code for the interface shown is available on GitHub online.<sup>3</sup> To test the interfaces, we have provided the interested reader with toy data online.<sup>4</sup>

## V. POTENTIAL OF THE DESDEO FRAMEWORK

Because the DESDEO framework contains various interactive methods, it enables versatile ways of applying them. As said, the DM can conveniently switch the method during the solution process. This can be desirable if (s)he wants to change the type of preference information in the middle of the solution process or get different types of information about the problem. This opens up vast possibilities when the DM is not forced to stick with a single method to be applied but can select methods that best suit the different phases of the solution process (e.g., learning and decision phases [54]). This potential has been considered in [55], where a generic multi-agent architecture for interactive methods was proposed to support DMs in selecting the most suited interactive method based on preferred preference type and their needs in different phases during the solution process. Without a framework like DESDEO, switching the method is inconvenient; the problem to be solved must be connected to individual multiobjective optimization methods separately, and the solution history with the previous method is not easily available.

DESDEO has clear potential in allowing researchers to hybridize EMO and MCDM methods in novel ways. This potential is not just limited to the example seen in Section IV, where an EMO method was used to compute a representation of a Pareto front, which was then explored using an MCDM method. More innovative and advanced ways of combining not just methods but also their individual components are possible. This is because of the modular fashion in which the various multiobjective optimization methods have been implemented in DESDEO. Combining individual components enables the development of new interactive methods, which can be also included in DESDEO extending the framework further. The IOPIS algorithm, described in [46], is an example of such a method.

Moreover, DESDEO offers a promising basis for implementing new interactive multiobjective optimization methods that are not based on combining existing components. Due to the modular structure, a developer can easily reuse already implemented components and only add those that are not yet available (if needed). For example, the *desdeo-tools* package has a wide variety of different tools ranging from achievement scalarizing functions to fast nondominated sorting, which

can prove useful in implementing new methods. In addition, experimenting with new methods and ideas in multiobjective optimization is also made easy thanks to DESDEO and the reusability of its components. DESDEO can also encourage and lower the threshold for researchers to implement their methods as open source code, contributing to the openness of the research conducted in multiobjective optimization. This way, DESDEO has the potential and is on a good track to becoming a central hub for open implementations of interactive multiobjective optimization methods.

Apart from being interesting from an academic perspective, DESDEO can naturally be utilized for modeling and solving real-life problems from any field as long as the problem can be modeled as a multiobjective optimization problem. Depending on the type and requirements of the problem, DESDEO might still lack certain features necessary for modeling and solving the problem, which is also one of the current limitations of DESDEO. However, due to DESDEO's modular structure and open source nature, implementing these missing features is possible by anyone. For example, the underlying optimization methods for single-objective optimization problems arising in various interactive methods in DESDEO can be changed to better account for the type of problem being solved. Similarly, the crossover and mutation operations in EMO methods can also be customized if need be. Lastly, in modeling a data-driven multiobjective optimization problem, almost any surrogate model can be implemented and used. Obviously, existing features in DESDEO can be combined with new features as well allowing practitioners to save time and help them focus on solving the problem at hand. In this way, DESDEO can be extended to account for any kind of multiobjective optimization problem from any field while decreasing the potential workload for practitioners.

Being a software framework, DESDEO has a learning curve to it, which means that a certain level of proficiency in Python and multiobjective optimization is to be expected from the user. This clearly limits the size of the potential user base of DESDEO and is, therefore, one of the framework's major limitations at the present time. We already offer a written documentation of DESDEO's features, but to make DESDEO even more accessible, we plan on including more topical guides in the documentation on how to use DESDEO (such as the ones presented in Section IV) and consider producing tutorial videos on how to use DESDEO in the future. This should help broaden DESDEO's user base and allow users to extend DESDEO to meet their individual needs. All of this will help DESDEO grow further as a software framework.

Comparison and identifying the best suited method for various needs are important. DESDEO does offer very promising opportunities for comparing and validating different interactive methods. This is vital and demanding because the DM plays an important role in the solution process and conducting experiments with human participants is challenging. To be able to compare interactive methods, their performance needs

<sup>3</sup><https://github.com/industrial-optimization-group/desdeo-dash>

<sup>4</sup>[https://github.com/industrial-optimization-group/DESDEO/blob/master/docs/notebooks/data/toy\\_data.csv](https://github.com/industrial-optimization-group/DESDEO/blob/master/docs/notebooks/data/toy_data.csv)

to be evaluated and validated using appropriate quality indicators. To the best of our knowledge, no quality indicators for interactive methods have been proposed. For such quality indicators, the desirable properties that qualify interactive solution processes should be defined. In [56], a systematic literature review of the assessments of interactive methods is provided along with desirable properties for interactive methods. This can be considered as the initial step towards developing quality indicators for interactive methods. Moreover, there has been some interest in comparing interactive methods with so-called artificial DMs in the literature (e.g., [57]–[59]). Within the DESDEO framework, an artificial DM has recently been proposed to compare reference point-based interactive EMO methods [60]. DESDEO provides an excellent platform for comparisons because it involves various interactive methods within the same framework. To utilize the opportunities available, we need artificial DMs capable of handling different types of preferences and methods.

## VI. CONCLUSION

In this paper, we fill a gap in the optimization software available. We introduced DESDEO: an open source multi-objective optimization framework implemented in Python. DESDEO makes interactive multiobjective optimization methods openly available for both users and developers. We introduced the modular structure of DESDEO and its different packages and their modules. We also described the purpose of each package and its dependencies and the framework's external dependencies. Besides, with a five-objective optimization problem, we demonstrated how to use the DESDEO framework to define different types of problems (i.e., with analytical expressions, data-driven, and computationally expensive problems) and solve them by applying and hybridizing interactive multiobjective optimization methods of MCDM and EMO types.

The modularity of DESDEO eases developing new methods and offers a convenient possibility of comparing different interactive methods. Furthermore, implementing different types of methods in the same framework, as done in DESDEO, will start a new era in hybridization and allows the DM to switch between methods in various iterations of the solution process.

We also noted that for efficient interaction with the DM, there is a need for interactive visualization tools and suitable (graphical) UIs in multiobjective optimization, which is lacking in the literature. We are addressing this practical concern by actively developing a D3 (<https://d3js.org/>) based TypeScript library of interactive visualization components, such as interactive parallel coordinate plots within DESDEO. Our primary goal with this library is to provide the multiobjective optimization community with new and needed tools to build their own interfaces for interactive multiobjective optimization; similar to the example seen in Section IV-F. To facilitate the use of the packages in DESDEO to be extended to other software, such as web based interfaces, we are also working

on a web API (application programming interface) through which we can expose interactive methods in DESDEO to enable their use in a variety of applications. The interested reader can follow the latest developments of DESDEO via its homepage ([desdeo.it.jyu.fi](https://desdeo.it.jyu.fi)). The realization of this vision should make interactive multiobjective optimization methods much more accessible in the future, not just for researchers developing them, but also for the needs of applications in various fields.

## ACKNOWLEDGMENT

The authors would like to thank all of those who have contributed to DESDEO in the past. Especially, they would like to thank Giomara Lárraga, Johanna Silvennoinen, Pouya Aghaei Pour, Juuso Pajasmaa, Stefan Otayagich, and Antti Luopajarvi. They would also like to thank Vesa Ojalehto for his pioneering work in developing the old version of the DESDEO framework. This work is a part of the thematic research area Decision Analytics Utilizing Causal Models and Multiobjective Optimization (DEMO, [jyu.fi/demo](https://desdeo.it.jyu.fi/demo)) at the University of Jyväskylä.

## REFERENCES

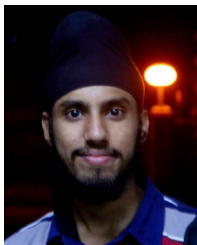
- [1] C.-L. Hwang and A. Masud, *Multiple Objective Decision Making—Methods and Applications: A State-of-the-Art Survey*. Berlin, Germany: Springer, 1979.
- [2] V. Chankong and Y. Y. Haimes, *Multiobjective Decision Making: Theory and Methodology*. New York, NY, USA: Elsevier, 1983.
- [3] K. Miettinen, *Nonlinear Multiobjective Optimization*. Boston, MA, USA: Kluwer, 1999.
- [4] C. A. C. Coello, G. B. Lamont, and D. A. Van Veldhuizen, *Evolutionary Algorithms for Solving Multi-Objective Problems*, 2nd ed. New York, NY, USA: Springer, 2007.
- [5] K. Deb, *Multi-Objective Optimization Using Evolutionary Algorithms*. Chichester, U.K.: Wiley, 2001.
- [6] K. Miettinen and M. M. Mäkelä, "On scalarizing functions in multiobjective optimization," *OR Spectr.*, vol. 24, no. 2, pp. 193–213, May 2002.
- [7] F. Ruiz, M. Luque, and J. M. Cabello, "A classification of the weighting schemes in reference point procedures for multiobjective programming," *J. Oper. Res. Soc.*, vol. 60, no. 4, pp. 544–553, Apr. 2009.
- [8] K. Miettinen, J. Hakanen, and D. Podkopaev, "Interactive nonlinear multi-objective optimization methods," in *Multiple Criteria Decision Analysis: State of the Art Surveys*, 2nd ed., S. Greco, M. Ehrgott, and J. Figueira, Eds. New York, NY, USA: Springer, 2016, pp. 931–980.
- [9] M. Luque, F. Ruiz, and K. Miettinen, "Global formulation for interactive multiobjective optimization," *OR Spectr.*, vol. 33, no. 1, pp. 27–48, Jan. 2011.
- [10] J. Branke, K. Deb, K. Miettinen, and R. Slowinski, Eds., *Multiobjective Optimization: Interactive and Evolutionary Approaches*. Berlin, Germany: Springer, 2008.
- [11] B. Xin, L. Chen, J. Chen, H. Ishibuchi, K. Hirota, and B. Liu, "Interactive multiobjective optimization: A review of the state-of-the-art," *IEEE Access*, vol. 6, pp. 41256–41279, 2018.
- [12] V. Ojalehto and K. Miettinen, "DESDEO: An open framework for interactive multiobjective optimization," in *Multiple Criteria Decision Making and Aiding*, S. Huber, M. J. Geiger, and A. T. de Almeida, Eds. Cham, Switzerland: Springer, 2019, pp. 67–94.
- [13] G. Rossus, "Python reference manual," NLD, Centrum voor Wiskunde en Informatica, Amsterdam, The Netherlands, Tech. Rep., 1995.
- [14] R. Benayoun, J. de Montgolfier, J. Tergny, and O. Laritchev, "Linear programming with multiple objective functions: Step method (STEM)," *Math. Program.*, vol. 1, no. 1, pp. 366–375, Dec. 1971.
- [15] X. Wang, Y. Jin, S. Schmitt, and M. Olhofer, "An adaptive Bayesian approach to surrogate-assisted evolutionary multi-objective optimization," *Inf. Sci.*, vol. 519, pp. 317–331, May 2020.

- [16] S. N. Qasem, S. M. Shamsuddin, S. Z. M. Hashim, M. Darus, and E. Al-Shammari, "Memetic multiobjective particle swarm optimization-based radial basis function network for classification problems," *Inf. Sci.*, vol. 239, pp. 165–190, Aug. 2013.
- [17] J. Knowles, "ParEGO: A hybrid algorithm with on-line landscape approximation for expensive multiobjective optimization problems," *IEEE Trans. Evol. Comput.*, vol. 10, no. 1, pp. 50–66, Feb. 2006.
- [18] M. Li, G. Li, and S. Azarm, "A kriging metamodel assisted multi-objective genetic algorithm for design optimization," *J. Mech. Design*, vol. 130, no. 3, pp. 1–10, Mar. 2008.
- [19] H. Aytuğ and S. Sayın, "Using support vector machines to learn the efficient set in multiple objective discrete optimization," *Eur. J. Oper. Res.*, vol. 193, no. 2, pp. 510–519, Mar. 2009.
- [20] G. Kourakos and A. Mantoglou, "Development of a multi-objective optimization algorithm using surrogate models for coastal aquifer management," *J. Hydrol.*, vol. 479, pp. 13–23, Feb. 2013.
- [21] K. Mitra and S. Majumder, "Successive approximate model based multi-objective optimization for an industrial straight grate iron ore induration process using evolutionary algorithm," *Chem. Eng. Sci.*, vol. 66, no. 15, pp. 3471–3481, Aug. 2011.
- [22] F.-A. Fortin, F.-M. De Rainville, M.-A. G. Gardner, M. Parizeau, and C. Gagné, "DEAP: Evolutionary algorithms made easy," *J. Mach. Lang. Res.*, vol. 13, pp. 2171–2175, Jul. 2012.
- [23] A. Garrett, *Inspyred: Bio-inspired algorithms in Python*. Accessed: Nov. 19, 2020. [Online]. Available: <https://github.com/aarongarrett/inspyred>
- [24] X. Gandibleux, G. Soleilhac, A. Przybylski, and S. Ruzika, "vOptSolver: An open source software environment for multiobjective mathematical optimization," in *Proc. 21st Conf. Int. Fed. Oper. Res. Societies (IFORS)*, 2017, pp. 17–21.
- [25] D. Hadka, *Platypus: Multiobjective Optimization in Python*. Accessed: Nov. 19, 2020. [Online]. Available: <https://platypus.readthedocs.io>
- [26] D. Hadka, *MOEA Framework: A Free and Open Source Java Framework for Multiobjective Optimization*. Accessed: Dec. 1, 2020. [Online]. Available: <http://moaeframework.org/>
- [27] D. Izzo and F. Biscani, *PyGMO: Python Parallel Global Multi-objective Optimizer*. Accessed: Nov. 19, 2020. [Online]. Available: <https://esa.github.io/pygmo>
- [28] F. Biscani, D. Izzo, and C. H. Yam, "A global optimisation toolbox for massively parallel engineering optimisation," 2010, *arXiv:1004.3824*.
- [29] A. Benítez-Hidalgo, A. J. Nebro, J. García-Nieto, I. Oregi, and J. Del Ser, "jMetalPy: A Python framework for multi-objective optimization with metaheuristics," *Swarm Evol. Comput.*, vol. 51, Dec. 2019, Art. no. 100598.
- [30] J. J. Durillo and A. J. Nebro, "jMetal: A Java framework for multi-objective optimization," *Adv. Eng. Softw.*, vol. 42, no. 10, pp. 760–771, 2011.
- [31] J. Blank and K. Deb, "Pymoo: Multi-objective optimization in Python," *IEEE Access*, vol. 8, pp. 89497–89509, 2020.
- [32] Y. Tian, R. Cheng, X. Zhang, and Y. Jin, "PlatEMO: A MATLAB platform for evolutionary multi-objective optimization," *IEEE Comput. Intell. Mag.*, vol. 12, no. 4, pp. 73–87, Nov. 2017.
- [33] A. P. Wierzbicki, "A mathematical basis for satisficing decision making," *Math. Model.*, vol. 3, no. 5, pp. 391–405, 1982.
- [34] K. Miettinen and M. M. Mäkelä, "Synchronous approach in interactive multiobjective optimization," *Eur. J. Oper. Res.*, vol. 170, no. 3, pp. 909–922, May 2006.
- [35] K. Miettinen and F. Ruiz, "NAUTILUS framework: Towards trade-off-free interaction in multiobjective optimization," *J. Bus. Econ.*, vol. 86, nos. 1–2, pp. 5–21, Jan. 2016.
- [36] A. B. Ruiz, K. Sindhya, K. Miettinen, F. Ruiz, and M. Luque, "E-NAUTILUS: A decision support system for complex multiobjective optimization problems based on the NAUTILUS method," *Eur. J. Oper. Res.*, vol. 246, no. 1, pp. 218–231, Oct. 2015.
- [37] R. Cheng, Y. Jin, M. Olhofer, and B. Sendhoff, "A reference vector guided evolutionary algorithm for many-objective optimization," *IEEE Trans. Evol. Comput.*, vol. 20, no. 5, pp. 773–791, Oct. 2016.
- [38] K. Deb and H. Jain, "An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part I: Solving problems with box constraints," *IEEE Trans. Evol. Comput.*, vol. 18, no. 4, pp. 577–601, Apr. 2013.
- [39] K. Miettinen, P. Eskelinen, F. Ruiz, and M. Luque, "NAUTILUS method: An interactive technique in multiobjective optimization based on the nadir point," *Eur. J. Oper. Res.*, vol. 206, no. 2, pp. 426–434, Oct. 2010.
- [40] K. Miettinen, D. Podkopaev, F. Ruiz, and M. Luque, "A new preference handling technique for interactive multiobjective optimization without trading-off," *J. Global Optim.*, vol. 63, no. 4, pp. 633–652, Dec. 2015.
- [41] A. B. Ruiz, F. Ruiz, K. Miettinen, L. Delgado-Antequera, and V. Ojalehto, "NAUTILUS Navigator: Free search interactive multiobjective optimization without trading-off," *J. Global Optim.*, vol. 74, no. 2, pp. 213–231, Jun. 2019.
- [42] M. Hartikainen, K. Miettinen, and K. Klamroth, "Interactive nonconvex Pareto navigator for multiobjective optimization," *Eur. J. Oper. Res.*, vol. 275, no. 1, pp. 238–251, May 2019.
- [43] J. Hakanen, T. Chugh, K. Sindhya, Y. Jin, and K. Miettinen, "Connections of reference vectors and different types of preference information in interactive multiobjective evolutionary algorithms," in *Proc. IEEE Symp. Ser. Comput. Intell. (SSCI)*, Dec. 2016, pp. 1–8.
- [44] K. Deb, L. Thiele, M. Laumanns, and E. Zitzler, "Scalable test problems for evolutionary multiobjective optimization," in *Evolutionary Multiobjective Optimization: Theoretical Advances and Applications*, A. Abraham, L. Jain, and R. Goldberg, Eds. London, U.K.: Springer, 2005, pp. 105–145.
- [45] A. P. Wierzbicki, "On the completeness and constructiveness of parametric characterizations to vector optimization problems," *Oper.-Res.-Spektrum*, vol. 8, no. 2, pp. 73–87, Jun. 1986.
- [46] B. S. Saini, J. Hakanen, and K. Miettinen, "A new paradigm in interactive evolutionary multiobjective optimization," in *Parallel Problem Solving From Nature—PPSN XVI*, T. Bäck, M. Preuss, A. Deutz, H. Wang, C. Doerr, M. Emmerich, and H. Trautmann, Eds. Cham, Switzerland: Springer, 2020, pp. 243–256.
- [47] S. van der Walt, S. C. Colbert, and G. Varoquaux, "The NumPy array: A structure for efficient numerical computation," *Comput. Sci. Eng.*, vol. 13, no. 2, pp. 22–30, 2011.
- [48] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, and S. J. Van Der Walt, "SciPy 1.0: Fundamental algorithms for scientific computing in Python," *Nature Methods*, vol. 17, no. 3, pp. 261–272, Feb. 2020.
- [49] W. McKinney, "Data structures for statistical computing in Python," in *Proc. 9th Python Sci. Conf.*, S. van der Walt and J. Millman, Eds. SciPy, 2010, pp. 56–61.
- [50] I. Steponavičė, S. Ruuska, and K. Miettinen, "A solution process for simulation-based multiobjective design optimization with an application in the paper industry," *Comput.-Aided Des.*, vol. 47, pp. 45–58, Feb. 2014.
- [51] S. C. Narula and H. R. Weistroffer, "A flexible method for nonlinear multicriteria decision-making problems," *IEEE Trans. Syst., Man, Cybern.*, vol. 19, no. 4, pp. 883–887, Jul. 1989.
- [52] K. Miettinen and M. M. Mäkelä, "Interactive method NIMBUS for nondifferentiable multiobjective optimization problems," in *Multicriteria Analysis*, J. Clímaco, Ed. Berlin, Germany: Springer, 1997, pp. 310–319.
- [53] M. D. McKay, R. J. Beckman, and W. J. Conover, "A comparison of three methods for selecting values of input variables in the analysis of output from a computer code," *Technometrics*, vol. 21, no. 2, pp. 239–245, 1979.
- [54] K. Miettinen, F. Ruiz, and A. P. Wierzbicki, "Introduction to multiobjective optimization: Interactive approaches," in *Multiobjective Optimization: Interactive and Evolutionary Approaches*, J. Branke, K. Deb, K. Miettinen, and R. Slowinski, Eds. Berlin, Germany: Springer, 2008, pp. 27–57.
- [55] B. Afsar, D. Podkopaev, and K. Miettinen, "Data-driven interactive multiobjective optimization: Challenges and a generic multi-agent architecture," *Proc. Comput. Sci.*, vol. 176, pp. 281–290, Jan. 2020.
- [56] B. Afsar, K. Miettinen, and F. Ruiz, "Assessing the performance of interactive multiobjective optimization methods: A survey," *ACM Comput. Surveys*, vol. 54, no. 4, p. 85, 2021.
- [57] C. Barba-González, V. Ojalehto, J. M. García-Nieto, A. J. Nebro, K. Miettinen, and J. F. Aldana-Montes, "Artificial decision maker driven by PSO: An approach for testing reference point based interactive methods," in *Proc. 15th Int. Conf. Parallel Problem Solving Nature—PPSN XV*, A. Auger, C. M. Fonseca, N. Lourenço, P. Machado, L. Paquete, and D. Whitley, Eds. Cham, Switzerland: Springer, 2018, pp. 274–285.
- [58] S. Huber, M. J. Geiger, and M. Sevaux, "Simulation of preference information in an interactive reference point-based method for the bi-objective inventory routing problem," *J. Multi-Criteria Decis. Anal.*, vol. 22, nos. 1–2, pp. 17–35, Jan. 2015.
- [59] V. Ojalehto, D. Podkopaev, and K. Miettinen, "Towards automatic testing of reference point based interactive methods," in *Parallel Problem Solving From Nature—PPSN XIV*, J. Handl, E. Hart, P. R. Lewis, M. López-Ibáñez, G. Ochoa, and B. Paechter, Eds. Cham, Switzerland: Springer, 2016, pp. 483–492.

- [60] B. Afsar, K. Miettinen, and A. B. Ruiz, "An artificial decision maker for comparing reference point based interactive evolutionary multiobjective optimization methods," in *Evolutionary Multi-Criterion Optimization*, H. Ishibuchi, Q. Zhang, R. Cheng, K. Li, H. Li, H. Wang, and A. Zhou, Eds. Cham, Switzerland: Springer, 2021, pp. 619–631.



**G. MISITANO** received the M.Sc. degree from the University of Jyväskylä, in 2020, where he is currently pursuing the Doctoral degree with the Multiobjective Optimization Group. His research interest includes the interpretability aspects of interactive multiobjective optimization. This includes, but is not limited to, researching new ways to make interactive multiobjective optimization methods less opaque to the decision maker and analyst alike. In addition, he is interested in studying how to apply interpretable and explainable artificial intelligence to multiobjective optimization in general. He is also one of the main contributors to the DESDEO Framework.



one of the primary contributors to the DESDEO Framework.

**B. S. SAINI** received the M.Tech. degree from IIT Kharagpur, in 2018. He is currently pursuing the Doctoral degree with the Multiobjective Optimization Group, University of Jyväskylä. His research interests include multiobjective optimization, data visualization, data-driven optimization, and development of evolutionary algorithms. He has worked on many open source implementations of the methods from the aforementioned topics with a focus on modularity and interpretability. He is also



performance of interactive multiobjective optimization methods with both artificial and human decision-makers.

**B. AFSAR** received the Ph.D. degree in computer engineering from Ege University, Izmir, Turkey, in 2014. He is currently a Postdoctoral Researcher with the Multiobjective Optimization Group, University of Jyväskylä. His main research interests include multiobjective optimization, data-driven multi-criteria decision-making, evolutionary computation, interactive multiobjective optimization methods and their applications, and multi-agent systems. He is also working on assessing the



scenario planning, and decision-making under (deep) uncertainty.

**B. SHAVAZIPOUR** received the Ph.D. degree in operations research from the University of Cape Town, Cape Town, South Africa, in 2018. He is currently a Postdoctoral Researcher with the Multiobjective Optimization Group, University of Jyväskylä. His principal research interests include multiobjective optimization and multi-criteria decision-making both in theory and applications, mathematical programming, data analysis and impacts upon the data envelopment analysis,



for interactive multiobjective optimization methods ([desdeo.it.jyu.fi](http://desdeo.it.jyu.fi)). She has authored about 190 refereed journals, proceedings, and collection papers; edited 17 proceedings, collections, and special issues; and written a monograph on nonlinear multiobjective optimization. Her research interests include theory, methods, applications, and software of nonlinear multiobjective optimization. She is a member of the Finnish Academy of Science and Letters, Section of Science, and the Steering Committee of Evolutionary Multi-Criterion Optimization. She has been the President of the International Society on Multiple Criteria Decision Making (MCDM). She has received the Georg Cantor Award of the International Society on MCDM for developing innovative ideas. She belongs to the editorial board of seven international journals.

...